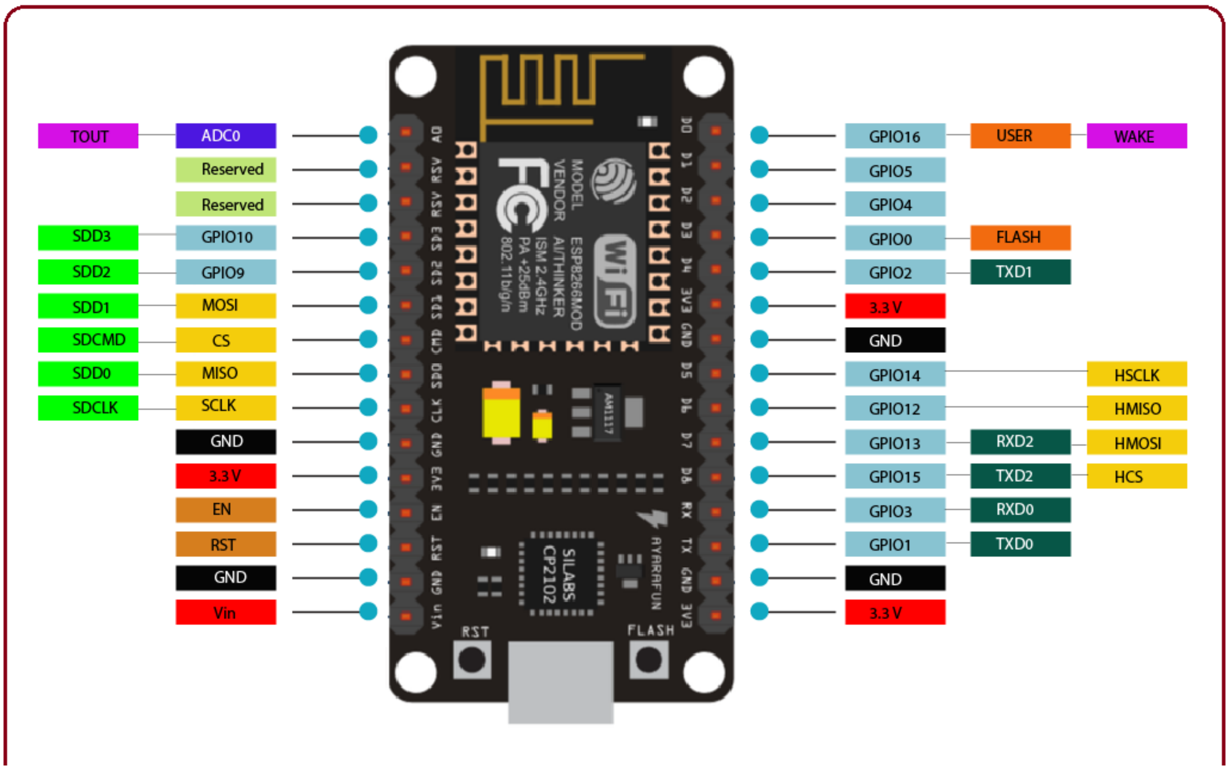
ESP8266 – Tutorial Demonstration II

You will need:

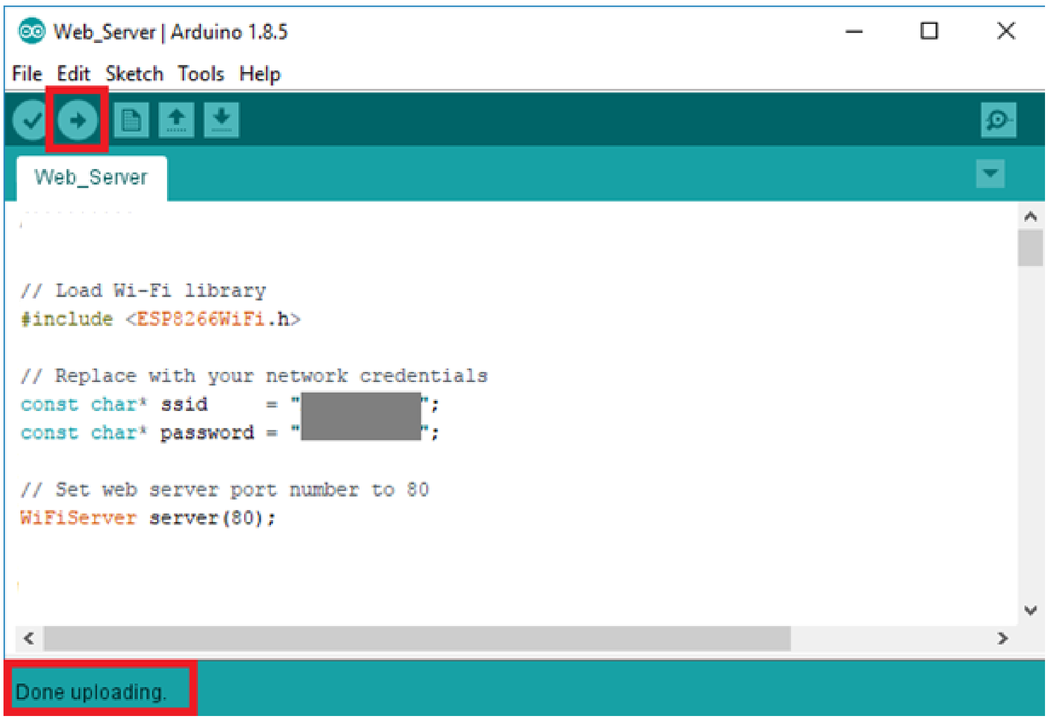
- ESP8266
- 2 LEDs
- 2 Resistors (220 or 330 ohms should work just fine)
- Some jumper wires (Female to Female) 4 to 5 probably
- A small breadboard will be extremely good. We can do the work without it as well.
- Good microUSB **Data** cable (not charging cable)
- Laptop

Datasheet:



Web Server App

Initial Code:

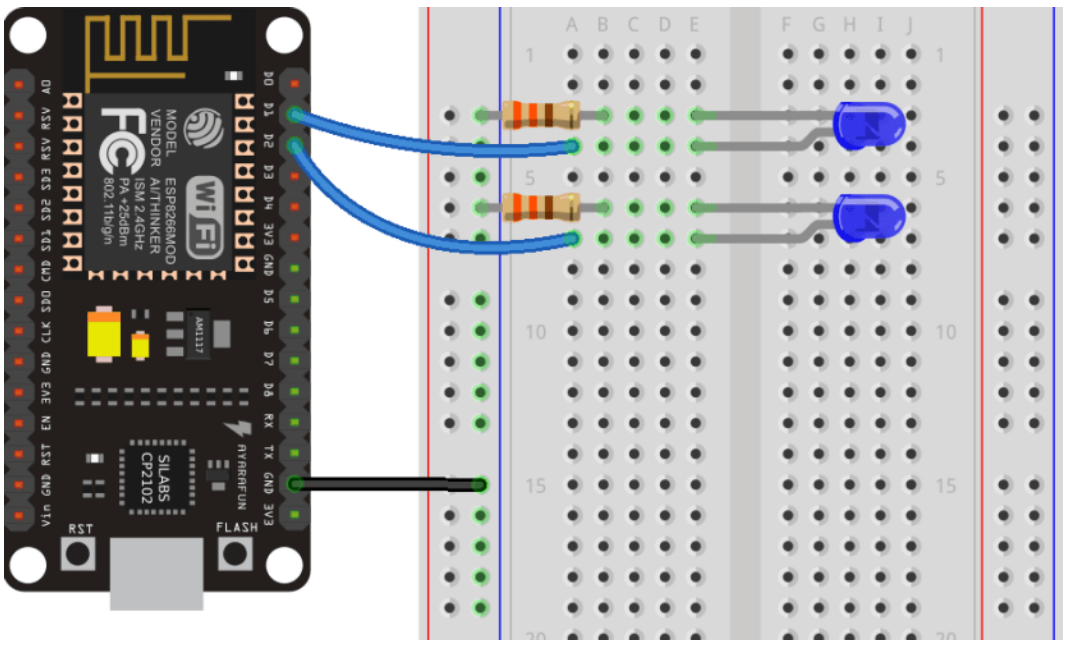


```
// Load Wi-Fi library
#include <ESP8266WiFi.h>

// Replace with your network credentials
const char* ssid     = "[redacted]";
const char* password = "[redacted]";

// Set web server port number to 80
WiFiServer server(80);
```

Done uploading.

Schematics

Connect two LEDs to your ESP8266 as shown in the following schematic diagram – with one LED connected to GPIO 4 (D2), and another to GPIO 5 (D1).

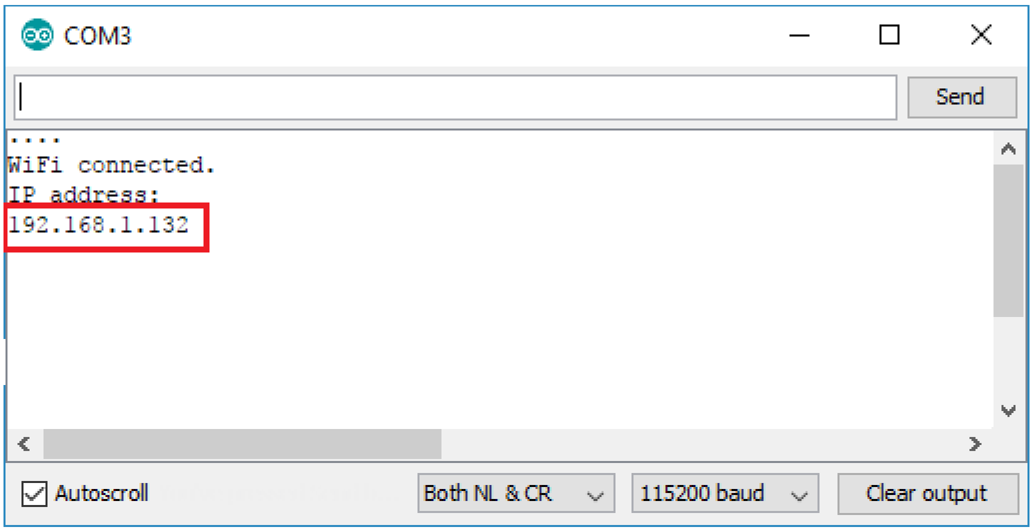


Testing the Web Server

Now, you can upload the code, and it will work straight away. Don't forget to check if you have the right board and COM port selected, otherwise you'll get an error when trying to upload. Open the Serial Monitor at a baud rate of 115200.

Finding the ESP IP Address

Press the ESP8266 RESET button, and it will output the ESP IP address on the Serial Monitor

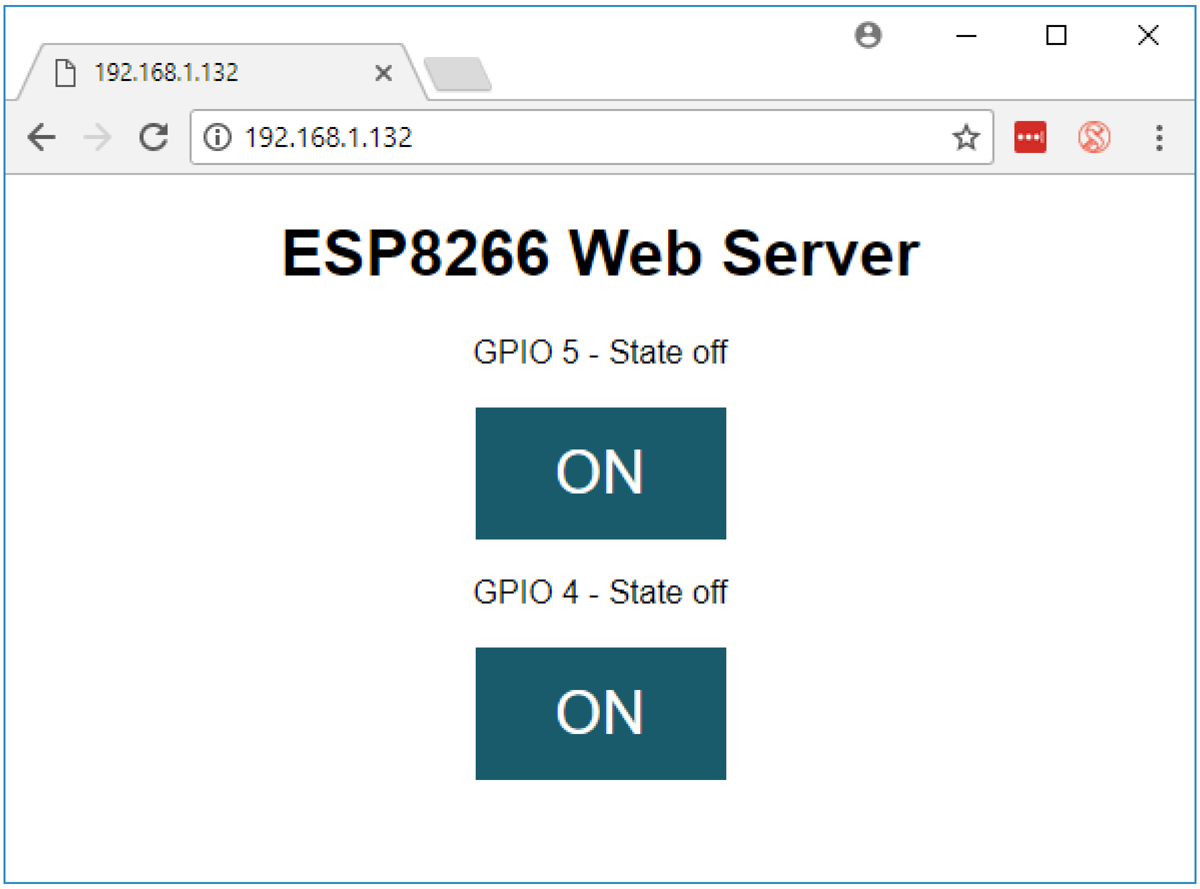


```
....
WiFi connected.
IP address:
192.168.1.132
```

Copy that IP address, because you need it to access the web server.

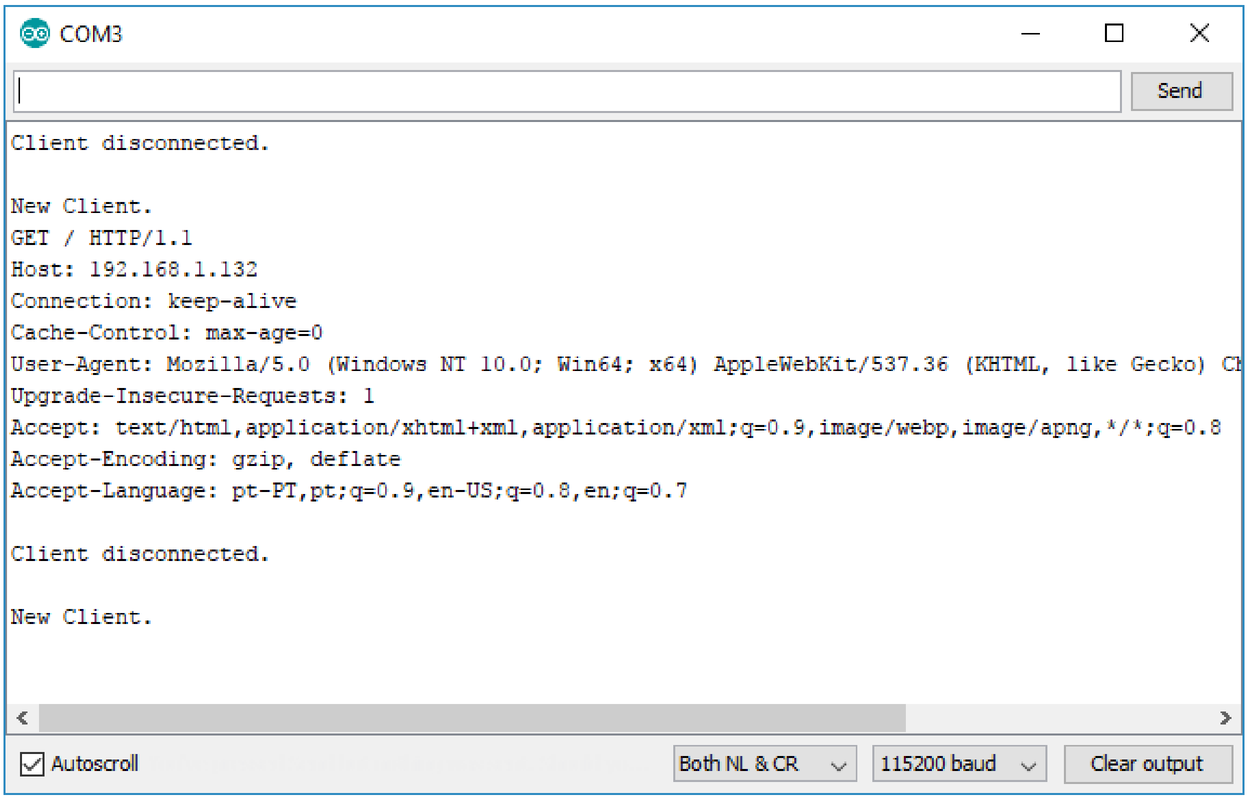
Accessing the Web Server

Open your browser, type the ESP IP address, and you'll see the following page. This page is sent by the ESP8266 when you make a request on the ESP IP address.



If you take a look at the serial monitor, you can see what's going on the background. The ESP receives an HTTP request from a new client – in this case, your browser.

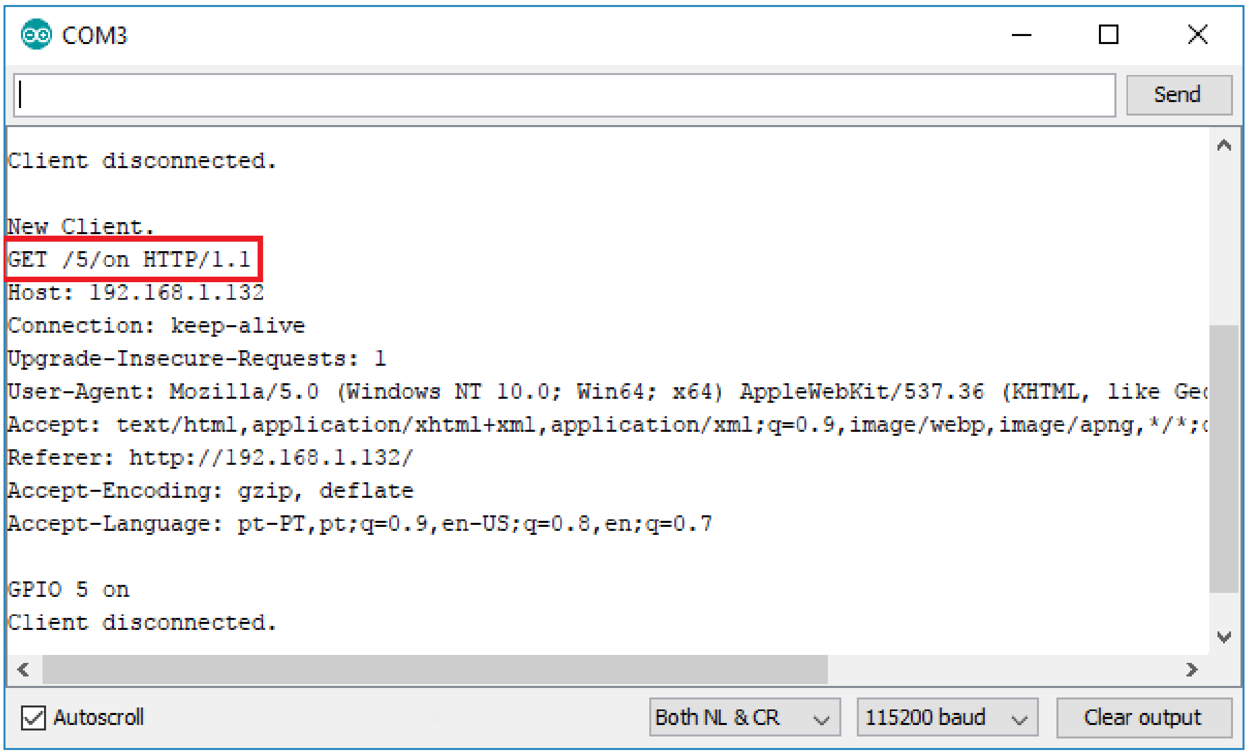
You can also see other information about the HTTP request – these fields are called HTTP header fields, and they define the operating parameters of an HTTP transaction.



```
Client disconnected.

New Client.
GET / HTTP/1.1
Host: 192.168.1.132
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7

Client disconnected.

New Client.
```

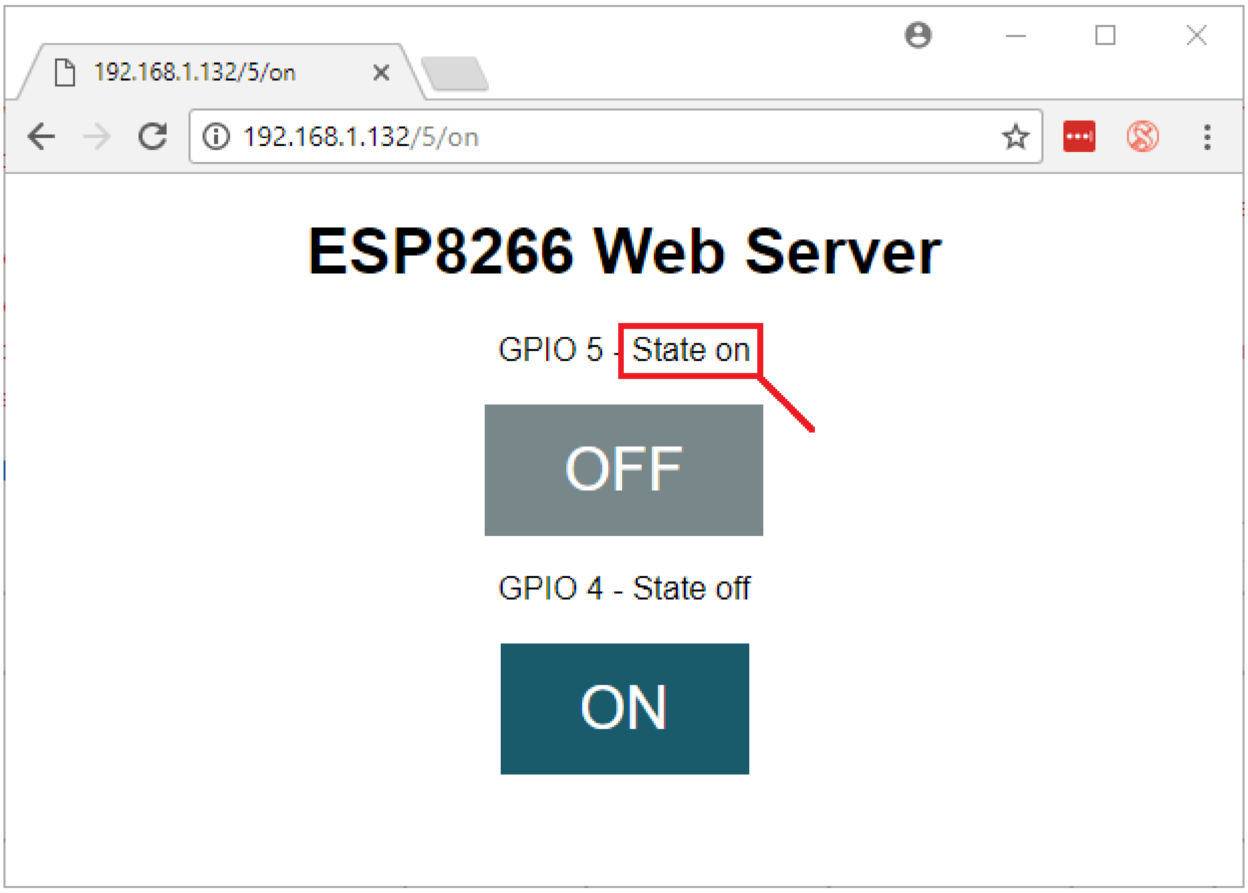
Testing the Web Server

Let’s test the web server. Click the button to turn GPIO 5 ON. The ESP receives a request on the /5/on URL, and turns LED 5 ON.



```
Client disconnected.

New Client.
GET /5/on HTTP/1.1
Host: 192.168.1.132
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gec
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q
Referer: http://192.168.1.132/
Accept-Encoding: gzip, deflate
Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7

GPIO 5 on
Client disconnected.
```

The LED state is also updated on the web page.



Test GPIO 4 button and check that it works in a similar way.

Code and Explaination

Now, let's take a closer look at the code to see how it works, so that you are able to modify it to fulfill your needs.

The first thing you need to do is to include the `ESP8266WiFi` library.

```
// Load Wi-Fi library
#include <ESP8266WiFi.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.

```
// Set web server port number to 80
WiFiServer server(80);
```
The following line creates a variable to store the header of the HTTP request:

```
String header;
```
Next, you create auxiliar 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.

```
// Auxiliar variables to store the current output state
String output5State = "off";
String output4State = "off";
```
You also need to assign a GPIO to each of your outputs. Here we are using GPIO 4 and GPIO 5. You can use any other suitable GPIOs.

```
// Assign output variables to GPIO pins
const int output5 = 5;
const int output4 = 4;
```

setup()

Now, let's go into the `setup()`. The `setup()` function only runs once when your ESP first boots. 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(output5, OUTPUT);
pinMode(output4, OUTPUT);
// Set outputs to LOW
digitalWrite(output5, LOW);
digitalWrite(output4, LOW);
```

The following lines begin the Wi-Fi connection with `WiFi.begin(ssid, password)`, wait for a successful connection and prints 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 ESP is always listening for incoming clients with this 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 we press.

```
// turns the GPIOs on and off
if (header.indexOf("GET /5/on") >= 0) {
  Serial.println("GPIO 5 on");
  output5State = "on";
  digitalWrite(output5, HIGH);
} else if (header.indexOf("GET /5/off") >= 0) {
  Serial.println("GPIO 5 off");
  output5State = "off";
  digitalWrite(output5, LOW);
} else if (header.indexOf("GET /4/on") >= 0) {
  Serial.println("GPIO 4 on");
  output4State = "on";
  digitalWrite(output4, HIGH);
} else if (header.indexOf("GET /4/off") >= 0) {
  Serial.println("GPIO 4 off");
  output4State = "off";
  digitalWrite(output4, LOW);
}
```

For example, if you've pressed the GPIO 5 ON button, the URL changes to the ESP IP address followed by /5/ON, and we receive that information on the HTTP header. So, we can check if the header contains the expression GET /5/on.
If it contains, the code prints a message on the serial monitor, changes the `output5State` variable to on, and 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 generate the web page. The ESP8266 will be sending a response to your browser with some HTML text to display the web page.

The web page is sent to the client using the `client.println()` function. You should enter what you want to send to the client as an argument.
The first text you should always send is the following line, that indicates that we're 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\">");
```

The next one is used to prevent requests related to 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 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 some properties to define color, size, border, etc…

```
client.println(".button { background-color: #195B6A; border: none; color: white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
```

Then, we 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: #77878A;}</style></head>");
```

Setting the Web Page First Heading

In the next line you set the first heading of your web page, you can change this text to whatever you like.

```
// Web Page Title
client.println("<h1>ESP8266 Web Server</h1>");
```

Displaying the Buttons and Corresponding State

Then, you write a paragraph to display the GPIO 5 current state. As you can see we use the `output5State` variable, so that the state updates instantly when this variable changes.

```
client.println("<p>GPIO 5 - State " + output5State + "</p>");
```

Then, we display the on or the off button, depending on the current state of the GPIO.

```
if (output5State=="off") {
 client.println("<p><a href=\"/5/on\"><button class=\"button\">ON</button></a></p>");
} else {
 client.println("<p><a href=\"/5/off\"><button class=\"button button2\">OFF</button></a></p>");
}
```

We use the same procedure for GPIO 4.

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();
```