Part 23
-
Webserver

Version: 2021-05-27

Which library to use

There are at least 3 ways to implement a webserver on ESP32 using 3 different libraries:

- WiFi which includes WiFIServer
- WebServer
- AsyncWebServer

In super-brief WiFiServer is effectively a socket server, not a web server.
WebServer is a full web server but can only handle one call at a time: even a post-back from the same client will cause problems.
AsyncWebServer is a great, full-featured web server that unfortunately suffers from heap corruption under load, so isn't ready for prime time. As far as I know, there isn't a high-availability web server for the ESP32 that you can use for REST, etc, because of the corruption bug. It's been there a year so I doubt it's going anywhere soon.

So, I'll go with the Webserver library first and at the end you find also the information on how to use the WiFi / WiFiServer library

Web Server using library "WebServer"

A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients.
To implement web server on ESP, there are two ways to make your first web server

1. connect to your WiFi network
2. make ESP as access point

Creating web server on ESP32 connected to existing WiFi network
We need these libraries to make web server.

- WiFi.h is required for doing all WiFi related functionalities
- WebServer.h it handles all HTTP protocols

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

Define your SSID and Password of your WiFi router, where the ESP connects

```
//SSID and Password of your WiFi router
const char* ssid = "your_ssid";
const char* password = "password";
```

Web server is on port 80, you can use other ports also, default HTTP port is 80, to open web page with different port number you have to enter port number after IP address. Ex. For port number 81 you have to type 192.168.2.2:81 in browser.

```
WebServer server(80); //Server on port 80
```

The following commands are used to connect to your WiFi Access point as a WiFi device or station. If the network is open you can remove password field from command.

```
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
```

After connection request we wait for WiFi to get connect. If the ESP32 was connected and disconnects afterwards due to signal loss or any reason, there is no need to give this command again, it will try to connect again automatically. This is handled by its OS, you may find some stack errors displayed in serial monitor. These errors come from its internal OS.

```
// Wait for connection
while (WiFi.status() != WL_CONNECTED)
{
  delay(500);
  Serial.print(".");
}
```

To know IP address i.e. assigned to ESP32 by your WiFi router use this command

```
WiFi.localIP();
```

When a client requests a web page by entering ESP32 IP address, which data to be sent is handled by subroutine and that subroutine name is defined in server.on(path,subroutine_name).

```
server.on("/", handleRoot); //to handle at root location
```

If you have two or more pages you can define like this

```
Server.on("/",root);              // this is root location
Server.on("/page1",First_page);  //"192.168.2.2/page1"  first page location
Server.on("/page2",Second_page); //"192.168.2.2/page2"  second page location
```

You must have three subroutines that handle client requests.

To start the server use this command

```
server.begin();
```

In main loop we handle client request

```
server.handleClient();
```

The following subroutine is called when you enter IP address in web browser and hit enter. This routine sends the test "hello from ESP32" to web browser.

```
void handleRoot()
{
  server.send(200, "text/plain", "hello from ESP32!");
}
```

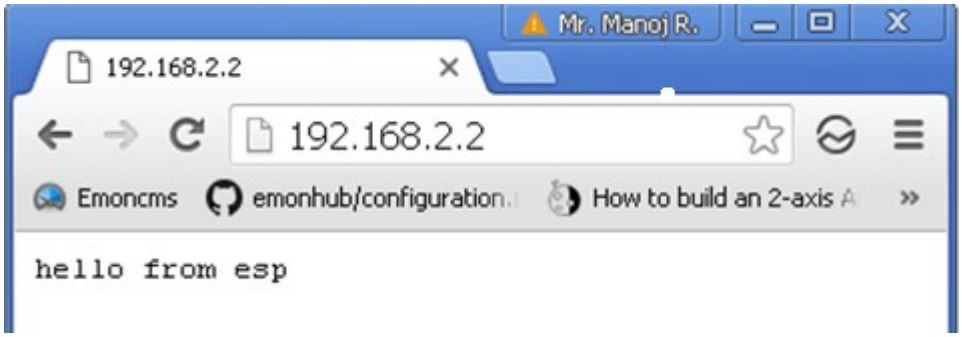
Complete Program

```
/*
 * Hello world web server
 */

#include <WiFi.h>
#include <WebServer.h>

//SSID and Password of your WiFi router
const char* ssid     = "<wifi-ssid>";
const char* password = "<wifi-pw>";

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

//===============================================================
//     This rutine is exicuted when you open its IP in browser
//===============================================================
void handleRoot()
{
  server.send(200, "text/plain", "hello from esp");
}

//===============================================================
//                  SETUP
//===============================================================
void setup(void)
{
  Serial.begin(9600);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  //If connection successful show IP address in serial monitor
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());  //IP address assigned to your ESP

  server.on("/", handleRoot);      //to handle at root location

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

//===============================================================
//                     LOOP
//===============================================================
void loop(void)
{
  server.handleClient();          //Handle client requests
}
```

Result
To see the result first get the IP address from serial monitor, open a serial monitor and press reset on ESP32. It shows IP and shows its connection status, if it is not able to connect it will show "……..." dots in serial monitor.

Once connected it will show following

Open web browser and enter the IP shown (in my case 192.168.2.2), Make sure that your laptop or phone is connected to the same network. You can see this web page in all the devices which are connected to the WiFi network, where the ESP32 is connected.



Creating web server on ESP32 configured as Access Point
In some application you may find it usefull that both AP and connection to WiFi network are available for making configuration changes in ESP32 and for data sending to cloud using WiFi connectivity. This way you can access ESP web page with two different IP address.

```
WiFi.mode(WIFI_AP_STA);  //Both hotspot and client are enabled
```

if you only want the ESP32 to be an access point, use this command.

```
WiFi.mode(WIFI_AP);         //Only Access point
```

To start ESP32 as Access point you have to use this simple command

```
//Start HOTspot removing password will disable security
WiFi.softAP(ssid, password);
```

To get IP address i.e. assigned to ESP32 by your WiFi router use this command

```
IPAddress myIP = WiFi.softAPIP();
```

When client request a web page by entering ESP32 IP address, data to be sent is handled by subroutine and that subroutine name is defined in server.on(path,subroutine_name).

```
server.on("/", handleRoot); //to handle at root location
```

Example: If you have two pages you can define like this

```
Server.on("/",root);              //192.168.2.2          root location
Server.on("/page1",First_page);   //"192.168.2.2/page1"  first page location
Server.on("/page2",Second_page); //"192.168.2.2/page2"  second page location
```

You have three subroutines that handle client requests.
To start the server use this command

```
server.begin();
```

In main loop we handle client request

```
server.handleClient();
```

This subroutine is called when you enter IP address in web browser and hit enter. This routine sends the test "hello from ESP32" to web browser.

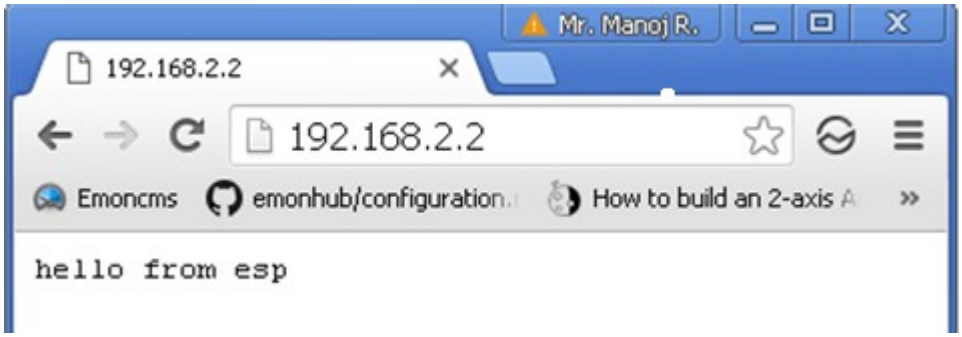
```
void handleRoot()
{
  server.send(200, "text/plain", "hello from esp");
}
```

Complete Program

```
/*
 * Hello world web server
 */

#include <WiFi.h>
#include <WebServer.h>

//SSID and Password to your ESP Access Point
const char* ssid     = "ESPWebServer";
const char* password = "123456789";

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

//===============================================================
//     This rutine is exicuted when you open its IP in browser
//===============================================================
void handleRoot()
{
  server.send(200, "text/plain", "hello from esp!");
}

//===============================================================
//                  SETUP
//===============================================================
void setup(void)
{
  Serial.begin(9600);
  Serial.println("");
  //Only Access point
  WiFi.mode(WIFI_AP);
  //Start HOTspot removing password will disable security
  WiFi.softAP(ssid, password);

  IPAddress myIP = WiFi.softAPIP();
  Serial.print("HotSpt IP:");
  Serial.println(myIP);

  server.on("/", handleRoot);      //to handle at root location

  server.begin();
  Serial.println("HTTP server started");
}
//===============================================================
//                     LOOP
//===============================================================
void loop(void)
{
  server.handleClient();          //Handle client requests
}
```

Result

After uploading program take your mobile turn on WiFi and in WiFi setting Scan for hot spot you will find "ESPWebServer" hot spot connect to it with password "12345678" as we have given in program. After connecting to ESP hot spot, open web browser in mobile phone and enter IP you will see "hello from esp" message. IP address can be found in serial monitor.



Web Server with HTML Web Page

Create a good looking Web page

Open your editor and start writing HTML code. Save as index.html.

```
<HTML>
        <HEAD>
                <TITLE>My first web page</TITLE>
        </HEAD>
<BODY>
        <CENTER>
                <B>Hello World.... </B>
        </CENTER>
</BODY>
</HTML>
```

`<HEAD>` and `<TITLE>` is used to give page title, which is visible in top of the browser. `<CENTER>` tag is used for center alignment of text, `<B>` is used to make text bold.

Test your web page

Open your web page in web browser. You can observe that at the top you see Title "`My first web page`". And Web page with **`Hello World`**… message.

To see the changes in your HTML code simply change you HTML program and press refresh in browser. It will reflect immediately. This way you can make your webpage test it, then deploy it on ESP32. It saves your lot of time.


Mr. Manoj R.
My first web page
file:///C:/Documents%20and%20Settin
Emoncms
emonhub/configuration
How to build an 2-axis
Hello World....

Upload your own HTML code as web page
We have learned how to create web server and its basics, now we want to upload our HTML web page. It's very simple, just replace "hello from esp" with HTML code.

```
server.send(200, "text/plain", "hello from esp");
```

First we take the webpage code in separate header file name it as "web-index.h", our web page is now a array of characters stored in variable **MAIN_page**. Do not use comments in this file. It is HTML data as a character array not a program. Now HTML code is in a header file .h not .html file.

```
const char MAIN_page[] PROGMEM = R"=====(
<HTML>
        <HEAD>
                        <TITLE>My first web page</TITLE>
        </HEAD>
<BODY>
        <CENTER>
                        <B>Hello World.... </B>
        </CENTER>
</BODY>
</HTML>
)=====";
```

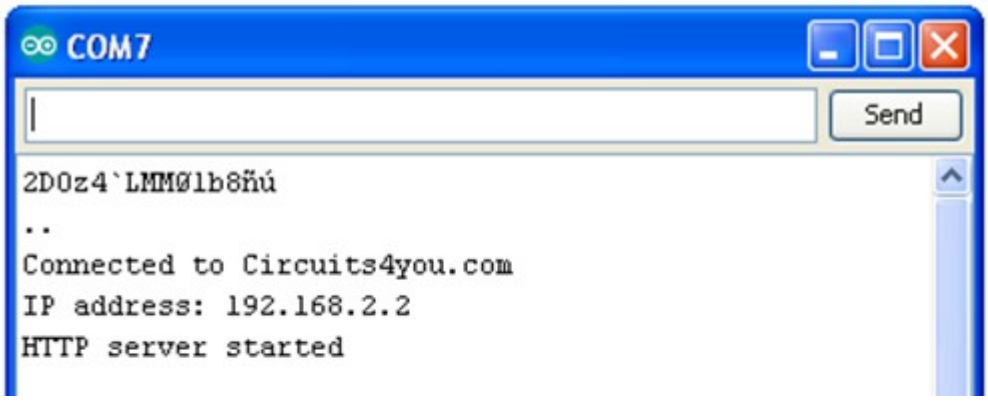
Now we import this header file in our program using `#import "web-index.h"`. Make sure that this file is be with arduino code file .ino
The changes in main programs are made in `handleRoot` subroutine which sends the web page to client, now we are sending html page change `text/plain` to `text/html`.
The modified handleRoot subroutine

```
void handleRoot()
{
  String s = MAIN_page;
  server.send(200, "text/html", s);
}
```

Complete Program

```
/*
 * Hello world web server
 */

#include <WiFi.h>
#include <WebServer.h>

#include "web-index.h" //Our HTML webpage contents

//SSID and Password of your WiFi router
const char* ssid     = "<wifi-ssid>";
const char* password = "<wifi-pw>";

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

//===============================================================
// This routine is executed when you open its IP in browser
//===============================================================
void handleRoot()
{
 String s = MAIN_page;             //Read HTML contents
 server.send(200, "text/html", s); //Send web page
}

//===============================================================
//                  SETUP
//===============================================================
void setup(void)
{
  Serial.begin(9600);

  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  //If connection successful show IP address in serial monitor
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  server.on("/", handleRoot);      //handle at root location

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

//===============================================================
//                     LOOP
//===============================================================
void loop(void)
{
  server.handleClient();
}
```

Result
To see the result first get the IP address from serial monitor, Open serial monitor and press reset. Once connected it will show following



Open web browser and enter the IP (in my case 192.168.2.2). Make sure that your laptop or phone must be connected to the same network. You can see this web page which is we have created in all the devices which are connected to the WiFi router, where the ESP32 is connected.

Update and display sensor values

In many IoT Applications we monitor sensor data and we want to display it in web page. The web page requires frequent refresh to get the update from ESP32.
To solve this problem you have two options, first is refresh page with HTML refresh tag: ex. refresh at every 30 seconds.

```
<head>
  <meta http-equiv=”refresh” content=”30”>
</head>
```

Disadvantage of using HTML refresh tag is, it flickers the screen and loads complete page again and again.

A second option is to display data and update without refreshing web page. You can do a lot of things with this. To make this possible we need to use javaScript Ajax**.**

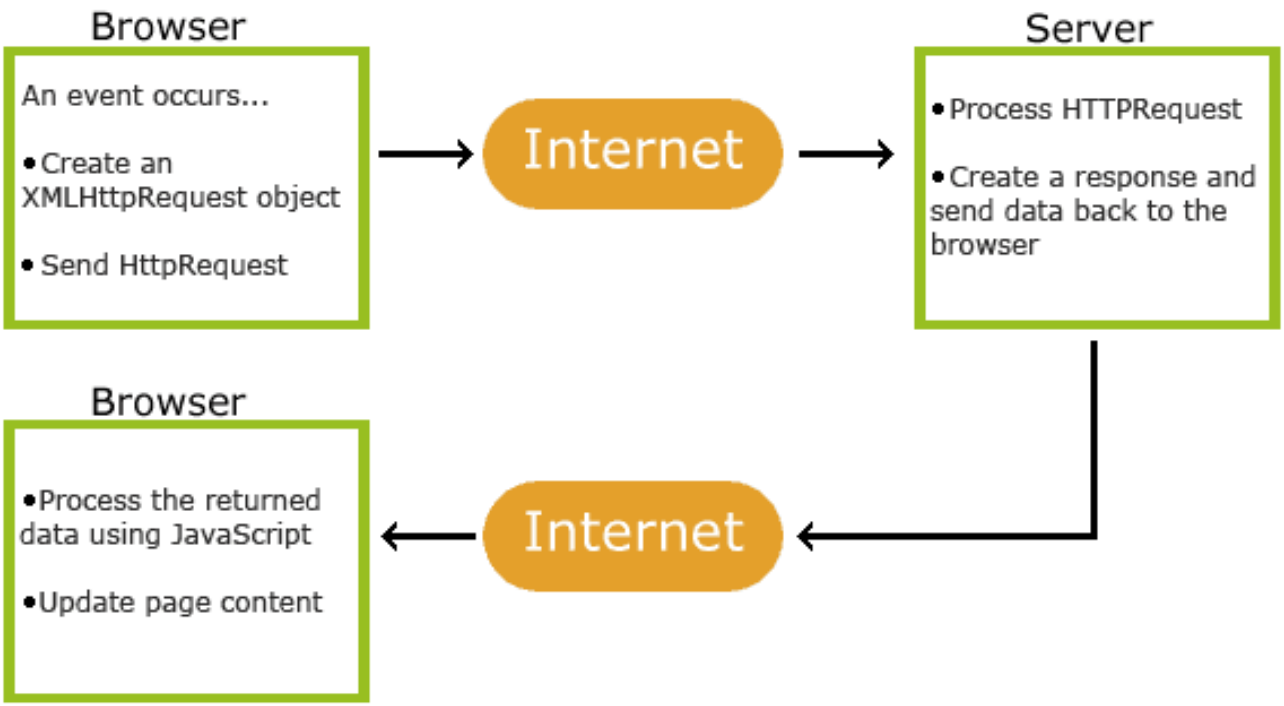
HTML and Java Script, AJAX basics
AJAX is about updating parts of a web page, without reloading the whole page.
AJAX stands for “Asynchronous JavaScript and XML”. It is a technique for creating fast and dynamic web pages. It allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

AJAX is based on internet standards, and uses a combination of:

- XMLHttpRequest object (to exchange data asynchronously with a server)
- JavaScript/DOM (to display/interact with the information)
- CSS (to style the data)
- XML (often used as the format for transferring data)

Read More on Ajax at https://www.w3schools.com/asp/asp_ajax_intro.asp


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

How ESP32 AJAX Works
In ESP32 we create two pages on server. First page loads as normal webpage and second webpage is behind the scene i.e. AJAX.

Create Web Page with AJAX
Make your HTML page and test it in browser. Then make it header file as shown in below code.

Create `web-index.h` file in your sketch folder. Copy and paste below code. Program is commented well. You can read more on HTML and AJAX at w3schools.com

```
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
    <style>
      .card{
          max-width: 400px;
          min-height: 250px;
          background: #02b875;
          padding: 30px;
          box-sizing: border-box;
          color: #FFF;
          margin:20px;
          box-shadow: 0px 2px 18px -4px rgba(0,0,0,0.75);
      }
    </style>
  <body>

    <div class="card">
      <h4>The ESP32 Update web page without refresh</h4><br>
      <h1>Sensor Value:<span id="ADCValue">0</span></h1><br>
      <br><a href="https://circuits4you.com">Circuits4you.com</a>
    </div>
    <script>
      // Call a function repetatively with 2 second interval
      setInterval(function(){getData();}, 2000);

      function getData()
      {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function()
        {
          if (this.readyState == 4 && this.status == 200)
          {
            document.getElementById("ADCValue").innerHTML = this.responseText;
          }
        };
        xhttp.open("GET", "readADC", true);
        xhttp.send();
      }
    </script>
  </body>
</html>
)=====";
```

ESP32 Web Server Program Main INO file

```
/*
 * ESP32 AJAX Demo
 */

#include <WiFi.h>
#include <WebServer.h>

#include "web-index.h"  //Web page header file

WebServer server(80);

//Enter your SSID and PASSWORD
const char* ssid     = ".........";
const char* password = ".........";

//===============================================================
// This routine is executed when you open its IP in browser
//===============================================================
void handleRoot()
{
  String s = MAIN_page;             //Read HTML contents
  server.send(200, "text/html", s); //Send web page
}

void handleADC()
{
  int a = analogRead(A0);
  String adcValue = String(a);

  //Send ADC value only to client ajax request
  server.send(200, "text/plane", adcValue);
}

//===============================================================
// Setup
//===============================================================

void setup(void)
{
  Serial.begin(115200);
  Serial.println();
  Serial.println("Booting Sketch...");

  /*
  //ESP32 As access point
  WiFi.mode(WIFI_AP); //Access Point mode
  WiFi.softAP(ssid, password);
  */

  //ESP32 connects to your wifi ----------------------------------
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

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

  //Wait for WiFi to connect
  while(WiFi.waitForConnectResult() != WL_CONNECTED)
  {
    Serial.print(".");
  }
```

```
  //If connection successful show IP address in serial monitor
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  server.on("/", handleRoot);        //display page
  server.on("/readADC", handleADC);  //get update of ADC Value only

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

//================================================================
// This routine is executed when you open its IP in browser
//================================================================
void loop(void)
{
  server.handleClient();
  delay(1);
}
```

Testing
Make WiFi network configuration changes as per your network. Upload the program and test it. Open serial monitor to observe the IP address and other actions.

Results
Open Serial monitor and get the IP address

Enter the IP address in web browser

Handling HTML web forms data

HTML Forms are one of the main points of interaction between a user and a web site or ESP32. They allow users to send data to the web site. Most of the time that data is sent to the web server, but the web page can also intercept it to use it on its own.
An HTML form is made of one or more widgets. Those widgets can be text fields (single line or multiline), select boxes, buttons, checkboxes, or radio buttons. Most of the time those widgets are paired with a label that describes their purpose.
The main difference between a HTML form and a regular HTML document is that most of the time, the data collected by the form is sent to a ESP32 web server. In that case, you need to set up a web server to receive and process the data.

The HTML code

The HTML `<form>` element defines a form that is used to collect user input:

```
<form>
.
form elements
.
</form>
```

An HTML form contains form elements.
Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more.

The `<input>` element is the most important form element. The `<input>` element can be displayed in several ways, depending on the *type* attribute. Here are some examples:

TYPE	DESCRIPTION
`<input type="text">`	Defines a one-line text input field
`<input type="radio">`	Defines a radio button (for selecting one of many choices)
`<input type="submit">`	Defines a submit button (for submitting the form)

The *action* attribute defines the action to be performed when the form is submitted.
Normally, the form data is sent to a web page on the server when the user clicks on the submit button.
In the example above, the form data is sent to a page on the server called "/action_page".
This page contains a server-side script that handles the form data:

```
<form action="/action_page">
```

If the *action* attribute is omitted, the action is set to the current page.

The *method* attribute specifies the HTTP method 'GET' or 'POST' to be used when submitting the form data:

```
<form action="/action_page" method="get">
```

or

```
<form action="/action_page" method="post">
```

The default method when submitting form data is GET.
However, when GET is used, the submitted form data will be visible in the page address field:

```
/action_page?firstname=Mickey&lastname=Mouse
```

Notes on GET

- Appends form-data into the URL in name/value pairs
- The length of a URL is limited (about 3000 characters)
- Never use GET to send sensitive data as it is will be visible in the URL
- Useful for form submissions where a user wants to bookmark the result
- GET is better for non-secure data, like query strings in Google

Always use POST if the form data contains sensitive or personal information. The POST method does not display the submitted form data in the page address field.

Notes on POST:

- POST has no size limitations, and can be used to send large amounts of data.
- Form submissions with POST cannot be bookmarked

Creating web form
Open your editor and create simple html web page. Test/open this html page in web browser to see results and do necessary adjustments, before uploading it to ESP32.

```
<!DOCTYPE html>
<html>
  <body>

    <h2>Circuits4you<h2>
    <h3> HTML Form ESP32</h3>

    <form action="/action_page">
      First name:<br>
      <input type="text" value="Mickey">
      <br>
      Last name:<br>
      <input type="text" name="lastname" value="Mouse">
      <br><br>
      <input type="submit" value="Submit">
    </form>

  </body>
</html>
```

Open web browser and test your web form

Creating ESP32 Web Server

Connect to WiFi

```
WiFi.begin(ssid, password);
Serial.println("");

// Wait for connection
while (WiFi.status() != WL_CONNECTED)
{
  delay(500);
  Serial.print(".");
}
```

Create web server on root

```
server.on("/", handleRoot);
server.begin();

void handleRoot()
{
  String s = MAIN_page;             //Read HTML contents
  server.send(200, "text/html", s); //Send web page
}
```

Similarly we create form handler, form action is set to “action_page”

```
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<body>

<h2>Circuits4you<h2>
<h3> HTML Form ESP8266</h3>

<form action="/action_page">
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
)=====";
```

```
  server.on("/", handleRoot);      //Which routine to h
  server.on("/action_page", handleForm); //form action

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

```
server.on("/action_page", handleForm); //form action is handled here
```

We can read the form data or user sent data through forms using **server.arg**.

```
String firstName = server.arg("firstname");
```

```
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<body>

<h2>Circuits4you<h2>
<h3> HTML Form ESP8266</h3>

<form action="/action_page">
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>

</body>
</html>
)=====";
```

```
//==============================================
// This routine is executed when you press sub
//==============================================
void handleForm() {
 String firstName = server.arg("firstname");
 String lastName = server.arg("lastname");

 Serial.print("First Name:");
 Serial.println(firstName);

 Serial.print("Last Name:");
 Serial.println(lastName);

 String s = "<a href='/'> Go Back </a>";
 server.send(200, "text/html", s); //Send web
}
```

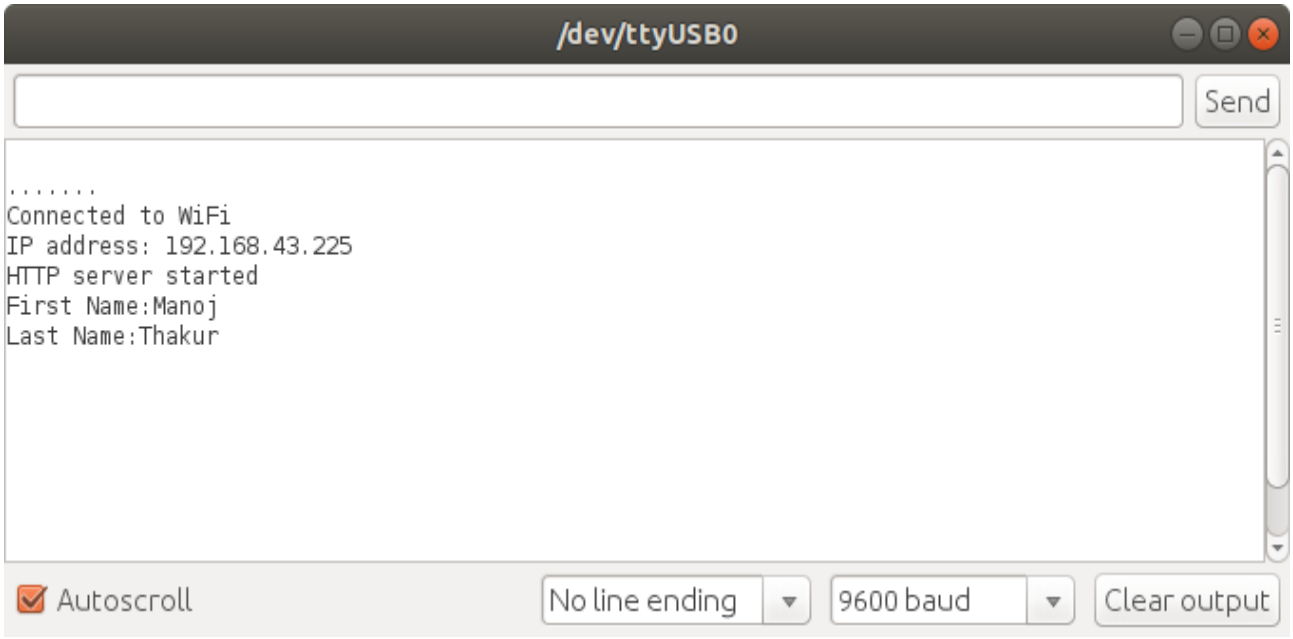
```
//================================================================
// This routine is executed when you press submit
//================================================================
void handleForm()
{
  String firstName = server.arg("firstname");
  String lastName = server.arg("lastname");

  Serial.print("First Name:");
  Serial.println(firstName);

  Serial.print("Last Name:");
  Serial.println(lastName);

  String s = "<a href='/'> Go Back </a>";
  server.send(200, "text/html", s); //Send web page
}
```

Complete code

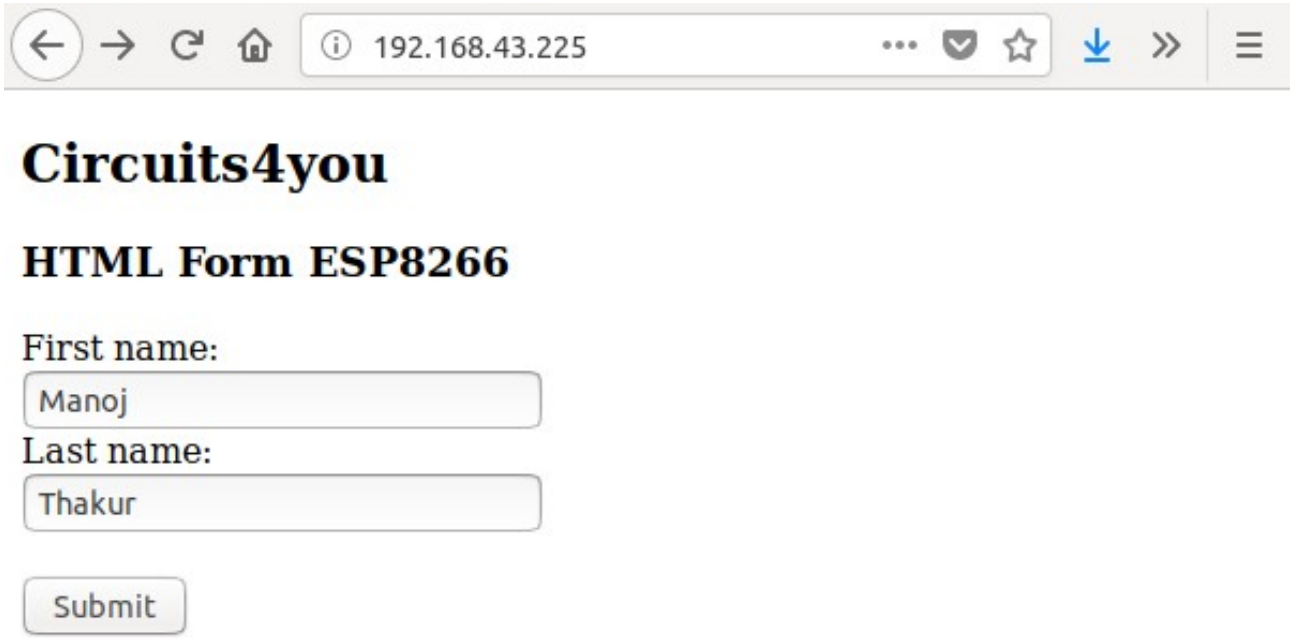
```
/*
 * ESP32 (NodeMCU) Handling Web form data basic example
 */

#include <WiFi.h>
#include <WebServer.h>

const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
  <body>

    <h2>Circuits4you<h2>
    <h3> HTML Form ESP32</h3>

    <form action="/action_page">
      First name:<br>
      <input type="text" name="firstname" value="Mickey">
      <br>
      Last name:<br>
      <input type="text" name="lastname" value="Mouse">
      <br><br>
      <input type="submit" value="Submit">
    </form>

  </body>
</html>
)=====";

//SSID and Password of your WiFi router
const char* ssid     = ".......";
const char* password = ".......";

WebServer server(80);

//===============================================================
// This routine is executed when you open its IP in browser
//===============================================================
void handleRoot()
{
  String s = MAIN_page; //Read HTML contents
  server.send(200, "text/html", s); //Send web page
}

//===============================================================
// This routine is executed when you press submit
//===============================================================
void handleForm()
{
  String firstName = server.arg("firstname");
  String lastName = server.arg("lastname");

  Serial.print("First Name:");
  Serial.println(firstName);

  Serial.print("Last Name:");
  Serial.println(lastName);

  String s = "<a href='/'> Go Back </a>";
  server.send(200, "text/html", s); //Send web page
}
//===============================================================
```

```
//                  SETUP
//=============================================================
void setup(void)
{
  Serial.begin(9600);

  WiFi.begin(ssid, password);
  Serial.println("");

  // Wait for connection
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  //If connection successful show IP address in serial monitor
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println("WiFi");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  server.on("/", handleRoot);
  server.on("/action_page", handleForm); //form action handle

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

//=============================================================
//                     LOOP
//=============================================================
void loop(void)
{
  server.handleClient();          //Handle client requests
}
```

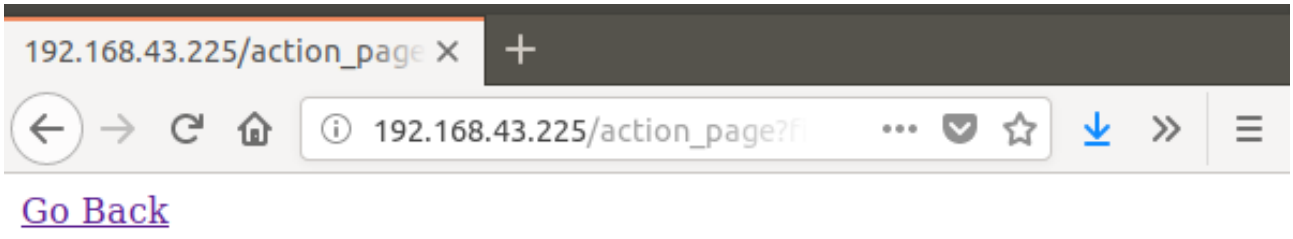
Testing
After uploading, open serial monitor with 9600 baud rate. First you will see only IP address.



Enter the IP address in web browser to open the web page.

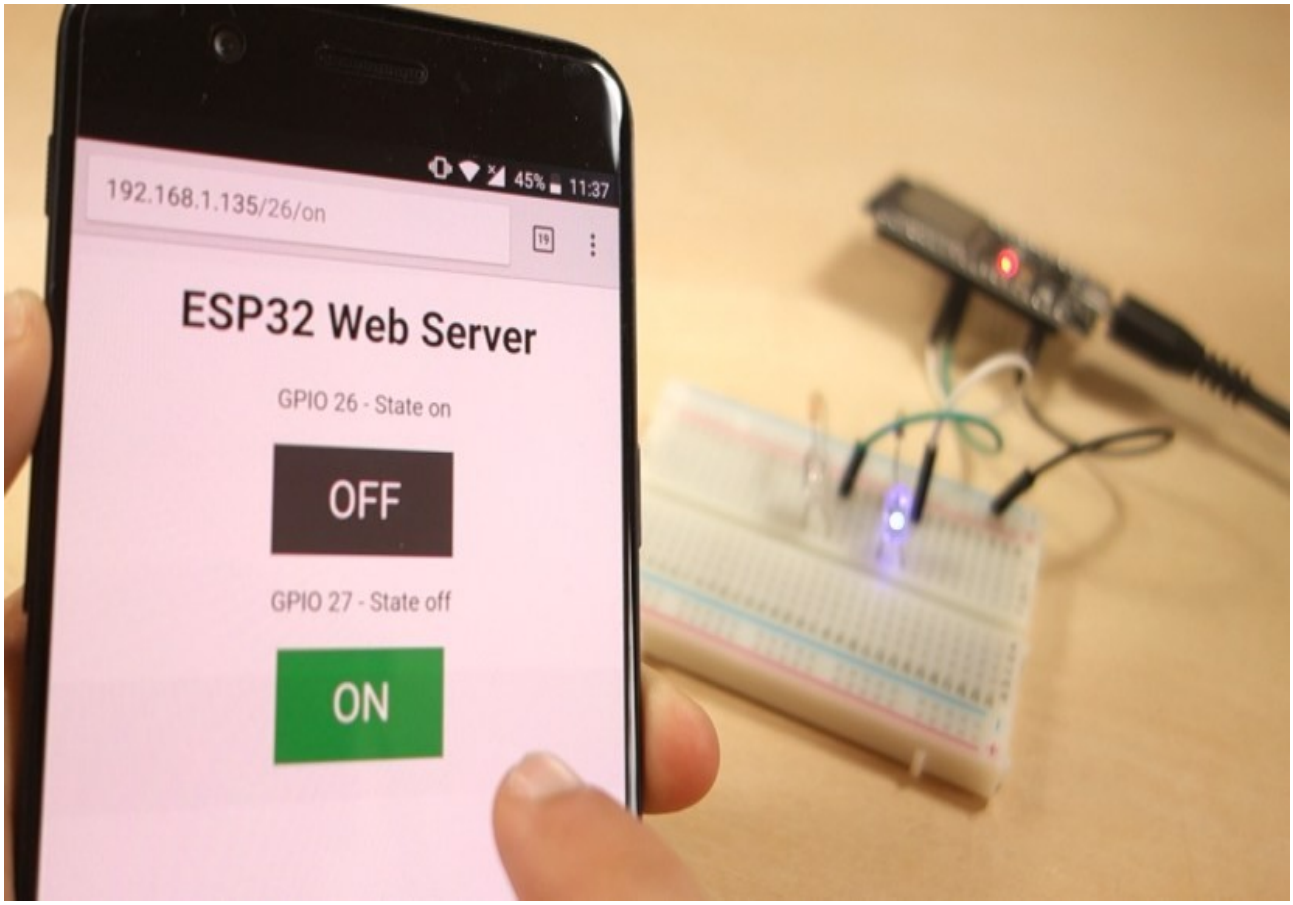


Click on submit button and observer the serial monitor



Web server using library “WiFi” with include library “WiFiServer”

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 as a browser on the local network. We’ll show you how to create the web server and how the code works step-by-step.



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.

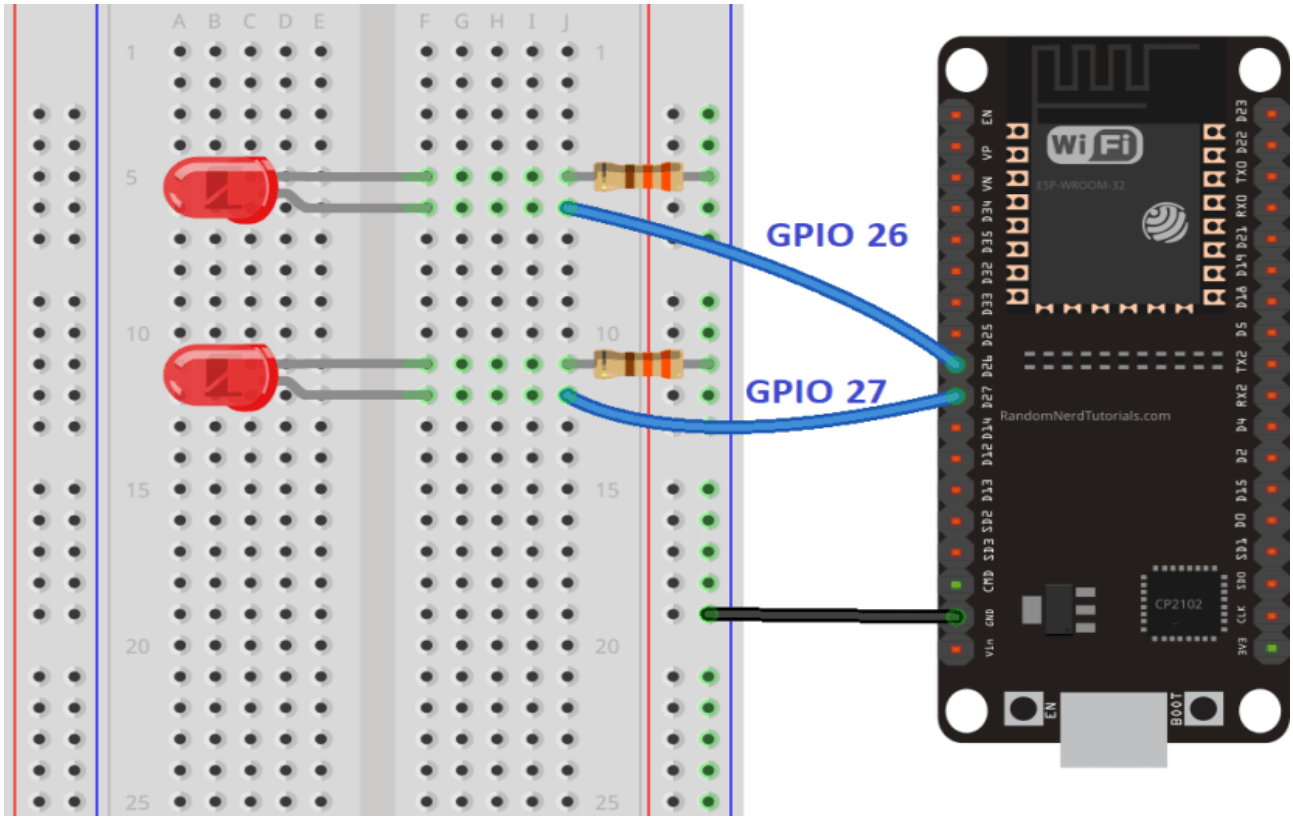
- The web server build controls two LEDs connected to the ESP32 GPIO 26 and GPIO 27;
- You can access the ESP32 web server by typing the ESP32 IP address on a browser in the local network;
- By clicking the buttons on your web server you can instantly change the state of each LED.

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.

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

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.

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

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

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

  // If a new client connects,
  if (client)
  {
    currentTime = millis();
    previousTime = currentTime;
    // print a message out in the serial port
    Serial.println("New Client.");
    // make a String to hold incoming data from the client
    String currentLine = "";
    // loop while the client's connected
    while (client.connected() && currentTime - previousTime <= timeoutTime)
    {
      currentTime = millis();
      // if there's bytes to read from the client,
      if (client.available())
      {
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        header += c;
        // if the byte is a newline character
        if (c == '\n')
        {

          // if the current line is blank, you got two newlines 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
            // 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);
```

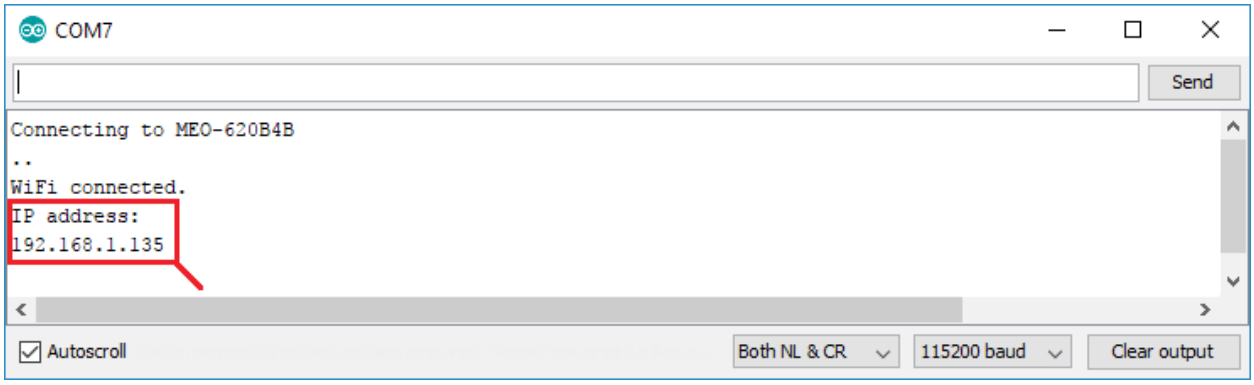
```
            }

            // 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
            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,
          // add it to the end of the currentLine
```

```
            currentLine += c;
          }
        }
      }

      // Clear the header variable
      header = "";

      // Close the connection
      client.stop();
      Serial.println("Client disconnected.");
      Serial.println("");
    }
  }
```

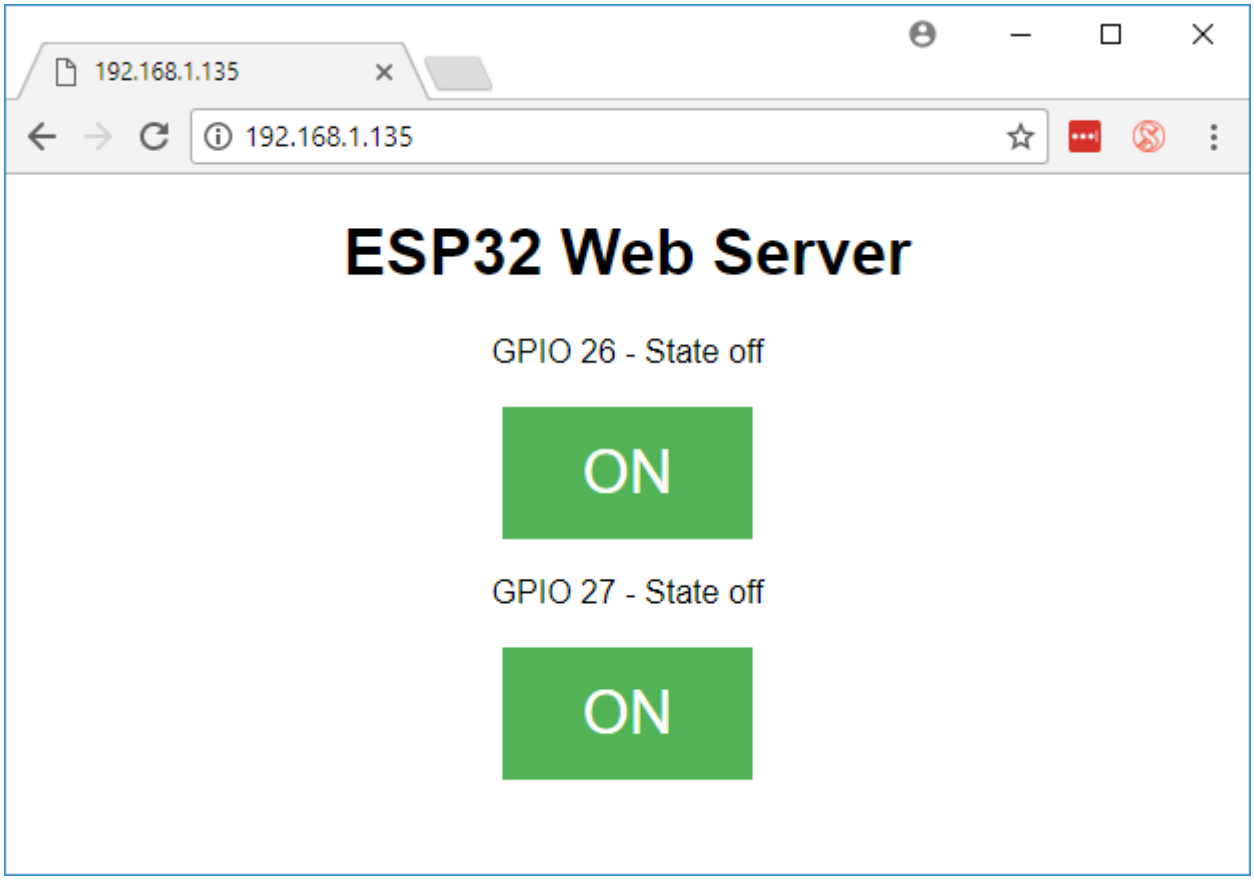
Setting Your Network Credentials
You need to modify the following lines with your network credentials: SSID and password. The code is well commented on where you should make the changes.

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

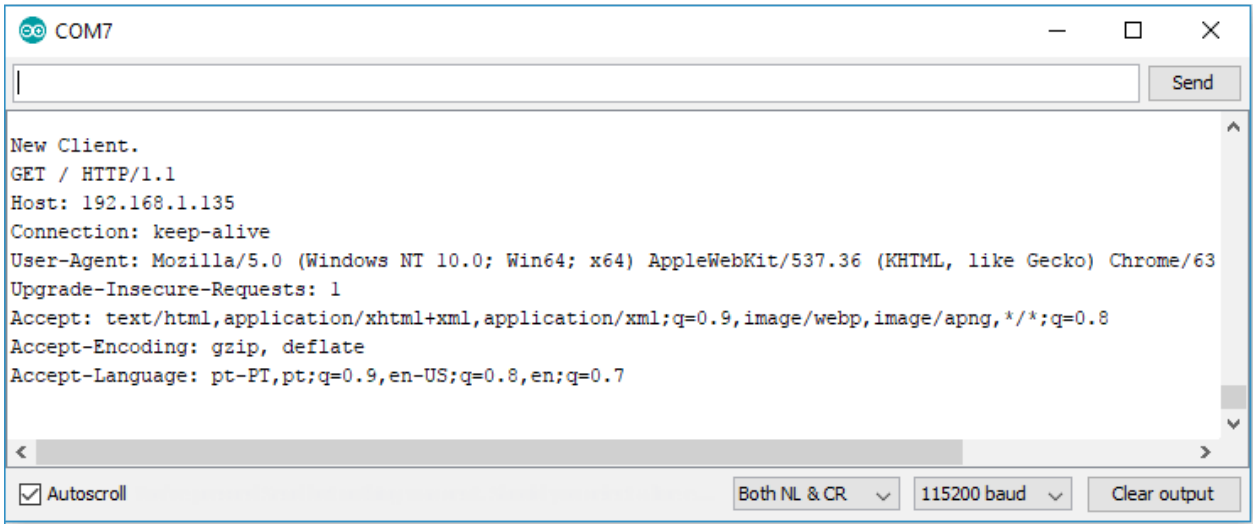
Finding the ESP IP Address
After uploading the code, open the Serial Monitor at a baud rate of 115200.
Press the ESP32 EN button (reset). The ESP32 connects to Wi-Fi, and outputs the ESP IP address on the Serial Monitor. Copy that IP address, because you need it to access the ESP32 web server.



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.

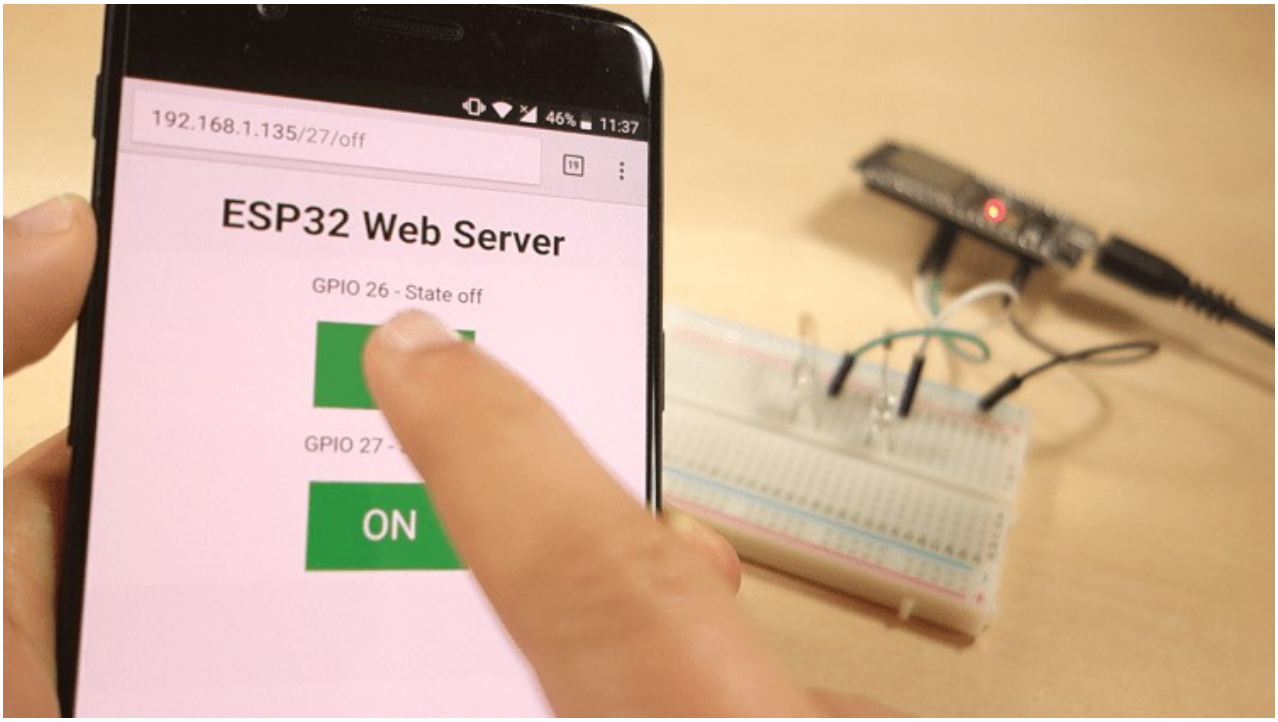


If you take a look at the Serial Monitor, you can see what's happening on the background. The ESP receives an HTTP request from a new client (in this case, your browser).

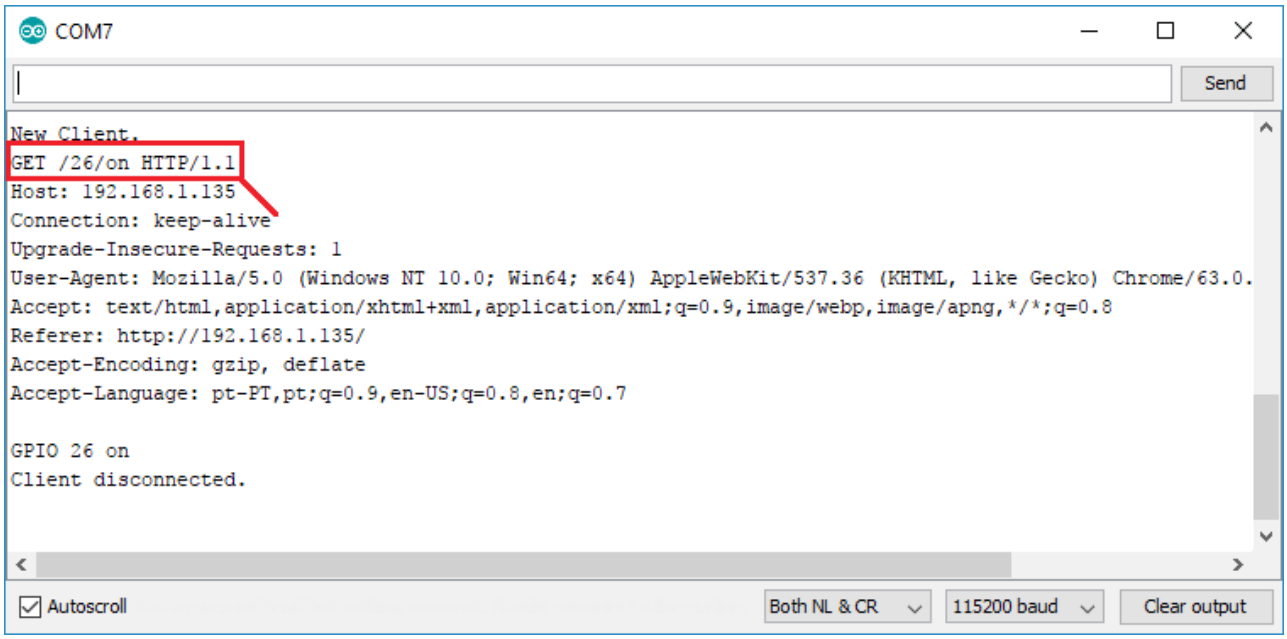


You can also see other information about the HTTP request.

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 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.

```
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:

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

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.

```
    // If a new client connects,
    if (client)
    {
      currentTime = millis();
      previousTime = currentTime;
      // print a message out in the serial port
      Serial.println("New Client.");
      // make a String to hold incoming data from the client
      String currentLine = "";
      // loop while the client's connected
      while (client.connected() && currentTime - previousTime <= timeoutTime)
      {
        currentTime = millis();
        // if there's bytes to read from the client,
        if (client.available())
        {
          char c = client.read();             // read a byte, then
          Serial.write(c);                    // print it out the serial monitor
          header += c;
          // if the byte is a newline character
          if (c == '\n')
          {

            // if the current line is blank, you got two newlines 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
              // 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.

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:,\">");
```

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>");
```

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("<body><h1>ESP32 Web Server</h1>");
```

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.

```
// Display current state, and ON/OFF buttons for GPIO 26
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 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>");
}
```

We use the same procedure for GPIO 27.

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();
```