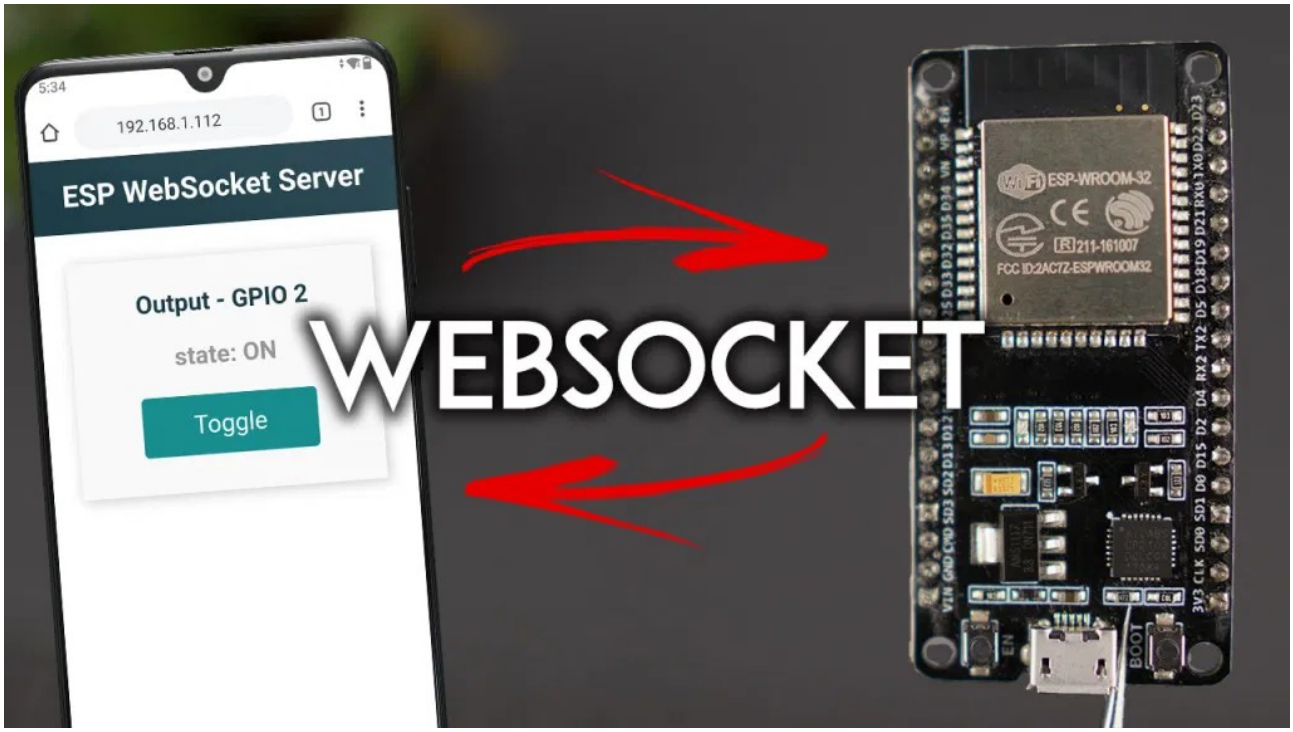
ESP32 WebSocket-Server: Kontrolle der Ausgänge (Arduino-IDE)

In diesem tutorial erfahren Sie, wie Sie bauen eine web server mit die ESP32 mit WebSocket-Kommunikation-Protokoll. Als Beispiel zeigen wir Ihnen, wie man eine web-Seite zum Steuern der ESP32-Ausgänge aus der Ferne. Der Zustand des Ausgangs wird angezeigt, auf der web-Seite, und es aktualisiert automatisch alle clients.



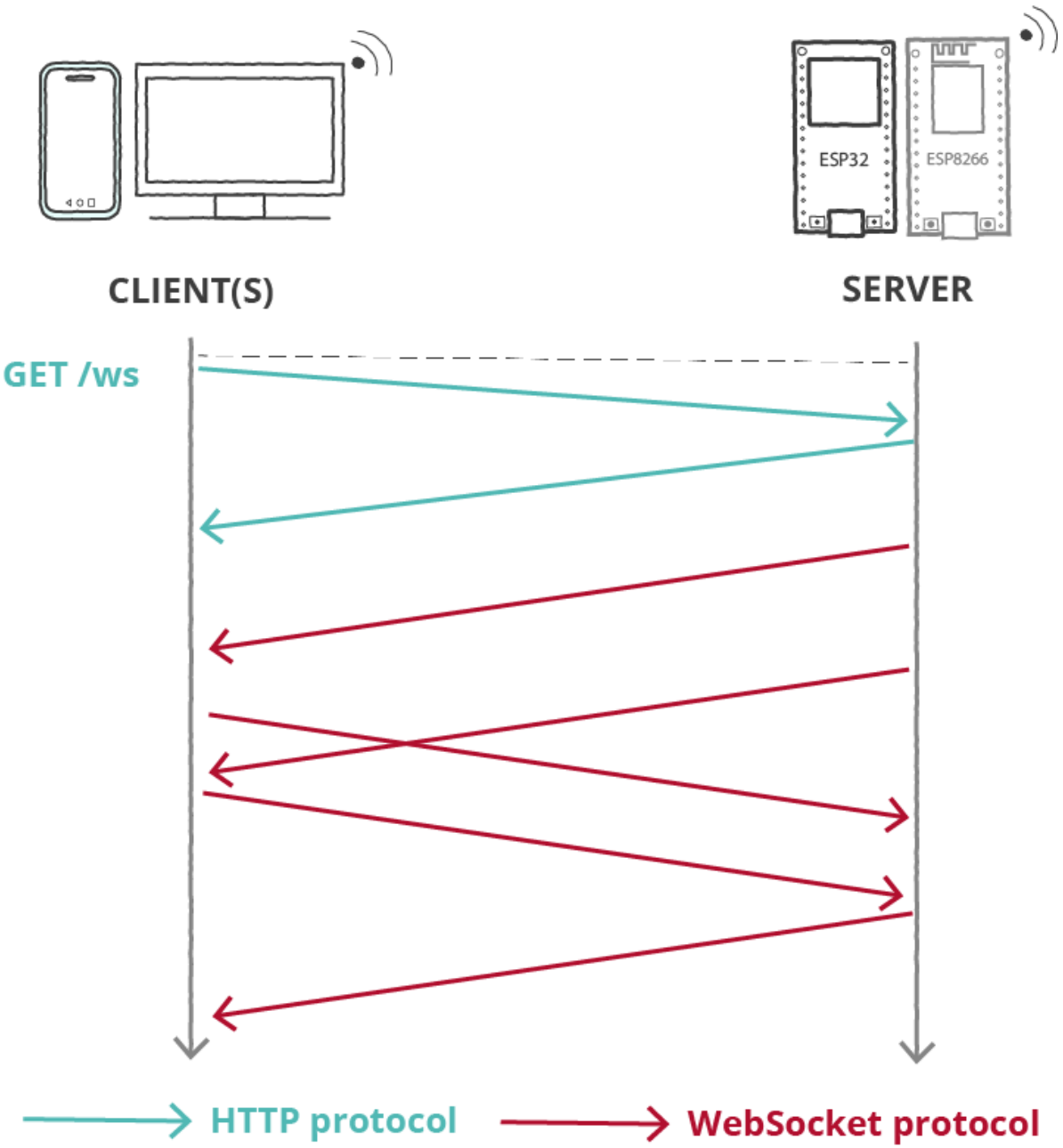
Der ESP32 wird programmiert mit der Arduino IDE und den ESPAsyncWebServer. Wir haben auch eine ähnliche WebSocket-guide für den ESP8266 .

Wenn Sie schon nach einigen von unseren vorherigen web-server Projekten wie diesem , haben Sie vielleicht bemerkt, dass, wenn Sie mehrere Registerkarten (in demselben oder auf unterschiedlichen Geräten) gleichzeitig geöffnet, wird der Status nicht aktualisieren alle tabs automatisch, es sei denn, Sie aktualisieren die web-Seite. Um dieses Problem zu lösen, wir verwenden können WebSocket – Protokoll alle clients benachrichtigt werden, wenn eine änderung und Aktualisierung der web-Seite entsprechend.

Dieses tutorial basiert auf einem Projekt, erstellt und dokumentiert von einem unserer Leser (Stéphane Calderoni). Lesen Sie seine hervorragende tutorial hier .

Einführung WebSocket

Eine WebSocket ist eine dauerhafte Verbindung zwischen einem client und einem server ermöglicht eine bidirektionale Kommunikation zwischen den beiden Parteien über eine TCP-Verbindung. Das bedeutet, Sie können das senden von Daten vom client zum server und vom server an den client zu jeder Zeit.

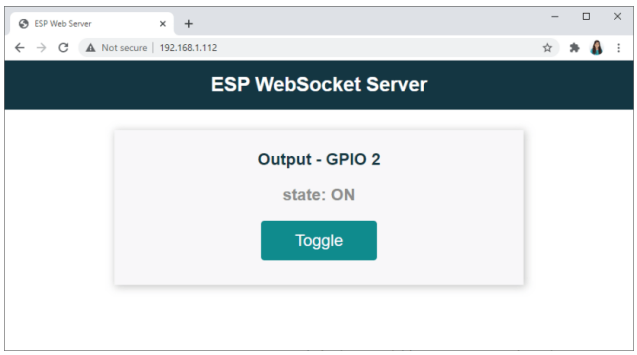


Baut der client eine WebSocket-Verbindung mit dem server durch eine Prozess bekannt als *der WebSocket-handshake* . Der Handschlag beginnt mit einer HTTP-Anforderung/Antwort -, so dass- Servern verarbeiten von HTTP-verbindungen sowie WebSocket-verbindungen auf dem gleichen port. Sobald die Verbindung hergestellt ist, der client und der server können senden WebSocket-Daten in voll-duplex-Modus.
Mit dem WebSockets-Protokoll, das die server (ESP32-board) können Sie senden Informationen an die Kunden oder an alle Kunden, unaufgefordert. Diese ermöglicht auch uns zu senden Informationen an die web-browser, wenn eine änderung Eintritt.

Diese änderung kann etwas sein, das geschehen auf der web-Seite (Sie klicken auf eine Schaltfläche) oder etwas, das geschehen auf dem ESP32 Seite wie das drücken einer physischen Taste auf einer Rennstrecke.

Projekt-Übersicht

Hier ist die web-Seite erstellen wir für dieses Projekt.



- Der ESP32 web server zeigt eine Seite mit einer Schaltfläche zum Umschalten der Zustand des GPIO-2;
- Für Einfachheit, wir sind controlling GPIO-2 – on-board LED. Sie können dieses Beispiel verwenden, um die Kontrolle über alle anderen GPIO;
- Die Oberfläche zeigt die aktuellen GPIO-Zustand. Wenn eine änderung Auftritt, die auf die GPIO-Zustand, die Schnittstelle werden sofort aktualisiert;
- Die GPIO-Status aktualisiert sich automatisch in alle Kunden. Dies bedeutet, dass, wenn Sie haben mehrere web-browser-tabs geöffnet, die auf demselben Gerät oder auf verschiedenen Geräten, Sie sind alle zur gleichen Zeit aktualisiert.

Wie es Funktioniert?

Das folgende Bild beschreibt, was passiert, wenn klicken Sie auf die "Toggle" - button.

Hier ist, was passiert, wenn Sie klicken Sie auf die "Toggle" - button:

1. Klicken Sie auf die "Toggle" - Taste;
2. Der client (browser) sendet Daten über WebSocket-Protokoll mit der "toggle" - Meldung;
3. Der ESP32 (server) empfängt diese Nachricht, damit Sie weiß, es sollten schalten Sie die LED-Status. Wenn die LED zuvor war off, turn it on;
4. Dann sendet es die Daten mit den neuen LED-Status für alle Kunden auch durch WebSocket-Protokoll;
5. Die clients erhalten die Nachricht und update die Status-led auf der web-Seite entsprechend. Dies ermöglicht uns zu aktualisieren Sie alle clients fast augenblicklich, wenn eine Veränderung geschieht.

Vorbereitung der Arduino IDE

Wir werden Programm die ESP32 board mit Arduino IDE, so stellen Sie sicher, dass Sie es installiert haben, in Ihrem Arduino-IDE.

- Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)

Installation Von Bibliotheken – Async-Web-Server

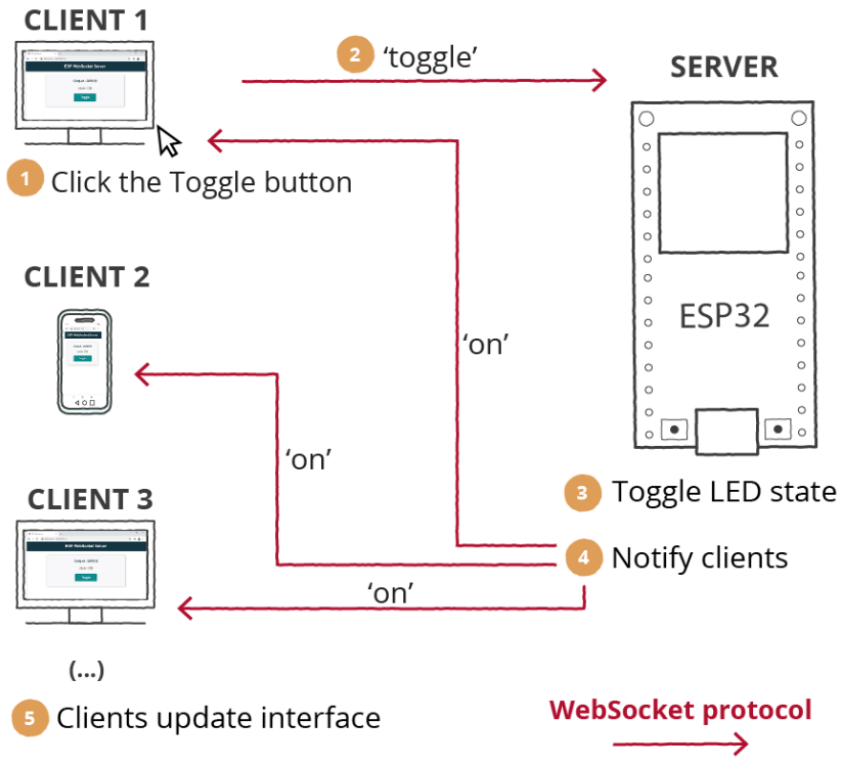
Zum erstellen der web-server verwenden wir den ESPAsyncWebServer Bibliothek. Diese Bibliothek muss die AsyncTCP Bibliothek, um richtig zu arbeiten. Klicken Sie auf die links unten, um download der Bibliotheken.

- ESPAsyncWebServer
- AsyncTCP

Diese Bibliotheken sind nicht verfügbar, um die Installation über den Arduino Bibliotheksmanager, so müssen Sie kopieren Sie die library-Dateien, um die Arduino-Installation Ordner "Libraries". 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.

Code für den ESP32 WebSocket-Server

Kopieren Sie den folgenden code in Ihre Arduino IDE.




```
/*********
  Rui Santos
  Complete project details at https://RandomNerdTutorials.com/esp32-websocket-server-arduino/
  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.
*********/

// Import required libraries
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

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

bool ledState = 0;
const int ledPin = 2;

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

```
AsyncWebSocket ws("/ws");

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <title>ESP Web Server</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <style>
  html {
    font-family: Arial, Helvetica, sans-serif;
    text-align: center;
  }
  h1 {
    font-size: 1.8rem;
    color: white;
  }
  h2{
    font-size: 1.5rem;
    font-weight: bold;
    color: #143642;
  }
  .topnav {
    overflow: hidden;
    background-color: #143642;
  }
  body {
    margin: 0;
  }
  .content {
    padding: 30px;
    max-width: 600px;
    margin: 0 auto;
  }
  .card {
    background-color: #F8F7F9;;
    box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
    padding-top:10px;
    padding-bottom:20px;
  }
  .button {
    padding: 15px 50px;
    font-size: 24px;
    text-align: center;
    outline: none;
    color: #fff;
    background-color: #0f8b8d;
    border: none;
    border-radius: 5px;
    -webkit-touch-callout: none;
    -webkit-user-select: none;
    -khtml-user-select: none;
    -moz-user-select: none;
    -ms-user-select: none;
    user-select: none;
    -webkit-tap-highlight-color: rgba(0,0,0,0);
   }
   /*.button:hover {background-color: #0f8b8d}*/
   .button:active {
     background-color: #0f8b8d;
     box-shadow: 2 2px #CDCDCD;
     transform: translateY(2px);
   }
   .state {
```

```
     font-size: 1.5rem;
     color:#8c8c8c;
     font-weight: bold;
   }
  </style>
<title>ESP Web Server</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" href="data:,">
</head>
<body>
  <div class="topnav">
    <h1>ESP WebSocket Server</h1>
  </div>
  <div class="content">
    <div class="card">
      <h2>Output - GPIO 2</h2>
      <p class="state">state: <span id="state">%STATE%</span></p>
      <p><button id="button" class="button">Toggle</button></p>
    </div>
  </div>
<script>
  var gateway = `ws://${window.location.hostname}/ws`;
  var websocket;
  window.addEventListener('load', onLoad);
  function initWebSocket() {
    console.log('Trying to open a WebSocket connection...');
    websocket = new WebSocket(gateway);
    websocket.onopen    = onOpen;
    websocket.onclose   = onClose;
    websocket.onmessage = onMessage; // <-- add this line
  }
  function onOpen(event) {
    console.log('Connection opened');
  }
  function onClose(event) {
    console.log('Connection closed');
    setTimeout(initWebSocket, 2000);
  }
  function onMessage(event) {
    var state;
    if (event.data == "1"){
      state = "ON";
    }
    else{
      state = "OFF";
    }
    document.getElementById('state').innerHTML = state;
  }
  function onLoad(event) {
    initWebSocket();
    initButton();
  }
  function initButton() {
    document.getElementById('button').addEventListener('click', toggle);
  }
  function toggle(){
    websocket.send('toggle');
  }
</script>
</body>
</html>
)rawliteral";

void notifyClients() {
```

```
  ws.textAll(String(ledState));
}

void handleWebSocketMessage(void *arg, uint8_t *data, size_t len) {
  AwsFrameInfo *info = (AwsFrameInfo*)arg;
  if (info->final && info->index == 0 && info->len == len && info->opcode ==
WS_TEXT) {
    data[len] = 0;
    if (strcmp((char*)data, "toggle") == 0) {
      ledState = !ledState;
      notifyClients();
    }
  }
}

void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType
type,
             void *arg, uint8_t *data, size_t len) {
  switch (type) {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n", client->id(),
client->remoteIP().toString().c_str());
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n", client->id());
      break;
    case WS_EVT_DATA:
      handleWebSocketMessage(arg, data, len);
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
  }
}

void initWebSocket() {
  ws.onEvent(onEvent);
  server.addHandler(&ws);
}

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

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

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
```

```
    Serial.println("Connecting to WiFi..");
  }

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

  initWebSocket();

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

  // Start server
  server.begin();
}

void loop() {
  ws.cleanupClients();
  digitalWrite(ledPin, ledState);
}
```

Legen Sie Ihre Netzwerk-Anmeldeinformationen in die folgenden Variablen und der code wird sofort mit der Arbeit.

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

Wie der Code Funktioniert

Lesen Sie weiter, um zu erfahren, wie der code funktioniert oder überspringen der Demonstration -Abschnitt.

Importieren Von Bibliotheken

Importieren Sie die erforderlichen Bibliotheken zu erstellen, die web-server.

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

Netzwerk-Anmeldeinformationen

Legen Sie Ihre Netzwerk-Anmeldeinformationen in den folgenden Variablen:

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

GPIO-Ausgang

Erstellen Sie eine variable namens ledState zu halten die GPIO-Zustand und eine variable mit dem Namen ledPin , bezieht sich das auf die GPIO-Sie Steuern möchten. In diesem Fall, wir werden das on-board-LED (connected to GPIO-2).

```
bool ledState = 0;
const int ledPin = 2;
```

AsyncWebServer und AsyncWebSocket

Erstellen Sie eine AsyncWebServer Objekt auf port 80.

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

Die ESPAsyncWebServer - Bibliothek enthält eine WebSocket-plugin, dass macht es einfach zu Griff WebSocket-verbindungen. Erstellen Sie eine AsyncWebSocket Objekt aufgerufen ws behandeln die verbindungen auf /ws Pfad.

```
AsyncWebSocket ws("/ws");
```

Aufbau der Web-Seite

Die index_html variable enthält den HTML -, CSS-und JavaScript notwendig zu bauen und Stil der web-Seite und die Bearbeitung von client-server-Interaktionen mittels WebSocket-Protokoll.

Hinweis: wir platzieren alles, was benötigt wird, um bauen Sie die web-Seite, auf der index_html variable, die wir auf der Arduino-Skizze. Beachten Sie, dass es möglicherweise praktischer zu haben, getrennt von HTML -, CSS-und JavaScript-Dateien, die dann den upload auf den ESP32-Dateisystem und verweisen Sie auf den code.

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

Hier ist der Inhalt der index_html variable:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>ESP Web Server</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <style>
  html {
    font-family: Arial, Helvetica, sans-serif;
    text-align: center;
  }
  h1 {
    font-size: 1.8rem;
    color: white;
  }
  h2{
    font-size: 1.5rem;
    font-weight: bold;
    color: #143642;
  }
  .topnav {
    overflow: hidden;
    background-color: #143642;
  }
  body {
    margin: 0;
  }
  .content {
    padding: 30px;
    max-width: 600px;
    margin: 0 auto;
  }
  .card {
    background-color: #F8F7F9;;
```

```
    box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
    padding-top:10px;
    padding-bottom:20px;
  }
  .button {
    padding: 15px 50px;
    font-size: 24px;
    text-align: center;
    outline: none;
    color: #fff;
    background-color: #0f8b8d;
    border: none;
    border-radius: 5px;
    -webkit-touch-callout: none;
    -webkit-user-select: none;
    -khtml-user-select: none;
    -moz-user-select: none;
    -ms-user-select: none;
    user-select: none;
    -webkit-tap-highlight-color: rgba(0,0,0,0);
   }
   .button:active {
     background-color: #0f8b8d;
     box-shadow: 2 2px #CDCDCD;
     transform: translateY(2px);
   }
   .state {
     font-size: 1.5rem;
     color:#8c8c8c;
     font-weight: bold;
   }
  </style>
<title>ESP Web Server</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" href="data:,">
</head>
<body>
  <div class="topnav">
    <h1>ESP WebSocket Server</h1>
  </div>
  <div class="content">
    <div class="card">
      <h2>Output - GPIO 2</h2>
      <p class="state">state: <span id="state">%STATE%</span></p>
      <p><button id="button" class="button">Toggle</button></p>
    </div>
  </div>
<script>
  var gateway = `ws://${window.location.hostname}/ws`;
  var websocket;
  function initWebSocket() {
    console.log('Trying to open a WebSocket connection...');
    websocket = new WebSocket(gateway);
    websocket.onopen    = onOpen;
    websocket.onclose   = onClose;
    websocket.onmessage = onMessage; // <-- add this line
  }
  function onOpen(event) {
    console.log('Connection opened');
  }

  function onClose(event) {
    console.log('Connection closed');
    setTimeout(initWebSocket, 2000);
```

```
  }
  function onMessage(event) {
    var state;
    if (event.data == "1"){
      state = "ON";
    }
    else{
      state = "OFF";
    }
    document.getElementById('state').innerHTML = state;
  }
  window.addEventListener('load', onLoad);
  function onLoad(event) {
    initWebSocket();
    initButton();
  }

  function initButton() {
    document.getElementById('button').addEventListener('click', toggle);
  }
  function toggle(){
    websocket.send('toggle');
  }
</script>
</body>
</html>
```

CSS

Zwischen den <style> </style> - tags wir sind die Stile, die Gestaltung der web-Seite mit CSS. Fühlen Sie sich frei, es zu ändern, um die web-Seite Aussehen, wie Sie es wünschen. Wir werden nicht erklären, wie die CSS-Datei für diese Webseite funktioniert, weil es nicht relevant ist für diese WebSocket-tutorial.

```
<style>
  html {
    font-family: Arial, Helvetica, sans-serif;
    text-align: center;
  }
  h1 {
    font-size: 1.8rem;
    color: white;
  }
  h2 {
    font-size: 1.5rem;
    font-weight: bold;
    color: #143642;
  }
  .topnav {
    overflow: hidden;
    background-color: #143642;
  }
  body {
    margin: 0;
  }
  .content {
    padding: 30px;
    max-width: 600px;
    margin: 0 auto;
  }
  .card {
    background-color: #F8F7F9;;
    box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
```

```
    padding-top:10px;
    padding-bottom:20px;
  }
  .button {
    padding: 15px 50px;
    font-size: 24px;
    text-align: center;
    outline: none;
    color: #fff;
    background-color: #0f8b8d;
    border: none;
    border-radius: 5px;
    -webkit-touch-callout: none;
    -webkit-user-select: none;
    -khtml-user-select: none;
    -moz-user-select: none;
    -ms-user-select: none;
    user-select: none;
    -webkit-tap-highlight-color: rgba(0,0,0,0);
   }
   .button:active {
     background-color: #0f8b8d;
     box-shadow: 2 2px #CDCDCD;
     transform: translateY(2px);
   }
   .state {
     font-size: 1.5rem;
     color:#8c8c8c;
     font-weight: bold;
   }
 </style>
```

HTML

Zwischen den <Körper> </body> - tags fügen wir die web-Seite Inhalte, die für den Benutzer sichtbar ist.

```
<div class="topnav">
  <h1>ESP WebSocket Server</h1>
</div>
<div class="content">
  <div class="card">
    <h2>Output - GPIO 2</h2>
    <p class="state">state: <span id="state">%STATE%</span></p>
    <p><button id="button" class="button">Toggle</button></p>
  </div>
</div>
```

Es gibt eine überschrift 1 mit dem text "ESP-WebSocket-Server". Fühlen Sie sich frei zu ändern text.

```
<h1>ESP WebSocket Server</h1>
```

Dann gibt es die "überschrift 2", mit der "Ausgabe – GPIO 2" text.

```
<h2>Output - GPIO 2</h2>
```

Nach, dass, wir haben einen Abschnitt, der zeigt die aktuelle GPIO-Zustand.

```
<p class="state">state: <span id="state">%STATE%</span></p>
```

Die %STATE% ist ein Platzhalter für die GPIO-Zustand. Es wird ersetzt durch den aktuellen Wert von der ESP32 mit dem Absenden der web-Seite. Die Platzhalter auf den HTML-text gehen soll zwischen % - Zeichen. Dies bedeutet, dass dieser %STATE% text ist wie eine variable, die dann durch den tatsächlichen Wert.

Nach dem senden der web-Seite an den client, der Staat muss dynamisch zu ändern, wenn es eine änderung in den GPIO-Status. Wir erhalten diese Informationen per WebSocket-Protokoll. Dann, JavaScript behandelt, was mit den erhaltenen Informationen, um aktualisieren Sie den Status entsprechend. Um in der Lage sein zu handhaben, dass text mithilfe von JavaScript, den text muss eine id haben, die wir verweisen können. In diesem Fall ist die id Zustand ( ).

Schließlich gibt es einen Absatz mit der Taste zum Umschalten der GPIO-Zustand.

```
<p><button id="button" class="button">Toggle</button></p>
```

Beachten Sie, dass wir haben eine id zu die-Taste (id="button").

JavaScript – Handling WebSockets

Die JavaScript-geht zwischen den <script> </script> tags. Es ist verantwortlich für die Initialisierung einer WebSocket-Verbindung mit dem server da bald das web-interface ist vollständig im browser geladen und Handhabung Datenaustausch über WebSockets.

```
<script>
  var gateway = `ws://${window.location.hostname}/ws`;
  var websocket;
  function initWebSocket() {
    console.log('Trying to open a WebSocket connection...');
    websocket = new WebSocket(gateway);
    websocket.onopen    = onOpen;
    websocket.onclose   = onClose;
    websocket.onmessage = onMessage; // <-- add this line
  }
  function onOpen(event) {
    console.log('Connection opened');
  }

  function onClose(event) {
    console.log('Connection closed');
    setTimeout(initWebSocket, 2000);
  }
  function onMessage(event) {
    var state;
    if (event.data == "1"){
      state = "ON";
    }
    else{
      state = "OFF";
    }
    document.getElementById('state').innerHTML = state;
  }

  window.addEventListener('load', onLoad);

  function onLoad(event) {
    initWebSocket();
    initButton();
  }
```

```
  function initButton() {
    document.getElementById('button').addEventListener('click', toggle);
  }

  function toggle(){
    websocket.send('toggle');
  }
</script>
```

Lassen Sie uns nehmen einen Blick an, wie dies funktioniert.

Das gateway ist der Einstieg in die WebSocket-Schnittstelle.

```
var gateway = `ws://${window.location.hostname}/ws`;
```

Fenster.Lage.hostname wird die aktuelle Adresse der Seite (die web-server-IP-Adresse).

Erstellen Sie eine neue Globale variable namens websocket .

```
var websocket;
```

Fügen Sie einen Ereignis-listener aufrufen, werden die onload - Funktion, wenn die Webseite geladen wird.

```
window.addEventListener('load', onload);
```

Der onload() - Funktion ruft die initWebSocket() - Funktion initialisiert eine WebSocket-Verbindung mit dem server und die initButton() Funktion zum hinzufügen von Ereignis-Listenern zu den Tasten.

```
function onload(event) {
  initWebSocket();
  initButton();
}
```

Die initWebSocket() - Funktion initialisiert eine WebSocket-Verbindung auf das gateway definiert haben. Wir vergeben auch mehrere callback-Funktionen für, wenn die WebSocket-Verbindung geöffnet ist, geschlossen ist oder wenn eine Nachricht empfangen wird.

```
function initWebSocket() {
  console.log('Trying to open a WebSocket connection…');
  websocket = new WebSocket(gateway);
  websocket.onopen    = onOpen;
  websocket.onclose   = onClose;
  websocket.onmessage = onMessage;
}
```

Wenn die Verbindung geöffnet ist, wir drucken Sie einfach eine Nachricht in die Konsole und eine Nachricht senden, sagen "Hallo". Der ESP32 empfängt diese Nachricht, so wissen wir, dass die Verbindung initialisiert wurde.

```
function onOpen(event) {
  console.log('Connection opened');
  websocket.send('hi');
}
```

Wenn für einige Grund die web-socket-Verbindung geschlossen ist, rufen wir die initWebSocket() - Funktion erneut nach 2000 Millisekunden ein (2 Sekunden).

```
function onClose(event) {
  console.log('Connection closed');
  setTimeout(initWebSocket, 2000);
}
```

Die setTimeout() - Methode ruft eine Funktion oder wertet einen Ausdruck nach einer bestimmten Anzahl von Millisekunden.

Endlich, wir müssen zu behandeln, was passiert, wenn wir empfangen eine neue Nachricht. Der server (Ihre ESP-board) wird entweder senden Sie bitte eine "1" oder eine "0" angezeigt. Entsprechend auf die empfangene Nachricht, wir wollen display ein "ON" oder "OFF" - Meldung auf dem Absatz, der zeigt den Stand. Denken Sie daran, dass die - tag mit id="state" ? Wir werden das element und setzen Sie den Wert AUF on oder OFF.

```
function onMessage(event) {
  var state;
  if (event.data == "1"){
    state = "ON";
  }
  else{
    state = "OFF";
  }
  document.getElementById('state').innerHTML = state;
}
```

Die initButton() - Funktion erhält der button durch seine id (button) und fügt einen Ereignis-listener vom Typ 'auf' .

```
function initButton() {
  document.getElementById('button').addEventListener('click', toggle);
}
```

Dies bedeutet, dass, wenn Sie auf die Schaltfläche klicken, wird der toggle - Funktion aufgerufen wird.

Die toggle - Funktion sendet eine Nachricht über die WebSocket-Verbindung mit dem 'toggle' - text.

```
function toggle(){
  websocket.send('toggle');
}
```

Dann das ESP32 behandeln soll, was passiert, wenn Sie diese Nachricht empfängt – wechseln der aktuellen GPIO-Zustand.

Umgang Mit WebSockets – Server

Zuvor haben Sie gesehen, wie Sie zu behandeln der WebSocket-Verbindung auf client-Seite (browser). Nun, lassen Sie uns nehmen einen Blick auf, wie zu Griff es auf der server-Seite.

Informieren Sie Alle Kunden

Die notifyClients() - Funktion benachrichtigt alle clients mit einer Nachricht mit, was Sie übergeben als argument. In diesem Fall, wir möchten zu informieren alle Kunden von den aktuellen LED-Status, wenn es ist eine ändern.

```
void notifyClients() {
  ws.textAll(String(ledState));
}
```

Die AsyncWebSocket - Klasse bietet eine textAll() - Methode für das senden derselben Nachricht an alle clients, die mit dem server verbunden werden zur gleichen Zeit.

Handle WebSocket-Nachrichten

Die handleWebSocketMessage() - Funktion eine callback-Funktion, die ausgeführt wird, wenn wir erhalten neue Daten von den clients, die über WebSocket-Protokoll.

```
void handleWebSocketMessage(void *arg, uint8_t *data, size_t len) {
  AwsFrameInfo *info = (AwsFrameInfo*)arg;
  if (info->final && info->index == 0 && info->len == len && info->opcode ==
WS_TEXT) {
    data[len] = 0;
    if (strcmp((char*)data, "toggle") == 0) {
      ledState = !ledState;
      notifyClients();
    }
  }
}
```

Wenn wir erhalten die "toggle" - Nachricht, wir schaltet den Wert der ledState variable. Darüber hinaus informieren wir alle Kunden durch den Aufruf der notifyClients() - Funktion. Auf diese Weise, alle Kunden sind von der änderung benachrichtigt und aktualisieren Sie die Schnittstelle entsprechend.

```
if (strcmp((char*)data, "toggle") == 0) {
  ledState = !ledState;
  notifyClients();
}
```

Konfigurieren Sie den WebSocket-server

Jetzt müssen wir festlegen, einen Ereignis-listener, mit denen die verschiedenen asynchronen Schritte des WebSocket-Protokolls. Dieser event-handler implementiert werden kann, durch die Definition der onEvent() wie folgt:

```
void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType
type,
 void *arg, uint8_t *data, size_t len) {
  switch (type) {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n", client->id(),
client->remoteIP().toString().c_str());
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n", client->id());
      break;
    case WS_EVT_DATA:
      handleWebSocketMessage(arg, data, len);
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
  }
}
```

Das Typ - argument repräsentiert das Ereignis, das Auftritt. Es kann die folgenden Werte annehmen:

- WS_EVT_CONNECT , wenn ein client angemeldet hat;
- WS_EVT_DISCONNECT wenn ein client sich abgemeldet hat;

- WS_EVT_DATA wenn ein Datenpaket empfangen wird, von den Kunden;
- WS_EVT_PONG in der Antwort auf eine ping-Anfrage;
- WS_EVT_ERROR , wenn ein Fehler vom client empfangen werden.

Initialisieren WebSocket

Schließlich, die initWebSocket() - Funktion initialisiert das WebSocket-Protokoll.

```
void initWebSocket() {
  ws.onEvent(onEvent);
  server.addHandler(&ws);
}
```

Prozessor()

Der Prozessor() - Funktion ist verantwortlich für die Suche nach Platzhaltern, die auf den HTML-text und ersetzen Sie Sie mit was auch immer wir wollen vor Absenden der web-Seite an den browser. In unserem Fall sind, werden wir ersetzen die %STATUS% - Platzhalter mit AUF , wenn die ledState ist 1 . Andernfalls ersetzen Sie es mit OFF .

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

setup()

In der setup() initialisieren Sie die Serielle Monitor für debugging-Zwecke.

```
Serial.begin(115200);
```

Einrichten der ledPin als AUSGANG und setzen Sie es auf NIEDRIG , wenn das Programm das erste mal startet.

```
pinMode(ledPin, OUTPUT);
digitalWrite(ledPin, LOW);
```

Initialisieren der Wi-Fi und das drucken des ESP32 IP-Adresse über den Serial Monitor.

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

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

Initialisieren WebSocket-Protokoll durch den Aufruf der initWebSocket() - Funktion die zuvor erstellt.

```
initWebSocket();
```

Anfragen

Servieren Sie den text gespeichert index_html variable, wenn Sie eine Anfrage erhalten Sie auf den root - / URL – Sie brauchen, um passieren die Prozessor - Funktion, die als argument, um ersetzen Sie die Platzhalter durch die aktuellen GPIO-Zustand.

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

Endlich, starten Sie den server.

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

loop()

Die LED wird physisch kontrolliert auf die Schleife() .

```
void loop() {
  ws.cleanupClients();
  digitalWrite(ledPin, ledState);
}
```

Beachten Sie, dass wir alle rufen die cleanupClients() - Methode. Hier ist, warum (Begründung des ESPAsyncWebServer Bibliothek GitHub Seite):

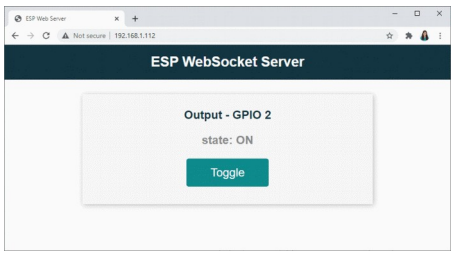
Browser manchmal nicht richtig schließen Sie die WebSocket-Verbindung, auch wenn die close() - Funktion aufgerufen, die in JavaScript. Dieser wird schließlich erschöpfen, die web-server-Ressourcen und bewirkt, dass der server abstürzt. In regelmäßigen Abständen aufrufen der cleanupClients() - Funktion aus der main loop() begrenzt die Anzahl der clients durch das schließen der älteste Kunde, wenn die maximale Anzahl an clients überschritten wurde. Diese können aufgerufen werden, in jedem Zyklus, aber wenn Sie möchten, um verwenden weniger power, dann ruft so selten wie einmal pro Sekunde ist ausreichend.

Demonstration

Nach dem einsetzen Ihrer Netzwerk-Anmeldeinformationen auf die ssid und das Passwort Variablen, können Sie laden den code auf Ihrem board. Vergessen Sie nicht, zu prüfen, ob das richtige board und COM-port ausgewählt ist.

Nach dem hochladen der code, öffnen Sie den Seriellen Monitor mit einer Baudrate von 115200 und drücken Sie die on-board-EN/RST-Taste. Die ESP-IP-Adresse gedruckt werden soll.

Öffnen Sie einen browser auf dem lokalen Netzwerk, und legen Sie die ESP32 IP-Adresse. Sie sollten erhalten Sie Zugriff auf die web-Seite kontrollieren Sie die Ausgabe.



Klicken Sie auf die Taste zum Umschalten der LED. Sie können öffnen Sie mehrere browser-Registerkarten gleichzeitig oder greifen Sie auf die web-server von verschiedenen Geräten und der LED-Status wird update automatisch in alle Kunden wenn es ist eine ändern.

Zusammenfassung

In diesem tutorial haben Sie gelernt, wie ein WebSocket-server mit dem ESP32. Das WebSocket-Protokoll ermöglicht eine Vollduplex-Kommunikation zwischen dem client und dem server. Nach der Initialisierung werden die server und die client Daten austauschen kann zu einem bestimmten Zeitpunkt.

Dies ist sehr nützlich, da der server kann Daten senden an den client, wenn etwas passiert. Für Beispiel, Sie können eine physische Taste , um das setup, die, wenn Sie gedrückt werden benachrichtigt alle clients aktualisieren Sie die web-Schnittstelle.
In diesem Beispiel haben wir Ihnen gezeigt, wie Sie Steuern einen GPIO des ESP32. Sie können diese Methode verwenden, um zu Steuern, mehr GPIOs. Sie können auch das WebSocket-Protokoll zum senden von sensor-Messwerte und Benachrichtigungen zu jeder Zeit.

Wir hoffen, Sie haben gefunden Sie dieses tutorial nützlich. Wollen wir schaffen mehr tutorials und Beispiele, die über das WebSocket-Protokoll. Also, bleiben Sie dran.

Erfahren Sie mehr über die ESP32 mit unseren Ressourcen:

- Lernen ESP32 mit Arduino-IDE
- MicroPython Programmierung mit ESP32 und ESP8266
- Mehr ESP32 Projekte und Anleitungen...