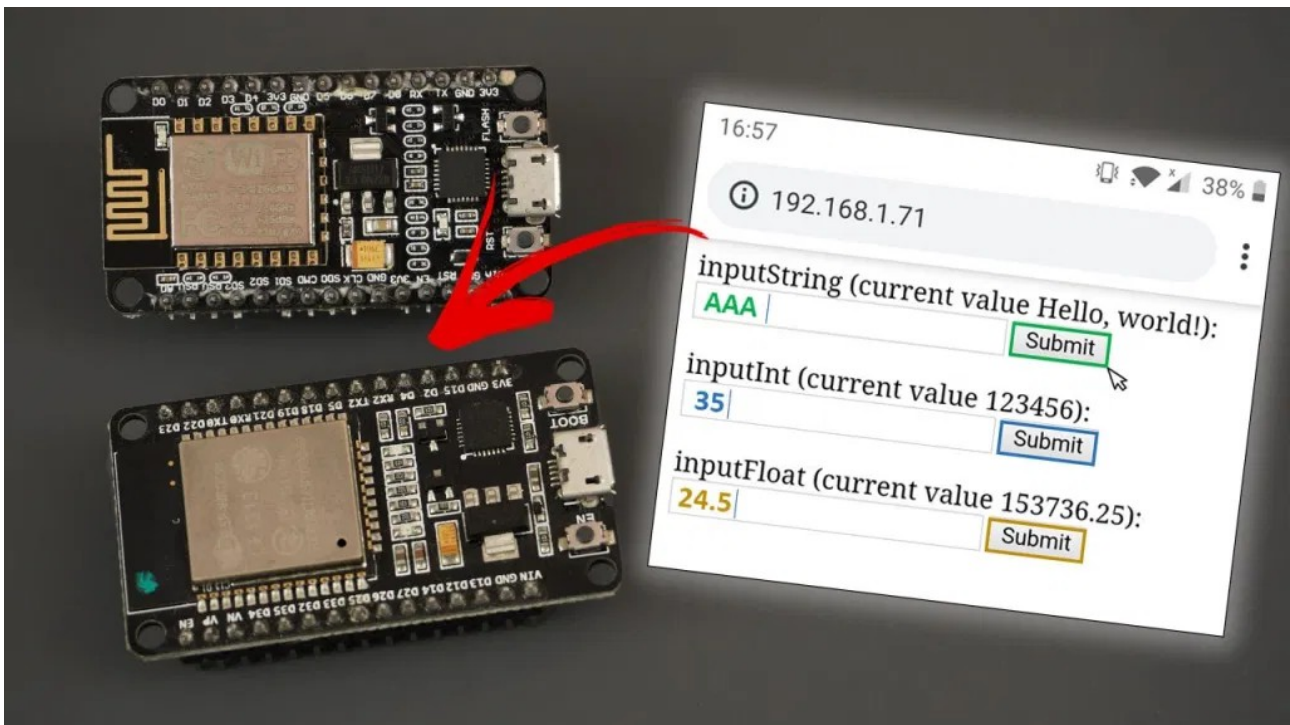


Input-Daten auf HTML-Formular ESP32/ESP8266 Web Server using Arduino IDE

<https://randomnerdtutorials.com/esp32-esp8266-input-data-html-form>

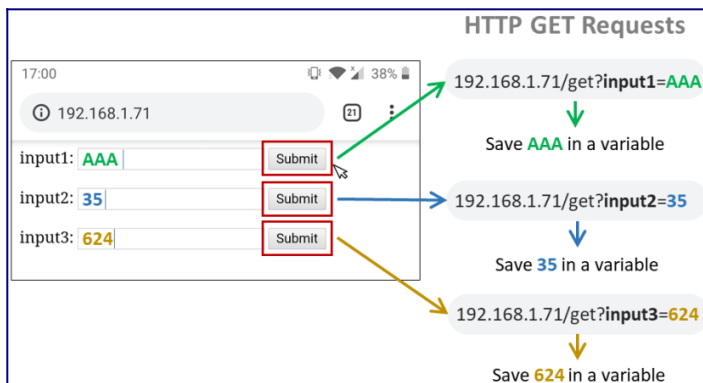
In dieser Anleitung erfahren Sie, wie Sie erstellen Sie eine ESP32/ESP8266 web server mit drei Eingabefelder für die übergabe von Werten an Ihren ESP mit einem HTML-Formular. Anschließend können Sie diese Werte als Variablen in Ihrem code. Wir werden mit der Arduino IDE zum Programmieren des boards.



Projekt-Übersicht

In diesem tutorial erstellen wir eine asynchrone web-server mithilfe des [ESPAsyncWebServer Bibliothek](#) zeigt, dass drei Eingabefelder für die Werte übergeben, die Sie in Ihrem code verwenden können, um update-Variablen.

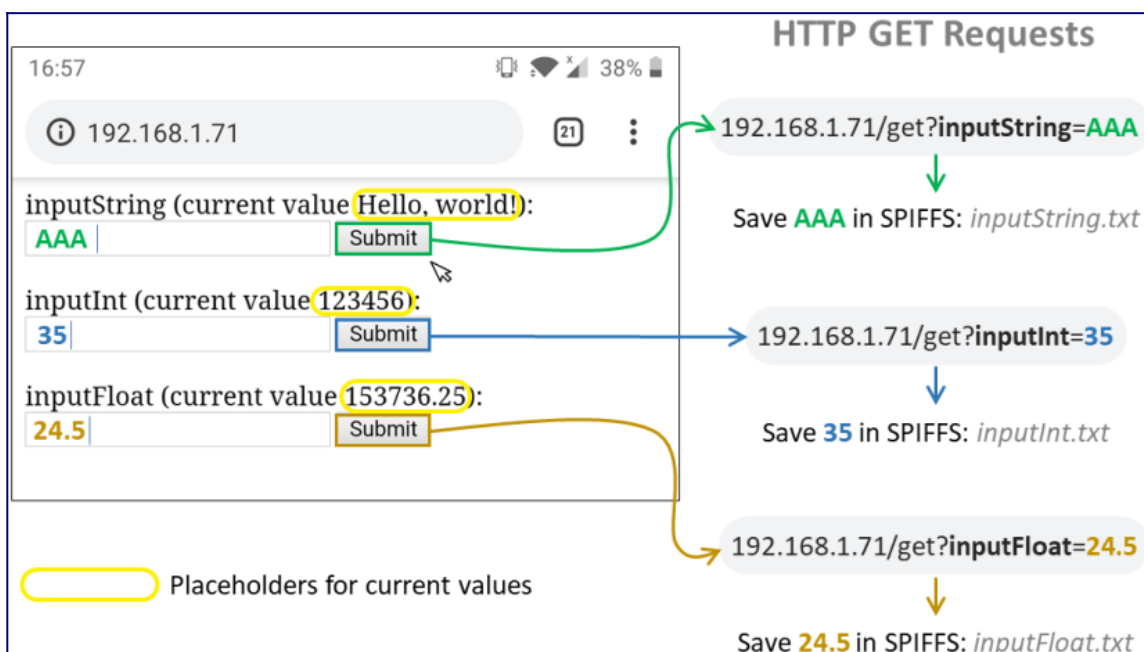
Werfen wir einen Blick auf zwei ähnliche Beispiele. Die folgende Abbildung veranschaulicht, wie das erste Beispiel funktioniert.



Sie haben eine Webseite mit drei Eingabefelder, Sie können den Zugriff mit jedem browser in Ihrem Netzwerk. Wenn Sie einen neuen Wert eingeben und die Taste "Senden" klicken, wird Ihre ESP-update eine variable mit dem neuen Wert.

Wenn Sie jemals benötigt, um update eine variable durch einen ESP-web-server, Sie sollte Folgen Sie diesem Projekt. Mit dieser Methode vermeiden Sie harte Kodierung-Variablen, weil Sie können, erstellen Sie ein Eingabefeld auf einer Webseite zu aktualisieren, eine variable mit einem neuen Wert. Dies kann besonders nützlich zu set Schwelle Werte, die eingestellt SSID/Passwort, ändern, API-Keys, etc...

Später, wir werden auch zeigen Sie wie zu speichern die Variablen, die dauerhaft mit SPIFFS und wie diese zugänglich sind. Hier ist, wie das zweite Beispiel funktioniert.



Diese web-Seite ermöglicht Ihnen die Eingabe von drei Arten von Variablen: String, Int und Float. Dann, jedes mal, wenn Sie Einreichen einen neuen Wert, wird dieser Wert gespeichert ist, in eine SPIFFS Datei. Diese web-Seite enthält auch einen Platzhalter, um zu zeigen Sie die aktuellen Werte.

Erfahren Sie mehr über den Aufbau einer web-server verwenden SPIFFS, Sie können finden Sie auf den folgenden tutorials:

[ESP32 Web Server mithilfe von SPIFFS \(SPI Flash File System\)](#)

- [ESP8266 NodeMCU Webserver mit SPIFFS \(SPI Flash File System\)](#)

Voraussetzungen

Stellen Sie sicher, dass Sie überprüfen Sie alle Voraussetzungen in diesem Abschnitt, bevor Sie das Projekt, um den code zu kompilieren.

1. Installieren ESP32/ESP8266 Board in Arduino IDE

Wir werden Programm die ESP32 und ESP8266 mit Arduino IDE. So müssen Sie den ESP32 oder ESP8266 add-on installiert. Folgen Sie einem der nächsten tutorials zu installieren die ESP-add-on:

- [Installing ESP32 Board in Arduino IDE \(Windows, Mac OS X, Linux\)](#)
- [Installing ESP8266 Board in Arduino IDE \(Windows, Mac OS X, Linux\)](#)

2. Installation Der Bibliotheken

Zum erstellen des asynchronen web-server zu installieren, müssen Sie diese Bibliotheken.

- **ESP32:** installieren Sie den [ESPAsyncWebServer](#) und die [AsyncTCP](#) Bibliotheken.
- **ESP8266:** installieren Sie den [ESPAsyncWebServer](#) und die [ESPAsyncTCP](#) Bibliotheken.

Diese Bibliotheken sind nicht verfügbar, um die Installation über den Arduino Bibliotheksmanager, so müssen Sie kopieren Sie die library-Dateien, um die Arduino-Installationsordner. Alternativ dazu können Sie in Ihrem Arduino-IDE, Sie können gehen Sie auf **Sketch > Include Library > Add .zip-Bibliothek** und wählen Sie die Bibliotheken haben Sie gerade heruntergeladen haben.

3. Benötigte Teile

Um diesem tutorial zu Folgen, Sie müssen nur einen [ESP32](#) oder [ESP8266](#) (Lesen [ESP32 vs ESP8266](#)). Es gibt keine Schaltung für dieses Projekt.

1. ESP32/ESP8266 Griff Eingabefelder auf der Webseite mit HTML-Formular

Kopieren Sie den folgenden code in die Arduino IDE. Dann geben Sie Ihre Netzwerk-Anmeldeinformationen (SSID und Passwort), um Sie für Sie arbeiten.

/*****

Rui Santos

Complete project details at <https://RandomNerdTutorials.com/esp32-esp8266-input-data-html-form/>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

*****/

```
#include <Arduino.h>

#ifdef ESP32

    #include <WiFi.h>

    #include <AsyncTCP.h>

#else

    #include <ESP8266WiFi.h>

    #include <ESPAsyncTCP.h>

#endif

#include <ESPAsyncWebServer.h>

AsyncWebServer server(80);

// REPLACE WITH YOUR NETWORK CREDENTIALS

const char* ssid = "REPLACE_WITH_YOUR_SSID";

const char* password = "REPLACE_WITH_YOUR_PASSWORD";

const char* PARAM_INPUT_1 = "input1";
```

```

const char* PARAM_INPUT_2 = "input2";

const char* PARAM_INPUT_3 = "input3";


// HTML web page to handle 3 input fields (input1, input2, input3)
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html><head>

  <title>ESP Input Form</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">

</head><body>

  <form action="/get">

    input1: <input type="text" name="input1">

    <input type="submit" value="Submit">

  </form><br>

  <form action="/get">

    input2: <input type="text" name="input2">

    <input type="submit" value="Submit">

  </form><br>

  <form action="/get">

    input3: <input type="text" name="input3">

    <input type="submit" value="Submit">

  </form>

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


void notFound(AsyncWebServerRequest *request) {

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

}


void setup() {

```

```

Serial.begin(115200);

WiFi.mode(WIFI_STA);

WiFi.begin(ssid, password);

if (WiFi.waitForConnectResult() != WL_CONNECTED) {

    Serial.println("WiFi Failed!");

    return;

}

Serial.println();

Serial.print("IP Address: ");

Serial.println(WiFi.localIP());


// Send web page with input fields to client
server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request){

    request->send_P(200, "text/html", index_html);

});


// Send a GET request to <ESP_IP>/get?input1=<inputMessage>
server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {

    String inputMessage;

    String inputParam;

    // GET input1 value on <ESP_IP>/get?input1=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {

        inputMessage = request->getParam(PARAM_INPUT_1)->value();

        inputParam = PARAM_INPUT_1;

    }

    // GET input2 value on <ESP_IP>/get?input2=<inputMessage>
    else if (request->hasParam(PARAM_INPUT_2)) {

        inputMessage = request->getParam(PARAM_INPUT_2)->value();
    }
}

```

```

    inputParam = PARAM_INPUT_2;
}

// GET input3 value on <ESP_IP>/get?input3=<inputMessage>
else if (request->hasParam(PARAM_INPUT_3)) {
    inputMessage = request->getParam(PARAM_INPUT_3)->value();
    inputParam = PARAM_INPUT_3;
}
else {
    inputMessage = "No message sent";
    inputParam = "none";
}

Serial.println(inputMessage);

request->send(200, "text/html", "HTTP GET request sent to your ESP on input
field ("

                                + inputParam + ") with value: " +
inputMessage +

                                "<br><a href=\"/\>Return to Home
Page</a>");

});

server.onNotFound(notFound);

server.begin();
}

void loop() {

}

```

RAW Code

```

/*****
Rui Santos

```

Complete project details at <https://RandomNerdTutorials.com/esp32-esp8266-input-data-html-form/>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

*****/

```
#include <Arduino.h>
#ifdef ESP32
    #include <WiFi.h>
    #include <AsyncTCP.h>
#else
    #include <ESP8266WiFi.h>
    #include <ESPAsyncTCP.h>
#endif
#include <ESPAsyncWebServer.h>

AsyncWebServer server(80);

// REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

const char* PARAM_INPUT_1 = "input1";
const char* PARAM_INPUT_2 = "input2";
const char* PARAM_INPUT_3 = "input3";

// HTML web page to handle 3 input fields (input1, input2, input3)
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html><head>
  <title>ESP Input Form</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head><body>
  <form action="/get">
    input1: <input type="text" name="input1">
    <input type="submit" value="Submit">
  </form><br>
  <form action="/get">
    input2: <input type="text" name="input2">
    <input type="submit" value="Submit">
  </form><br>
  <form action="/get">
    input3: <input type="text" name="input3">
    <input type="submit" value="Submit">
  </form>
</body></html>)rawliteral";

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

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  if (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("WiFi Failed!");
    return;
  }
  Serial.println();
  Serial.print("IP Address: ");
```



```

Serial.println(WiFi.localIP());

// Send web page with input fields to client
server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
});

// Send a GET request to <ESP_IP>/get?input1=<inputMessage>
server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String inputMessage;
    String inputParam;
    // GET input1 value on <ESP_IP>/get?input1=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
        inputMessage = request->getParam(PARAM_INPUT_1)->value();
        inputParam = PARAM_INPUT_1;
    }
    // GET input2 value on <ESP_IP>/get?input2=<inputMessage>
    else if (request->hasParam(PARAM_INPUT_2)) {
        inputMessage = request->getParam(PARAM_INPUT_2)->value();
        inputParam = PARAM_INPUT_2;
    }
    // GET input3 value on <ESP_IP>/get?input3=<inputMessage>
    else if (request->hasParam(PARAM_INPUT_3)) {
        inputMessage = request->getParam(PARAM_INPUT_3)->value();
        inputParam = PARAM_INPUT_3;
    }
    else {
        inputMessage = "No message sent";
        inputParam = "none";
    }
    Serial.println(inputMessage);
    request->send(200, "text/html", "HTTP GET request sent to your ESP on input
field ("
                                + inputParam + ") with value: " +
inputMessage +
                                "<br><a href=\\\"/\\\">Return to Home
Page</a>");
});
server.onNotFound(notFound);
server.begin();
}

void loop() {
}

```

Wie der code funktioniert

Werfen wir einen kurzen Blick auf den code werfen und sehen, wie es funktioniert.

Einschließlich Bibliotheken

Zuerst, die erforderlichen Bibliotheken. Sie gehören verschiedene Bibliotheken, die je auf dem Brett, die Sie verwenden. Wenn Sie über einen ESP32, lädt der code die folgenden Bibliotheken:

```

#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

```

Wenn Sie über einen ESP8266, Sie gehören folgende Bibliotheken:

```

#include <ESP8266WiFi.h>

```

```
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
```

Netzwerk-Anmeldeinformationen

Vergessen Sie nicht, geben Sie Ihre Netzwerk-Anmeldeinformationen in die folgenden Variablen, so dass die ESP32 oder ESP8266 kann eine Verbindung zu Ihrem Netzwerk.

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

HTML-Formulare und Eingabefelder

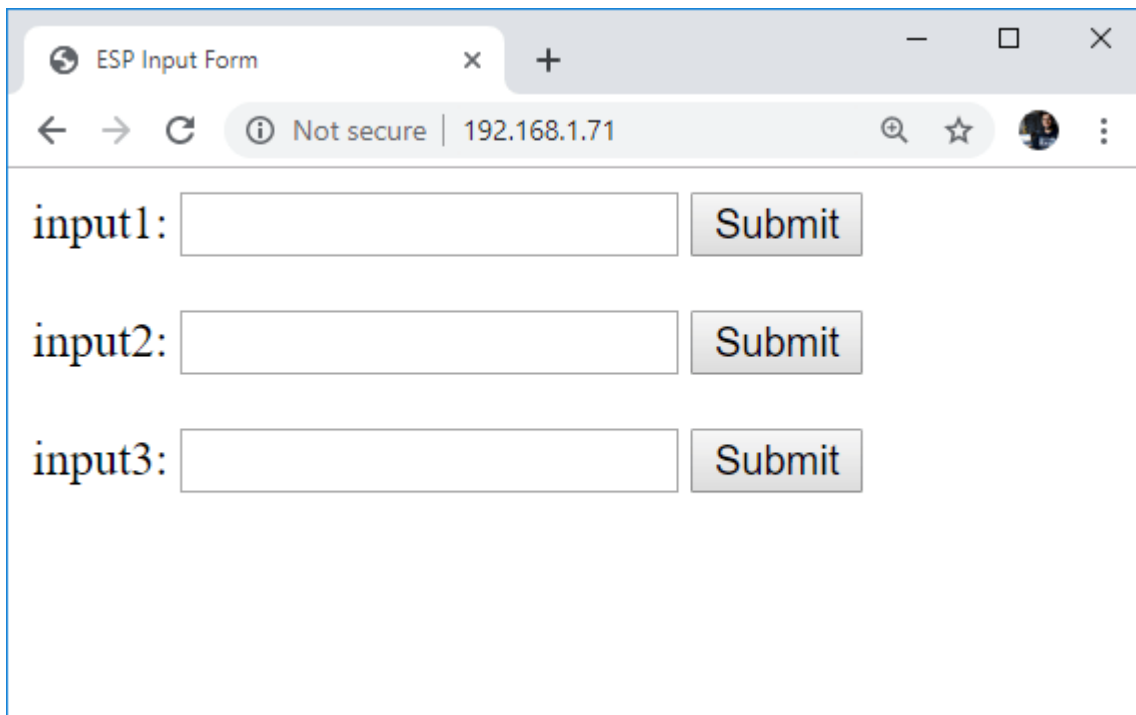
Erste, werfen wir einen Blick auf die HTML, die wir benötigen, um die Anzeige der Eingabefelder.

In unserem Beispiel haben wir drei input-Felder und jedes Feld hat einen "Submit" - button. Wenn der Benutzer Daten eingibt und drückt auf den "Senden" - Knopf, der Wert wird an den ESP und aktualisiert die variable.

Für diesen erstellen Sie in drei Formen:

```
<form action="/get">
  input1: <input type="text" name="input1">
  <input type="submit" value="Submit">
</form><br>
<form action="/get">
  input2: <input type="text" name="input2">
  <input type="submit" value="Submit">
</form><br>
<form action="/get">
  input3: <input type="text" name="input3">
  <input type="submit" value="Submit">
</form>
```

In HTML, the [<form>](#) - tag wird verwendet, um ein HTML-Formular erstellen um Benutzereingaben. In unserem Fall sollte das Formular enthält ein Eingabefeld und eine Schaltfläche.



Werfen wir einen Blick auf die erste form, um zu sehen, wie es funktioniert (die anderen Formen der Arbeit in einer ähnlichen Art und Weise).

```
<form action="/get">
  input1: <input type="text" name="input1">
  <input type="submit" value="Submit">
</form>
```

Das action - Attribut gibt an, wo die Daten eingefügt werden auf die form, nachdem Sie "senden" drücken. In diesem Fall, es wird eine HTTP GET-Anforderung an /Holen?input1=Wert . Der Wert bezieht sich auf den text geben Sie in das Eingabefeld.

Dann, wir definieren zwei input-Felder: ein Textfeld und die Schaltfläche "senden".

Die folgende Zeile definiert einen einzeiligen Eingabefeld.

```
input1: <input type="text" name="input1">
```

Das Typ - Attribut legt fest, wir wollen ein text-Eingabefeld, und das name - Attribut gibt den Namen des input-Elements.

Die nächste Zeile definiert einen button für das senden des HTML-Formular-Daten.

```
<input type="submit" value="Submit">
```

In diesem Fall werden die type - Attribut gibt an, Sie möchten eine Schaltfläche, und der Wert gibt den text auf der Schaltfläche.

Verbindung zu Ihrem Netzwerk herstellen

In der setup() eine Verbindung zu Ihrem lokalen Netzwerk.

```
Serial.begin(115200);
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
if (WiFi.waitForConnectResult() != WL_CONNECTED) {
  Serial.println("WiFi Failed!");
}
```

```

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

```

Behandeln von HTTP-GET-Anforderungen

Dann müssen Sie behandeln die HTTP-GET-Anfragen.

Wenn Sie Zugang zu der route, die URL, die Sie senden, wird die HTML-Seite an den client. In diesem Fall wird der HTML-text wird gespeichert index_html variable.

```

server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
});

```

Dann müssen Sie behandeln, was passiert, wenn Sie eine Anfrage erhalten Sie auf den /get - Routen.

```

server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {

```

Wir erstellen Sie zwei Variablen: inputMessage und inputParam zum speichern der input-Wert und die Eingabe-Feld.

Dann brauchen wir, um zu überprüfen, ob der HTTP-GET-Anforderung enthält, die input1 , input2 oder input3 Parameter. Diese werden gespeichert, auf die PARAM_INPUT_1 , PARAM_INPUT_2 und PARAM_INPUT_3 Variablen.

Wenn die Anfrage enthält die PARAM_INPUT_1 (z.B. eingang1), setzen wir die inputMessage der Wert eingefügt in die input1-Feld.

```

inputMessage = request->getParam(PARAM_INPUT_1)->value();

```

Jetzt haben Sie den Wert, den Sie haben zunächst nur auf die erste form gespeichert inputMessage variable.

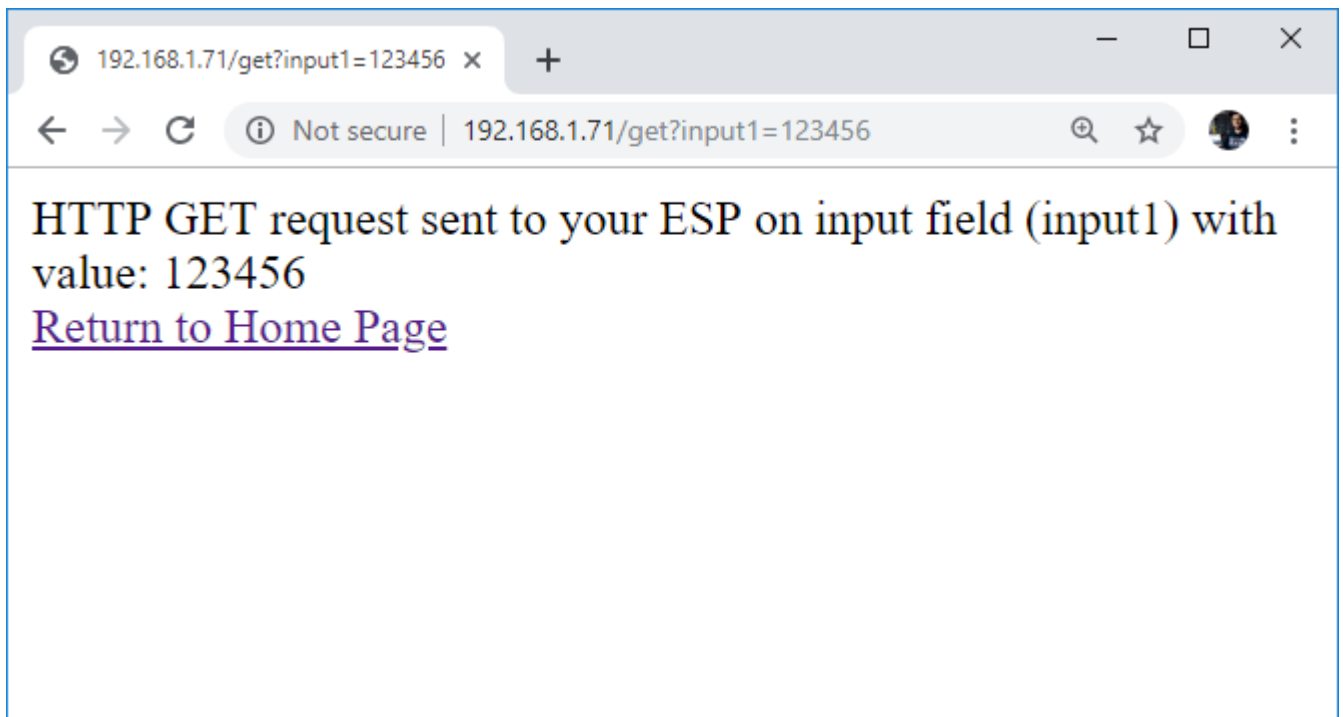
Setzen Sie dann die inputParam variable PARAM_INPUT_1 , so dass wir wissen, wo der Eingang Wert kommt.

Wenn Sie das Formular abschicken, erhalten Sie eine Meldung, dass der Wert, den Sie eingefügt haben und in welchem Bereich. Wir zeigen auch einen link, um wieder auf die Strecke URL (Homepage).

```

request->send(200, "text/html", "HTTP GET request sent to your ESP on input
field ("
    + inputParam + ") with value: " + inputMessage +
    "<br><a href=\"/\">Return to Home Page</a>");

```



Wenn Sie einen Antrag auf eine ungültige URL, nennen wir die `notFound()` - Funktion, definiert am Anfang die Skizze.

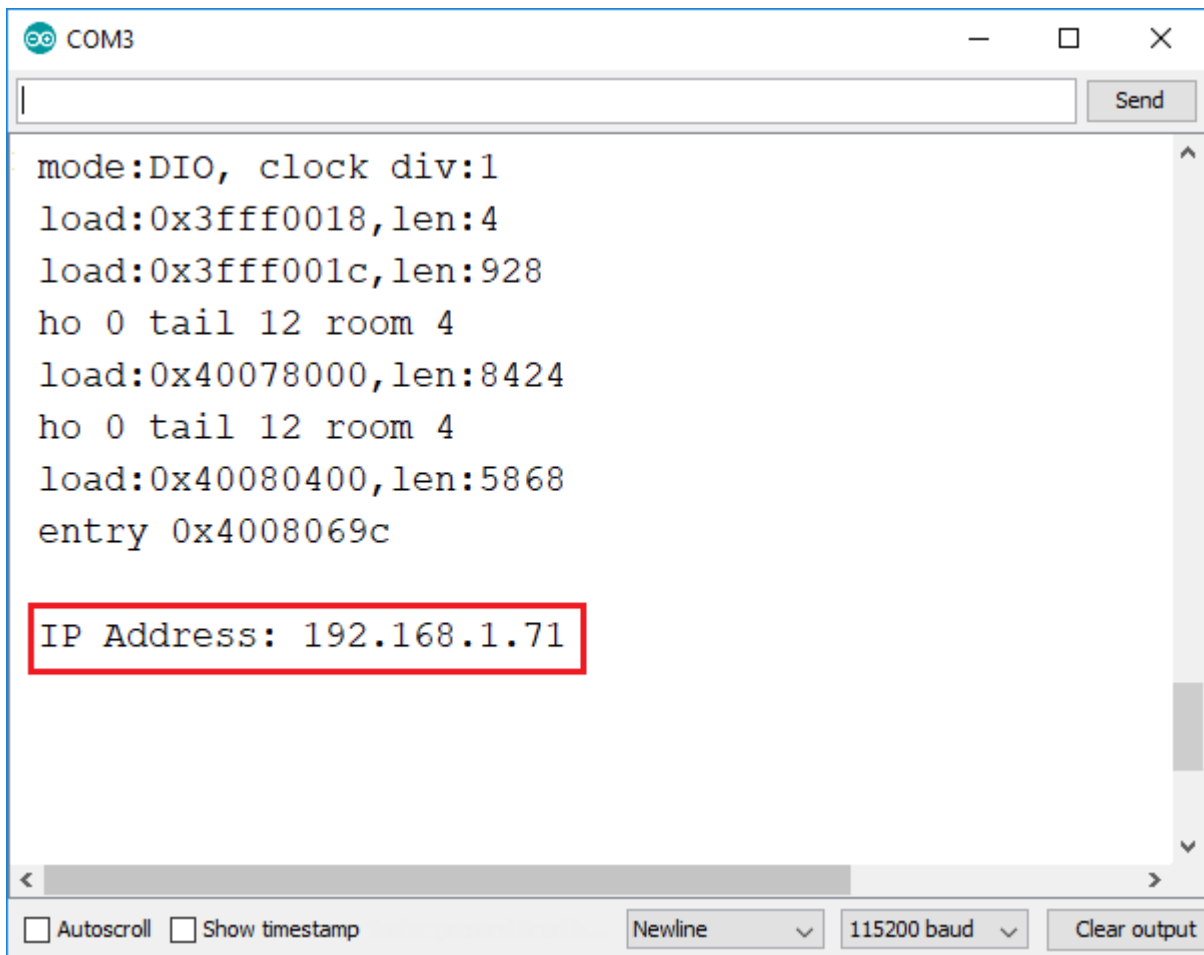
```
void notFound(AsyncWebServerRequest *request) {  
    request->send(404, "text/plain", "Not found");  
}
```

Endlich, starten Sie den server mit den clients.

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

Demonstration

Nach dem hochladen code zu Ihrem board, öffnen Sie Ihre Arduino IDE Serial Monitor mit einer Baudrate von 115200 zu finden, die ESP-IP-Adresse.



```
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:928
ho 0 tail 12 room 4
load:0x40078000,len:8424
ho 0 tail 12 room 4
load:0x40080400,len:5868
entry 0x4008069c

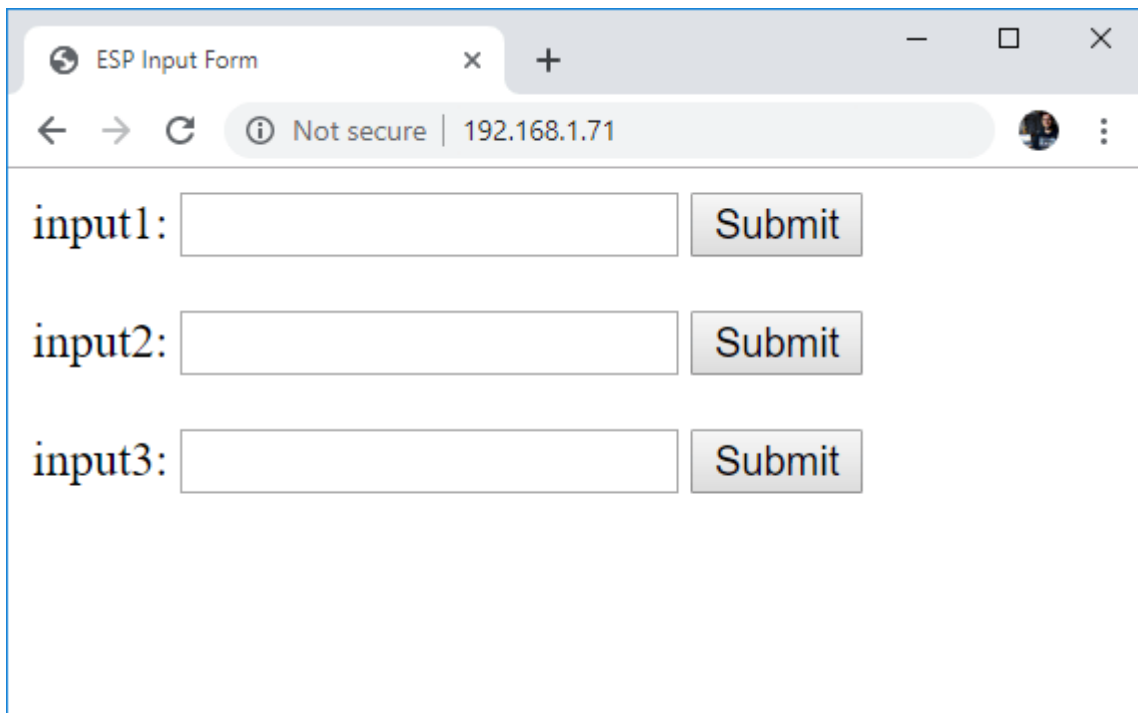
IP Address: 192.168.1.71
```

COM3

Send

Autoscroll Show timestamp Newline 115200 baud Clear output

Öffnen Sie dann Ihren browser und geben Sie die IP-Adresse. Diese web-Seite laden soll:



ESP Input Form

Not secure | 192.168.1.71

input1: Submit

input2: Submit

input3: Submit

Geben Sie beispielsweise **123456** auf input1 - Feld, und drücken Sie dann die Schaltfläche "Submit". Eine neue Seite laden sollte sagen, dass Wert **123456** wurde abgeschickt ESP:



2. ESP32/ESP8266 Speichern Eingabefelder SPIFFS

Jetzt gehen wir zum zweiten Beispiel. In diesem Beispiel speichert die eingefügten Daten auf der Eingabefelder dauerhaft auf SPIFFS. Wir haben auch Hinzugefügt Platzhalter auf der web-Seite zur Anzeige der aktuellen Werte.

Kopieren Sie die folgende Skizze auf der Arduino IDE. Dann, vor dem Upload geben Sie Ihre Netzwerk-Anmeldeinformationen (SSID und Kennwort).

```
/*  
  Rui Santos  
  Complete project details at https://RandomNerdTutorials.com/esp32-esp8266-  
  input-data-html-form/  
  
  Permission is hereby granted, free of charge, to any person obtaining a copy  
  of this software and associated documentation files.  
  
  The above copyright notice and this permission notice shall be included in all  
  copies or substantial portions of the Software.  
  */  
  
#include <Arduino.h>  
#ifdef ESP32  
  #include <WiFi.h>  
  #include <AsyncTCP.h>  
  #include <SPIFFS.h>  
#else  
  #include <ESP8266WiFi.h>  
  #include <ESPAsyncTCP.h>  
  #include <Hash.h>  
  #include <FS.h>  
#endif  
#include <ESPAsyncWebServer.h>
```

```

AsyncWebServer server(80);

// REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

const char* PARAM_STRING = "inputString";
const char* PARAM_INT = "inputInt";
const char* PARAM_FLOAT = "inputFloat";

// HTML web page to handle 3 input fields (inputString, inputInt, inputFloat)
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html><head>
  <title>ESP Input Form</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script>
    function submitMessage() {
      alert("Saved value to ESP SPIFFS");
      setTimeout(function(){ document.location.reload(false); }, 500);
    }
  </script></head><body>
    <form action="/get" target="hidden-form">
      inputString (current value %inputString%): <input type="text"
name="inputString">
      <input type="submit" value="Submit" onclick="submitMessage()">
    </form><br>
    <form action="/get" target="hidden-form">
      inputInt (current value %inputInt%): <input type="number " name="inputInt">
      <input type="submit" value="Submit" onclick="submitMessage()">
    </form><br>
    <form action="/get" target="hidden-form">
      inputFloat (current value %inputFloat%): <input type="number "
name="inputFloat">
      <input type="submit" value="Submit" onclick="submitMessage()">
    </form>
    <iframe style="display:none" name="hidden-form"></iframe>
  </body></html>)rawliteral";

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

String readFile(fs::FS &fs, const char * path){
  Serial.printf("Reading file: %s\r\n", path);
  File file = fs.open(path, "r");
  if(!file || file.isDirectory()){
    Serial.println("- empty file or failed to open file");
    return String();
  }
  Serial.println("- read from file:");
  String fileContent;
  while(file.available()){
    fileContent+=String((char)file.read());
  }
  file.close();
  Serial.println(fileContent);
  return fileContent;
}

void writeFile(fs::FS &fs, const char * path, const char * message){
  Serial.printf("Writing file: %s\r\n", path);
  File file = fs.open(path, "w");
  if(!file){
    Serial.println("- failed to open file for writing");
  }
}

```



```

        return;
    }
    if(file.print(message)){
        Serial.println("- file written");
    } else {
        Serial.println("- write failed");
    }
    file.close();
}

// Replaces placeholder with stored values
String processor(const String& var){
    //Serial.println(var);
    if(var == "inputString"){
        return readFile(SPIFFS, "/inputString.txt");
    }
    else if(var == "inputInt"){
        return readFile(SPIFFS, "/inputInt.txt");
    }
    else if(var == "inputFloat"){
        return readFile(SPIFFS, "/inputFloat.txt");
    }
    return String();
}

void setup() {
    Serial.begin(115200);
    // Initialize SPIFFS
    #ifdef ESP32
        if(!SPIFFS.begin(true)){
            Serial.println("An Error has occurred while mounting SPIFFS");
            return;
        }
    #else
        if(!SPIFFS.begin()){
            Serial.println("An Error has occurred while mounting SPIFFS");
            return;
        }
    #endif

    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    if (WiFi.waitForConnectResult() != WL_CONNECTED) {
        Serial.println("WiFi Failed!");
        return;
    }
    Serial.println();
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());

    // Send web page with input fields to client
    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
        request->send_P(200, "text/html", index_html, processor);
    });

    // Send a GET request to <ESP_IP>/get?inputString=<inputMessage>
    server.on("/get", HTTP_GET, [](AsyncWebServerRequest *request) {
        String inputMessage;
        // GET inputString value on <ESP_IP>/get?inputString=<inputMessage>
        if (request->hasParam(PARAM_STRING)) {
            inputMessage = request->getParam(PARAM_STRING)->value();
            writeFile(SPIFFS, "/inputString.txt", inputMessage.c_str());
        }
        // GET inputInt value on <ESP_IP>/get?inputInt=<inputMessage>

```

```

    else if (request->hasParam(PARAM_INT)) {
        inputMessage = request->getParam(PARAM_INT)->value();
        writeFile(SPIFFS, "/inputInt.txt", inputMessage.c_str());
    }
    // GET inputFloat value on <ESP_IP>/get?inputFloat=<inputMessage>
    else if (request->hasParam(PARAM_FLOAT)) {
        inputMessage = request->getParam(PARAM_FLOAT)->value();
        writeFile(SPIFFS, "/inputFloat.txt", inputMessage.c_str());
    }
    else {
        inputMessage = "No message sent";
    }
    Serial.println(inputMessage);
    request->send(200, "text/text", inputMessage);
});
server.onNotFound(notFound);
server.begin();
}

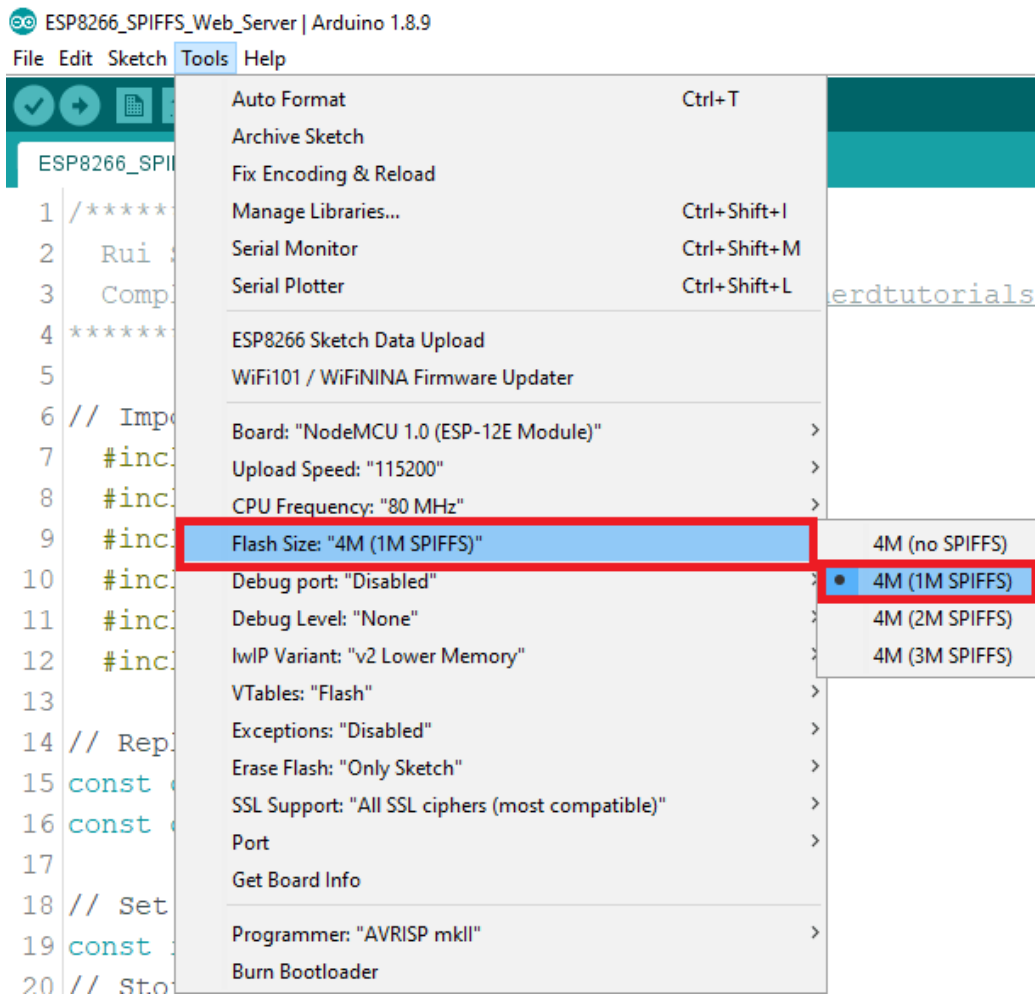
void loop() {
    // To access your stored values on inputString, inputInt, inputFloat
    String yourInputString = readFile(SPIFFS, "/inputString.txt");
    Serial.print("**** Your inputString: ");
    Serial.println(yourInputString);

    int yourInputInt = readFile(SPIFFS, "/inputInt.txt").toInt();
    Serial.print("**** Your inputInt: ");
    Serial.println(yourInputInt);

    float yourInputFloat = readFile(SPIFFS, "/inputFloat.txt").toFloat();
    Serial.print("**** Your inputFloat: ");
    Serial.println(yourInputFloat);
    delay(5000);
}

```

Important: if you're using an ESP8266, make sure you've enabled SPIFFS in **Tools > Flash Size** menu:



Wie der Code funktioniert

Dieser code ist sehr ähnlich mit der vorherigen mit ein paar tweaks. Werfen wir einen kurzen Blick auf Sie und sehen, wie es funktioniert.

Einschließlich Bibliotheken

Der code lädt die folgenden Bibliotheken, wenn Sie mit dem ESP32. Sie müssen die Last SPIFFS library zum schreiben in SPIFFS.

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <AsyncTCP.h>
#include <SPIFFS.h>
```

Wenn Sie über einen ESP8266, müssen Sie den FS - Bibliothek, um eine Schnittstelle mit SPIFFS.

```
#include <ESP8266WiFi.h>
#include <ESPAsyncWebServer.h>
#include <ESPAsyncTCP.h>
#include <Hash.h>
#include <FS.h>
```

HTML-Formular

In diesem Beispiel, wenn Sie Einreichen die Werte, öffnet sich ein Fenster sagen, der Wert wurde gespeichert, SPIFFS, statt umgeleitet zu eine andere Seite wie im vorherigen Beispiel.

Für, dass, wir müssen hinzufügen eine JavaScript-Funktion, in diesem Fall heißt submitMessage() , das angezeigt wird, eine Warnung Nachricht, dass der Wert gespeichert wurde SPIFFS. Nachdem das pop-up, lädt es die web-Seite zeigt die aktuellen Werte.

```
<script>
  function submitMessage() {
    alert("Saved value to ESP SPIFFS");
    setTimeout(function(){ document.location.reload(false); }, 500);
  }
</script>
```

Die Formulare sind auch ein bisschen anders als die vorherigen. Hier ist das Formular für die erste Eingabe.

```
<form action="/get" target="hidden-form">
  inputString (current value %inputString%): <input type="text"
name="inputString">
  <input type="submit" value="Submit" onclick="submitMessage()">
</form>
```

In diesem Fall werden die target - Attribut und ein `<iframe>` - verwendet werden, so dass Sie bleiben auf der gleichen Seite nach dem Absenden des Formulars.

Der name, der sich zeigt für die Eingabefeld enthält einen Platzhalter %inputString% , die wird dann ersetzt durch den aktuellen Wert der inputString variable.

Die onclick="submitMessage()" ruft die submitMessage() JavaScript-Funktion nach klicken des "Senden" - Knopf.

Lesen und Schreiben in SPIFFS

Dann haben wir einige Funktionen zum Lesen und schreiben von SPIFFS.

Die readFile () - liest den Inhalt einer Datei:

```
String readFile(fs::FS &fs, const char * path){
  Serial.printf("Reading file: %s\r\n", path);
  File file = fs.open(path, "r");
  if(!file || file.isDirectory()){
    Serial.println("- empty file or failed to open file");
    return String();
  }
  Serial.println("- read from file:");
  String fileContent;
  while(file.available()){
    fileContent+=String((char)file.read());
  }
  Serial.println(fileContent);
  return fileContent;
}
```

Die writeFile() schreibt Inhalt in eine Datei:

```
void writeFile(fs::FS &fs, const char * path, const char * message){
  Serial.printf("Writing file: %s\r\n", path);
  File file = fs.open(path, "w");
```

```

if(!file){
    Serial.println("- failed to open file for writing");
    return;
}
if(file.print(message)){
    Serial.println("- file written");
} else {
    Serial.println("- write failed");
}
}

```

Prozessor()

Der Prozessor() ist zuständig für die Suche nach Platzhaltern in den HTML-text und ersetzen Sie durch die tatsächlichen Werte gespeichert SPIFFS.

```

String processor(const String& var){
    //Serial.println(var);
    if(var == "inputString"){
        return readFile(SPIFFS, "/inputString.txt");
    }
    else if(var == "inputInt"){
        return readFile(SPIFFS, "/inputInt.txt");
    }
    else if(var == "inputFloat"){
        return readFile(SPIFFS, "/inputFloat.txt");
    }
    return String();
}

```

Behandeln von HTTP-GET-Anforderungen

Die Bearbeitung der HTTP-GET-Anforderungen, die funktioniert die gleiche Weise wie im vorherigen Beispiel, aber dieses mal retten wir die Variablen auf SPIFFS.

Für Beispiel, für die inputString Feld:

```

if (request->hasParam(PARAM_STRING)) {
    inputMessage = request->getParam(PARAM_STRING)->value();
    writeFile(SPIFFS, "/inputString.txt", inputMessage.c_str());
}

```

Wenn die Anfrage enthält inputString (z.B. PARAM_STRING), setzen wir die inputMessage variable auf den Wert der eingereichten inputString form.

```
inputMessage = request->getParam(PARAM_STRING)->value();
```

Speichern Sie dann diesen Wert SPIFFS.

```
writeFile(SPIFFS, "/inputString.txt", inputMessage.c_str());
```

Ein ähnlicher Prozess geschieht für die anderen Formen.

Zugriff auf die Variablen

In der loop() , wir zeigen, wie Sie Zugriff auf die Variablen.

Zum Beispiel, erstellen Sie eine String-variable namens yourInputString , dass liest den Inhalt des inputString.txt Datei auf SPIFFS.

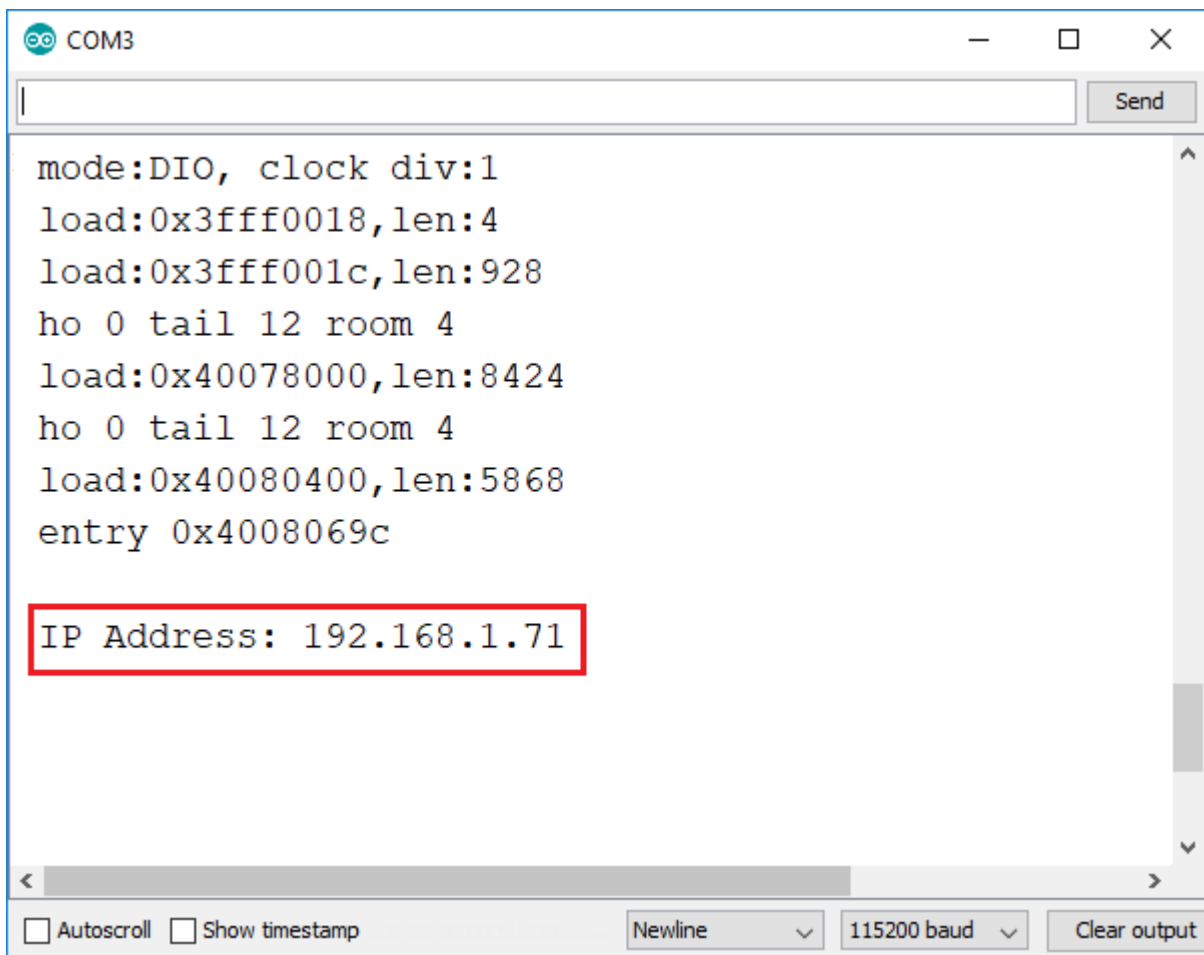
```
String yourInputString = readFile(SPIFFS, "/inputString.txt");
```

Drucken Sie dann die variable in den Seriellen Monitor.

```
Serial.println(yourInputString);
```

Demonstration

Nach dem hochladen code zu Ihrem board, öffnen Sie Ihre Arduino IDE Serial Monitor mit einer Baudrate von 115200 zu finden, die ESP-IP-Adresse.



Öffnen Sie Ihren browser und geben Sie die IP-Adresse. Eine ähnliche web-Seite laden soll (zunächst Ihre aktuellen Werte leer sein).

ESP Input Form x +

← → ↻ ⓘ Not secure | 192.168.1.71

inputString (current value Hello, world!): Submit

inputInt (current value 123456): Submit

inputFloat (current value 153736.25): Submit

Geben Sie eine Zeichenfolge in das erste Eingabefeld und drücken Sie "Abschicken", wiederholen Sie den gleichen Vorgang für die Int- und Float-Werte einzugeben. Jedes mal, wenn Sie drücken Sie die Senden-Schaltfläche für ein Feld, sehen Sie eine Warnung Nachricht wie diese:

ESP Input Form x +

← → ↻ ⚠ Not secure | 192.168.1.71

inputString (current Submit

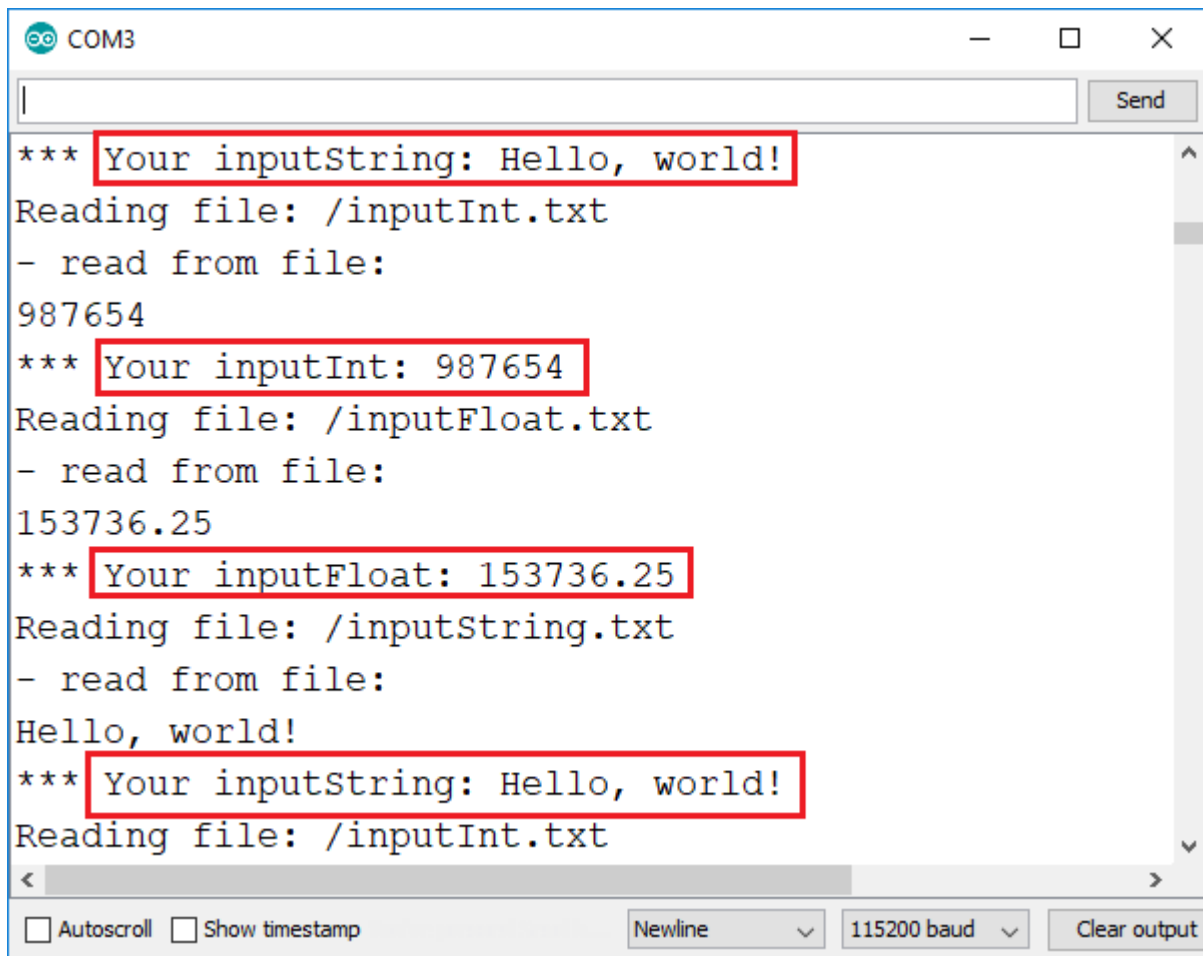
inputInt (current val Submit

inputFloat (current value 153736.25): Submit

192.168.1.71 says
Saved value to ESP SPIFFS
OK

Drücken Sie die "OK" - Taste zum laden der web-Seite und sehen Sie sich die aktuellen Werte aktualisiert.

Wenn Sie den Arduino IDE Serial Monitor öffnen, sehen Sie, dass die gespeicherten Werte gedruckt werden über und über wieder:



The screenshot shows the Arduino IDE Serial Monitor window titled 'COM3'. It displays a repeating sequence of operations: reading a string from '/inputString.txt', reading an integer from '/inputInt.txt', and reading a float from '/inputFloat.txt'. Each read is followed by a print statement. The printed values are: 'Hello, world!', '987654', and '153736.25'. The sequence repeats. The input fields for each read are highlighted with red boxes in the original image. The bottom of the window shows settings: 'Autoscroll' and 'Show timestamp' are unchecked, 'Newline' is selected, '115200 baud' is set, and there is a 'Clear output' button.

```
*** Your inputString: Hello, world!
Reading file: /inputInt.txt
- read from file:
987654
*** Your inputInt: 987654
Reading file: /inputFloat.txt
- read from file:
153736.25
*** Your inputFloat: 153736.25
Reading file: /inputString.txt
- read from file:
Hello, world!
*** Your inputString: Hello, world!
Reading file: /inputInt.txt
```

Für ein abschließendes Projekt, Sie können löschen Sie alle diese Zeilen in der loop() , die drucken Sie alle gespeicherten Werte werden alle 5 Sekunden, wir nur Links auf Zweck für die Fehlersuche.

Zusammenfassung

In diesem tutorial haben Sie gelernt, wie handle input-Felder in einer web-Seite zu aktualisieren, die ESP-Werte der Variablen. Sie können ändern, dass in diesem Projekt legen Sie Schwellenwerte, ändern Sie API-Schlüssel Werte, legen Sie einen PWM-Wert, timer, set SSID/password, etc.

Hier zu den anderen Projekten, die Ihnen gefallen könnten:

- [Anzeigen von Bildern in ESP32 und ESP8266 Web Server](#)
- [ESP32-CAM PIR Motion Detektor mit Foto-Capture \(speichert auf microSD-Karte\)](#)
- [ESP32 Web Server mit BME280 – Advanced-Wetter-Station](#)
- [ESP32 Pinout Reference: Which GPIO pins sollten Sie verwenden?](#)

Erfahren Sie mehr über die ESP32 mit unserem Bestseller-Kurs:

- [Lernen ESP32 mit der Arduino IDE \(eBook + Video-Kurs\)](#)

Danke für das Lesen.

