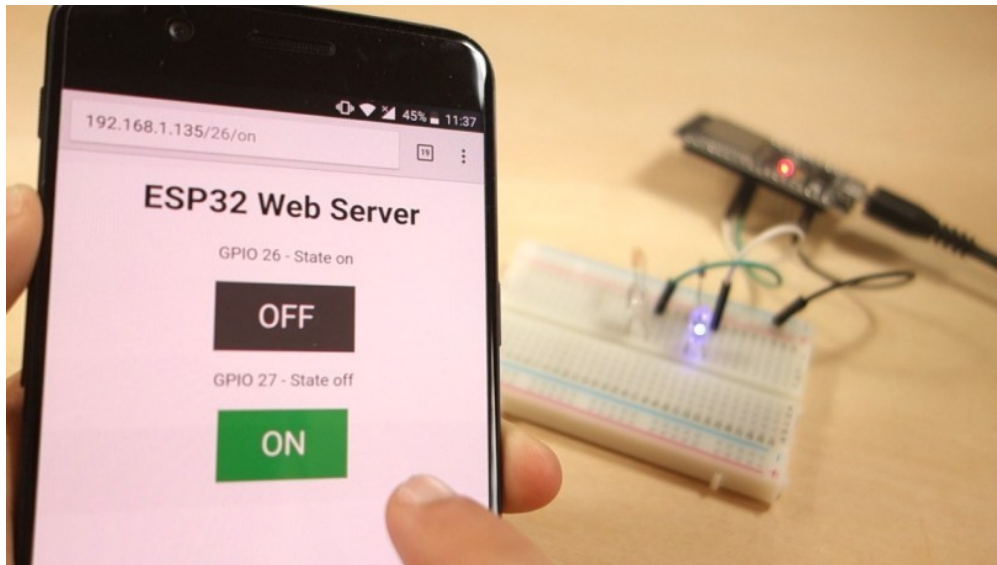


# ESP32 WIKI

## 1- ESP32 WEB SERVER

### ESP32 Web Server – Arduino IDE

In this project you'll create a standalone web server with an ESP32 that controls outputs (two LEDs) using the Arduino IDE programming environment. The web server is mobile responsive and can be accessed with any device that has a browser on the local network. We'll show you how to create the web server and how the code works step-by-step.



If you want to learn more about the ESP32, read [Getting Started Guide with ESP32](#).

### Watch the Video Tutorial

*This tutorial is available in video format (watch below) and in written format (continue reading this page).*

<https://youtu.be/ApGwxX6VVzk>

### Project Overview

Before going straight to the project, it is important to outline what our web server will do, so that it is easier to follow the steps later on.

- 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.

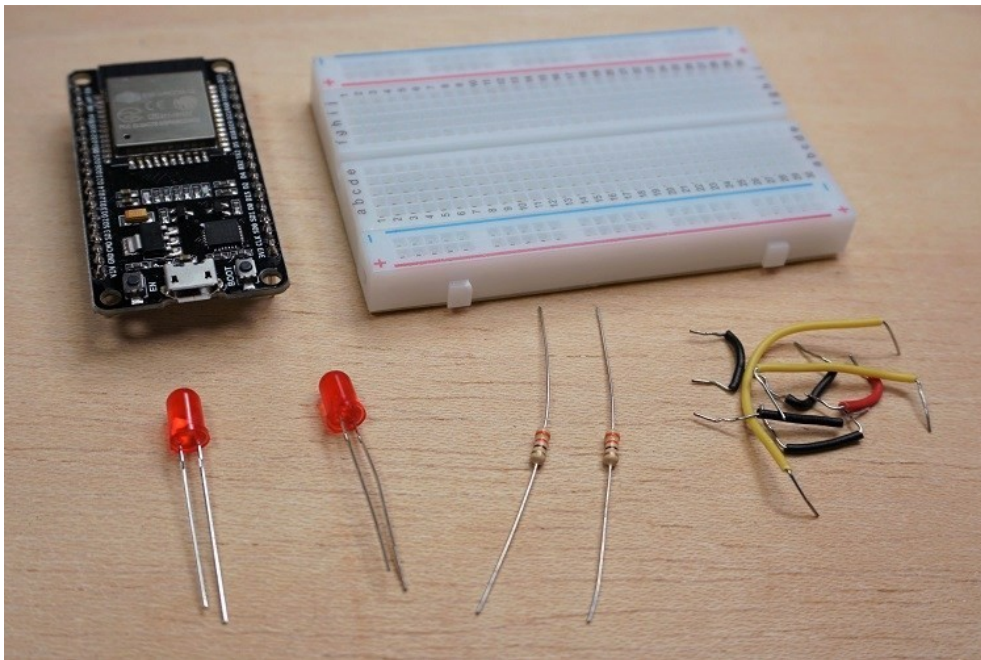
# Installing the ESP32 board in Arduino IDE

There's an add-on for the Arduino IDE that allows you to program the ESP32 using the Arduino IDE and its programming language. Follow one of the following tutorials to prepare your Arduino IDE:

- [Windows instructions – Installing the ESP32 Board in Arduino IDE](#)
- [Mac and Linux instructions – Installing the ESP32 Board in Arduino IDE](#)

## Parts Required

For this tutorial you'll need the following parts:



[ESP32 development board](#) – [read ESP32 Development Boards Review and Comparison](#)

- [2x 5mm LED](#)
- [2x 330 Ohm resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

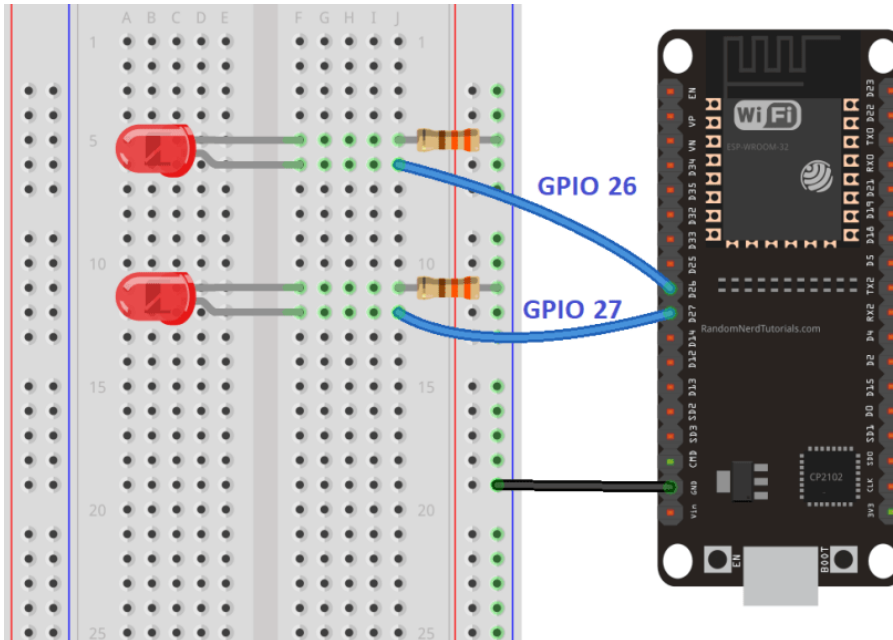
You can use the preceding links or go directly to [MakerAdvisor.com/tools](https://makeradvisor.com/tools) to find all the parts for your projects at the best price!



## Schematic

Start by building the circuit. Connect two LEDs to the ESP32 as shown in the following schematic diagram – one LED connected to GPIO 26, and the other to GPIO 27.

**Note:** We're using the ESP32 DEVKIT DOIT board with 36 pins. Before assembling the circuit, make sure you check the pinout for the board you're using.



# ESP32 Web Server Code

Here we provide the code that creates the ESP32 web server. Copy the following code to your Arduino IDE, but don't upload it yet. You need to make some changes to make it work for you.

```
/*  
  Rui Santos  
  Complete project details at http://randomnerdtutorials.com  
  */  
  
// Load Wi-Fi library  
#include <WiFi.h>  
  
// Replace with your network credentials  
const char* ssid      = "";  
const char* password = "";  
  
// Set web server port number to 80  
WiFiServer server(80);  
  
// Variable to store the HTTP request  
String header;  
  
// Auxiliar variables to store the current output state  
String output26State = "off";  
String output27State = "off";  
  
// Assign output variables to GPIO pins  
const int output26 = 26;  
const int output27 = 27;  
  
void setup() {  
  Serial.begin(115200);  
  // Initialize the output variables as outputs  
  pinMode(output26, OUTPUT);  
  pinMode(output27, OUTPUT);  
  // Set outputs to LOW  
  digitalWrite(output26, LOW);  
  digitalWrite(output27, 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,  
    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()) {           // loop while the client's
connected
        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 /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);
                    }

                    // 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 26
        client.println("<p>GPIO 26 - State " + 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>");
        }

        // Display current state, and ON/OFF buttons for GPIO 27
        client.println("<p>GPIO 27 - State " + 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>");

        // 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("");
}
}

```

## Setting Your Network Credentials

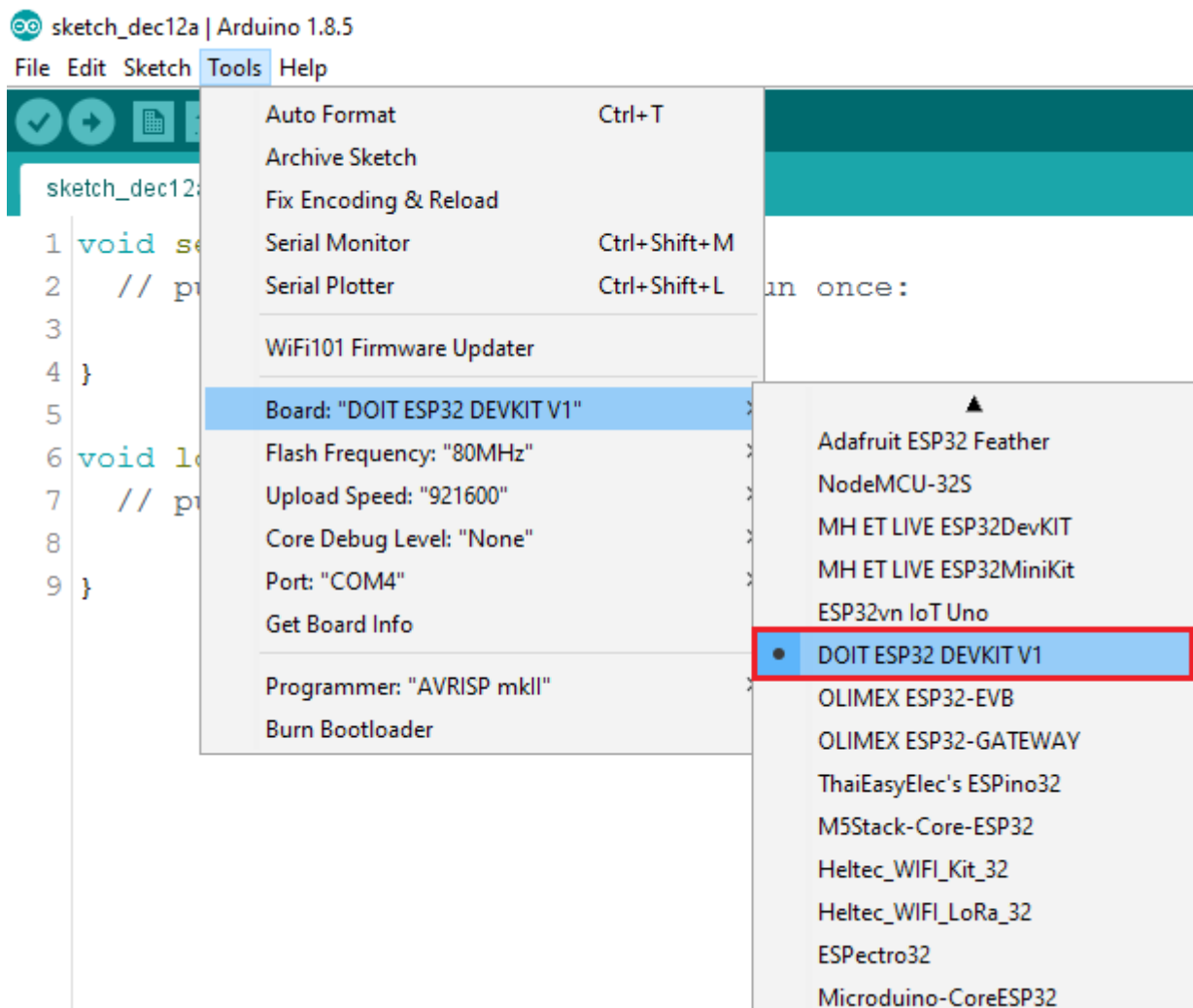
You need to modify the following lines with your network credentials: SSID and password. The code is well commented on where you should make the changes.

```
// Replace with your network credentials
const char* ssid      = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

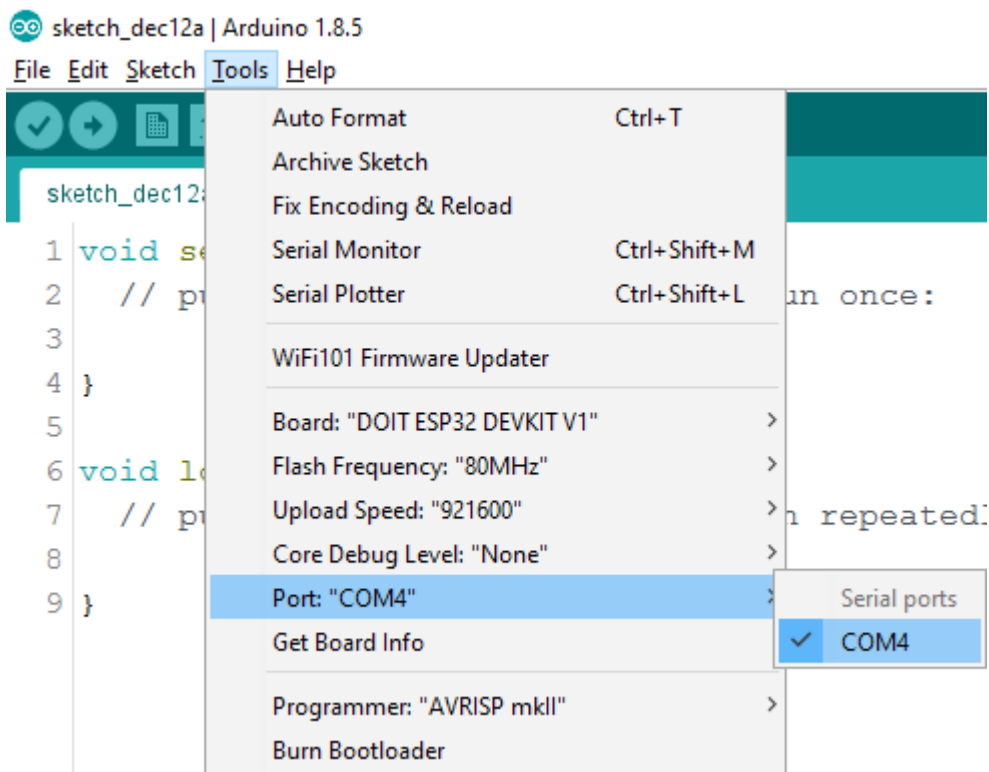
## Uploading the Code

Now, you can upload the code and the web server will work straight away. Follow the next steps to upload code to the ESP32:

- 1) Plug your ESP32 board in your computer;
- 2) In the Arduino IDE select your board in **Tools > Board** (in our case we're using the ESP32 DEVKIT DOIT board);



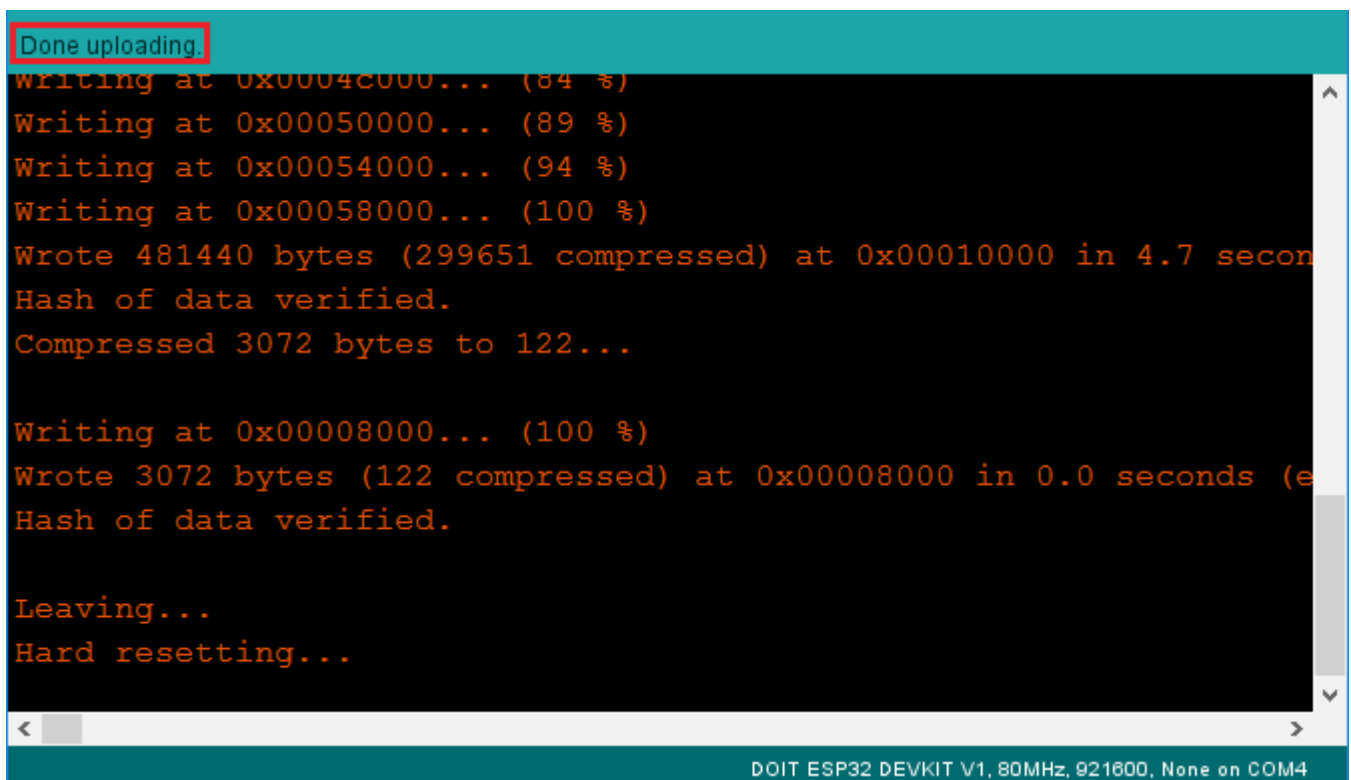
- 3) Select the COM port in **Tools > Port**.



4) Press the **Upload** button in the Arduino IDE and wait a few seconds while the code compiles and uploads to your board.



5) Wait for the “**Done uploading**” message.



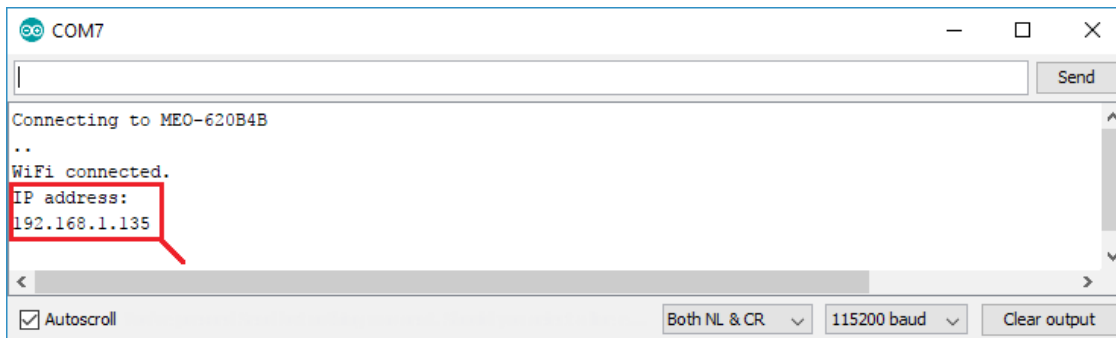


## Finding the ESP IP Address

After uploading the code, open the Serial Monitor at a baud rate of 115200.

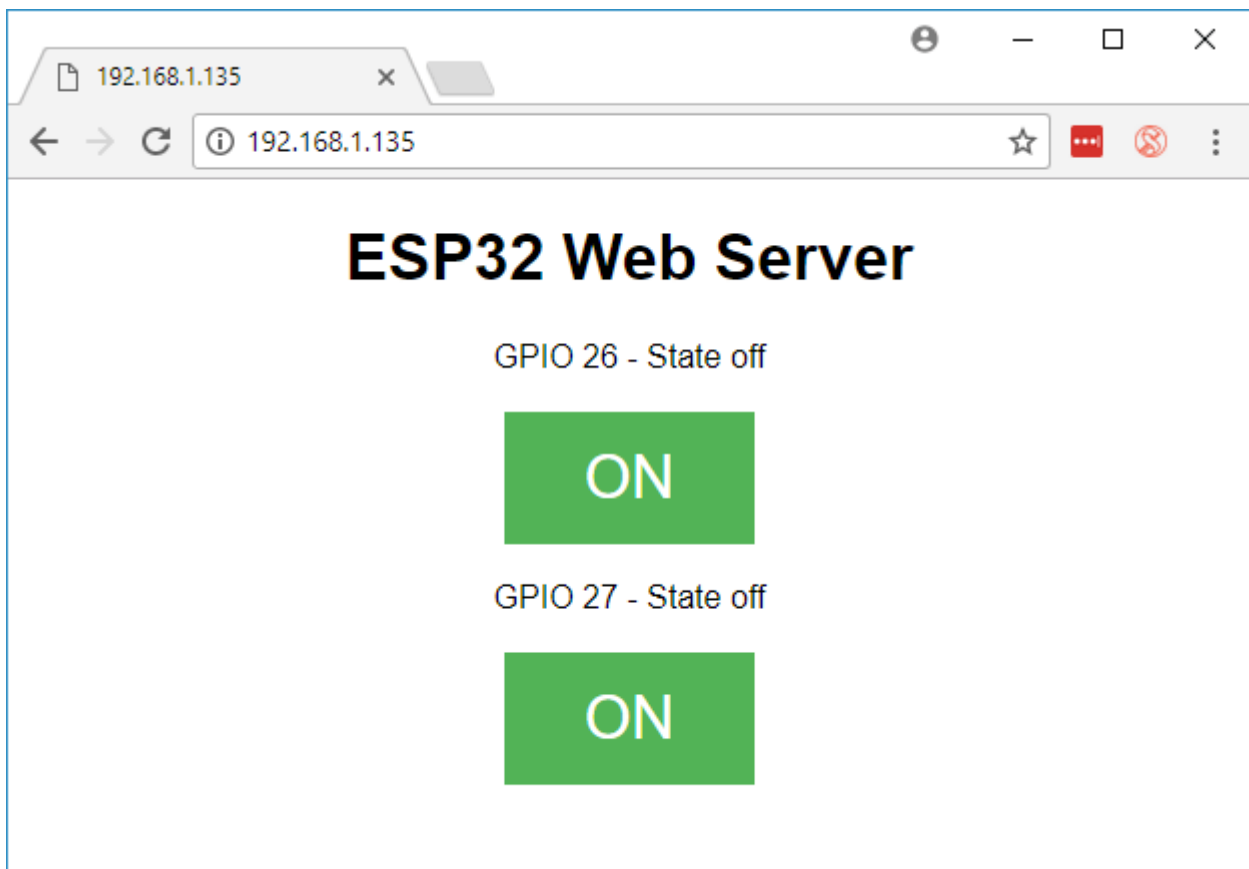


Press the ESP32 EN button (reset). The ESP32 connects to Wi-Fi, and outputs the ESP IP address on the Serial Monitor. Copy that IP address, because you need it to access the ESP32 web server.



## Accessing the Web Server

To access the web server, open your browser, paste the ESP32 IP address, and you'll see the following page. In our case it is **192.168.1.135**.



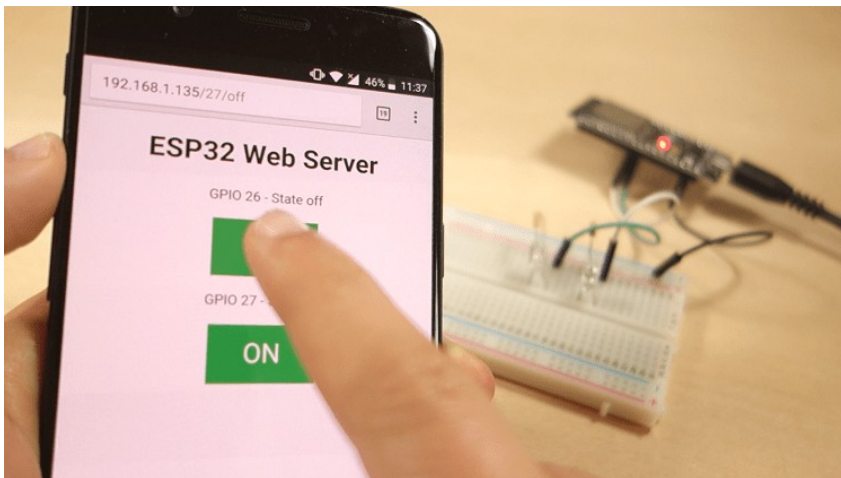
If you take a look at the Serial Monitor, you can see what's happening on the background. The ESP receives an HTTP request from a new client (in this case, your browser).



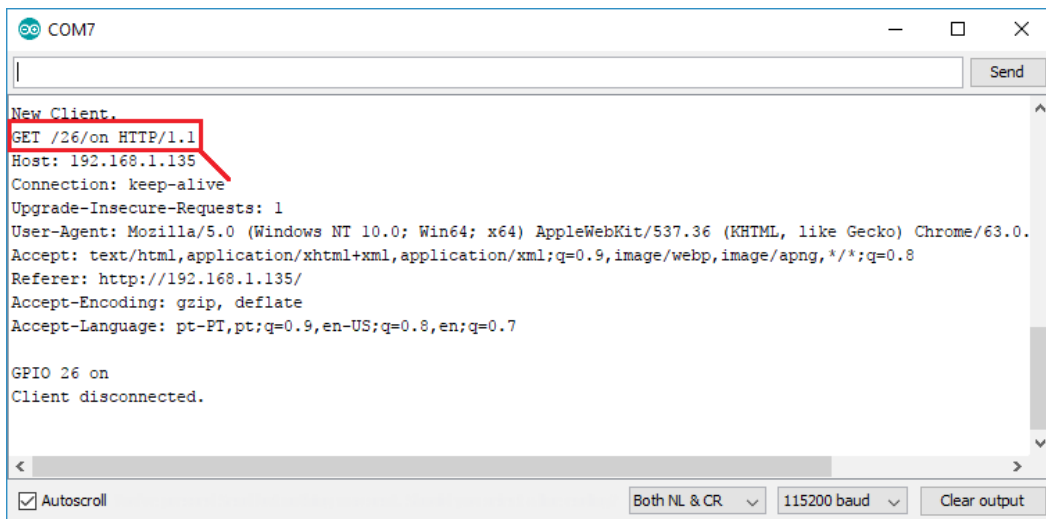
You can also see other information about the HTTP request.

## Testing the Web Server

Now you can test if your web server is working properly. Click the buttons to control the LEDs.



At the same time, you can take a look at the Serial Monitor to see what's going on in the background. For example, when you click the button to turn GPIO 26 ON, ESP32 receives a request on the **/26/on** URL.



When the ESP32 receives that request, it turns the LED attached to GPIO 26 ON and updates its state on the web page.



The button for GPIO 27 works in a similar way. Test that it is working properly.

## How the Code Works

In this section will take a closer look at the code to see how it works.

The first thing you need to do is to include the WiFi library.

```
#include <WiFi.h>
```

As mentioned previously, you need to insert your ssid and password in the following lines inside the double quotes.

```
const char* ssid = "";  
const char* password = "";
```

Then, you set your web server to port 80.

```
WiFiServer server(80);
```

The following line creates a variable to store the header of the HTTP request:

```
String header;
```

Next, you create auxiliary variables to store the current state of your outputs. If you want to add more outputs and save its state, you need to create more variables.

```
String output26State = "off";  
String output27State = "off";
```

You also need to assign a GPIO to each of your outputs. Here we are using GPIO 26 and GPIO 27. You can use any other suitable GPIOs.

```
const int output26 = 26;  
const int output27 = 27;
```

## **setup()**

Now, let's go into the `setup()`. First, we start a serial communication at a baud rate of 115200 for debugging purposes.

```
Serial.begin(115200);
```

You also define your GPIOs as OUTPUTs and set them to LOW.

```
// Initialize the output variables as outputs  
pinMode(output26, OUTPUT);  
pinMode(output27, OUTPUT);  
  
// Set outputs to LOW  
digitalWrite(output26, LOW);  
digitalWrite(output27, LOW);
```

The following lines begin the Wi-Fi connection with `WiFi.begin(ssid, password)`, wait for a successful connection and print the ESP IP address in the Serial Monitor.

```
// 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();
```

## **loop()**

In the `loop()` we program what happens when a new client establishes a connection with the web server.

The ESP32 is always listening for incoming clients with the following line:

```
WiFiClient client = server.available(); // Listen for incoming clients
```

When a request is received from a client, we'll save the incoming data. The while loop that follows will be running as long as the client stays connected. We don't recommend changing the following part of the code unless you know exactly what you are doing.

```
if (client) { // If a new client connects,
  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()) { // loop while the client's connected
    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();
```

The next section of if and else statements checks which button was pressed in your web page, and controls the outputs accordingly. As we've seen previously, we make a request on different URLs depending on the button pressed.

```
// turns the GPIOs on and off
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);
}
```

For example, if you've press the GPIO 26 ON button, the ESP32 receives a request on the **/26/ON URL** (we can see that that information on the HTTP header on the Serial Monitor). So, we can check if the header contains the expression **GET /26/on**. If it contains, we change the output26state variable to ON, and the ESP32 turns the LED on.

This works similarly for the other buttons. So, if you want to add more outputs, you should modify this part of the code to include them.

## Displaying the HTML web page

The next thing you need to do, is creating the web page. The ESP32 will be sending a response to your browser with some HTML code to build the web page.

The web page is sent to the client using this expressing `client.println()`. You should enter what you want to send to the client as an argument.

The first thing we should send is always the following line, that indicates that we are sending HTML.

```
<!DOCTYPE HTML><html>
```

Then, the following line makes the web page responsive in any web browser.

```
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
```

And the following is used to prevent requests on the favicon. – You don't need to worry about this line.

```
client.println("<link rel=\"icon\" href=\"data:,\">");
```

## Styling the Web Page

Next, we have some CSS text to style the buttons and the web page appearance. We choose the Helvetica font, define the content to be displayed as a block and aligned at the center.

```
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
```

We style our buttons with the #4CAF50 color, without border, text in white color, and with this padding: 16px 40px. We also set the text-decoration to none, define the font size, the margin, and the cursor to a pointer.

```
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;}");
```

We also define the style for a second button, with all the properties of the button we've defined earlier, but with a different color. This will be the style for the off button.

```
client.println(".button2 {background-color: #555555;}</style></head>");
```

## Setting the Web Page First Heading

In the next line you can set the first heading of your web page. Here we have “ESP32 Web Server”, but you can change this text to whatever you like.

```
// Web Page Heading
client.println("<h1>ESP32 Web Server</h1>");
```

## Displaying the Buttons and Corresponding State

Then, you write a paragraph to display the GPIO 26 current state. As you can see we use the `output26State` variable, so that the state updates instantly when this variable changes.

```
client.println("<p>GPIO 26 - State " + output26State + "</p>");
```

Then, we display the on or the off button, depending on the current state of the GPIO. If the current state of the GPIO is off, we show the ON button, if not, we display the OFF 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>");  
}
```

We use the same procedure for GPIO 27.

## Closing the Connection

Finally, when the response ends, we clear the header variable, and stop the connection with the client with `client.stop()`.

```
// Clear the header variable  
header = "";  
// Close the connection  
client.stop();
```

## Wrapping Up

In this tutorial we've shown you how to build a web server with the ESP32. We've shown you a simple example that controls two LEDs, but the idea is to replace those LEDs with a relay, or any other output you want to control. For more projects with ESP32, check the following tutorials: