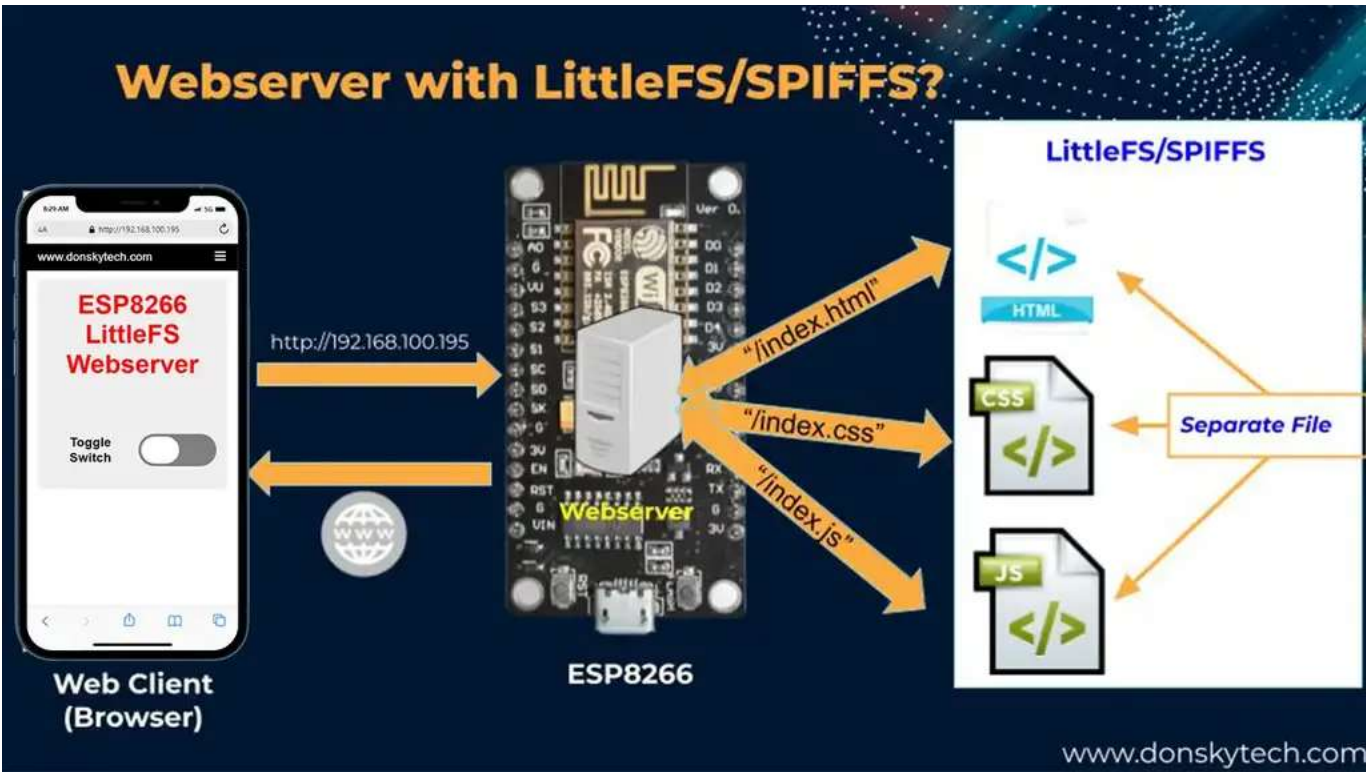


Email: donskey@donskeytech.com




donskeytech.com



Posted on [September 14, 2022](#) by donskey in [ESP8266](#)

ESP8266 Webserver Using LittleFS

Table of Contents 

[Introduction](#)

[Prerequisites](#)

[What we are building?](#)

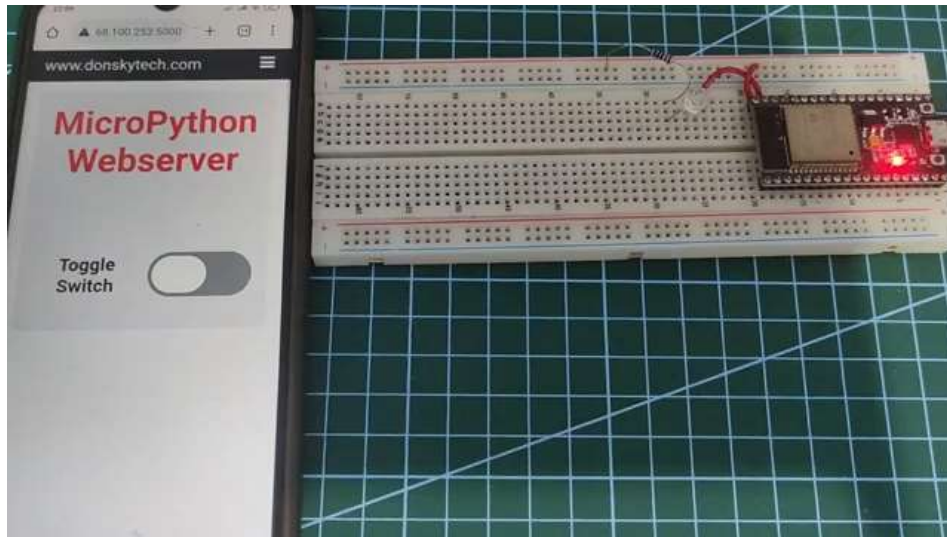
[What is a Webserver?](#)

[Changes to platform.ini](#)[Wrap up](#)

Introduction

Our Microcontrollers Units (MCU) especially the ESP8266 and ESP32 modules are capable of hoisting their own Webserver which you can use in your Internet of Things (IoT) projects. In this post, we will be creating our own ESP8266 Webserver using the LittleFS Filesystem that will serve web content to the calling client. I will show you how easy and simple your code will be by using the LittleFS in your program.

This post is the 2nd part of my ESP8266 LittleFS Tutorial Series where we try to explore the use cases in which knowledge of this feature will greatly help your project.



How to create a MicroPython Web Server the easy way!

- [Part 1 – ESP8266 LittleFS Tutorial Series – Overview](#)
- Part 2 – ESP8266 LittleFS Tutorial Series – Using LittleFS to serve web content in ESP8266 Webserver
- Part 3 – ESP8266 LittleFS Tutorial Series – Using LittleFS to save configurations in ESP8266

If you don't have an idea of what LittleFS or SPIFFS Filesystem is then please read through [Part 1](#) of this series.

Prerequisites

We will only need the following components in this project to follow along.

- ESP8266 (I used NodeMCU ESP12E) – [Amazon](#) | [AliExpress](#) | [Bangood](#)
- Breadboard – [Amazon](#) | [AliExpress](#) | [Bangood](#)
- Connecting Wires – [Amazon](#) | [AliExpress](#) | [Bangood](#)

Disclosure: These are affiliate links and I will earn small commissions to support my site when you buy through these links.

I used Visual Studio Code with the PlatformIO extension installed in developing this project. If you are in Windows and have not yet installed Visual Studio Code then you can check my post about how to [Install Visual Studio Code or VSCode in Windows](#). If you are not familiar with PlatformIO then please read this [PlatformIO Tutorial for Arduino Development](#) where I detailed how to get started with development using the PlatformIO IDE extension

I am using the [mincss](#) stylesheet in this program so that it would render better on a mobile phone. When you see the file **entireframework.min.css** then it refers to the mincss. You can take a look at the example there to know more about this awesome small library.

What we are building?

We are building our own ESP8266 Webserver that will turn on or off the onboard LED of our NodeMCU ESP8266 using our mobile phone. To do this, we will be creating our own NodeMCU ESP8266 Webserver using LittleFS Filesystem.

We will first explore creating a Webserver without using LittleFS and point out the issues then we will show how using LittleFS (or SPIFFS) will solve those issues.

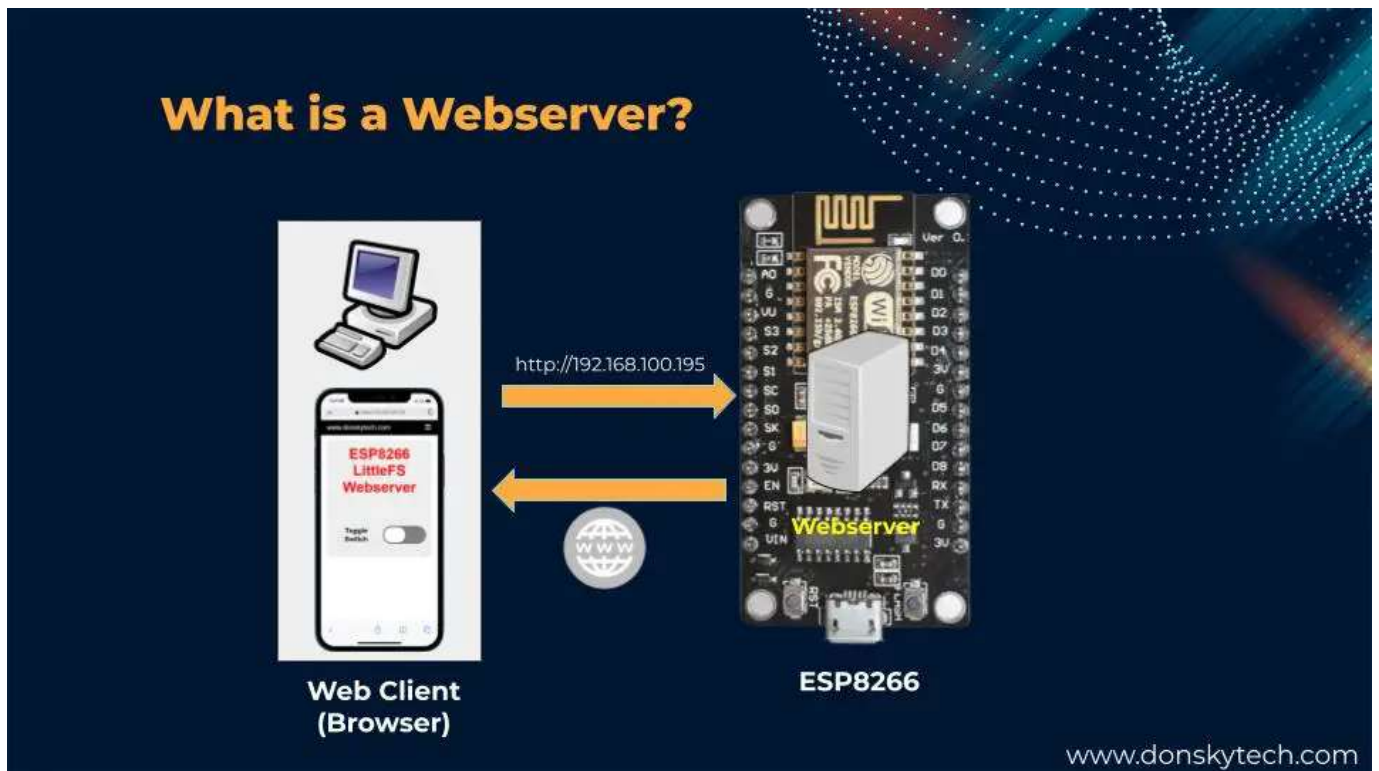
donskeytech
@donskeytech



What is a Webserver?

A Webserver is both hardware and software that responds to client requests using [HTTP](#) (Hypertext Transfer Protocol). In our case below, a Webserver serves a webpage whenever a client like a web browser on your laptop or your mobile phone types in the address of the Webserver.

The NodeMCU ESP8266 is capable of running a Webserver and at the same time controls sensors connected thru it by responding to requests coming from our webpage.



ESP8266 Webserver – No LittleFS/SPIFFS

The following is the code for our ESP8266 Webserver with no Filesystem like LittleFS or SPIFFS involved. This is the only code in our project that powers our project and it includes the Webserver code and the logic to turn on the built-in LED.

The code for my project is available on my GitHub page which you can access from [here](#). Let's go over the code for you to understand.


```

        { request->send(200, "text/css", prepareIndexMinCSS()); });

server.on("/index.js", HTTP_GET, [](AsyncWebServerRequest *request)
    { request->send(200, "text/javascript", prepareIndexJS()); });

// Respond to toggle event
server.on("/toggle", HTTP_GET, [](AsyncWebServerRequest *request)
    {
        String status;
        if (request->hasParam(PARAM_MESSAGE)) {
            status = request->getParam(PARAM_MESSAGE)->value();
            if(status == "ON"){
                toggleLED("ON");
            }else{
                toggleLED("OFF");
            }
        } else {
            status = "No message sent";
        }
        request->send(200, "text/plain", "Turning Built In LED : " + status); });

server.onNotFound(notFound);

server.begin();
}

void loop()
{
}

```

```

#include <Arduino.h>
#ifdef ESP32
#include <WiFi.h>
#include <AsyncTCP.h>
#elif defined(ESP8266)
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#endif
#include <ESPAsyncWebServer.h>

```

Include all the header files necessary to run this project.

```

/*
  Replace the SSID and Password according to your wifi
*/
const char *ssid = "<REPLACE_WITH_YOUR_WIFI_SSID>";
const char *password = "<REPLACE_WITH_YOUR_WIFI_PASSWORD>";

```

Replace this with your own wifi credentials.

```

// Webserver
AsyncWebServer server(80);

String PARAM_MESSAGE = "status";

```

```
request->send(404, "text/plain", "Not found");
}
```

We define our Webserver here and our built in LED pin. We also define a ***notFound()*** function here that gets called when you request resources from our ESP8266 WebServer that is not present.

```
// Prepare the HTML String
String prepareHTML()
{
    return "<!DOCTYPE html>"
    .
    . // More Code
}

// Prepare the index.css
String prepareIndexCSS()
{
    return "/* mincss */"
    .
    . // More Code
}

// Prepare the entireframework.min.css
String prepareIndexMinCSS()
{
    return "/*
    .
    . // More Code
}

// Prepare the index.js
String prepareIndexJS()
{
    .
    . // More Code
}
```

The functions ***prepareHTML()*** , ***prepareIndexCSS()*** , ***prepareIndexMinCSS()*** and the ***prepareIndexJS()*** returns a String representation of our web content. As you can see, we have coded our HTML, CSS (Cascading Stylesheets), and Javascript inside our Arduino Sketch. Yikes!

Isn't this code very easy and lovely to read? You can imagine the nightmare of what will happen if we need to change this code.

```
    digitalWrite(LED_PIN, LOW);  
else  
    digitalWrite(LED_PIN, HIGH);  
}
```

Toggle the LED status by calling this function.

```
void setup()  
{  
  
    Serial.begin(115200);  
  
    // Connect to WIFI  
    WiFi.mode(WIFI_STA);  
    WiFi.begin(ssid, password);  
    if (WiFi.waitForConnectResult() != WL_CONNECTED)  
    {  
        Serial.printf("WiFi Failed!\n");  
        return;  
    }  
  
    Serial.print("IP Address: ");  
    Serial.println(WiFi.localIP());  
  
    // LED PIN  
    pinMode(LED_PIN, OUTPUT);  
    digitalWrite(LED_PIN, HIGH);  
}
```

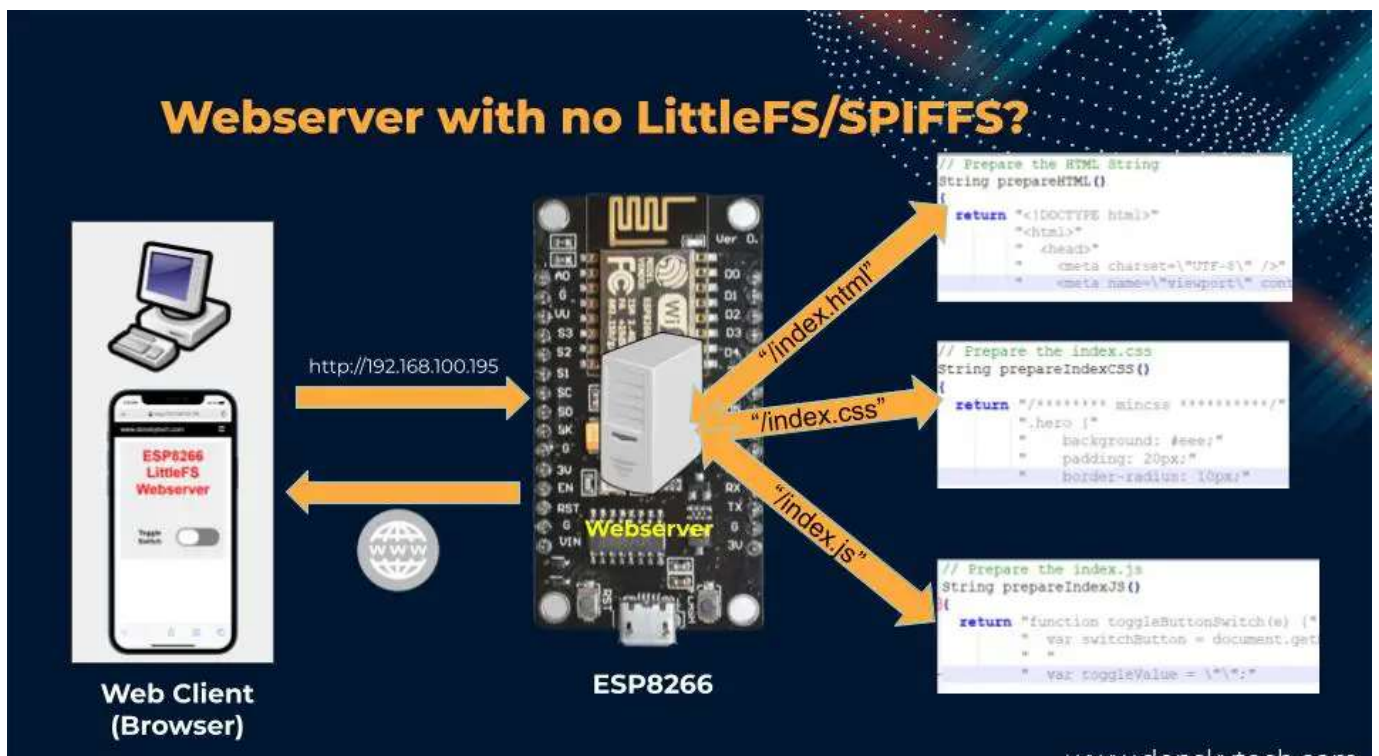
Initialize our Serial monitor baud rate then connect to our Wifi and then configure our LED pin.

```
// Route for root index.html  
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)  
    { request->send(200, "text/html", prepareHTML()); });  
  
// Route for root index.css  
server.on("/index.css", HTTP_GET, [](AsyncWebServerRequest *request)  
    { request->send(200, "text/css", prepareIndexCSS()); });  
  
// Route for root entireframework.min.css  
server.on("/entireframework.min.css", HTTP_GET, [](AsyncWebServerRequest *request)  
    { request->send(200, "text/css", prepareIndexMinCSS()); });  
  
// Route for root index.js  
server.on("/index.js", HTTP_GET, [](AsyncWebServerRequest *request)  
    { request->send(200, "text/javascript", prepareIndexJS()); });  
  
// Respond to toggle event  
server.on("/toggle", HTTP_GET, [](AsyncWebServerRequest *request)  
    {  
        String status;  
        if (request->hasParam(PARAM_MESSAGE)) {  
            status = request->getParam(PARAM_MESSAGE)->value();  
            if(status == "ON"){  
                toggleLED("ON");  
            }else{  
                toggleLED("OFF");  
            }  
        }  
    })
```

For every route or request from our browser, we respond here by using the code above. So for example,

- We then start the Webserver by calling the `server.begin()`;

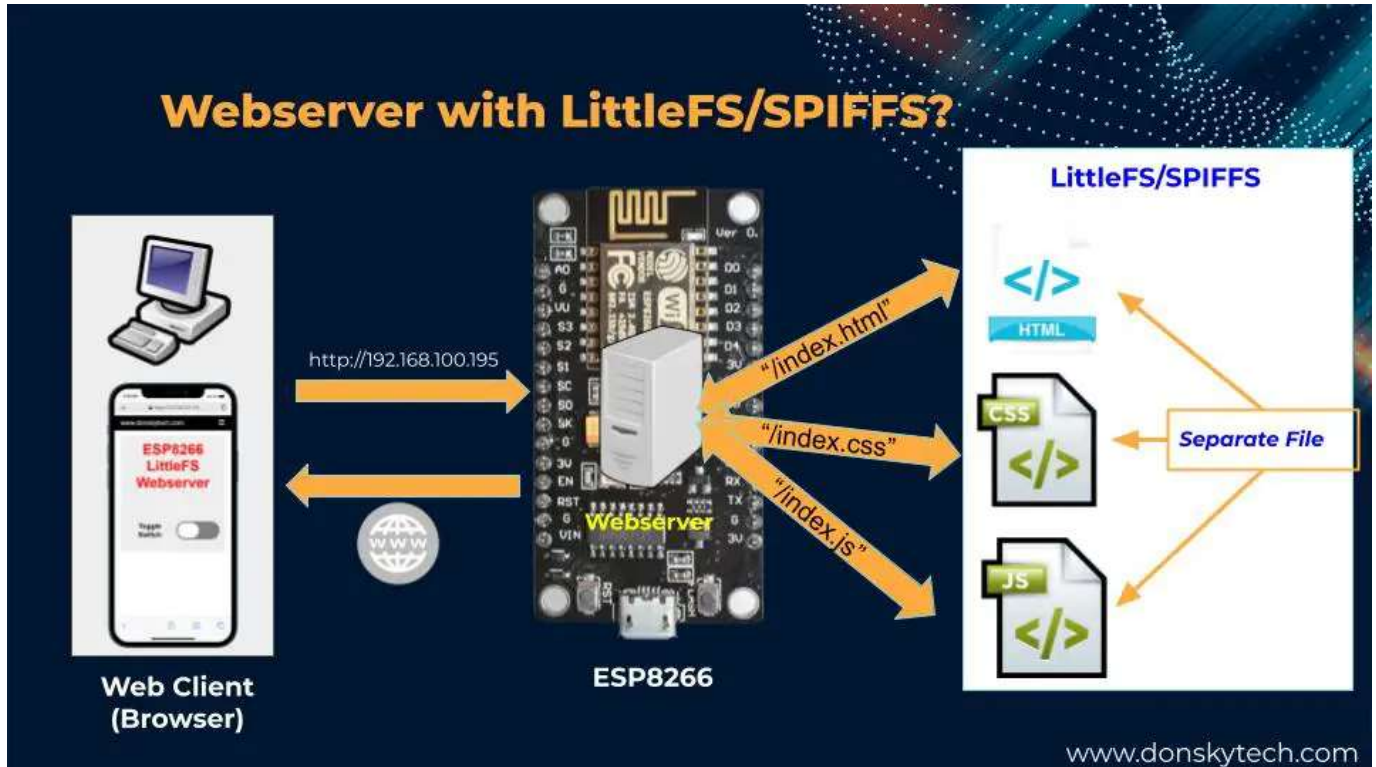
Can you now see the problem? We are coding our HTML/CSS/Javascript inside our Arduino sketches then it would be hard to maintain later. Please see the following image for a better representation of what is happening.



ESP8266 Webserver – Using LittleFS/SPIFFS

To work around the problem above then we can leverage the power of LittleFS or SPIFFS. Let us look at the diagram below for you to see better.

We have separately coded our HTML/CSS/Javascript in separate files. After which, we upload those files into our LittleFS or SPIFFS filesystem. We are using LittleFS in this post.



The code for this is available on my [Github](#) page. We have created separate files for our web content so that it would be easier to do the changes.

- index.html
- index.css

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset="UTF-8" />
5    <meta name="viewport" content="width=device-width, initial-scale=1" />
6    <title>ESP8266 LittleFS Webserver</title>
7    <link rel="stylesheet" href="index.css" />
8    <link rel="stylesheet" href="entireframework.min.css" />
9    <script src="index.js"></script>
10 </head>
11 <body>
12   <nav class="nav" tabindex="-1" onclick="this.focus()">
13     <div class="container">
14       <a class="pagename current" href="#">www.donskeytech.com</a>
15       <a href="#">One</a>
16       <a href="#">Two</a>
17       <a href="#">Three</a>
18     </div>
19   </nav>
20   <button class="btn-close btn btn-sm"></button>
21   <div class="container">
22     <div class="hero">
23       <h1 class="title">ESP8266 LittleFS Webserver</h1>
24       <div class="toggle-div">
25         <p class="label">Toggle Switch</p>
26         <input type="checkbox" id="switch" onclick="toggleButtonSwitch()" /> <label for="switch">Toggle</label>
27       </div>
28     </div>
29   </div>
30 </body>
31 </html>

```

The **data** folder will then need to be uploaded to the NodeMCU ESP8266 Filesystem.

What changed in the code?

main.cpp - using LittleFS

```

/*
  Title:  ESP8266 Webserver - No Filesystem
  Description:  An ESP8266 webserver that uses LittleFS to load web content
  Author:  donskey
  For:    www.donskeytech.com
  Date:   September 14, 2022
*/

#include <Arduino.h>
#include <ESP32>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <LittleFS.h>

/*
  Replace the SSID and Password according to your wifi
*/
const char *ssid = "<REPLACE_WITH_YOUR_WIFI_SSID>";
const char *password = "<REPLACE_WITH_YOUR_WIFI_PASSWORD>";

// Webserver
AsyncWebServer server(80);

```

```
void notFound(AsyncWebServerRequest *request)
{
    request->send(404, "text/plain", "Not found");
}

void toggleLED(String status)
{
    if (status == "ON")
        digitalWrite(LED_PIN, LOW);
    else
        digitalWrite(LED_PIN, HIGH);
}

void setup()
{
    Serial.begin(115200);
    Serial.println("Starting the LittleFS Webserver..");

    // Begin LittleFS
    if (!LittleFS.begin())
    {
        Serial.println("An Error has occurred while mounting LittleFS");
        return;
    }

    // Connect to WIFI
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    if (WiFi.waitForConnectResult() != WL_CONNECTED)
    {
        Serial.printf("WiFi Failed!\n");
        return;
    }

    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());

    // LED PIN
    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, HIGH);

    // Route for root index.html
    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
        { request->send(LittleFS, "/index.html", "text/html"); });

    // Route for root index.css
    server.on("/index.css", HTTP_GET, [](AsyncWebServerRequest *request)
        { request->send(LittleFS, "/index.css", "text/css"); });

    // Route for root entireframework.min.css
    server.on("/entireframework.min.css", HTTP_GET, [](AsyncWebServerRequest *request)
        { request->send(LittleFS, "/entireframework.min.css", "text/css"); });

    // Route for root index.js
    server.on("/index.js", HTTP_GET, [](AsyncWebServerRequest *request)
        { request->send(LittleFS, "/index.js", "text/javascript"); });

    // Respond to toggle event
```

```
    if (request->hasParam(PARAM_MESSAGE)) {
        status = request->getParam(PARAM_MESSAGE)->value();
        if(status == "ON"){
            toggleLED("ON");
        }else{
            toggleLED("OFF");
        }
    } else {
        status = "No message sent";
    }
    request->send(200, "text/plain", "Turning Built In LED : " + status); });

server.onNotFound(notFound);

server.begin();
}

void loop()
{
}
```

It's almost similar to the earlier code except that we are not coding the HTML/CSS/Javascript files inside our *main.cpp*.

```
#include <LittleFS.h>
```

Import the header file for LittleFS.

```
// Begin LittleFS
if (!LittleFS.begin())
{
    Serial.println("An Error has occurred while mounting LittleFS");
    return;
}
```


Start the LittleFS filesystem.

```
// Route for root index.html
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
    { request->send(LittleFS, "/index.html", "text/html"); });

// Route for root index.css
server.on("/index.css", HTTP_GET, [](AsyncWebServerRequest *request)
    { request->send(LittleFS, "/index.css", "text/css"); });

// Route for root entireframework.min.css
server.on("/entireframework.min.css", HTTP_GET, [](AsyncWebServerRequest *request)
    { request->send(LittleFS, "/entireframework.min.css", "text/css"); });

// Route for root index.js
server.on("/index.js", HTTP_GET, [](AsyncWebServerRequest *request)
    { request->send(LittleFS, "/index.js", "text/javascript"); });
```

We have removed all those functions that return string (e.g. `prepareHTML()`). We are loading now the web contents (HTML/CSS/Javascript) from the LittleFS filesystem. Isn't this much better rather than coding those long strings?

Once you understand the code then start uploading it to the NodeMCU ESP8266 Filesystem.

Changes to platform.ini

We need to set the **platform.ini** filesystem to use LittleFS.

Wrap up

I have discussed how to create your own NodeMCU ESP8266 Webserver using LittleFS filesystem in this post. I have shown how using LittleFS in serving your web content such as HTML/CSS/Javascript would make your code readable and easier to maintain.

In the last part of this series, we will discuss how LittleFS could help you in saving configurations or settings in your IOT projects.

Stay Tuned! Happy Exploring!

If you like my post then please consider sharing this. Thanks!

PlatformIO Tutorial for Arduino Development – donskeytech.com

[September 28, 2022](#)

[...] succeeding steps that we are going to do is based on my ESP8266 Webserver Using LittleFS. It is perfectly okay if you don't understand how it works because the goal of this post is [...]

[Reply](#)

Gautam

[November 7, 2022](#)

This 192.168.43.238 page can't be foundNo webpage was found for the web address: <http://192.168.43.238/>
HTTP ERROR 404

I tried ESP8266 Webserver – Using LittleFS. But I am getting above error. Kindly help.

[Reply](#)

donskey

[November 8, 2022](#)

Did you execute the Upload to Filesystem command?

[Reply](#)

Using Arduino with BME280 plus a weather station project

[December 22, 2022](#)

[...] Related Content: ESP8266 Webserver Using LittleFS [...]

Your email address will not be published. Required fields are marked *

Comment *

Name *

Email *

Website

☐ Save my name, email, and website in this browser for the next time I comment.

Post Comment



report this ad

Search

Search

Search

Categories

- [Arduino](#)
- [ESP32](#)
- [ESP8266](#)
- [Features](#)
- [Internet of Things](#)
- [MicroPython](#)
- [MQTT](#)
- [Node-Red](#)
- [Programming](#)
- [Projects](#)
- [Raspberry Pi](#)

Archives

- [June 2023](#)
- [May 2023](#)
- [April 2023](#)
- [March 2023](#)
- [February 2023](#)
- [January 2023](#)
- [December 2022](#)
- [November 2022](#)
- [October 2022](#)
- [September 2022](#)
- [August 2022](#)
- [July 2022](#)
- [April 2022](#)
- [March 2022](#)
- [August 2021](#)
- [July 2021](#)
- [May 2021](#)
- [January 2021](#)
- [December 2020](#)
- [November 2020](#)
- [October 2020](#)



report this ad

Tags

arduino arduinojson database database-application dht11 dht22 esp-01 **esp32** **esp8266** esp8266-core
express **flask** flask-socketio installation **iot** javascript keypad ldr microdot **micropython** mongodb
mosquitto **mqtt** mqtt.js mqttx node-red node.js nodemcu pico-w platformio **project** **python**
raspberrypi raspberrypi-pico-w raspberry-pi-sensor **raspberrypi** raspberry pi rfid thonny weatherstation web-server webserver
websocket websockets wifi-car windows



report this ad

[YouTube](#)[Facebook](#)[Twitter](#)

About www.donskeytech.com

I am fond of microcontroller programming and the Internet of Things (IOT). I am a hobbyist and a programmer from Manila, Philippines. Connect with me thru my different social media channels!

Happy Exploring!

Categories

- Arduino
- ESP32
- ESP8266
- Features
- Internet of Things
- MicroPython
- MQTT
- Node-Red
- Programming
- Projects
- Raspberry Pi

Archives

- June 2023
 - May 2023
 - April 2023
 - March 2023
 - February 2023
 - January 2023
 - December 2022
 - November 2022
 - October 2022
 - September 2022
 - August 2022
 - July 2022
 - April 2022
 - March 2022
 - August 2021
-

- January 2021
- December 2020
- November 2020
- October 2020

© 2022 donskeytech.com



[report this ad](#)