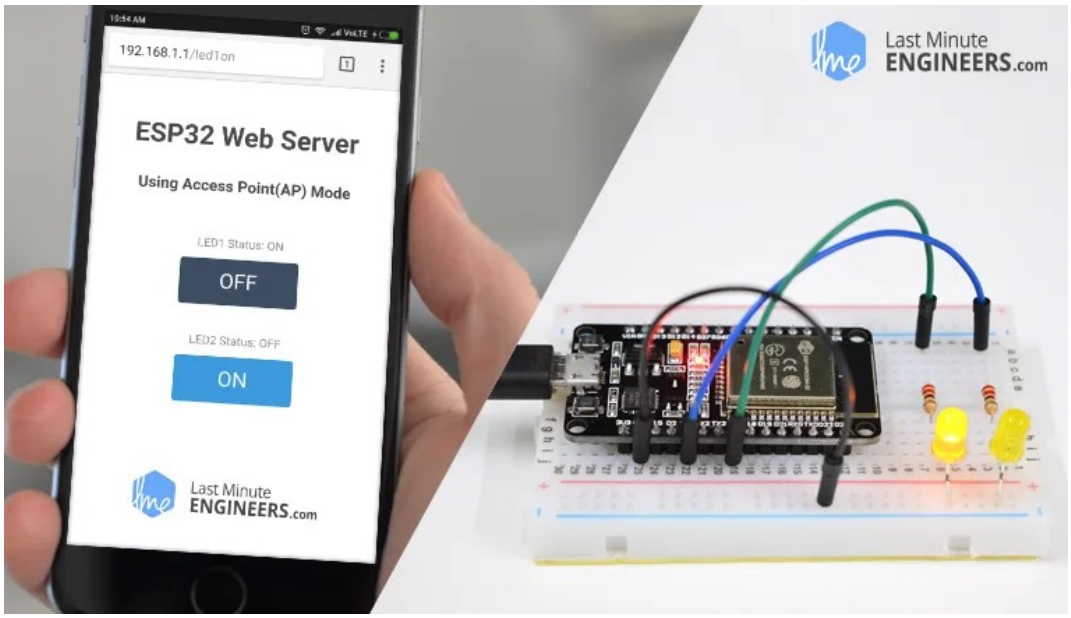
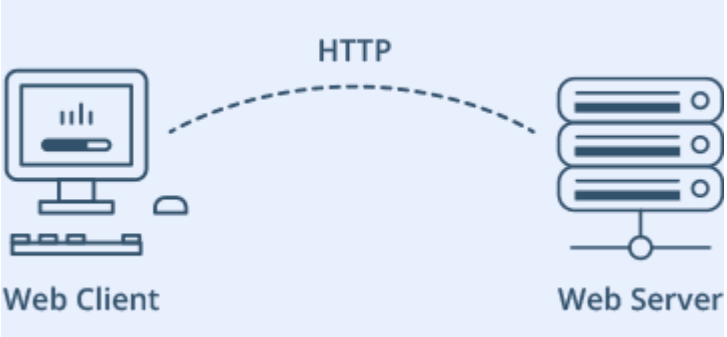
Create A Simple ESP32 Web Server In Arduino IDE



The newly launched successor of ESP8266 – the **ESP32** has been a growing star among IoT or WiFi-related projects. It's an extremely cost-effective WiFi module that – with a little extra effort – can be programmed to **build a standalone web server**. How cool is that!

What is a Web server and how it works?

Web server is a place which stores, processes and delivers web pages to Web clients. Web client is nothing but a web browser on our laptops and smartphones. The communication between client and server takes place using a special protocol called Hypertext Transfer Protocol (HTTP).



In this protocol, a client initiates communication by making a request for a specific web page using HTTP and the server responds with the content of that web page or an error message if unable to do so (like famous 404 Error). Pages delivered by a server are mostly HTML documents.

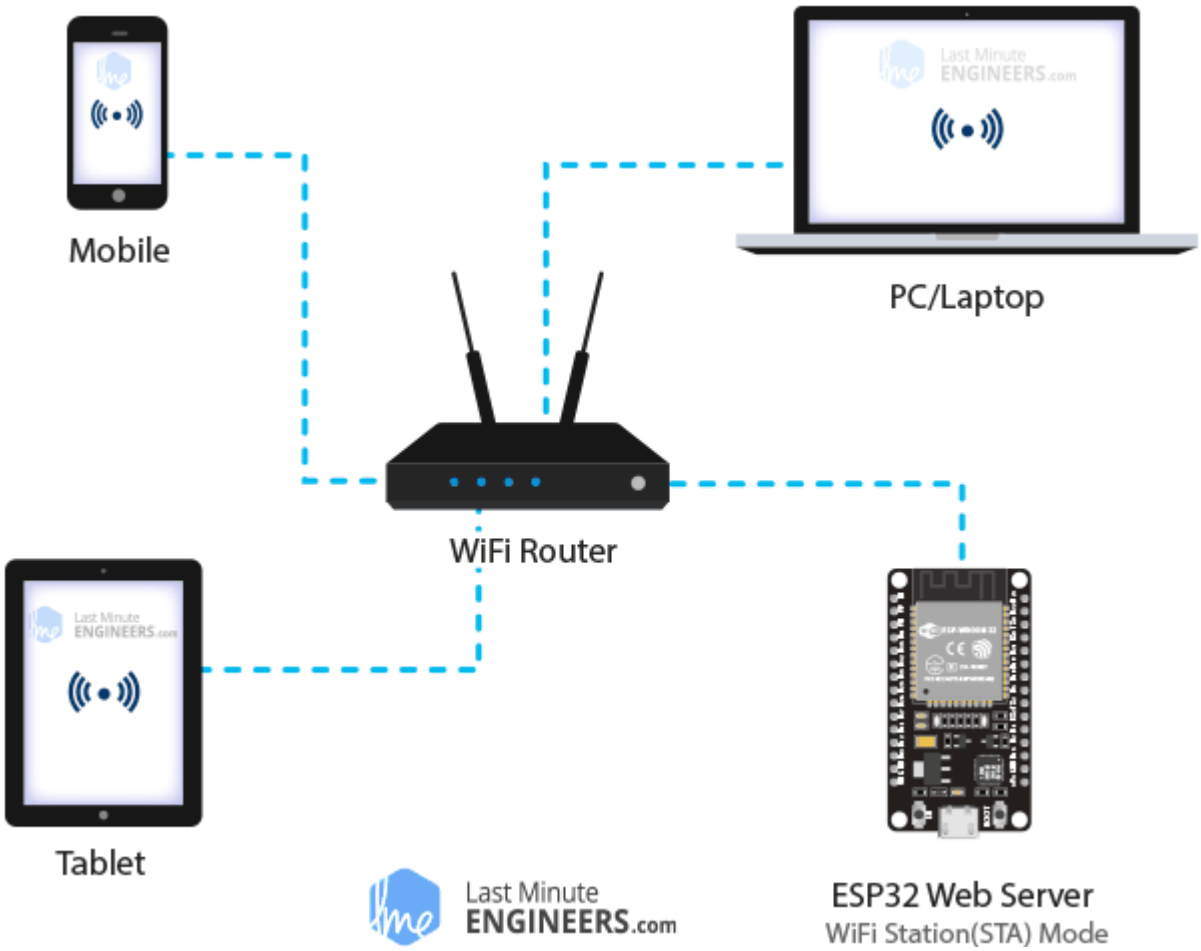
ESP32 Operating Modes

One of the greatest features ESP32 provides is that it cannot only connect to an existing WiFi network and act as a Web Server, but it can also set up a network of its own, allowing other devices

to connect directly to it and access web pages. This is possible because ESP32 can operate in three different modes: Station mode, Soft Access Point mode, and both at the same time. This provides possibility of building mesh networks.

Station (STA) Mode

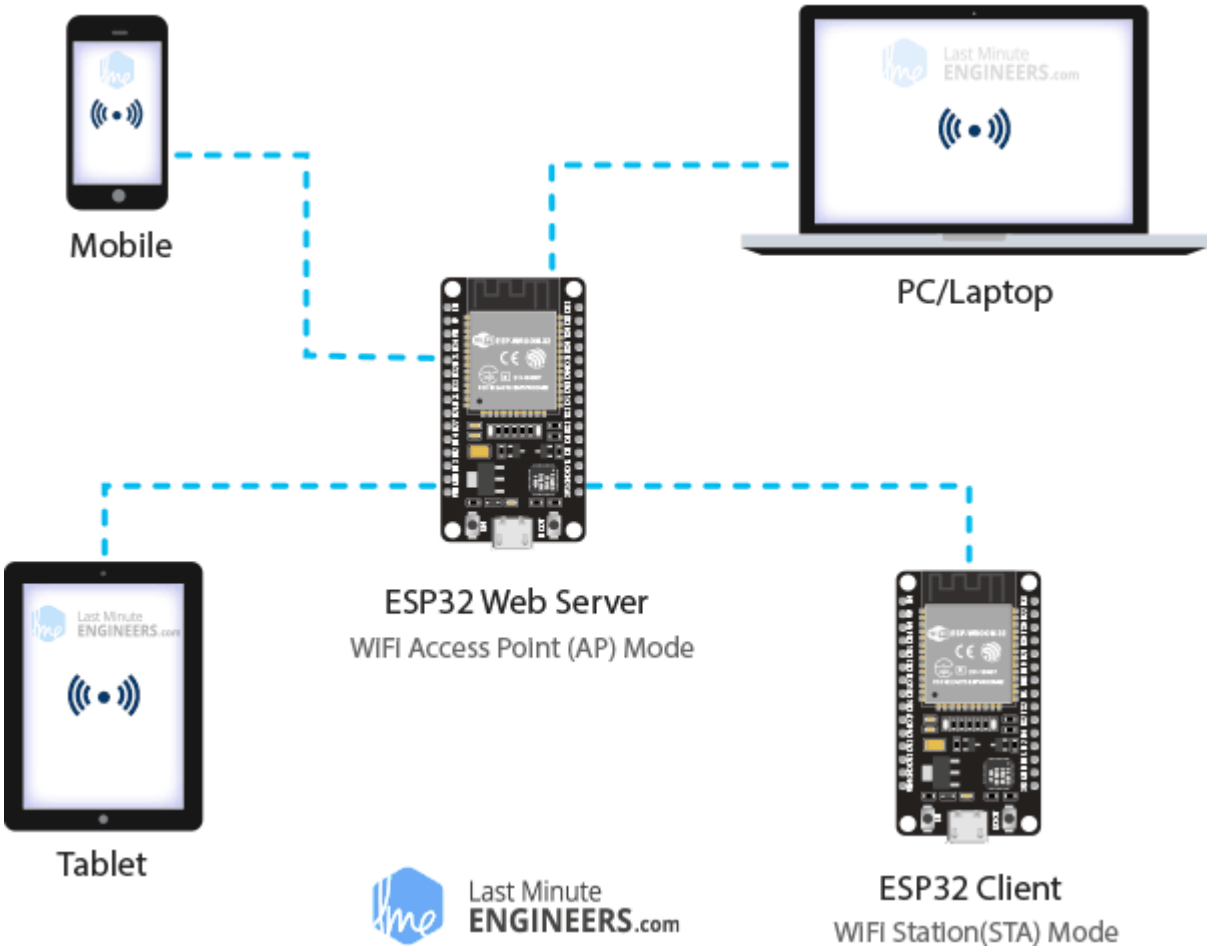
The ESP32 that connects to an existing WiFi network (one created by your wireless router) is called **Station** (STA)



In STA mode ESP32 gets IP from wireless router to which it is connected. With this IP address, it can set up a web server and **deliver web pages to all connected devices under existing WiFi network**.

Soft Access Point (AP) Mode

The ESP32 that creates its own WiFi network and acts as a hub (Just like WiFi router) for one or more stations is called **Access Point** (AP). Unlike WiFi router, it does not have interface to a wired network. So, such mode of operation is called **Soft Access Point** (soft-AP). Also the maximum number of stations that can connect to it is limited to five.



In AP mode ESP32 creates a new WiFi network and sets SSID (Name of the network) and IP address to it. With this IP address, it can **deliver web pages to all connected devices under its own network**.

Wiring – Connecting LEDs to ESP32

Now that we know the basics of how web server works, and in which modes ESP32 can create a web server, it's time to connect some LEDs to ESP32 that we want to control over WiFi.

Start by placing the ESP32 on to your breadboard, ensuring each side of the board is on a separate side of the breadboard. Next, connect two LEDs to digital GPIO 4 and 5 through a 220Ω current limiting resistor.

When you're done you should have something that looks similar to the illustration shown below.

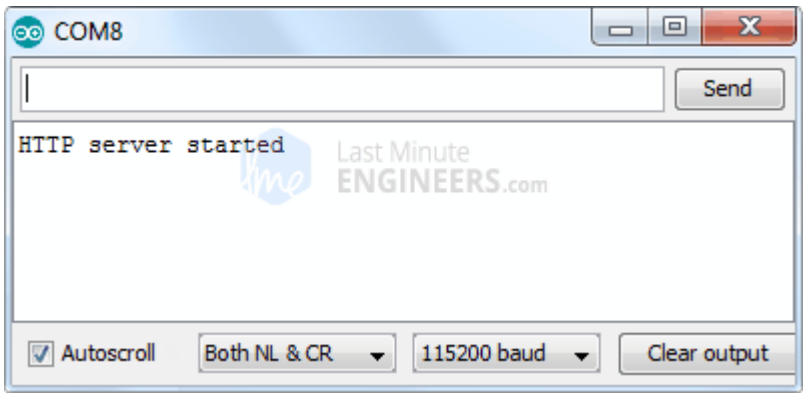
```
#include <WiFi.h>
#include <WebServer.h>

/* Put your SSID & Password */
const char* ssid = "ESP32";  // Enter SSID here
const char* password = "12345678";  //Enter Password here

/* Put IP Address details */
IPAddress local_ip(192,168,1,1);
IPAddress gateway(192,168,1,1);
IPAddress subnet(255,255,255,0);
```

```
WebServer server(80);

uint8_t LED1pin = 4;
bool LED1status = LOW;

uint8_t LED2pin = 5;
bool LED2status = LOW;

void setup() {
  Serial.begin(115200);
  pinMode(LED1pin, OUTPUT);
  pinMode(LED2pin, OUTPUT);

  WiFi.softAP(ssid, password);
  WiFi.softAPConfig(local_ip, gateway, subnet);
  delay(100);

  server.on("/", handle_OnConnect);
  server.on("/led1on", handle_led1on);
  server.on("/led1off", handle_led1off);
  server.on("/led2on", handle_led2on);
  server.on("/led2off", handle_led2off);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");
}
void loop() {
  server.handleClient();
  if(LED1status)
  {digitalWrite(LED1pin, HIGH);}
  else
  {digitalWrite(LED1pin, LOW);}

  if(LED2status)
  {digitalWrite(LED2pin, HIGH);}
  else
  {digitalWrite(LED2pin, LOW);}
}

void handle_OnConnect() {
  LED1status = LOW;
  LED2status = LOW;
  Serial.println("GPIO4 Status: OFF | GPIO5 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,LED2status));
}

void handle_led1on() {
  LED1status = HIGH;
  Serial.println("GPIO4 Status: ON");
  server.send(200, "text/html", SendHTML(true,LED2status));
}

void handle_led1off() {
  LED1status = LOW;
  Serial.println("GPIO4 Status: OFF");
  server.send(200, "text/html", SendHTML(false,LED2status));
}

void handle_led2on() {
  LED2status = HIGH;
  Serial.println("GPIO5 Status: ON");
  server.send(200, "text/html", SendHTML(LED1status,true));
```

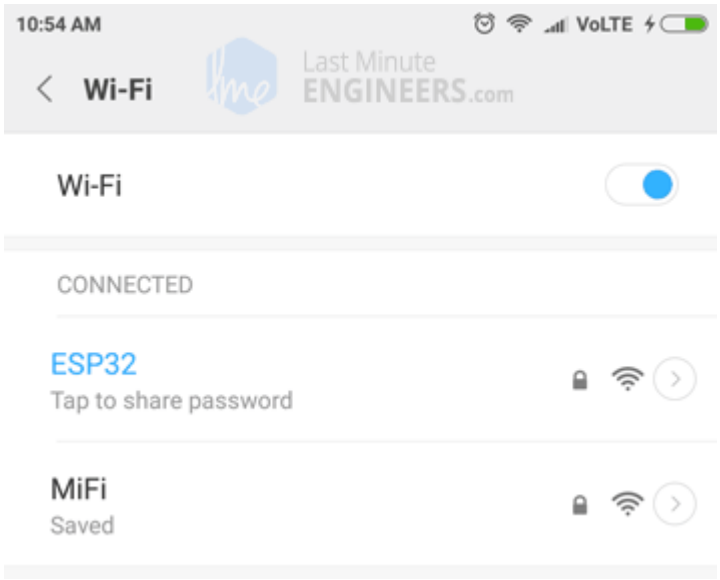
```
}

void handle_led2off() {
  LED2status = LOW;
  Serial.println("GPIO5 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,false));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}

String SendHTML(uint8_t led1stat,uint8_t led2stat){
  String ptr = "<!DOCTYPE html> <html>\n";
  ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1.0, user-scalable=no\">\n";
  ptr +="<title>LED Control</title>\n";
  ptr +="<style>html { font-family: Helvetica; display: inline-block; margin:
0px auto; text-align: center;}\n";
  ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3
{color: #444444;margin-bottom: 50px;}\n";
  ptr +=".button {display: block;width: 80px;background-color: #3498db;border:
none;color: white;padding: 13px 30px;text-decoration: none;font-size:
25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";
  ptr +=".button-on {background-color: #3498db;}\n";
  ptr +=".button-on:active {background-color: #2980b9;}\n";
  ptr +=".button-off {background-color: #34495e;}\n";
  ptr +=".button-off:active {background-color: #2c3e50;}\n";
  ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";
  ptr +="</style>\n";
  ptr +="</head>\n";
  ptr +="<body>\n";
  ptr +="<h1>ESP32 Web Server</h1>\n";
  ptr +="<h3>Using Access Point(AP) Mode</h3>\n";

   if(led1stat)
  {ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\"
href=\"/led1off\">OFF</a>\n";}
  else
  {ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\"
href=\"/led1on\">ON</a>\n";}

  if(led2stat)
  {ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\"
href=\"/led2off\">OFF</a>\n";}
  else
  {ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\"
href=\"/led2on\">ON</a>\n";}

  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

Accessing the Web Server in AP mode

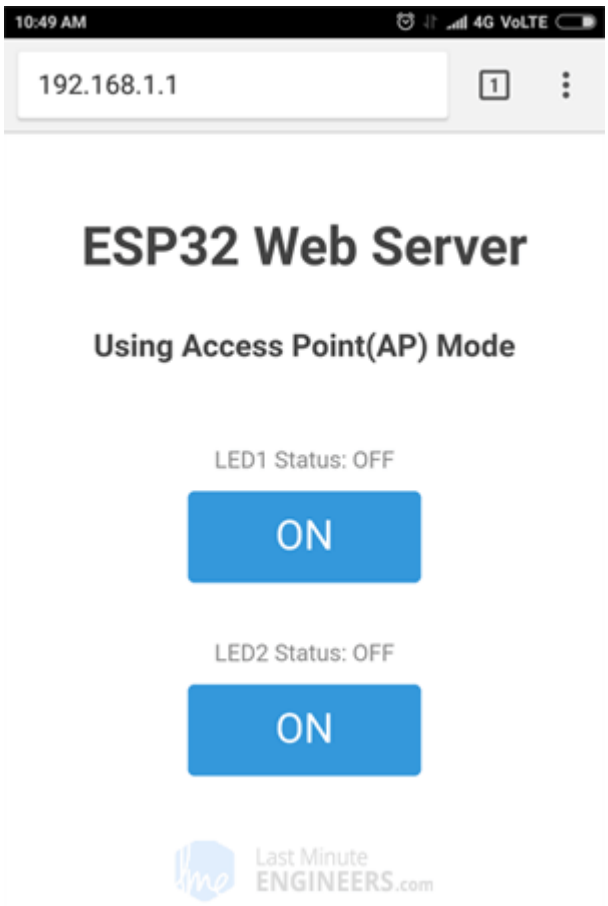
After uploading the sketch, open the Serial Monitor at a baud rate of 115200. And press the RESET button on ESP32. If everything is OK, it will show **HTTP server started** message.

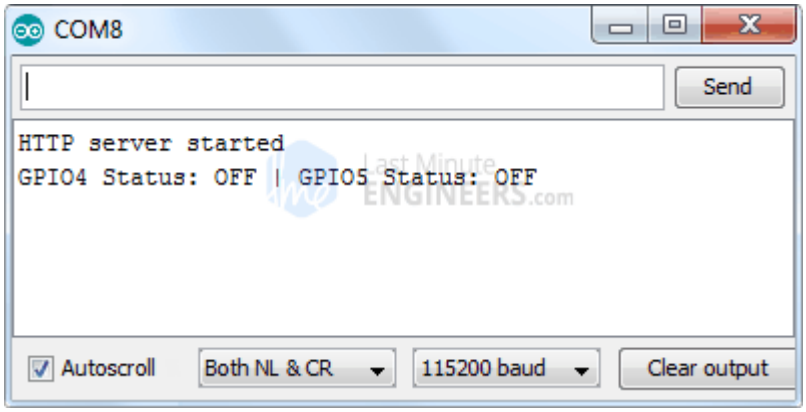


Next, find any device that you can connect to a WiFi network – phone, laptop, etc. And look for a network called **ESP32**. Join the network with password **123456789**.

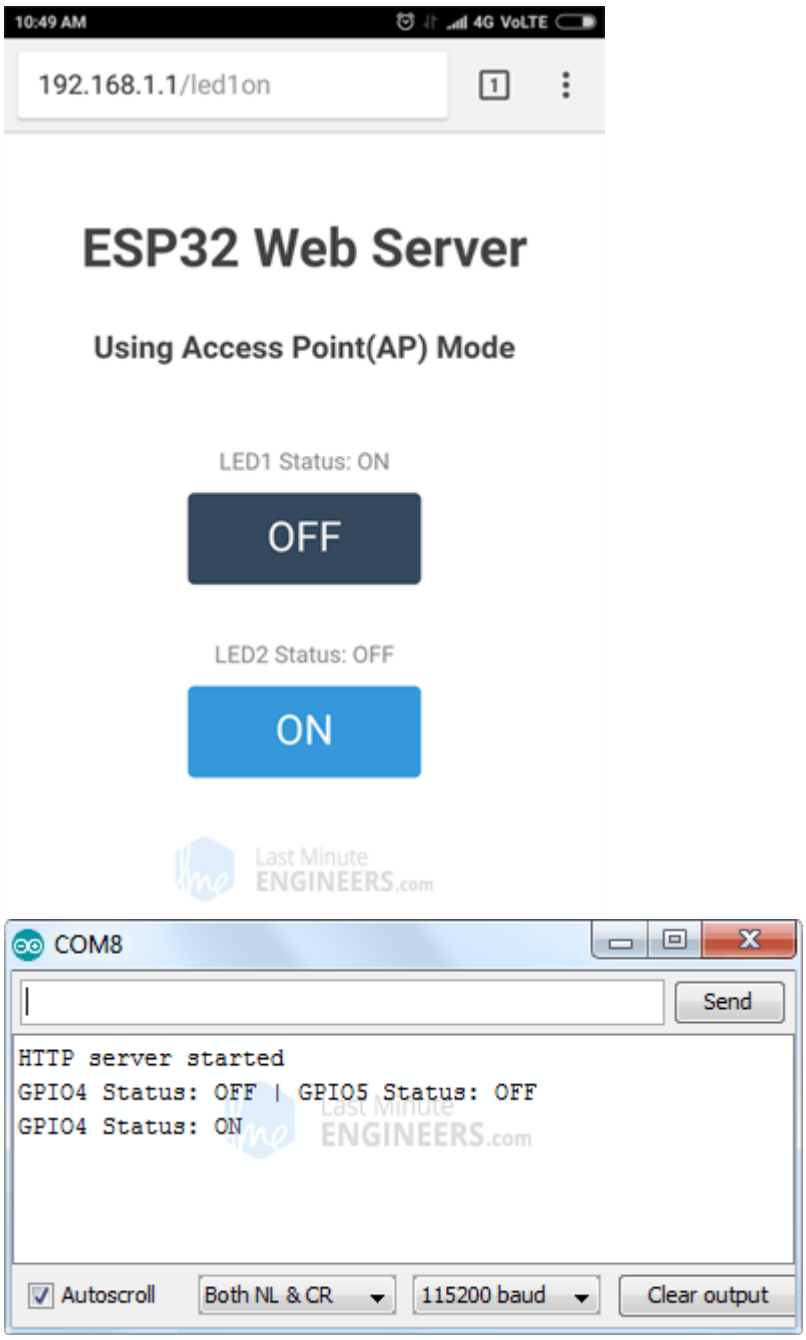


After connecting to your ESP32 AP network, load up a browser and point it to 192.168.1.1 The ESP32 should serve up a web page showing current status of LEDs and two buttons to control them. If take a look at the serial monitor at the same time, you can see status of ESP32's GPIO pins.





Now, click the button to turn LED1 ON while keeping an eye on the URL. Once you click the button, the ESP32 receives a request for **/led1on** URL. It then turns the LED1 ON and serves a web page with status of LED updated. It also prints the status of GPIO pin on the serial monitor.



You can test LED2 button and check that it works in a similar way.

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.

Detailed Code Explanation

The sketch starts by including **WiFi.h** library. This library provides ESP32 specific WiFi methods we are calling to connect to network. Following that we also include the **WebServer.h** library, which has some methods available that will help us setting up a server and handle incoming HTTP requests without needing to worry about low level implementation details.

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

As we are setting the ESP32 in Access Point (AP) mode, it will create a WiFi network. Hence, we need to set its SSID, Password, IP address, IP subnet mask and IP gateway.

```
/* Put your SSID & Password */
const char* ssid = "ESP32";  // Enter SSID here
const char* password = "12345678";  //Enter Password here

/* Put IP Address details */
IPAddress local_ip(192,168,1,1);
IPAddress gateway(192,168,1,1);
IPAddress subnet(255,255,255,0);
```

Next, we declare an object of **WebServer** library, so we can access its functions. The constructor of this object takes port (where the server will be listening to) as a parameter. Since 80 is the default port for HTTP, we will use this value. Now you can access the server without needing to specify the port in the URL.

```
// declare an object of WebServer library
WebServer server(80);
```

Next, we declare the ESP32's GPIO pins to which LEDs are connected and their initial state.

```
uint8_t LED1pin = 4;
bool LED1status = LOW;

uint8_t LED2pin = 5;
bool LED2status = LOW;
```

Inside Setup() Function

We configure our HTTP server before actually running it. First of all, we open a serial connection for debugging purpose and set GPIO ports to OUTPUT.

```
Serial.begin(115200);
pinMode(LED1pin, OUTPUT);
pinMode(LED2pin, OUTPUT);
```

Then, we set up a soft access point to establish a Wi-Fi network by proving SSID, Password, IP address, IP subnet mask and IP gateway.

```
WiFi.softAP(ssid, password);
WiFi.softAPConfig(local_ip, gateway, subnet);
delay(100);
```

In order to handle incoming HTTP requests, we need to specify which code to execute when a particular URL is hit. To do so, we use **on** method. This method takes two parameters. First one is a URL path and second one is the name of function which we want to execute when that URL is hit.

For example, the first line of below code snippet indicates that when a server receives an HTTP request on the root (/) path, it will trigger the `handle_OnConnect()` function. Note that the URL specified is a relative path.

Likewise, we need to specify 4 more URLs to handle two states of 2 LEDs.

```
server.on("/", handle_OnConnect);
server.on("/led1on", handle_led1on);
server.on("/led1off", handle_led1off);
server.on("/led2on", handle_led2on);
```

```
server.on("/led2off", handle_led2off);
```

We haven't specified what the server should do if the client requests any URL other than specified with `server.on()` . It should respond with an HTTP status 404 (Not Found) and a message for the user. We put this in a function as well, and use `server.onNotFound()` to tell it that it should execute it when it receives a request for a URI that wasn't specified with server.on

```
server.onNotFound(handle_NotFound);
```

Now, to start our server, we call the begin method on the server object.

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

Inside Loop() Function

To handle the actual incoming HTTP requests, we need to call the `handleClient()` method on the server object. We also change the state of LED as per the request.

```
void loop() {
  server.handleClient();
  if(LED1status)
  {digitalWrite(LED1pin, HIGH);}
  else
  {digitalWrite(LED1pin, LOW);}

  if(LED2status)
  {digitalWrite(LED2pin, HIGH);}
  else
  {digitalWrite(LED2pin, LOW);}
}
```

Next, we need to create a function we attached to root (/) URL with server.on. Remember? At the start of this function, we set the status of both the LEDs to LOW (Initial state of LEDs) and print it on serial monitor. In order to respond to the HTTP request, we use the **send** method. Although the method can be called with a different set of arguments, its simplest form consists of the HTTP response code, the content type and the content.

In our case, we are sending the code **200** (one of the HTTP status codes), which corresponds to the **OK** response. Then, we are specifying the content type as "text/html", and finally we are calling SendHTML() custom function which creates a dynamic HTML page containing status of LEDs.

```
void handle_OnConnect() {
  LED1status = LOW;
  LED2status = LOW;
  Serial.println("GPIO4 Status: OFF | GPIO5 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,LED2status));
}
```

Likewise, we need to create four functions to handle LED ON/OFF requests and 404 Error page.

```
void handle_led1on() {
  LED1status = HIGH;
  Serial.println("GPIO4 Status: ON");
  server.send(200, "text/html", SendHTML(true,LED2status));
}

void handle_led1off() {
```

```
  LED1status = LOW;
  Serial.println("GPIO4 Status: OFF");
  server.send(200, "text/html", SendHTML(false,LED2status));
}

void handle_led2on() {
  LED2status = HIGH;
  Serial.println("GPIO5 Status: ON");
  server.send(200, "text/html", SendHTML(LED1status,true));
}

void handle_led2off() {
  LED2status = LOW;
  Serial.println("GPIO5 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,false));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}
```

Displaying the HTML Web Page

`SendHTML()` function is responsible for generating a web page whenever the ESP32 web server gets a request from a web client. It merely concatenates HTML code into a big string and returns to the `server.send()` function we discussed earlier. The function takes status of LEDs as a parameter to dynamically generate the HTML content.

The first text you should always send is the **<!DOCTYPE>** declaration that indicates that we're sending HTML code.

```
String SendHTML(uint8_t led1stat,uint8_t led2stat){
String ptr = "<!DOCTYPE html> <html>\n";
```

Next, the <meta> viewport element makes the web page responsive in any web browser. While title tag sets the title of the page.

```
ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1.0, user-scalable=no\">\n";
ptr +="<title>LED Control</title>\n";
```

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 an inline-block and aligned at the center.

```
ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px
auto; text-align: center;}\n";
```

Following code then sets color, font and margin around the body, H1, H3 and p tags.

```
ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3
{color: #444444;margin-bottom: 50px;}\n";
ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";
```

Some styling is applied to the buttons as well with properties like color, size, margin, etc. The ON and OFF button has different background color while :active selector for buttons ensure button click effect.

```
ptr +=".button {display: block;width: 80px;background-color: #3498db;border:
none;color: white;padding: 13px 30px;text-decoration: none;font-size:
25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";
ptr +=".button-on {background-color: #3498db;}\n";
ptr +=".button-on:active {background-color: #2980b9;}\n";
ptr +=".button-off {background-color: #34495e;}\n";
ptr +=".button-off:active {background-color: #2c3e50;}\n";
```

Setting the Web Page Heading

Next, heading of the web page is set; you can change this text to anything that suits your application.

```
ptr +="<h1>ESP32 Web Server</h1>\n";
ptr +="<h3>Using Access Point(AP) Mode</h3>\n";
```

Displaying the Buttons and Corresponding State

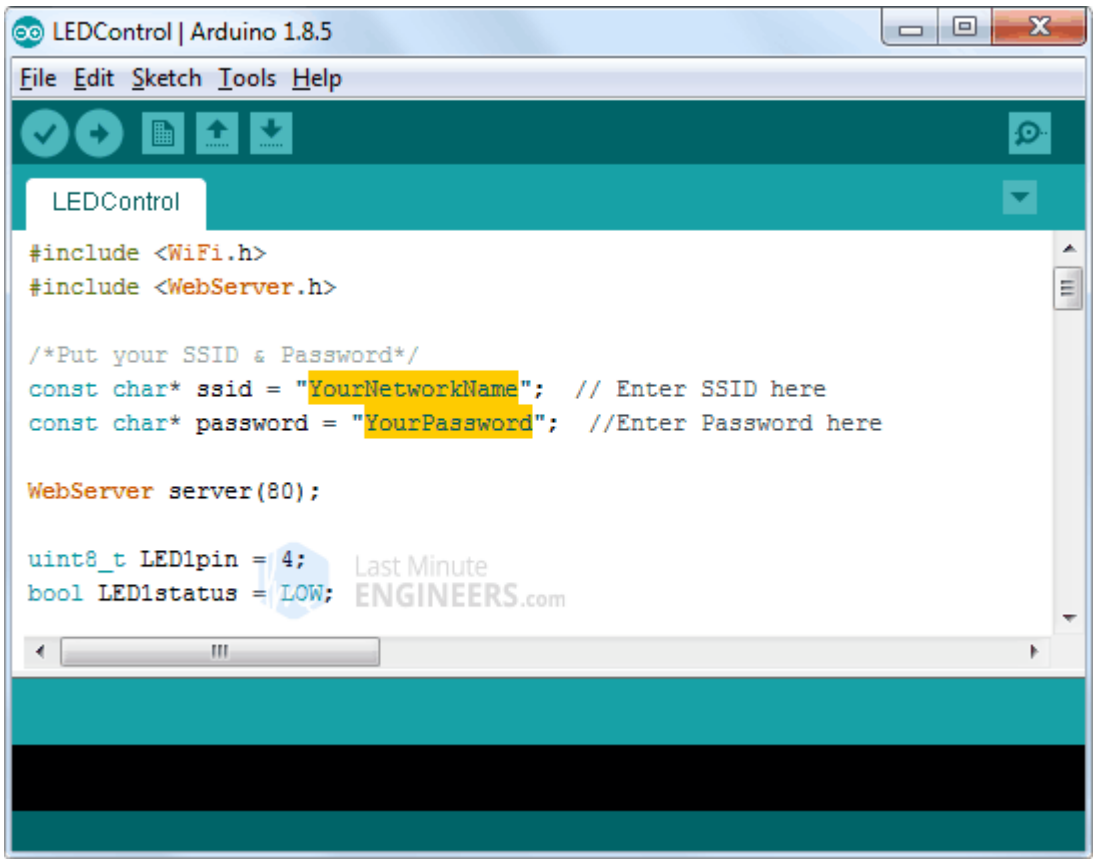
To dynamically generate the buttons and LED status, we use if statement. So, depending upon the status of the GPIO pins, ON/OFF button is displayed.

```
if(led1stat)
 {ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\"
href=\"/led1off\">OFF</a>\n";}
else
 {ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\"
href=\"/led1on\">ON</a>\n";}

if(led2stat)
 {ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\"
href=\"/led2off\">OFF</a>\n";}
else
 {ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\"
href=\"/led2on\">ON</a>\n";}
```

ESP32 as HTTP Server using WiFi Station (STA) mode

Now let’s move on to our next example which demonstrates how to turn the ESP32 into Station (STA) mode, and serve up web pages to any connected client under existing network.

Before you head for uploading the sketch, you need to make some changes to make it work for you. You need to modify the following two variables with your network credentials, so that ESP32 can establish a connection with existing network.



Once you are done, go ahead and try the sketch out.

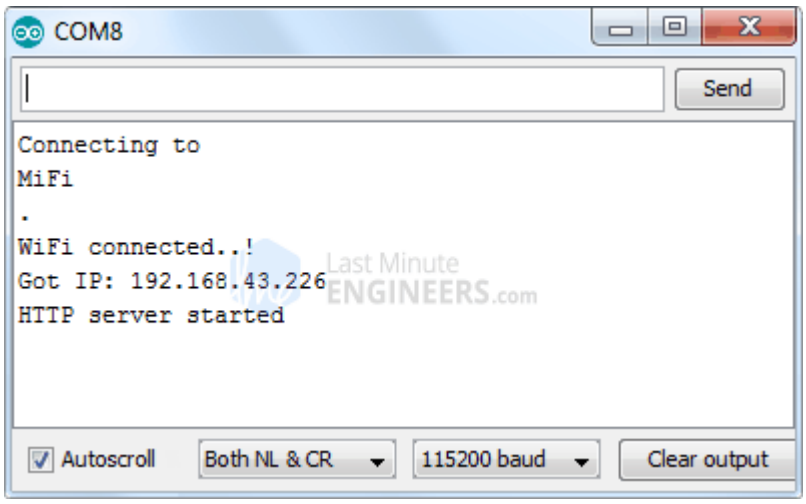
```
#include <WiFi.h>
#include <WebServer.h>

/*Put your SSID & Password*/
const char* ssid = " YourNetworkName";  // Enter SSID here
const char* password = " YourPassword";  //Enter Password here

WebServer server(80);

uint8_t LED1pin = 4;
bool LED1status = LOW;

uint8_t LED2pin = 5;
bool LED2status = LOW;

void setup() {
  Serial.begin(115200);
  delay(100);
  pinMode(LED1pin, OUTPUT);
  pinMode(LED2pin, OUTPUT);

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

  //connect to your local wi-fi network
  WiFi.begin(ssid, password);

  //check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
  }
  Serial.println("");
```

```
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: ");  Serial.println(WiFi.localIP());

  server.on("/", handle_OnConnect);
  server.on("/led1on", handle_led1on);
  server.on("/led1off", handle_led1off);
  server.on("/led2on", handle_led2on);
  server.on("/led2off", handle_led2off);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");
}
void loop() {
  server.handleClient();
  if(LED1status)
  {digitalWrite(LED1pin, HIGH);}
  else
  {digitalWrite(LED1pin, LOW);}

  if(LED2status)
  {digitalWrite(LED2pin, HIGH);}
  else
  {digitalWrite(LED2pin, LOW);}
}

void handle_OnConnect() {
  LED1status = LOW;
  LED2status = LOW;
  Serial.println("GPIO4 Status: OFF | GPIO5 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,LED2status));
}

void handle_led1on() {
  LED1status = HIGH;
  Serial.println("GPIO4 Status: ON");
  server.send(200, "text/html", SendHTML(true,LED2status));
}

void handle_led1off() {
  LED1status = LOW;
  Serial.println("GPIO4 Status: OFF");
  server.send(200, "text/html", SendHTML(false,LED2status));
}

void handle_led2on() {
  LED2status = HIGH;
  Serial.println("GPIO5 Status: ON");
  server.send(200, "text/html", SendHTML(LED1status,true));
}

void handle_led2off() {
  LED2status = LOW;
  Serial.println("GPIO5 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,false));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}

String SendHTML(uint8_t led1stat,uint8_t led2stat){
  String ptr = "<!DOCTYPE html> <html>\n";
```

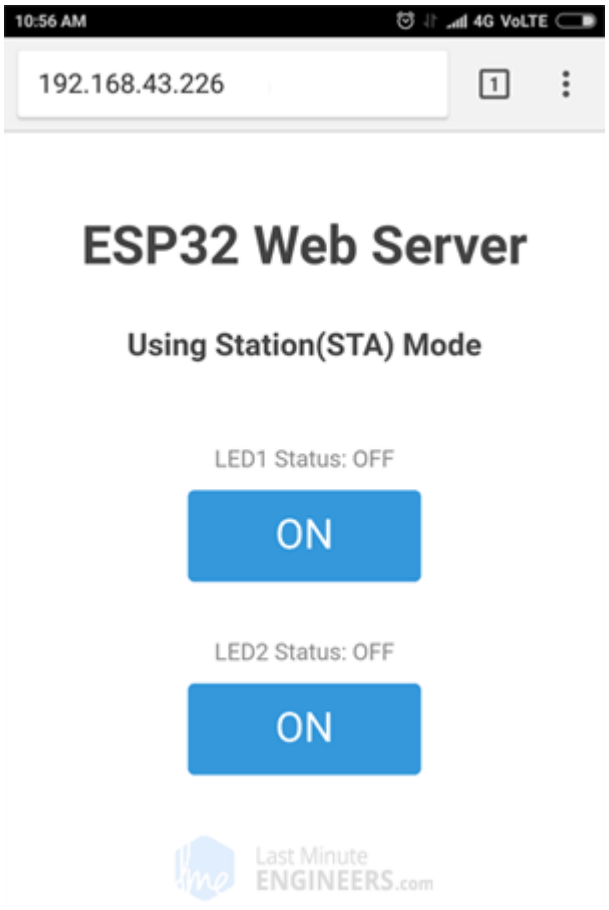
```
  ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1.0, user-scalable=no\">\n";
  ptr +="<title>LED Control</title>\n";
  ptr +="<style>html { font-family: Helvetica; display: inline-block; margin:
0px auto; text-align: center;}\n";
  ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3
{color: #444444;margin-bottom: 50px;}\n";
  ptr +=".button {display: block;width: 80px;background-color: #3498db;border:
none;color: white;padding: 13px 30px;text-decoration: none;font-size:
25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";
  ptr +=".button-on {background-color: #3498db;}\n";
  ptr +=".button-on:active {background-color: #2980b9;}\n";
  ptr +=".button-off {background-color: #34495e;}\n";
  ptr +=".button-off:active {background-color: #2c3e50;}\n";
  ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";
  ptr +="</style>\n";
  ptr +="</head>\n";
  ptr +="<body>\n";
  ptr +="<h1>ESP32 Web Server</h1>\n";
    ptr +="<h3>Using Station(STA) Mode</h3>\n";

   if(led1stat)
  {ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\"
href=\"/led1off\">OFF</a>\n";}
  else
  {ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\"
href=\"/led1on\">ON</a>\n";}

  if(led2stat)
  {ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\"
href=\"/led2off\">OFF</a>\n";}
  else
  {ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\"
href=\"/led2on\">ON</a>\n";}

  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

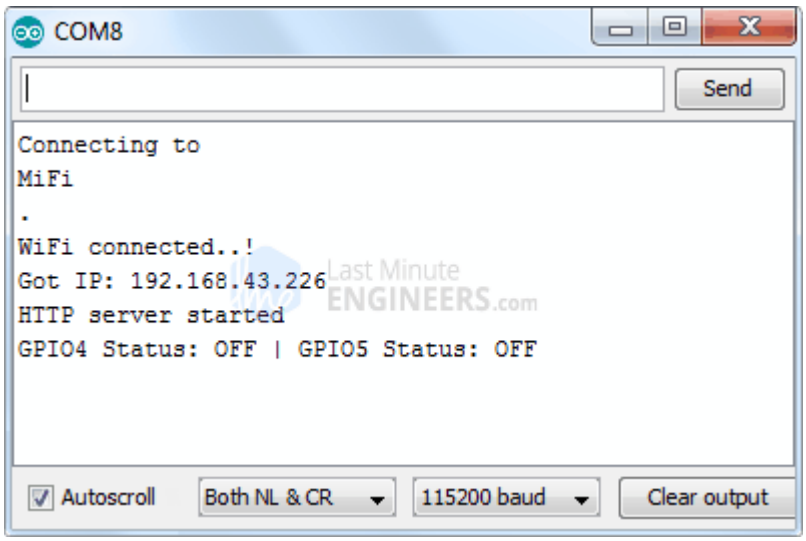
Accessing the Web Server in STA mode

After uploading the sketch, open the Serial Monitor at a baud rate of 115200. And press the RESET button on ESP32. If everything is OK, it will output the dynamic IP address obtained from your router and show **HTTP server started** message.

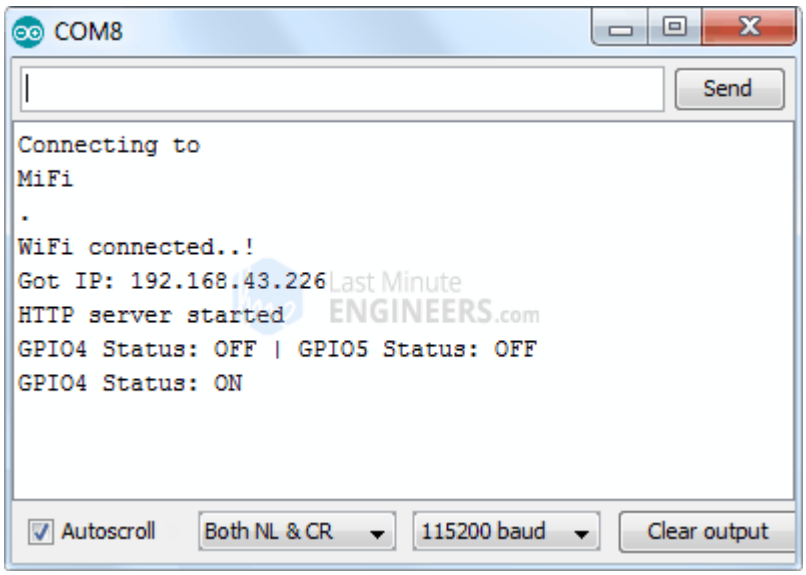


Next, load up a browser and point it to the IP address shown on the serial monitor. The ESP32 should serve up a web page showing current status of LEDs and two buttons to control them. If take a look at the serial monitor at the same time, you can see status of ESP32's GPIO pins.





Now, click the button to turn LED1 ON while keeping an eye on the URL. Once you click the button, the ESP32 receives a request for **/led1on** URL. It then turns the LED1 ON and serves a web page with status of LED updated. It also prints the status of GPIO pin on the serial monitor.



You can test LED2 button and check that it works in a similar way.

Code Explanation

If you observe this code with the previous code, the only difference is that we are not setting the soft Access Point, Instead we are joining existing network using `WiFi.begin()` function.

```
//connect to your local wi-fi network
WiFi.begin(ssid, password);
```

While the ESP32 tries to connect to the network, we can check the connectivity status with `WiFi.status()` function.

```
//check wi-fi is connected to wi-fi network
while (WiFi.status() != WL_CONNECTED)
{
delay(1000);
Serial.print(".");
}
```

Just for your information, this function returns the following statuses:

- **WL_CONNECTED**: assigned when connected to a Wi-Fi network
- **WL_NO_SHIELD**: assigned when no Wi-Fi shield is present
- **WL_IDLE_STATUS**: a temporary status assigned when WiFi.begin() is called and remains active until the number of attempts expires (resulting in WL_CONNECT_FAILED) or a connection is established (resulting in WL_CONNECTED)
- **WL_NO_SSID_AVAIL**: assigned when no SSID are available
- **WL_SCAN_COMPLETED**: assigned when the scan networks is completed
- **WL_CONNECT_FAILED**: assigned when the connection fails for all the attempts
- **WL_CONNECTION_LOST**: assigned when the connection is lost
- **WL_DISCONNECTED**: assigned when disconnected from a network

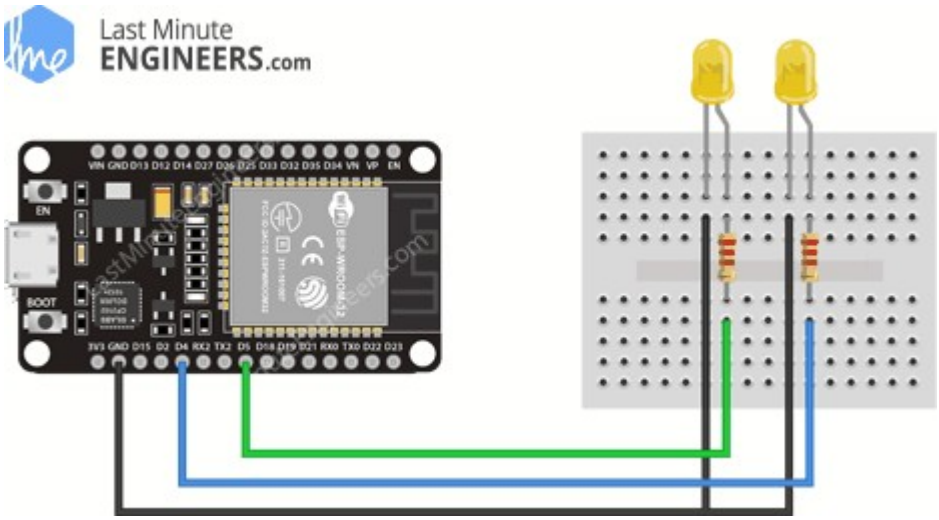
Once the ESP32 is connected to the network, the sketch prints the IP address assigned to ESP32 by displaying `WiFi.localIP()` value on serial monitor.

```
Serial.println("");
```

```
Serial.println("WiFi connected..!");
Serial.print("Got IP: ");  Serial.println(WiFi.localIP());
```

The only difference between AP & STA mode is one creates the network and other joins the existing network. So, rest of the code for handling HTTP requests and serving web page in STA mode is same as that of AP mode explained above. This includes:

- Declaring ESP32's GPIO pins to which LEDs are connected
- Defining multiple server.on() methods to handle incoming HTTP requests
- Defining server.onNotFound() method to handle HTTP 404 error
- Creating custom functions that are executed when specific URL is hit
- Creating HTML page
- Styling the web page
- Creating buttons and displaying their status



Concept Behind Controlling Things From ESP32 Web Server

So, you might be thinking, "How am I going to control things from a web server that merely processes and delivers web pages?" Well, then you need to understand what's going on behind the scene.

When you type a URL in a web browser and hit ENTER, the browser sends a HTTP request (a.k.a. GET request) to a web server. It's a job of web server to handle this request by doing something. You might have figured it out by now that we are going to control things by accessing a specific URL. For example, suppose we entered a URL like http://192.168.1.1/**ledon** in a browser. The browser then sends a HTTP request to ESP32 to handle this request. When ESP32 reads this request, it knows that user wants to turn the LED ON. So, it turns the LED ON and sends a dynamic webpage to a browser showing **LED status : ON**. As easy as Pie!

ESP32 as HTTP Server using WiFi Access Point (AP) mode

Now let's move on to the interesting stuff!

As the heading suggests, this example demonstrates how to turn the ESP32 into an access point (AP), and serve up web pages to any connected client. To start with, plug your ESP32 into your computer and Try the sketch out; and then we will dissect it in some detail.