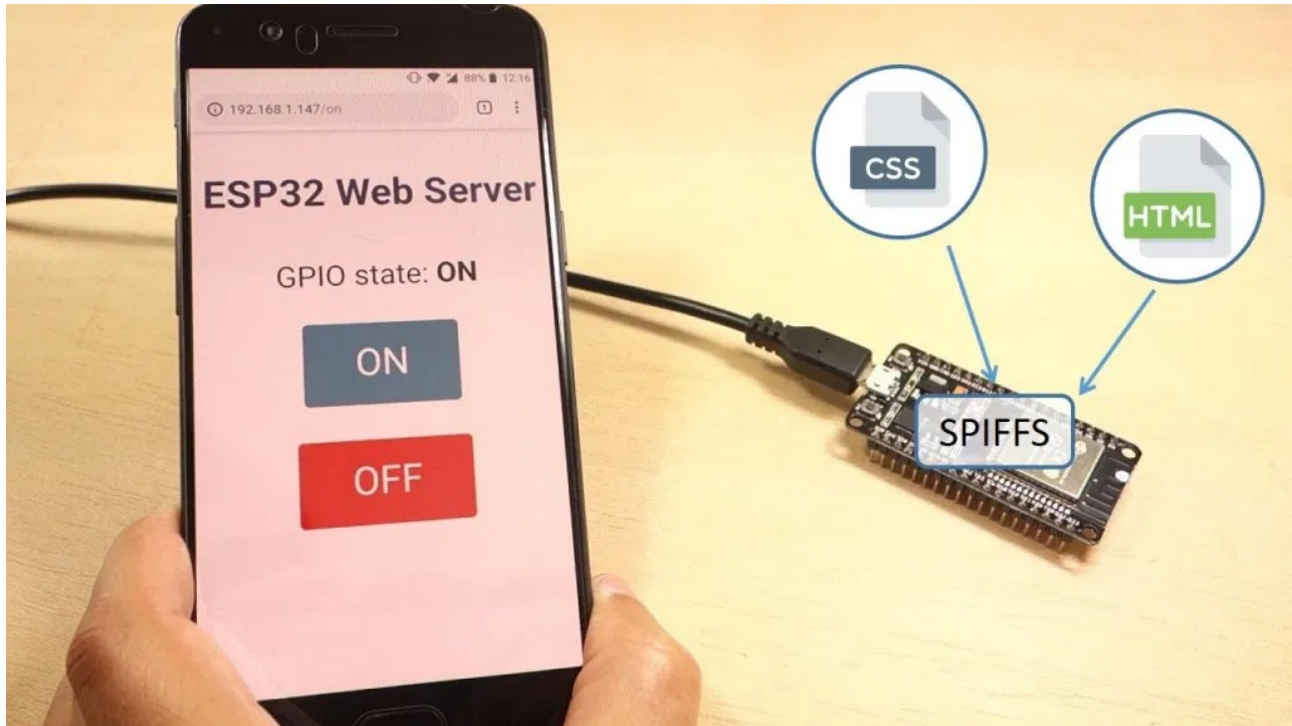


ESP32 Web Server mithilfe von SPIFFS (SPI Flash File System)

In diesem tutorial zeigen wir Ihnen, wie man einen web-server, der HTML-und CSS-Dateien gespeichert, auf die ESP32-Dateisystem. Anstatt zu schreiben, die HTML-und CSS-text in die Arduino-Skizze erstellen wir getrennt HTML-und CSS-Dateien.



Für demonstration Zwecke, die web-server wir bauen werde Kontrollen ist ein ESP32-Ausgang, aber es kann leicht angepasst werden für andere Zwecke, wie das anzeigen-sensor-Lesungen.

Zum Lesen empfohlen: [ESP8266 Web Server verwenden SPIFFS](#)

ESP32-Dateisystem Uploader Plugin

Um diesem tutorial zu Folgen, sollten Sie die ESP32-Dateisystem-Uploader-plugin installiert ist in Ihrem Arduino-IDE. Wenn Sie dies nicht getan haben, Folgen Sie dem nächsten tutorial zu installieren, es zuerst:

- [Installieren ESP32-Dateisystem Uploader auf Arduino IDE](#)

Hinweis: stellen Sie sicher, dass Sie die neueste Arduino IDE installiert, sowie die ESP32-add-on für die Arduino IDE. Wenn Sie nicht, Folgen Sie einem der nächsten tutorials um es zu installieren:

- [Windows Anleitungen – Installation der ESP32 Board in Arduino IDE](#)
- [Mac-und Linux - Anweisungen – Installation der ESP32 Board in Arduino IDE](#)

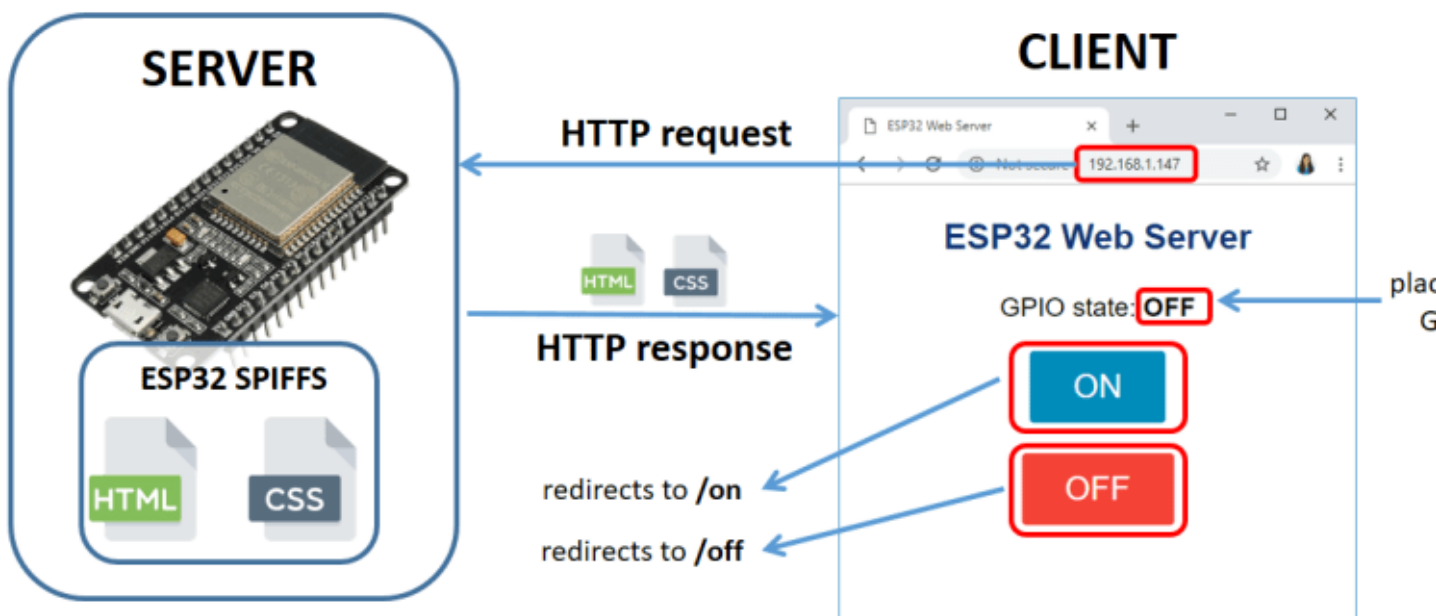
Projekt-Übersicht

Bevor Sie direkt zu dem Projekt, es ist wichtig, um zu skizzieren, was unsere web-server, so dass es einfacher ist, zu verstehen.



- Der web-server Sie bauen steuert eine LED angeschlossen, um den ESP32 GPIO-2. Dies ist der ESP32 auf-board LED. Sie können Steuern, alle anderen GPIO;
- Der web-server Seite zeigt zwei Tasten: AUF und OFF – schalten GPIO 2 auf und off;
- Der web-server Seite zeigt die aktuellen GPIO-Zustand.

Die folgende Abbildung zeigt ein Vereinfachtes Diagramm, um zu demonstrieren, wie alles funktioniert.



- Der ESP32 läuft ein web-server-code auf der Basis von [ESPAsyncWebServer Bibliothek](#);
- Die HTML-und CSS-Dateien gespeichert sind, auf dem ESP32 SPIFFS (Serial Peripheral Interface Flash filesystem);
- Wenn Sie machen eine Anfrage an eine bestimmte URL mit Ihrem browser, die ESP32 antwortet mit den angeforderten Dateien;
- Wenn Sie klicken Sie AUF die Schaltfläche, die Sie umgeleitet werden, um die Stamm-URL gefolgt von /an und die LED ist eingeschaltet;
- Wenn Sie auf die OFF-Taste und Sie werden umgeleitet auf die root-URL gefolgt von /aus und die LED ausgeschaltet;
- Auf der web-Seite ist ein Platzhalter für die GPIO-Zustand. Die Platzhalter für die GPIO-Zustand geschrieben wird direkt in der HTML-Datei zwischen % - Zeichen, zum Beispiel %STATE% .

Installation Der Bibliotheken

In den meisten unserer Projekte, die wir erstellt haben, den HTML-und CSS-Dateien des web-Servers als String direkt auf der Arduino-Skizze. Mit SPIFFS, können Sie die HTML-und CSS in getrennten Dateien und speichern Sie Sie auf dem ESP32-Dateisystem.

Eine der einfachsten Möglichkeiten, um ein web-server mit Dateien aus dem Dateisystem ist die Verwendung des ESPAsyncWebServer Bibliothek. Die ESPAsyncWebServer Bibliothek ist gut dokumentiert auf seiner GitHub-Seite. Für mehr Informationen über die Bibliothek, überprüfen Sie die folgenden link:

- <https://github.com/me-no-dev/ESPAsyncWebServer>

Die Installation des ESPAsyncWebServer Bibliothek

Befolgen Sie die nächsten Schritte zum installieren des [ESPAsyncWebServer](#) Bibliothek:

1. [Klicken Sie hier, um den download](#) des ESPAsyncWebServer Bibliothek. Sie sollten eine .zip-Ordner in Ihrem Downloads-Ordner
2. Entpacken Sie die .zip-Ordner und Sie sollten ESPAsyncWebServer-master-Ordner
3. Benennen Sie Ihren Ordner aus ESPAsyncWebServer-master zu ESPAsyncWebServer
4. Bewegen Sie den ESPAsyncWebServer Ordner, um Ihre Arduino IDE-installation Ordner Bibliotheken

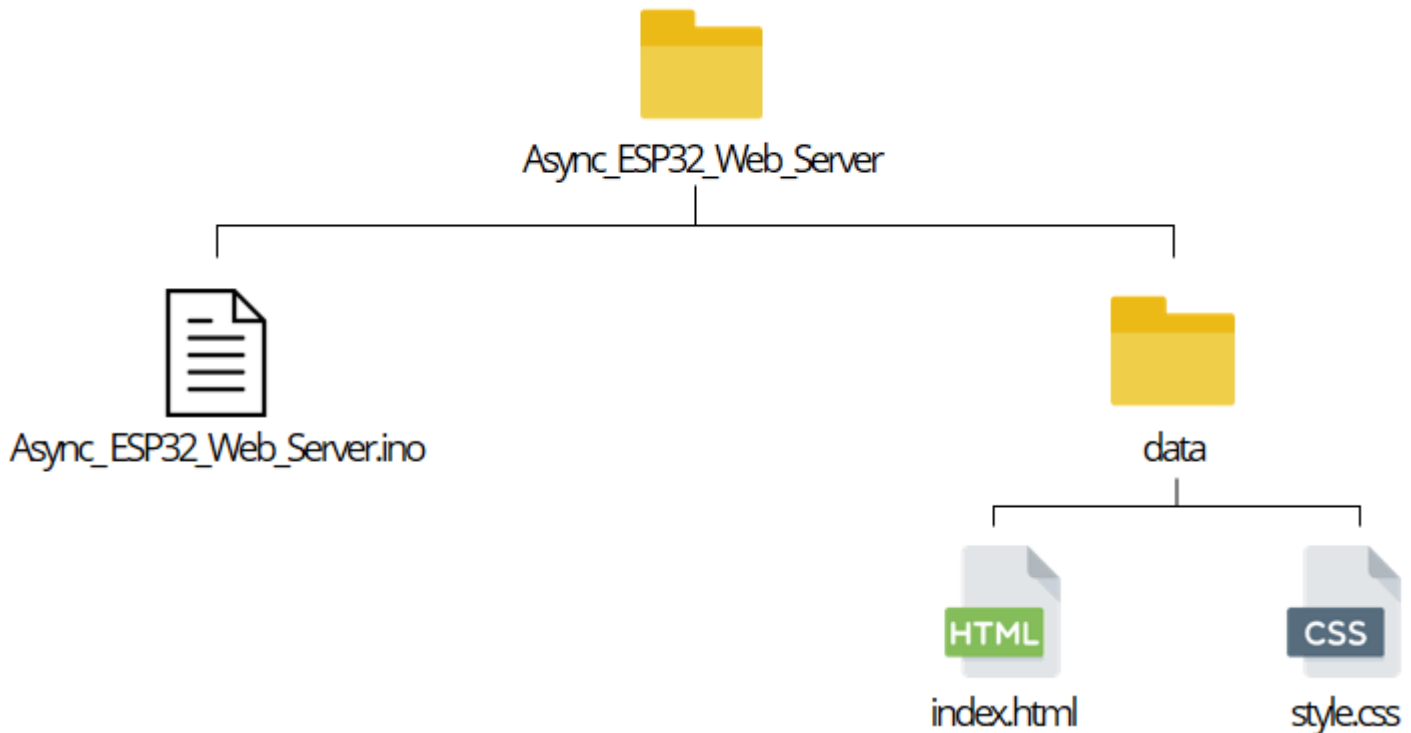
Installing the Async TCP Library for ESP32

Die [ESPAsyncWebServer](#) Bibliothek erfordert die [AsyncTCP](#) Bibliothek zu arbeiten. Befolgen Sie die nächsten Schritte zum installieren der Bibliothek:

1. [Klicken Sie hier, um den download](#) der AsyncTCP Bibliothek. Sie sollten eine .zip-Ordner in Ihrem Downloads-Ordner
2. Entpacken Sie die .zip-Ordner und Sie sollten AsyncTCP-master-Ordner
3. Benennen Sie Ihren Ordner aus AsyncTCP-master zu AsyncTCP
4. Bewegen Sie den AsyncTCPfolder zu Ihrer Arduino IDE-installation Ordner Bibliotheken
5. Schließlich, re-öffnen Sie Ihre Arduino IDE

Die Organisation Ihrer Dateien

Zum erstellen der web-server benötigen Sie drei unterschiedliche Dateien. Die Arduino-Skizze, die HTML-Datei und die CSS-Datei. Die HTML-und CSS-Dateien gespeichert werden sollen, in einem Ordner namens **Daten** innerhalb der Arduino sketch Ordner, wie unten gezeigt:



Erstellen der HTML-Datei

Der HTML-Code für dieses Projekt ist sehr einfach. Wir brauchen nur zu erstellen Sie eine Überschrift für die web-Seite, einen Absatz, um die Anzeige der GPIO-Zustand und zwei Tasten.

Erstellen Sie eine *index.html* Datei mit dem folgenden Inhalt, oder [laden Sie alle Dateien des Projekts hier](#):

```
<!DOCTYPE html>
<html>
<head>
  <title>ESP32 Web Server</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
  <h1>ESP32 Web Server</h1>
  <p>GPIO state: <strong> %STATE%</strong></p>
  <p><a href="/on"><button class="button">ON</button></a></p>
  <p><a href="/off"><button class="button button2">OFF</button></a></p>
</body>
</html>
```

Denn wir arbeiten mit CSS und HTML in verschiedenen Dateien, müssen wir die Referenz der CSS-Datei auf den HTML-text. Sollte die folgende Zeile Hinzugefügt werden zwischen die `<head>` `</head>` - tags:

```
<link rel="stylesheet" type="text/css" href="style.css">
```

Das `<link>` - tag wird die HTML-Datei, die Sie mithilfe eines externen Stylesheets zu formatieren, wie die Seite aussieht. Das **rel** - Attribut gibt die Art des externen Datei, in diesem Fall, dass es ist eine **stylesheet** —CSS-Datei—wird verwendet, um das Aussehen zu verändern Sie die Seite.

Das **type** - Attribut auf festgelegt ist "**text/css**" , um anzuzeigen, dass Sie über eine CSS-Datei für die Formate. Das **href** - Attribut gibt die Datei Lage, da sowohl die CSS-und HTML-Dateien werden im selben Ordner befinden, müssen Sie nur die Referenz mit dem Namen: **Stil.css** .

In der folgenden Zeile schreiben wir das erste Kapitel unserer web-Seite. In diesem Fall haben wir das "ESP32 Web Server". Sie können die änderung der überschrift zu jedem text, den Sie möchten:

```
<h1>ESP32 Web Server</h1>
```

Dann fügen wir einen Absatz mit dem text "GPIO-Status: ", gefolgt von den GPIO-Status. Da die GPIO-Status ändert sich entsprechend dem Zustand des GPIO, wir können fügen Sie einen Platzhalter, der dann ersetzt werden, die für jeden Wert legen wir auf die Arduino-Skizze.

Hinzufügen von Platzhalter verwenden wir % - Zeichen. Erstellen Sie einen Platzhalter für den Zustand, die wir verwenden können, %ZUSTAND% , zum Beispiel.

```
<p>GPIO state: <strong>%STATE%</strong></p>
```

Zuordnung eines Werts zu den STAATLICHEN Platzhalter erfolgt in der Arduino-Skizze.

Dann erstellen wir ein AUF und ein AB-Tasten. Wenn Sie auf die auf-Taste, leiten wir die web-Seite, um zu root, gefolgt von /auf - url. Wenn Sie auf die off-Taste, Sie werden umgeleitet, um die / aus der url.

```
<p><a href="/on"><button class="button">ON</button></a></p>
<p><a href="/off"><button class="button button2">OFF</button></a></p>
```

Das erstellen der CSS-Datei

Erstellen Sie den *Stil.css-Datei* mit folgendem Inhalt oder [downloaden Sie alle Dateien des Projekts hier](#):

```
html {
  font-family: Helvetica;
  display: inline-block;
  margin: 0px auto;
  text-align: center;
}
h1{
  color: #0F3376;
  padding: 2vh;
}
p{
  font-size: 1.5rem;
}
.button {
  display: inline-block;
  background-color: #008CBA;
  border: none;
  border-radius: 4px;
  color: white;
```

```
padding: 16px 40px;
text-decoration: none;
font-size: 30px;
margin: 2px;
cursor: pointer;
}
.button2 {
background-color: #f44336;
}
```

Dies ist nur eine grundlegende CSS-Datei festlegen, die Schriftgröße, den Stil und die Farbe der Knöpfe und richten Sie die Seite. Wir werden nicht erklären, wie CSS funktioniert. Ein guter Ort, um zu erfahren, über CSS ist der [W3Schools-website](https://www.w3schools.com/).

Arduino Sketch

Kopieren Sie den folgenden code in die Arduino IDE oder [downloaden Sie alle Dateien des Projekts hier](#). Dann, geben Sie Ihre Netzwerk-Anmeldeinformationen (SSID und Passwort) zu machen es Arbeit.

```
/******
  Rui Santos
  Complete project details at https://randomnerdtutorials.com
  *****/

// Import required libraries
#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include "SPIFFS.h"

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

// Set LED GPIO
const int ledPin = 2;
// Stores LED state
String ledState;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

// Replaces placeholder with LED state value
String processor(const String& var){
  Serial.println(var);
  if(var == "STATE"){
    if(digitalRead(ledPin)){
      ledState = "ON";
    }
    else{
      ledState = "OFF";
    }
    Serial.print(ledState);
    return ledState;
  }
  return String();
}

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);
```

```

pinMode(ledPin, OUTPUT);

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

// Connect to Wi-Fi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
}

// Print ESP32 Local IP Address
Serial.println(WiFi.localIP());

// Route for root / web page
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Route to load style.css file
server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(SPIFFS, "/style.css", "text/css");
});

// Route to set GPIO to HIGH
server.on("/on", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin, HIGH);
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Route to set GPIO to LOW
server.on("/off", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin, LOW);
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Start server
server.begin();
}

void loop(){
}

```

Wie der Code funktioniert

Zuerst, die erforderlichen Bibliotheken:

```

#include "WiFi.h"
#include "ESPAsyncWebServer.h"
#include "SPIFFS.h"

```

Sie müssen Ihre Netzwerk-Anmeldeinformationen in den folgenden Variablen:

```

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

```

Als Nächstes erstellen Sie eine variable bezieht sich auf die GPIO-2 genannt ledPin , und eine String-variable für die led-Status: ledState .

```
const int ledPin = 2;  
String ledState;
```

Erstellen Sie eine AsyncWebServer Objekt aufgerufen server überwacht port 80.

```
AsyncWebServer server(80);
```

Prozessor()

Der Prozessor() - Funktion wird das Attribut einen Wert für den Platzhalter die wir erstellt haben, auf der HTML-Datei. Es nimmt als argument den Platzhalter und sollte einen String zurückgeben, ersetzen die Platzhalter. Der Prozessor() - Funktion sollte folgenden Aufbau haben:

```
String processor(const String& var){  
  Serial.println(var);  
  if(var == "STATE"){  
    if(digitalRead(ledPin)){  
      ledState = "ON";  
    }  
    else{  
      ledState = "OFF";  
    }  
    Serial.print(ledState);  
    return ledState;  
  }  
  return String();  
}
```

Diese Funktion testet zuerst, ob der Platzhalter ist der ZUSTAND, die wir erstellt haben, auf der HTML-Datei.

```
if(var == "STATE"){
```

Wenn es ist, dann, entsprechend der LED-Status ist, setzen wir die ledState variable entweder ON oder OFF.

```
  if(digitalRead(ledPin)){  
    ledState = "ON";  
  }  
  else{  
    ledState = "OFF";  
  }  
}
```

Schließlich kehren wir die ledState variable. Dieser ersetzt den Platzhalter mit dem ledState string-Wert.

```
return ledState;
```

setup()

In der setup () - starten Sie durch Initialisierung der Seriellen Monitor und Einstellung der GPIO als Ausgang.


```
Serial.begin(115200);  
pinMode(ledPin, OUTPUT);
```

Initialisieren SPIFFS:

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

Wi-Fi-Verbindung

Verbindung zu Wi-Fi und das drucken des ESP32 IP-Adresse:

```
WiFi.begin(ssid, password);  
while (WiFi.status() != WL_CONNECTED) {  
    delay(1000);  
    Serial.println("Connecting to WiFi..");  
}  
Serial.println(WiFi.localIP());
```

Async-Web-Server

Die ESPAsyncWebServer-Bibliothek erlaubt uns die Konfiguration der Routen, auf denen der server empfangsbereit für eingehende HTTP-Anforderungen und Funktionen ausführen, wenn eine Anfrage empfangen wird auf diese route. Für, dass, benutzen Sie die auf() - Methode auf dem server-Objekt wie folgt:

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){  
    request->send(SPIFFS, "/index.html", String(), false, processor);  
});
```

Wenn der server eine Anforderung empfängt, auf der Wurzel "/" URL, es wird senden die *index.html* Datei in den client. Das Letzte argument der send() Funktion ist die Prozessor, so dass wir sind in der Lage zu ersetzen Sie die Platzhalter für den Wert, den wir wollen – in diesem Fall die ledState .

Da haben wir auf die CSS-Datei auf die HTML-Datei, der client eine Anfrage, für die CSS-Datei. Wenn das geschieht, wird die CSS-Datei an den client gesendet wird:

```
server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request){  
    request->send(SPIFFS, "/style.css", "text/css");  
});
```

Schließlich müssen Sie definieren, was passiert, auf der /on und - /off - Routen. Wenn eine Anforderung auf diesen Routen, die LED ist entweder an-oder ausgeschaltet, und der ESP32 dient die HTML-Datei.

```
server.on("/on", HTTP_GET, [](AsyncWebServerRequest *request){  
    digitalWrite(ledPin, HIGH);  
    request->send(SPIFFS, "/index.html", String(), false, processor);  
});  
server.on("/off", HTTP_GET, [](AsyncWebServerRequest *request){  
    digitalWrite(ledPin, LOW);  
    request->send(SPIFFS, "/index.html", String(), false, processor);  
});
```

In das Ende, wir nutzen die `begin()` - Methode auf dem `server` - Objekt, so dass der server gestartet wird, empfangsbereit für eingehende clients.

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

Denn dies ist eines asynchronen web server, Sie können definieren die Anforderungen, die in der `setup()` . Sie können dann weiteren code hinzufügen, um die `loop()` , während der server empfangsbereit für eingehende clients.

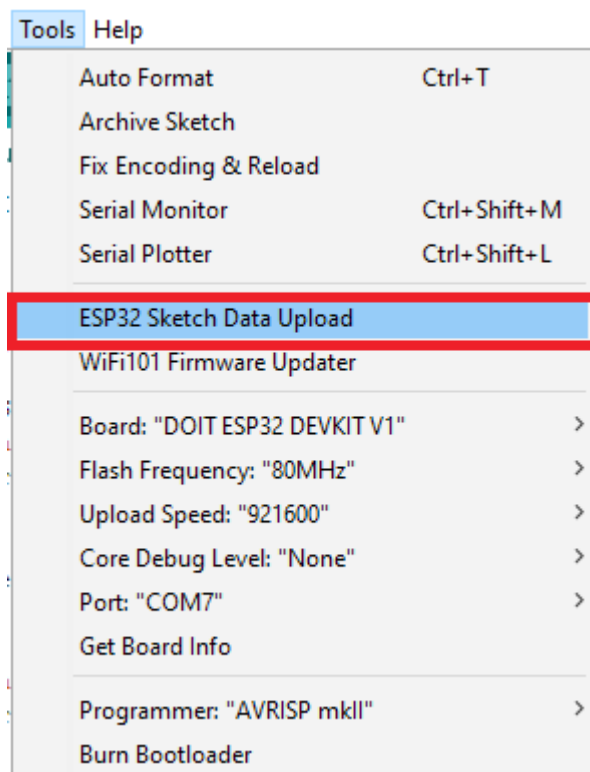
Hochladen von Code und Dateien

Speichern Sie den code als *Async_ESP32_Web_Server* oder [downloaden Sie alle Dateien des Projekts hier](#) . Gehen Sie auf **Sketch** > **Show Sketch-Ordner** , und erstellen Sie einen Ordner namens **Daten** . In diesem Ordner speichern, sollten Sie die HTML-und CSS-Dateien.

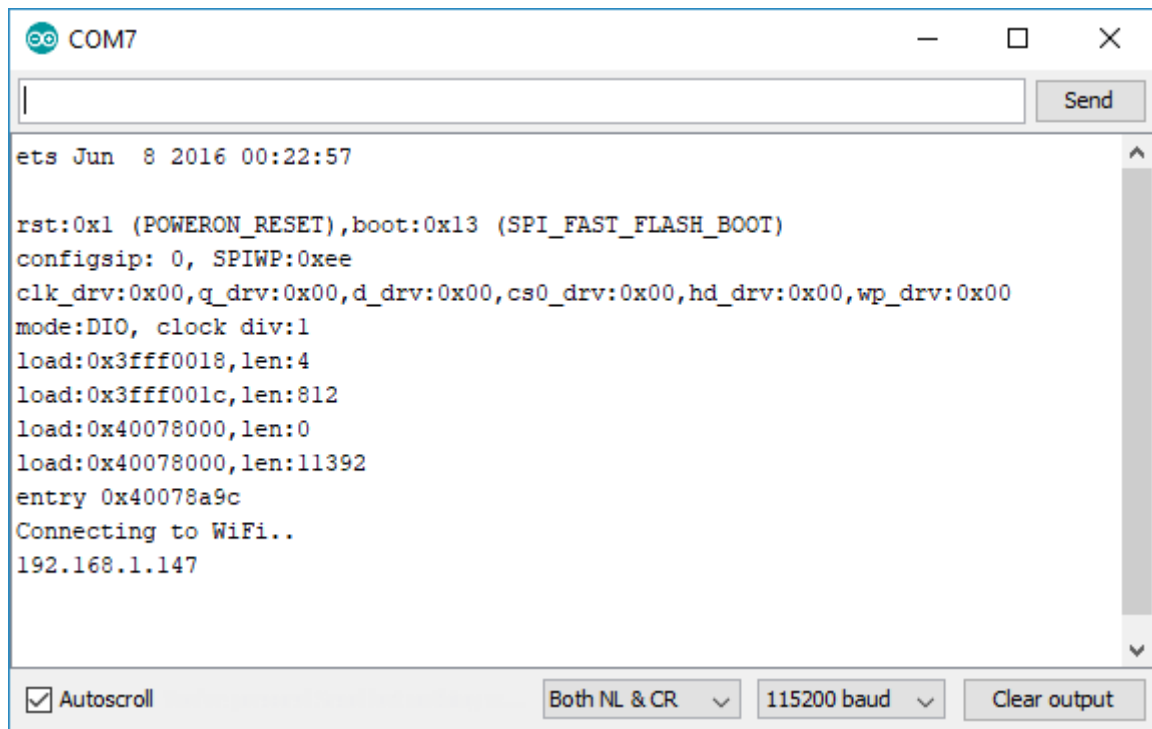
Dann, laden Sie die code zu Ihrem ESP32 board. Stellen Sie sicher, dass Sie das Recht haben, board und COM-port ausgewählt ist. Auch, machen sicher, Sie haben Hinzugefügt Ihre Netzwerke Anmeldeinformationen im code.



Nach dem hochladen der code, den Sie benötigen, um die Dateien hochzuladen. Gehen Sie zu **Tools** - > **ESP32 Daten Sketch Hochladen** und warten Sie, bis die Dateien hochgeladen werden.



Wenn alles hochgeladen wurde, öffnen Sie den Seriellen Monitor mit einer Baudrate von 115200. Drücken Sie die ESP32 " **AKTIVIEREN** " - button und es sollte drucken des ESP32 IP-Adresse.



```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:812
load:0x40078000,len:0
load:0x40078000,len:11392
entry 0x40078a9c
Connecting to WiFi..
192.168.1.147
```

COM7

Send

☒ Autoscroll

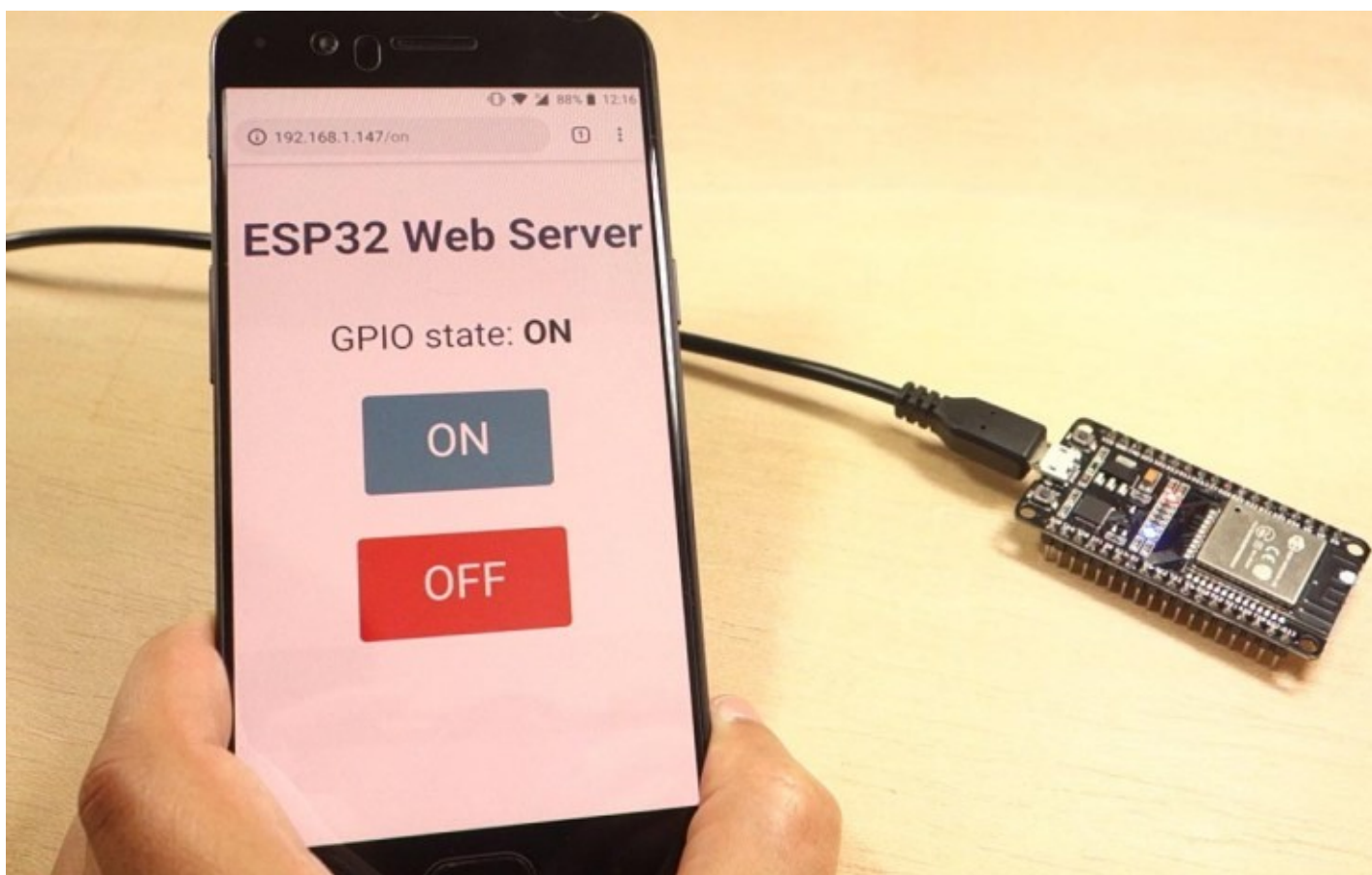
Both NL & CR

115200 baud

Clear output

Demonstration

Öffnen Sie Ihren browser und geben Sie die ESP32 IP-Adresse. Drücken Sie die AUF-und AB-Tasten zur Steuerung der ESP32 auf-board LED. Überprüfen Sie auch, dass die GPIO-Status wird korrekt aktualisiert.



Zusammenfassung

Unter SPI Flash File System (SPIFFS) ist besonders nützlich zum speichern von HTML-und CSS-Dateien dienen dazu, einen client – anstatt zu schreiben den code in der Arduino-Skizze.

Die ESPAsyncWebServer-Bibliothek ermöglicht das erstellen eines web-server durch ausführen einer bestimmten Funktion in der Antwort auf eine bestimmte Anfrage. Sie können auch fügen Sie Platzhalter, um die HTML-Datei, dass können ersetzt werden mit Variablen – wie Sensorwerte, oder GPIO-Staaten, zum Beispiel.

Wenn du gerne dieses Projekt, Sie können auch mögen:

- [ESP32 Web Server mit BME280 – Mini-Wetter-Station](#)
- [ESP32 Web Server – Arduino IDE](#)
- [Getting Started with MicroPython auf dem ESP32 und ESP8266](#)

*Dies ist ein Auszug aus unserem Kurs: **Lernen ESP32 mit der Arduino IDE**. Wenn Sie wie ESP32 und Sie mehr erfahren möchten, empfehlen wir die Einschreibung in [Lernen ESP32 mit der Arduino IDE natürlich](#).*