Tutorial Webserver mit Fetch-API

Inhaltsverzeichnis

Vorwort

Dieses Tutorial ist für den Anfänger gedacht. Es werden nur geringe Grundkenntnisse vorausgesetzt. Alle Beispiele sind sehr einfach gehalten und beschränken sich auf Grundsätzliches. Ziel ist es nicht die perfekte Anwendung oder schicke Webseiten zu erstellen. Vielmehr geht es darum Grundlagen zu erlernen mit denen man dann selber in der Lage ist eine möglichst optimale Lösung für das eigene Projekt zu erstellen.

Will man ein Projekt realisieren, kommt man schnell an die Stelle an der bestimmte Messwerte anzuzeigen sind oder auch bestimmte Eingaben als Sollwerte zu machen sind. Dabei kann es sich um Schaltschwellen , Positionen, Zeiten , Geschwindigkeiten oder was auch immer handen. Was es letztlich ist hängt vom Projekt ab. Neudeutsch wird das als HMI bezeichnet. **H**uman **M**aschine **I**nterface. Das kann mittels einer Anzeige oder Display mit Taster, Drehencoder oder Touchdisplay erfolgen. Alle diese Lösungen erfordern jedoch zusätzliche Hardware. Zur Anzeige und Eingabe sind dabei oft recht aufwendige Menü Strukturen zu realisieren. Mit einem geeigneten Controller, der als Webserver arbeitet, und einem normalen Browser, lässt sich das jedoch ebenso realisieren, und mit einem Handy ist ein Touch Display bereits sogar drahtlos vorhanden. Im Internet sind dazu tausende von Beispielen für die unterschiedlichsten Anwendungsfälle zu finden. Gerade der Anfänger ist, mit dem was da zu finden ist, jedoch häufig völlig überfordert. Aus diesem Grunde möchte ich mit dem vorliegenden Tutorial Schritt für Schritt vorgehen und ganz einfach anfangen. An Ende haben wir zwei Webseiten, eine zur Anzeige von Messwerten und mit zwei Tastern um etwas ein und aus zu schalten. Auf einer zweiten Seite können wir numerische Werte eingeben.

Als Controller wird ein ESP 8266 NodeMcu Modul verwendet, ein anderes geht natürlich ebenso. Für einem ESP32 sind Änderungen zu machen. Basierend auf dem Beispiel „Hallo Server“ werden wir den Sketch schrittweise erweitern. Die HTML Seiten werden mit einem Editor erstellt und auf dem Filesystem des ESP gespeichert. Das hat den Vorteil, das wir zumindest den statische Teil der HTML Seiten ohne ESP, zunächst auf dem Browser testen, und bei Bedarf wunschgemäß formatieren können. Als Editor verwende ich dazu Notepad++ . Das Erstellen von HTML Text im Code, wie bei dem Beispiel „Hallo Server“, ist doch oft recht fehlerbehaftet und lässt sich nur schlecht editieren. Es ist damit nur für sehr einfache Webseiten sinnvoll zu verwenden. Wenn man das mit dem

“SPIFFS“ Filesystem erst mal verstanden hat ist es sehr einfach zu verwenden. Um das Filesystem nutzen zu können müssen wir die Arduino IDE mit einem Tool erweitern. Dazu diese Hilfe, oder auch hier diese Hilfe.

Die in diesem Tutorial verwendeten Beispiele gibt es als lauffähige Versionen in einer Zip Datei jeweils in den entsprechenden Verzeichnissen. Die HTML Seiten für das ESP Filesystem liegen immer in dem Projektordner und da in einem Verzeichnis data.

Viele gute Beispiele für fast jeden Anwendungsfall gibt’s bei https://fipsok.de/. Dem Autor möchte ich an dieser Stelle ganz besonders danken, er hat mir bei einigen Stellen auf die Sprünge geholfen. Auch hier kann man vieles finden Rothschopf Werner .

So jetzt geht es aber los.

Die statische HTML Seite und der Grundsketch.

Zu diesem Abschnitt gehört das Beispiel im Verzeichnis „StaticServer“.

Das Erste Beispiel besteht aus der statischen HTML Seite und dem Grundsketch für den Controller, auf beides bauen wir im weiteren Verlauf auf. Die HTML Seite ist absichtlich ganz einfach gehalten. Wer mehr zu HTML wissen will, kann das zum Beispiel auf w3schools oder auch Selfhtml. Aber es gibt auch viele andere Information im Netz dazu. Unsere HTML Seite, die wir als Grundlage verwenden wollen, sieht zunächst so aus.

```
<!DOCTYPE html>
<html lang="de">
<head>
<meta name = "viewport" content = "width = device-width, initial-scale = 1.0">
<meta charset="UTF-8">
<title> Tutorial Web Server</title>
<style>
p{font-size:1.5em}
</style>

</head>
<body>
<h1>Messwerte</h1>
<p>
Messwert 1 : <span id="value1">0</span><br>
Messwert 2 : <span id="value2">0</span><br>
Messwert 3 : <span id="value3">0</span><br>
</p>
</body>
</html>
```

Das ist zunächst mal nichts Besonderes. Im `head` Bereich stehen zwei `meta` Zeilen die eigentlich nicht erforderlich sind. Die Zeile mit `viewport` sorgt dafür das die Anzeige der Seite auf Geräten, mit unterschiedlichem Bildschirmauflösungen, optimiert angezeigt werden kann. Zum Beispiel auf einem Handy. Die zweite Zeile sorgt dafür das auch Umlaute und gängige Sonderzeichen einfach verwendet werden können. Es macht also Sinn diese beiden Zeilen immer zu verwenden.

Ebenfalls im `Head` Bereich finden wir einen `style` tag mit dem die Schriftgröße für den Absatz „`p` tag“ festgelegt wird. Ist vergleichbar mit der Absatz Formatvorlage einer Textverarbeitung.

Im Body Bereich, das ist der Teil den wir anschließend auf unserer Web Seite sehen, finden wir schon mal drei Platzhalter die jeweils eine unterschiedliche „id“ zugewiesen bekommen. Hier sollen später unsere Messwerte angezeigt werden. Solange die noch nicht vorhanden sind wird allerdings

eine 0 angezeigt. Wer will kann die HTML Seite mal mit dem Browser starten. Sie befindet sich in dem Ordner *TutorialWebserver\StaticServer\data\index.html.*

> **!** Damit es zu keinen Missverständnissen kommt, möchte ich ausdrücklich darauf hinweisen, das es sich bei der Bezeichnung „value1“ nicht um ein Arry handelt die angehängte 1 dient lediglich dazu das die id unterschiedlich ist. Das gilt im weiteren Verlauf für alle ähnlichen Bezeichner ebenso. Array´s haben [ ] Klammern.

Um die Texte „Messwert 1“, und die eigentlichen Daten vernünftig zu formatieren, könnte man eine Tabelle mit 2 Spalten anlegen. Da es aber nicht wirklich nötig ist, und den Anfänger eventuell verwirrt, lass ich es hier mal weg.

Als Sketch auf dem ESP verwenden wir das Beispiel „Hallo Server“ das wir allerdings ein wenig umbauen müssen. Die wichtigste Änderung ist die Einbindung der Lib für das ESP Filesystem “SPIFFS“ und die Anpassung des Handles zum versenden unserer Webseite. Damit man beim Start des ESP sehen kann was in dem Filesystem enthalten ist, macht es Sinn sich den Inhalt mittels einer kleinen Funktion anzeigen zu lassen. In dem verwendeten Sketch macht das die Funktion `„FSzeigen()“.`

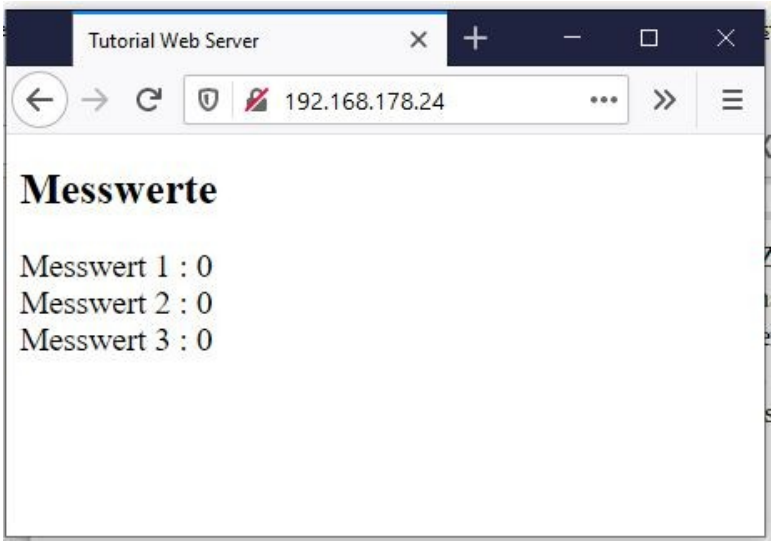
Der ESP8266 bietet die Möglichkeit Dateien aus dem Filesystem direkt zu versenden. In unserem Fall geschieht das mit der Zeile :

```
// --Server Handle einrichten---
     server.serveStatic("/", SPIFFS, "/index.html");
     server.onNotFound(handleNotFound);   // Fehler bearbeiten
```

Damit wird beim Aufruf der url , IP Adresse des ESP im Browser, ohne ein zusätzliches Argument die Datei index.html versendet. Das ist ziemlich einfach und man spart sich das Auslesen aus der Datei in eine temporäre Zeichenkette und das manuelle senden mit Server.send().

Um den ersten Sketch nun zum laufen zu bringen laden wir zunächst die HTML Seite auf den ESP. Dabei ist wichtig das der Monitor der IDE geschlossen ist, sonst gibt es eine Fehlermeldung. In der IDE unter *Werkzeuge / ESP8266 Data Sketch Upload* finden wir die zuvor installierte Erweiterung.

Anschließend ändern wir in dem Sketch noch die Zugangsdaten für das eigene Netzwerk und laden den Sketch auf den ESP. Auf dem Monitor wird uns die IP Adresse angezeigt die der ESP im Netzwerk bekommen hat. Durch die Eingabe dieser Adresse wird die HTML Seite dann im Browser angezeigt. Eventuell muss man die Seite mehrfach aufrufen bis der Router bemerkt hat das sich die Seite im eigenen Netzwerk befindet. Natürlich passiert da jetzt noch nicht viel, aber die Grundlage ist schon mal vorhanden.



Statische Webseite

Zusammenfassung: Wir haben eine HTML Grundseite als Datei erstellt und den Sketch so umgebaut das wir die statische Seite anzeigen können.

Messwerte dynamisch anzeigen

Zu diesem Abschnitt gehört das Beispiel : TutorialWebserver\Messwert\.

Um Daten auf einer HTML Seite anzuzeigen gibt es mehrere Möglichkeiten. Wir werden hier Javascript und die Fetch API verwenden. Was Javascript genau ist kann man z.B. auf w3schools Wiki oder auch selfhtml lernen. Wer hier in dem Tutorial etwas nicht verstanden hat, kann dann dort mal nachlesen. Für unseren Fall geht es um eine Ansammlung von Funktionen, die durch ein Ereignis gestartet, und vom Browser auf dem Client ausgeführt werden. Ein Ereignis kann z.B das Clicken auf einen Button oder ein zeitgesteuertes Ereignis sein. Was es da alles an möglichen Ereignissen gibt kann man z.B auf dieser Seite erfahren.

Im weiteren Verlauf taucht noch der Begriff JSON auf. Im Wesentlichen handelt es sich dabei um einen Datenstrom in einem bestimmten Format. In unserem Fall werden die Messwerte vom ESP als eine Zeichenkette im Json Array Format versendet. Dieses Format ist sehr einfach, die Messwerte sind dabei durch ein Komma getrennt, ähnlich einer CSV Datei. Der Browser empfängt nun die Daten, zerlegt sie anhand der Kommas, und speichert sie in einem Array ab. Darauf haben wir dann mit Javascript Zugriff.

Was Fetch-API ist gibt's z.B hier oder hier. Dabei handet es sich im Wesentlichen um eine Anfrage des Clients an den Server, der darauf eine Antwort (response) sendet. Unsere HTML Seite soll also etwas holen (fetch) . Sie sendet eine Anfrage an den Server und erwartet dann als Antwort die Daten mit den Messwerten. Um Fetch einzusetzen muss man sich an eine festgelegtes Konstrukt aus mehreren Zeilen halten. Die vielen Klammern sehen zunächst etwas verwirrend aus. Als minimal Variante ohne Fehlererkennung kann das so aussehen.

```
fetch('/uri') // Anfrage erstellen an
      .then( function(response){          // erwarten eine Antwort
            return response.json();       // Antwort als Json
            })
      .then(function(myArray){
```

```
            // hier steht der Code zur Auswertung der Antwort
      });
```

Wichtig für uns als Anwender sind die markierten Bereiche. (stark vereinfachte Erklärung)

/uri = Ziel zum Handle auf dem Server , die uri muss vorhanden sein.
json() = Datenformat in dem die Daten erwartet werden
MyArray = Variable zur Datenablage aus der Antwort
Der Bereich der für den Code zur Verarbeitung der Daten benötigt wird.

Die Fetch API ist natürlich wesentlich umfangreicher als es hier in diesem Tutorial gezeigt wird, aber für unsere Aufgabe reicht das erst einmal.

Die HTML Seite Messwerte

Wir werden also zunächst unsere HTLM Seite erweitern und fügen den Teil mit dem Javascript vor dem Ende des body Bereich mit ein. Das ist in dem angeführten Beispiel bereist geschehen, die Änderungen werden hier besprochen.

Der Javaskript Teil

```
<script>
setInterval(loadDaten, 2000);
loadDaten();
function loadDaten(){
fetch('/daten')
      .then( function(response){
            return response.json();
            })
      .then(function(myArr){
            document.getElementById("value1").innerHTML=myArr[0];
            document.getElementById("value2").innerHTML=myArr[1];
            document.getElementById("value3").innerHTML=myArr[2];

      });
}
</script>
```

Mit *`SetInterval(loadDaten,2000)`* wird ein Intervall-Ereignis erzeugt das die Function `loadDaten()` alle 2000ms aufruft. Anschließend wird direkt die Funktion `loadDaten()` aufgerufen dadurch wird die Funktion beim Aufruf der Seite direkt ausgeführt.

Als nächsten finden wir die `Function loadDaten()` *selbst u*nd darin das Konstrukt für den Fetch Aufruf. In der Anfragen ist die uri `/daten` angegeben. Eine Antwort wird als Json erwartet. Die Daten werden in dem Array `myArr` abgespeichert.

Mit den Zeile `document.getElementById("value1").innerHTML=myArr[0]` wird der Inhalt der Array variable `myArr[0]` dem Platzhalter mit der `Id ="value1"` zugewiesen und dort angezeigt. Das war´s eigentlich schon.

Der ESP Sketch Messwerte

Da wir in unserem Projekt keine realen Messwerte haben simulieren wir die drei Messwerte. Wir nenne sie nur wert1 bis wert3 . Sie werden im globalen Teil des Sketches definiert. Im loop gibt es dazu einen Teil der alle 500ms die Messwerte aktualisiert. Als Handle fügen wir eine Zeile mit der

uri `/daten` neu ein und rufen darin die Funktion `sendeDaten()` auf, diese sendet die Messwerte dann als Antwort (response) zurück.

```
// --Server Handle einrichten---
  server.serveStatic("/", SPIFFS, "/index.html");
  server.on("/daten", sendeDaten);
  server.onNotFound(handleNotFound);    // Fehler bearbeiten
```

die Funktion `sendDaten()` selbst sieht dann in unserem Beispiel so aus:

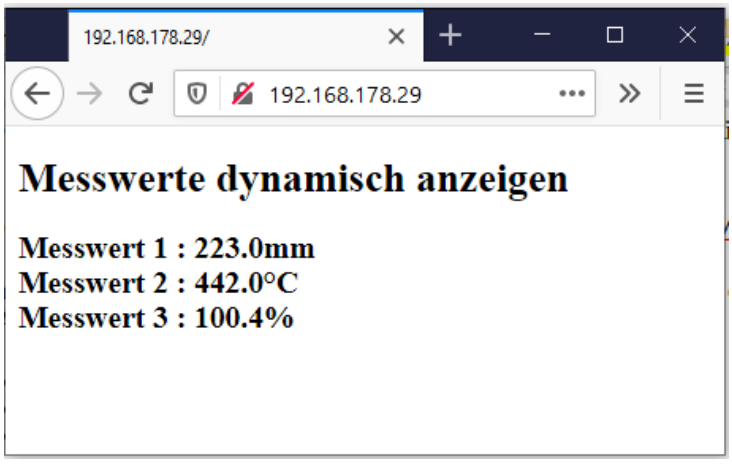
```
void sendeDaten() {
  char sendbuffer[50];// sendbuffer dimensionieren

   // JSON String als Array anlegen im Format:["20.0","572.6","24.0"]
   sprintf(sendbuffer, "[\"%4.1f\",\"%4.1f\",\"%4.1f\"]", wert1, wert2, wert3);
   server.send(200, "application/json", sendbuffer);
   Serial.println(sendbuffer);
   Serial.println(strlen(sendbuffer));
}
```

Zum senden der Daten benötigen wir eine Sendebuffer den wir als Zeichenkette definieren, er muss so groß sein das er die gesamte Zeichenkette plus Endzeichen aufnehmen kann. Etwas Reserve ist da immer gut, es kann schlimme Fehler geben, wenn über den reservierten Bereich hinaus, in den Speicher geschrieben wird. Mit der function `sprintf()` setzen wir uns diese Zeichenkette formatiert im Json Format zusammen und versenden sie. Die beiden letzten Zeilen zeigen uns dann noch Inhalt und Format der übertragen Daten und die Länge im Monitor an. Wer sprintf() und die verschiedene Platzhalter nicht kennt sollte sich das noch mal ansehen. z.B. hier .

Falls man hinter den Messwerte noch Einheiten anzeigen möchte geht das auch ganz einfach mit einer kleinen Änderung in der HTML Seite :

```
Messwert 1 : <span id="value1">0</span>mm<br>
Messwert 2 : <span id="value2">0</span>°C<br>
Messwert 3 : <span id="value3">0</span>%<br>
```

Zusammenfassung : Wir haben bisher eine HTML Seite auf der wir zyklisch dynamische Messwerte anzeigen können ohne die Seite komplett neu laden zu müssen.



HTML Seite mit dynamischen Daten

Button als Ein/Aus Schalter

Zu diesem Abschnitt gehört das Beispiel : TutorialWebserver\ButtonServer

Im diesem Abschritt wollen wir zwei Buttons dazu bauen, mit denen wir zwei Ausgänge des ESP ein und aus schalten können. Zusätzlich soll der aktuelle Zustand jeweils durch eine unterschiedliche Farbe des Buttons dargestellt werden. Für das ESP8266 Node Modul ist zu beachten das die interne LED am pin D4 gegen plus geschaltet ist, und somit bei einem Low Pegel leuchtete. Wen das eventuell verwirrt der kann zum Testen zwei normale LED über Vorwiderstand an die Pin‘s nach GND anschließen.

Die HTML Seite Messwerte mit Button´s

Wir werden zunächst die HTML Seite erweitern um die Buttons anzuzeigen. Dazu nutzen wir den *input* Tag. Den beiden Buttons weisen wir ein *onclick* Ereignis zu. Über eine neue Funktion in dem Script Teil arbeiten wir wieder mit einem Fetch um das click Ereignis an den ESP zu senden. Dieser schaltet damit den betreffende Ausgang und antwortete mit dem aktuellen Schaltzustand.

Aber langsam Schritt für Schritt, zunächst mal einen Button dazu. Den `style` Bereich erweitern wir zunächst auch für die `input` tags der beiden Buttons.

```
input{
      width:5.5em;
      height:2em;
      font-size:1em
}
```

Der input tag selbst sieht dann so aus.

```
<input type="button" id="1" onclick="btnPressed(id)" value="Aus">
```

Der erste Button hat die id=“1“ . Mit dem Ereignis `onclick` rufen wir eine Funktion `btnPressed()` auf und übergeben der Funktion die id als Parameter. Value ist der Text der in dem Button stehen soll.

Die von dem click Ereignis aufgerufene Funktion `btnPressed()` sieht nun so aus und ist damit auch ziemlich übersichtlich. Im wesentlichen kennen wir das ja auch bereits.

```
function btnPressed(btnId){
fetch('/btnclick?id='+ btnId)
      .then( function(response){
            return response.text();
      })
      .then (function(text){
            buttonstate(btnId,text)
      });
}
```

Als Parameter wird die variable `btnId` übergeben. Anschließend erfolgt ein `fetch` Aufruf. Neu ist jetzt, das wir an die uri *`btnclick`* mit dem Zeichen *“?“* ein Argument anhängen. Dieses besteht hier aus den Zeichen `id=` und dem Wert der Variablen `btnId` selbst. Die gesamte uri sieht demnach dann so aus.

`/btnclick?id=1` oder `/btnclick?id=2`. Name und Wert des Arguments können wir mit dem ESP dann abfragen und auswerten.

Wir erwarten als Antwort auf den Fetch diesmal einen Text. Wenn die Daten dann angekommen sind rufen wir eine weitere Funktion `buttonstate()` auf. Ihr übergeben wir die `btnId` des zuvor gedrückten Buttons (1 oder 2) und und den vom ESP gesendeten Text aus der Antwort. In diesem Text steht dann entweder einen 0 oder eine 1, abhängig vom aktuellen Schaltzustand.

Die Funktion buttonstate()

```
function buttonstate(btnId,text){

if (text=="1"){
      document.getElementById(btnId).value="Ein";
      document.getElementById(btnId).style.background="green";
      }
      else{
      document.getElementById(btnId).value="Aus";
      document.getElementById(btnId).style.background="red";
      }
}
```

Diese `function buttonstate()` fragt eigentlich nur ab ob der Text ==“1“ ist oder nicht und verändert die Eigenschaft value und background des Buttons. Der Zugriff erfolgt dabei über die Id des HTML Elements mit `getElementById()`. Da wir diese Funktion nochmals benötigen, wurde das in eine eigene Funktion ausgelagert.

Damit hätten wir es fast geschafft. Es fehlt jetzt noch das Aktualisieren der Buttons zusammen mit den Messwerten. Dazu erweitern wir die bereits bestehende Funktion `loadDaten()` um zwei weitere Zeilen.

```
function loadDaten(){
fetch('/daten')
      .then( function(response){
            return response.json();
            })
      .then(function(myArr){
            value1.innerHTML=myArr[0];
            value2.innerHTML=myArr[1];
            value3.innerHTML=myArr[2];
            buttonstate(1,myArr[3]);
            buttonstate(2,myArr[4]);
      });
```

Hier wird ebenfalls die Funktion `buttonstate()` aufgerufen, einmal für das HTML Element mit der Id=1 und dann mit der Id=2. In Array Elemente `myArr[3]` und `myArr[4]`wird dann der vom ESP gesendete aktuelle Schaltzustand erwartet. Damit hätten wir die HTML Seite fertig. Nun geht es noch an den ESP Sketch.

Der Sketch Button Server

Im Sketch für den ESP benötigen wir zunächst globale Statusvariable für den Zustand der Buttons (ein oder aus) und zwei Pins für die Ausgänge.

In der Funktion `sendeDaten()` erweitern wir die Zeile zum senden der Daten um die aktuellen Satuswerte der Buttons.

```
sprintf(sendbuffer, "[\"%4.1f\",\"%4.1f\",\"%4.1f\",\"%1u\",\"%1u\"]",
wert1, wert2, wert3,btnstate1,btnstate2);
```

Dann benötigen wir eine neuen Handle für dir uri */btnclick* bei dem die Funktion `getDaten()` aufgerufen wird.

```
// --Server Handle einrichten---
  server.serveStatic("/", SPIFFS, "/index.html");
  server.on("/daten", sendDaten);
  server.on("/btnclick", getDaten);
  server.onNotFound(handleNotFound);    // Fehler bearbeiten
```

Und natürlich die Funktion `getDaten()` selbst um die Daten zu lesen und die geänderten Zustände gleich wieder als Antwort (response) zu senden.

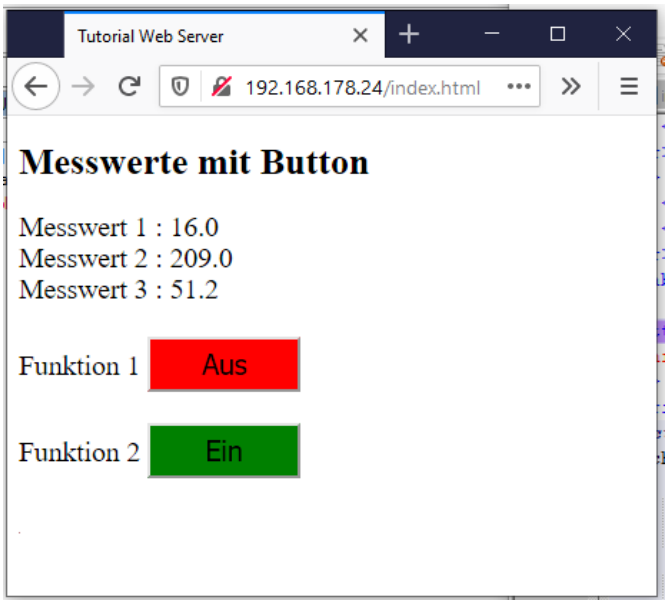
```
void getDaten() {
  char state[] = {"0"};

  if (server.arg(0) == "1") {             // Button mit Id 1 wurde gedrückt
    btnstate1 = !btnstate1;               // toggle Zustand
    if (btnstate1) strcpy(state, "1");    // aktuellen Zustand in sendebuffer
  }
  if (server.arg(0) == "2") {             // Button mit Id 2 wurde gedrückt
    btnstate2 = !btnstate2;
    if (btnstate2) strcpy(state, "1");
  }
  server.send(200, "text/plane", state);  // Antwort senden als Text
}
```

Die Zeichenkette `state` wird zunächst mit dem Zeichen für die 0 beschrieben. So jetzt wird es etwas verzwickt, ist aber eigentlich ganz einfach. Das Objekt Server liefert mit der Methode .arg(0) ein String Objekt mit dem Inhalt des ersten Arugments der uri zurück. Deshalb steht die 1 in „“ es handelt sich also um einen String Objekt mit dem Inhalt für das Zeichen 1 . Im Weiteren wird je nach dem welcher Button gedrückt wurde der Zustand getoggelt. Mittels `if()` wird die Zeichenkette state mit dem Zeichen für eine 1 beschrieben. `strcpy(state, "1")`. Anschließend wird der Wert als Text versendet.

Natürlich wäre an dieser Stelle ebenfalls eine andere Variante möglich , man könnte für buttonstate1 und buttonstate2 ein Array verwenden, und mit Wert der Variablen server.arg(0) direkt auf das Array Element zugreifen. Auf der anderen Seite kann die Variable `btnnstate1` ja aber auch z.B. `btnLuefter` heißen und damit ist das Programm dann wieder einfacher lesbar. Der Name der Variablen sagt somit etwas über die Funktion aus. Wenn man jetzt viele Buttons auf der Seite hat macht es sicherlich Sinn mit einem Array zu arbeiten.

An diese Stelle möchte ich nun noch auf etwas hinweisen. Im Javascript haben wir an die uri ja id=1 angehängt. Diesen Namen „id“ können wir im ESP Sketch ebenfalls abfragen. Das geht mit der Methode `server.argName()`.

Zusammenfassung : Wir haben eine Webseite auf der wir dynamisch Daten ausgeben und zwei Buttons mit denen sich zwei Zustände schalten lassen. Der Schaltzustand des Buttons wird farblich und inhaltlich dynamisch geändert dargestellt.



Messwerte mit Buttons

Seite zur Eingabe von Sollwerten

Zu diesem Abschnitt gehört das Beispiel: TutorialWebserver\SollwertServer

Bei einem realen Projekt könnte jetzt noch die Möglichkeit bestehen, das Sollwerte oder auch Zeitwerte eingegeben werden müssen. Das soll hier über eine zweite HTML Seite gemacht werden. Wir benötigen also eine HTML Seite für z.B drei numerische Eingaben und einen Send Button. Diese Variante bietet sich immer dann an wenn mehrere Daten gleichzeitig mit einem Send Button übertragen werden können. HTML technisch kann man dazu die drei Eingaben in einem form tag unterbringen und einen SUBMIT Button verwenden. Das machen wir aber nicht, da wir uns jetzt schon so gut mit fetch angefreundet haben wollen wir auch dabei bleiben.

Auf jeder Seite befindet sich zusätzlich noch ein Link auf die jeweils andere Seite.

Zunächst ergänzen wir also die HTML Seite *index.html* um einen Link auf die Seite *sollw.html*

```
<a href="sollw.html">Eingabe Sollwerte </a>
```

Die HTML Seite Sollwerte eingeben

Auf der neuen Seite sind drei Texte, drei Eingabefelder, ein Send Button und ein Link auf die Seite index.html , um zurück auf die Startseite zu gelangen. Eingeben wollen wir Dezimalzahlen. Im Style Bereich befindet sich unterschiedliche Angaben für die verschieden Tags.

```
<style>
p,table{font-size:1.5em}
input{
	width:5em;
	height:1.5em;
	font-size:1em
}
button{
	width:5.5em;
	height:2em;
	font-size:1em
```

```
}
</style>
```

Texte und Eingabefelder befinden sich diesmal in einer Tabelle mit 3 Zeilen und 2 Spalten. Damit kann der Text unterschiedlich lang sein , die Eingabefelder stehen dennoch untereinander. Wenn man physikalische Dimensionen dahinter haben will bietet sich eine dritte Spalte an. Man kann aber ja auch die Dimension links im Text mit angeben. z.B so : Wartezeit (s)

```
<body>
<h1>Sollwerte eingeben</h1>
<table>
<tr>
	<td>Sollwert 1</td>
	<td><input type="text" id="s1" value="0"></td>
</tr>
<tr>
	<td>Sollwert 2</td>
	<td><input type="text" id="s2" value="0"></td>
</tr>
<tr>
	<td>Sollwert 3</td>
	<td><input type="text" id="s3" value="0"></td>
</tr>
</table>
<P>
<button type="button" onclick="sendbtn()">senden</button><br><br>
<a href="index.html">zurück zur Anzeige </a>
</p>
```

Für die Eingabefelder habe ich den Typ „text“ verwendet, für numerische Eingaben gibt es eigentlich noch den Typ „number“. Der lässt aber kein Komma oder Punkt für die Eingabe einer Dezimalzahl zu. Jeder input tag hat eine eigene id.

Im Javascript Teil gibt es zwei Funktionen. Eine benötigen wir zur Anzeige der aktuellen Sollwerte wenn die Seite aufgerufen wird. Die zweite Funktion, welche mit dem „senden“ Button aufgerufen wird zum versenden der Eingaben.

Die Funktion `loadSollw()` wird wieder als erstes im Javascript aufgerufen. Das kennen wir bereits.

```
<script>
loadSollw();
function loadSollw(){
fetch('/sollwert')
	.then( function(response){
		return response.json();
		})
	.then(function(myArr){
		document.getElementById("s1").value=myArr[0];
		document.getElementById("s2").value=myArr[1];
		document.getElementById("s3").value=myArr[2];
	});
}
```

Mit fetch wird eine Anfrage an die uri `/sollwert` erstellt. Die ankommenden Daten der Antwort liegen als Json vor und werden den html Elementen über die id zugeordnet.

Die zweite Funktion bringt etwas neues mit, aber auch das kennen wir eigentlich schon. Es sollen diesmal die Daten der drei Eingabefelder als Argumentliste an die uri angehängt werden.

```
function sendbtn(){
var value1=document.getElementById("s1").value;
value1=value1.substr(0,8);
var value2=document.getElementById("s2").value;
value2=value2.substr(0,8);
var value3=document.getElementById("s3").value;
value3=value3.substr(0,8);
var arg="?s1="+value1+";s2="+value2+";s3="+value3;
fetch('/sendbtn'+arg)
      .then( function(response){
            return response.json();
            })
      .then(function(myArr){
            document.getElementById("s1").value=myArr[0];
            document.getElementById("s2").value=myArr[1];
            document.getElementById("s3").value=myArr[2];
      });
```

Zunächst laden wir dazu die den Inhalt des Eingabefeldes in eine Variable. Der Zugriff erfolgt wieder über `id` und `Value` Eigenschaft. In der nächsten Zeile wird die Länge der Eingabe auf 8 Zeichen begrenzt, falls die Tastatur klemmt ;-) . Das geschieht für alle drei Eingabefelder. Dann wird eine variable `arg` zusammengesetzt, die so die Argumentliste für die uri bereitstellt `fetch('/sendbtn'+arg)`. Man hätte das auch alles direkt beim fetch mit angeben können, aber ich denke das wäre etwas unübersichtlich geworden. Als Antwort erwarten wir ebenfalls ein Json dessen Werte wir wieder den Eingabefeldern über *id* und *value* Eigenschaft zuordnen. In der Antwort sind also die vom ESP aktualisierten und zulässigen Werte enthalten. Das macht dann Sinn wenn z.B ein unzulässiger Wert eingegeben wurde. Fetch bietet auch noch weitere Möglichkeiten um Daten zu versenden, darauf gehe ich hier aber nicht ein.

Die beiden HTML Seiten laufen auch auf dem Browser ganz ohne ESP, aber natürlich ohne Daten. Somit ist es also möglich auch recht umfangreiche Menü Strukturen zu entwickeln. In einem größeren realen Projekt würde man ohnehin zunächst die HTML Seiten als statische Seiten erstellen, optisch gestalten und dann nach und nach mit Leben füllen. In dem Zusammenhang möchte ich noch auf die Möglichkeit hinweisen das man in den meisten Browsern die HTML Seiten auch debuggen kann. Dabei kann man Haltepunkte in den Script Teilen setzen, und sich Variablen Inhalte ansehen. Allerdings kommt es manchmal, bei Fehlern im Javaskript, zu seltsamen Fehlermeldungen, die mich auch schon völlig auf den Holzweg geführt haben.

Der Sketch Sollwert Server

Auf der ESP Seite definieren wir drei neue float Variable sollw1 bis sollw3. Dann benötigen wir 4 neue handles.

```
// --Server Handle einrichten---
  server.serveStatic("/", SPIFFS, "/index.html");
  server.serveStatic("/index.html", SPIFFS, "/index.html");// seite wechseln
  server.serveStatic("/sollw.html", SPIFFS, "/sollw.html");
  server.on("/daten", sendeDaten);
  server.on("/btnclick", handleGetDaten);
  server.on("/sollwert", sendeSollw);     // senden der aktuelle Sollwerte
  server.on("/sendbtn", getSollw);        // Übernahme der Sollwerte
```

Die beiden oberen markierten Zeilen werden für den Seitenwechsel benötigt. Wenn der Client z.B. die Seite *sollw.html* haben will wir diese aus dem Filesystem gesendet. Das kann man sicherlich auch anders machen, ich finde aber diese Variante hat den Vorteil das es auch ohne ESP auf dem PC läuft, und man es da schon mal testen kann. Die dritte markierte Zeile dient zum Aufruf der

`sendeSollw()` Funktion wenn die Seite geladen wurde. Die vierte Zeile dient zur Übernahme der Daten wenn der Button gedrückt wurde.

Die Funktion `sendeSollw()` beinhaltet bereits Bekanntes.

```
void sendeSollw() {
  char sendbuffer[50];
  // JSON String als Array anlegen ["20.0","572.6","24.0"]
  sprintf(sendbuffer, "[\"%4.1f\",\"%4.1f\",\"%4.1f\"]",
          sollw1, sollw2, sollw3);
  server.send(200, "application/json", sendbuffer);
  Serial.println(sendbuffer);
  Serial.println(strlen(sendbuffer));
}
```

Die Funktion `getSollw()` muss die Daten aus der HTML Seite aufnehmen und verarbeiten. Die Daten werden durch das Server Objekt als String-Objekte bereitgestellt. Auch hier könnte man wieder auf die Namen mit `server.argName()` zugreifen. Im Beispiel Sketch werden diese mit auf dem Monitor angezeigt.

```
void getSollw() {
  sollw1 = StringToFloat(server.arg(0));
  sollw2 = StringToFloat(server.arg(1));
  sollw3 = StringToFloat(server.arg(2));

  sendeSollw();// response senden
}
```

Falls in dem Wert ein Komma enthalten ist, soll es durch einen Punkt ersetzt werden. Dazu gibt es noch eine kleine Funktion `StringToFloat()`, die als Rückgabewert einen float Wert liefert.

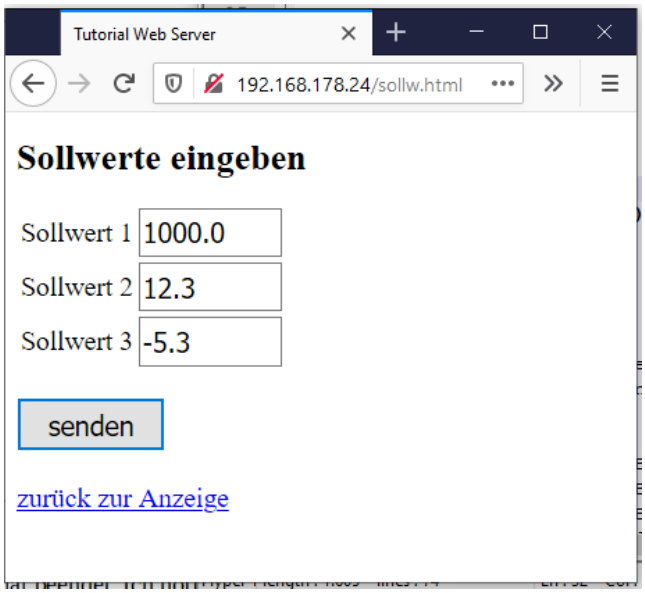
```
// ----- String Objekt to float mit Komma tausch -----------
float StringToFloat(String t) {
  t.replace(",", ".");  // replace im String Objekt
  return t.toFloat();   // String convert to float
}
```

In einem realen Projekt würde ich in diese Funktion noch eine Begrenzung auf min/max Werte integrieren. Mit Aufruf der function `sendeSollw()` werden die Daten dann aktualisiert als response zurück gesendet und auf der Web Seite angezeigt.

Nach dem wir alles auf den ESP geladen haben, können wir auf die Seite „*Sollwert eingeben*“ wechseln und dort die Daten eingeben und absenden.

Zusammenfassung:

Wir haben eine zweite Webseite erstellt auf der sich drei numerische Werte eingeben lassen. Mit einem „senden“ Button werden die Werte an den ESP gesendet um sie dort weiter zu verarbeiten.



Sollwert Eingabe

Daten mit Post versenden

Zu diesem Abschnitt gehört das Beispiel TutorialWebserver\SollwertServerPost

Zum Abschluss gibt es jetzt noch eine Variante bei der die Daten der Seite Sollwerte mittels Post versendet werden. Ebenso wird in dem Beispiel gezeigt wie eine gemeinsame Style Sheet Datei verwendet werden kann. Damit ist, bei Verwendung von mehren Seiten, die Optik aller Seiten einfacher zentral zu verwalten. Im head Bereich wird ein link zu der verwendeten CSS Datei angegeben.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" type="text/css" href="mystyle.css">
    <title>Fetch Post</title>
  </head>
```

Damit werden die style Anweisungen in der Datei geladen und so verwendet als würden sie dort stehen. Beide Seiten haben eine gemeinsame Hintergrundfarbe, und diesmal wird alles mittig angezeigt. Auf der Seite Messwerte gibt es zwei input Elemente die als Button benutzt werden. Das habe ich gemacht um innerhalb des Button den Text dynamisch ändern zu können. Beide sollen eine angemessene Größe haben. Auf der Seite „Sollwerte eingeben“ gibt es drei input Elemente zur Eingabe von Zahlen, diese benötigen andere Abmessungen. Damit das mit einer gemeinsamen Stylesheet Datei funktioniert wurde für die betreffenden Elemente die als Button arbeiten ein CSS class selector verwendet.

Um Daten mit Post zu versenden werden diese in ein form tag eingeschlossen. Der Zugriff erfolgt dabei über das Namen Attribut.

```
<form>
      <label>
      Erster Wert:
      <input name="soll1">
      </label>
      <label>
      Zweiter Wert
      <input name="soll2">
      </label>
      <label>
      Dritter Wert
      <input name="soll3">
      </label>
      <button class="btn" type="button" onclick="sendbtn()">
      Senden
      </button>
    </form>
```

Im javascript Teil werden wieder zwei Funktionen benötigt. Zunächst den Teil der zum laden der Werte beim Aufruf der Seite benötigt wird. Das kennen wir bereits aus dem letzten Beispiel.

```
loadSollw();
function loadSollw(){
  fetch('/sollwert')
  .then(function(response) {
     return response.json();
  })
  .then(function(myArr) {
     document.querySelectorAll('input').forEach ((el,i) => el.value = myArr[i]);
  });
}
```

Da die input Elemente nun keine id mehr haben geht ein Zugriff über `document.getElementById` nicht mehr. Es wird also eine andere Variante verwendet.

Mit `document.querySelectorAll('input')` werden alle Elemente mit einem input tag selektiert. Im weiteren Verlauf wird mittels der `forEach` Methode auf jedes einzelne dieser Element zugegriffen und ihm der Wert der Variablen `myArr[i]` zugewiesen.

Die zweite Funktion zum Versenden der Daten nach dem drücken des „Senden“ Buttons sieht nun so aus.

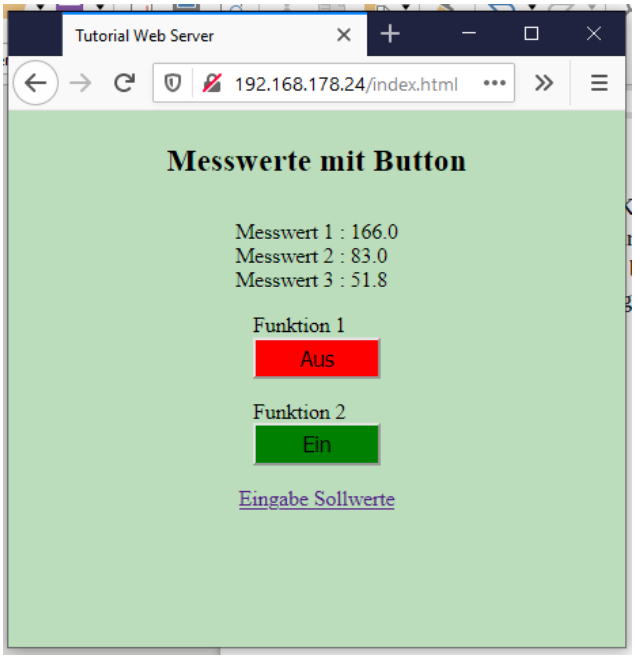
```
function sendbtn(){
  const data = document.querySelector('form');
  fetch('/sendbtn',{
      method: 'post',
      body: new FormData(data)
      })
  .then( function(response) {
    return response.json();
      })
  .then(function(myArr) {
    document.querySelectorAll('input').forEach ((el,i) => el.value = myArr[i]
      });
 }
```

Mit `const data = document.querySelector('form')` werden zunächst die Daten der Form in eine Variable data kopiert. Im fetch Aufruf verwenden wir hinter der uri nun ein sogenanntes *Init* Objekt. In einer geschweiften Klammer können zusätzliche Optionen zur Anfrage an den Server gemacht werden. Mehrere Angaben werden durch ein Komma getrennt. Eigentlich steht das alles in einer Zeile, wegen der besseren Übersicht wird es allgemein jedoch untereinander geschrieben. Was da an umfangreichen Möglichkeiten alles gibt kann man hier. nachlesen. Wir geben die Angabe zur Post Methode und unsere Formdaten hier an. Der Rest der Funktion ist bekannt.

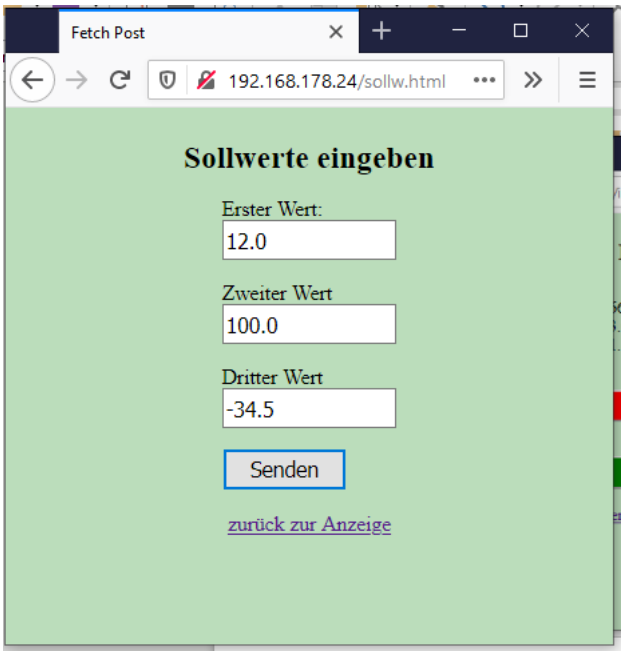
Im Sketch für den ESP müssen wir nun noch einen neuen Handle für die Stylesheet Datei erstellen.

```
server.serveStatic("/", SPIFFS, "/index.html");
server.serveStatic("/index.html", SPIFFS, "/index.html");// seite wechseln
server.serveStatic("/sollw.html", SPIFFS, "/sollw.html");
server.serveStatic("/mystyle.css", SPIFFS, "/mystyle.css");
server.on("/daten", sendeDaten);
```

Jetzt können wir alle aus dem Beispiel Ordner auf den ESP laden und probieren.



Seite Messwerte



Seite Sollwerte

Damit ist das Tutorial beendet. Ich hoffe ich habe mich verständlich ausgedrückt. Ich denke die Beispiele sind recht einfach und man kann sie bei Bedarf anpassen. Es gibt sicher auch elegantere Lösungen. Insbesondere die Gestaltung der Web Seiten und auch die Fetch-API bietet hier noch viel Spielraum. Mit dem hier gezeigten lassen sich aber bereits recht umfangreiche Menü´s mit mehreren Seiten erstellen und mit Leben füllen.

So jetzt aber viel Spaß beim probieren.