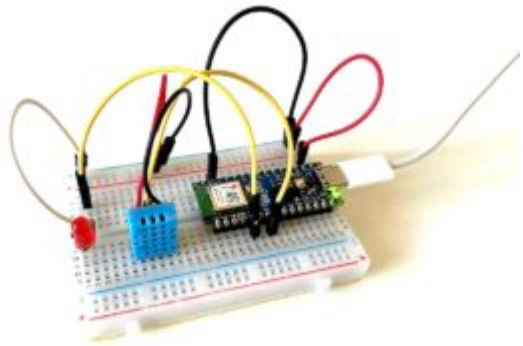
So baust du einen ESP32 Webserver



In diesem Projekt baust du dir deinen eigenen ESP32 Webserver. Hierbei arbeitest du dich schrittweise voran: vom einfachen „Hello World“ über die Steuerung einer LED bis zur Abfrage der aktuellen Temperatur. All dies stellt dein Microcontroller auf einer Webseite zur Verfügung, die du **von einem Gerät deiner Wahl aufrufen** kannst. Los geht’s!

Warum ein Webserver auf dem ESP32?

Der ESP32 ist ein echtes Kraftpaket. Mit seinem integrierten WLAN-Modul ist er wie geschaffen für Projekte im „Internet der Dinge“ (IoT). Ein eigener kleiner Webserver auf dem ESP32 erlaubt es euch, von jedem Gerät in eurem WLAN-Netzwerk – sei es euer Smartphone, Tablet oder Laptop – auf den Mikrocontroller zuzugreifen und ihn zu steuern. Ihr könnt Zustände abfragen, Aktionen auslösen und Daten visualisieren, und das alles über einen einfachen Webbrowser.

Teil 1: „Hallo Welt!“ – Dein erster ESP32 Webserver

Wie bei jedem guten Programmier-Tutorial starten wir auch hier mit den absoluten Grundlagen. Unser erstes Ziel: eine Webseite

zu erstellen, die uns im Browser ein freundliches „Hallo Welt von deinem ESP32!“ anzeigt.

Was du hierfür brauchst:

- Ein ESP32-Entwicklungsboard (z.B. DevKit oder Arduino ESP32)
- Ein passendes USB-Kabel
- Die Arduino IDE, bereits für den ESP32 eingerichtet.

Der Sketch

Da du für diesen ersten Test deines ESP32 Webservers außer dem Microcontroller keine weitere Hardware brauchst, kann es direkt mit dem Code losgehen. Kopiere den folgenden Sketch, **trage im oberen Bereich des Sketchs deine WLAN-Zugangsdaten ein** und lade ihn auf deinen ESP32:

```
//ESP32 Webserver
//polluxlabs.net

#include <WiFi.h>

// Deine WLAN-Zugangsdaten
const char* ssid = "DEIN_WLAN_NAME";
const char* password = "DEIN_WLAN_PASSWORT";

// Wir erstellen ein Server-Objekt auf Port 80 (Standard für HTTP)
WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  while (!Serial) { ; }

  // Mit dem WLAN verbinden
  Serial.println();
```

```
  Serial.print("Verbinde mit ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WLAN verbunden.");
  Serial.println("IP-Adresse: ");
  Serial.println(WiFi.localIP());

  server.begin();
}

void loop() {
   WiFiClient client = server.available();    // Auf neue
Clients warten

  if (client) {                              // Wenn sich ein
Client verbindet...
    Serial.println("Neuer Client verbunden.");
    String currentLine = "";                 // String, um die
ankommenden Daten vom Client zu speichern
    while (client.connected()) {             // solange der
Client verbunden ist...
      if (client.available()) {             // wenn der Client
Daten sendet...
        char c = client.read();              // ...lese ein
Byte
       Serial.write(c);
        if (c == '\n') {                     // wenn das Byte
ein Zeilenumbruch ist...
          // Eine leere Zeile vom Client bedeutet das Ende der
HTTP-Anfrage
          if (currentLine.length() == 0) {
            // HTTP-Header senden
            client.println("HTTP/1.1 200 OK");
```

```
            client.println("Content-type:text/html");
            client.println();

            // Die eigentliche Webseite
                client.print("<h1>Hallo Welt von deinem
ESP32!</h1>");
            // Die HTTP-Antwort endet mit einer leeren Zeile
            client.println();
            break;
          } else {
            currentLine = "";
          }
        } else if (c != '\r') {
          currentLine += c;
        }
      }
    }
    // Verbindung schließen
    client.stop();
    Serial.println("Client getrennt.");
  }
}
```

Was passiert hier?

1. **setup()**: Der ESP32 verbindet sich mit dem von dir angegebenen WLAN. Sobald die Verbindung steht, **gibt er seine IP-Adresse im Seriellen Monitor aus**. Diese Adresse brauchen wir gleich! Anschließend wird der Webserver gestartet.
2. **loop()**: Die Schleife wartet auf eingehende Verbindungen. Sobald du die IP-Adresse in einem Browser öffnest, wird der Browser zu einem „Client“. Der ESP32 erkennt das, sendet einen simplen HTML-Code (<h1>Hallo Welt...</h1>) zurück und schließt die Verbindung wieder.

Kopiere dir die IP-Adresse deines Webservers und öffne sie in einem Browser deiner Wahl. Du solltest nun die folgende (sehr einfache) Webseite sehen:

Teil 2: Licht an! – Steuerung einer LED über den Webserver

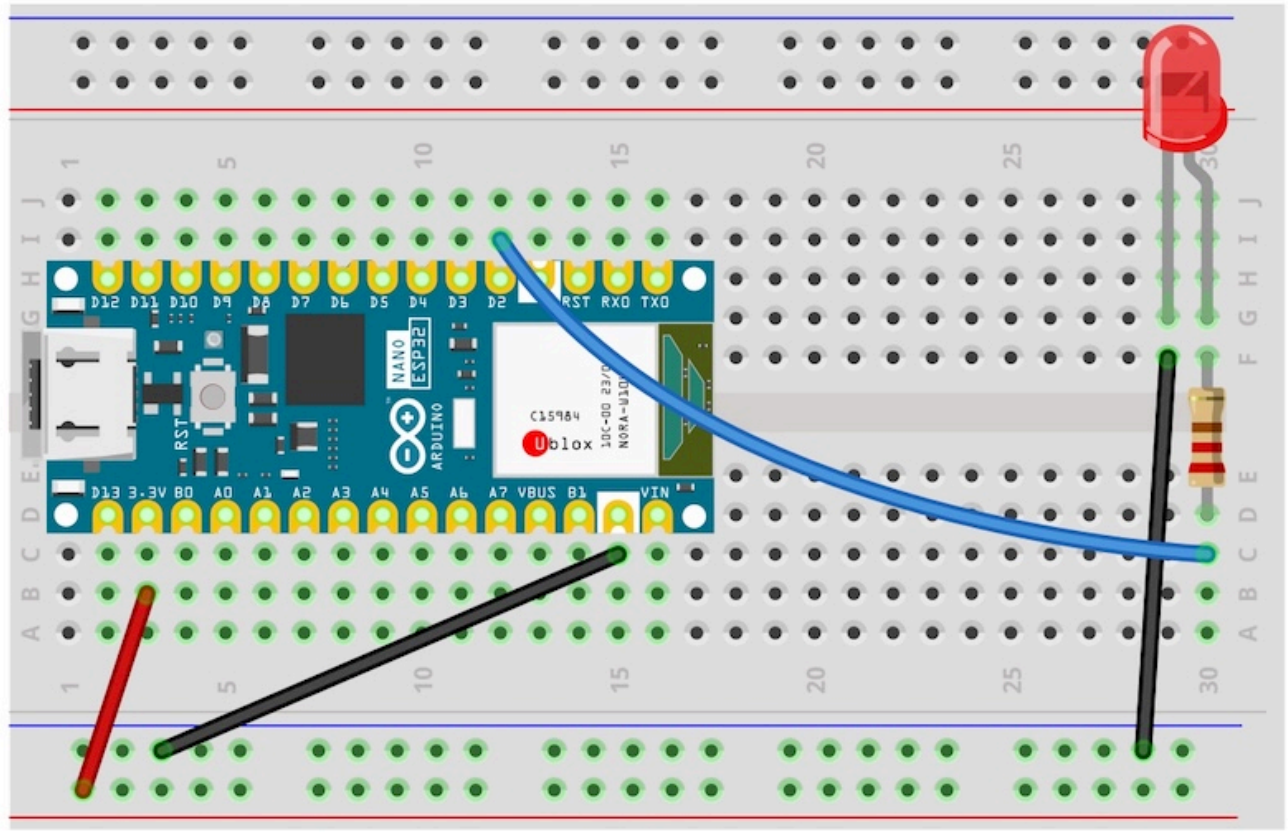
Nachdem die Grundlagen sitzen, wollen wir nun etwas Handfestes steuern. Eine LED ist dafür das perfekte Beispiel. Wir erweitern unseren Code so, dass auf unserer Webseite zwei simple Schaltflächen erscheinen, mit denen wir eine LED ein- und ausschalten können.

Was du zusätzlich brauchst:

- Eine LED
- Ein 220-Ohm-Widerstand
- Ein Breadboard und ein paar Jumperkabel

Der Aufbau auf dem Breadboard

Verbinde die Anode (das lange Beinchen) der LED über den Widerstand mit einem GPIO-Pin des ESP32 (z.B. am Arduino ESP32 der Pin D2). Die Kathode (das kurze Beinchen) verbindest du mit GND.



Der Sketch

Du baust auf deinen ersten Sketch auf. Die größten Änderungen finden in der loop()-Funktion statt, wo wir die Anfrage des Browsers auswerten müssen.

___STEADY_PAYWALL___

```
//ESP32 Webserver mit LED
//polluxlabs.net

#include <WiFi.h>
```

```
// Deine WLAN-Zugangsdaten
const char* ssid = "DEIN_WLAN_NAME";
const char* password = "DEIN_WLAN_PASSWORT";

WiFiServer server(80);

// Pin für die LED definieren
const int ledPin = D2; //Am Arduino ESP32 Pin D2
String ledStatus = "aus";

void setup() {
  Serial.begin(115200);
  while (!Serial) { ; }

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

  // ... (WLAN-Verbindungscode aus Teil 1)
  // Mit dem WLAN verbinden
  Serial.println();
  Serial.print("Verbinde mit ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WLAN verbunden.");
  Serial.println("IP-Adresse: ");
  Serial.println(WiFi.localIP());

  server.begin();
}

void loop() {
  WiFiClient client = server.available();
```

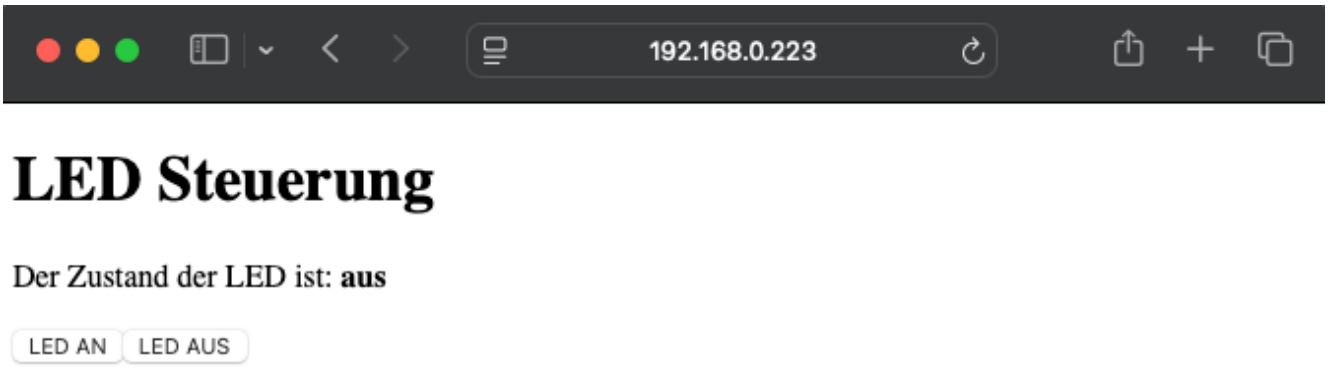
```
  if (client) {
    Serial.println("Neuer Client verbunden.");
    String header = ""; // Speichert den Header der Anfrage
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        header += c;
        if (c == '\n') {
          if (header.indexOf("GET /led/an") >= 0) {
            digitalWrite(ledPin, HIGH);
            ledStatus = "an";
          } else if (header.indexOf("GET /led/aus") >= 0) {
            digitalWrite(ledPin, LOW);
            ledStatus = "aus";
          }
          if (header.endsWith("\r\n\r\n")) {
            // HTTP-Header senden
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();

            // Die Webseite mit Schaltflächen
                                client.print("<!DOCTYPE
html><html><head><title>ESP32                          LED
Steuerung</title></head><body>");
            client.print("<h1>LED Steuerung</h1>");
               client.print("<p>Der Zustand der LED ist:
<strong>" + ledStatus + "</strong></p>");
              client.print("<a href=\"/led/an\"><button>LED
AN</button></a>");
             client.print("<a href=\"/led/aus\"><button>LED
AUS</button></a>");
            client.print("</body></html>");
            client.println();
            break;
          }
        }
      }
    }
    client.stop();
    Serial.println("Client getrennt.");
```

```
  }
}
```

Was ist neu?

- Wir lesen den kompletten „Header“ der Anfrage vom Browser.
- Wir suchen im Header nach bestimmten URLs, die wir selbst definieren: /led/an und /led/aus.
- Je nachdem, welche URL aufgerufen wird (indem ihr auf den entsprechenden Button klickt), schalten wir die LED mit digitalWrite() ein oder aus.
- Die Webseite selbst enthält nun klickbare Links (<a>-Tags), die genau diese URLs aufrufen.

Lade den neuen Code hoch und rufe die IP-Adresse auf. Du kannst jetzt die LED über die Webseite steuern:



Teil 3: Temperaturanzeige mit einem

DHT11

Jetzt wird es richtig spannend! Wir lesen einen Sensor aus und zeigen die Daten live auf unserer Webseite an. Dafür nutzen wir den beliebten und günstigen DHT11 Temperatur- und Feuchtigkeitssensor. Auf Pollux Labs findest du jedoch auch Anleitungen für den Anschluss anderer Temperatursensoren wie den DHT22 oder den BMP180.

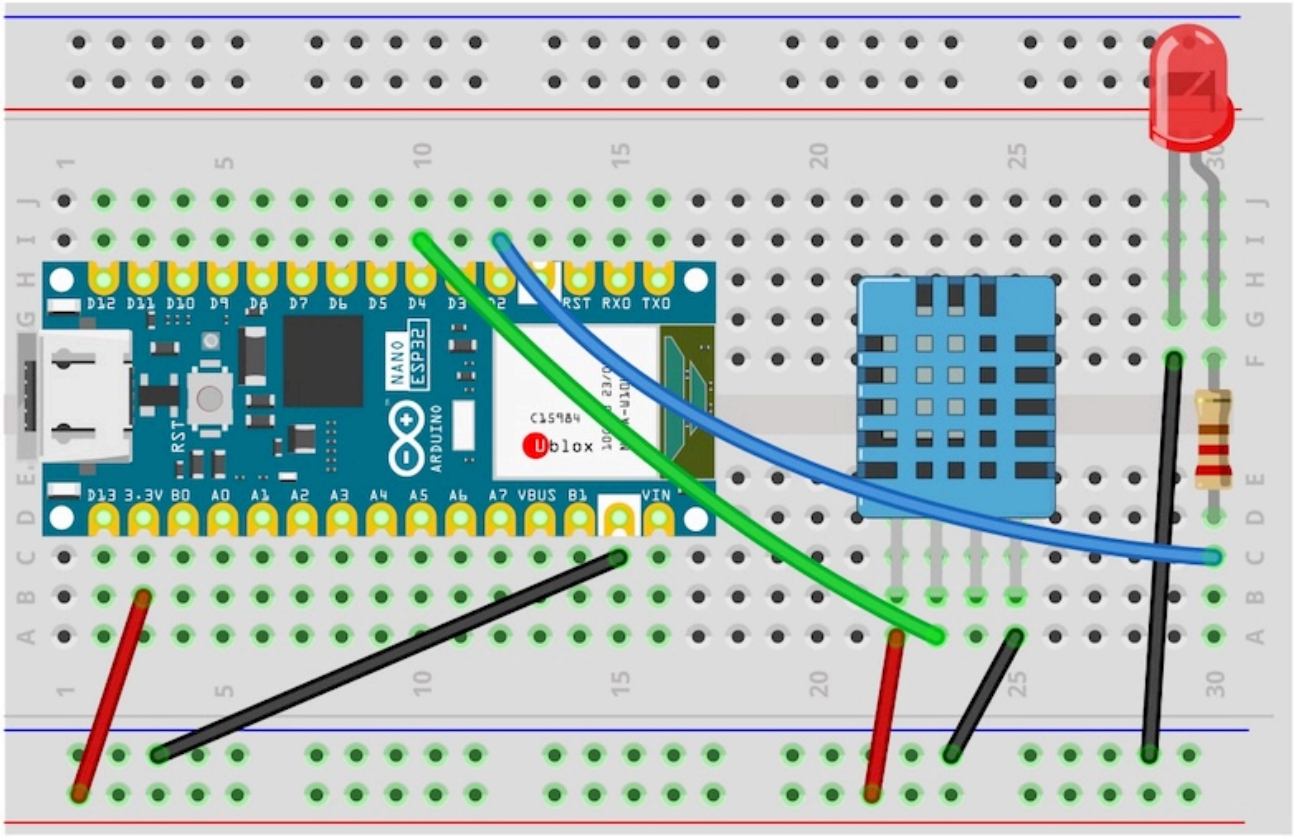
Was du zusätzlich brauchst:

- Einen DHT11-Sensor
- Die Adafruit DHT Sensor Library und die Adafruit Unified Sensor Library. Beide findest du im Bibliotheksmanager der Arduino IDE.

Der Aufbau auf dem Breadboard

Der DHT11 hat meistens vier Pins, von denen jedoch nur drei verwendet werden: VCC (oder +), DATA und GND. Orientiere dich beim Aufbau an folgender Skizze. Die LED kannst du auf dem Breadboard lassen, im nächsten Teil kombinieren wir die LED-Steuerung und die Temperaturmessung für deinen ESP32 Webserver.

- Verbinde VCC mit dem 3.3V-Pin des ESP32.
- Verbinde GND mit der Erde.
- Verbinde den DATA-Pin mit einem digitalen Pin, zum Beispiel Pin D4.



Der Sketch

Hier der Sketch, mit dem dein ESP32 die Temperatur misst und auf der Webseite anzeigt.

```
//ESP32 Webserver mit Temperaturmessung
//polluxlabs.net

#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

// Deine WLAN-Zugangsdaten
const char* ssid = "DEIN_WLAN_NAME";
const char* password = "DEIN_WLAN_PASSWORT";

WiFiServer server(80);

// DHT Sensor Konfiguration
#define DHTPIN D4       // Pin, an dem der DHT11 angeschlossen
```

```
ist
#define DHTTYPE DHT11  // Sensortyp
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  while (!Serial) { ; }
  // ... (WLAN-Verbindungscode aus Teil 1)
  // Mit dem WLAN verbinden
  Serial.println();
  Serial.print("Verbinde mit ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WLAN verbunden.");
  Serial.println("IP-Adresse: ");
  Serial.println(WiFi.localIP());

  dht.begin();
  server.begin();
}

void loop() {
  WiFiClient client = server.available();

  if (client) {
    Serial.println("Neuer Client verbunden.");
    String header = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        header += c;
        if (c == '\n' && header.endsWith("\r\n\r\n")) {
          // Temperatur auslesen
```

```
          float temperatur = dht.readTemperature();

          // Prüfen, ob das Auslesen erfolgreich war
          if (isnan(temperatur)) {
              Serial.println("Fehler beim Auslesen des DHT-
Sensors!");
            temperatur = 0; // Standardwert bei Fehler
          }

          // HTTP-Header senden
          client.println("HTTP/1.1 200 OK");
               client.println("Content-type:text/html;
charset=UTF-8"); // Wichtig für Umlaute wie °
          client.println();

          // Die Webseite mit Temperaturanzeige
                                client.print("<!DOCTYPE
html><html><head><title>ESP32 Wetterstation</title>");
              client.print("<meta http-equiv='refresh'
content='10'>"); // Seite alle 10s neu laden
          client.print("</head><body>");
          client.print("<h1>ESP32 Mini-Wetterstation</h1>");
           client.print("<p>Aktuelle Temperatur: <strong>" +
String(temperatur) + " &deg;C</strong></p>");
          client.print("</body></html>");
          client.println();
          break;
        }
      }
    }
    client.stop();
    Serial.println("Client getrennt.");
  }
}
```

Das ist neu

- Wir binden die DHT.h-Bibliothek ein und initialisieren

den Sensor.
- In der loop() lesen wir bei jeder Anfrage die Temperatur mit dht.readTemperature() aus.
- Wir haben einen kleinen meta-Tag in unser HTML eingefügt, der den Browser anweist, **die Seite alle 10 Sekunden automatisch neu zu laden**. So siehst du immer die aktuellen Werte!
- Die Temperatur wird dann als Teil der Webseite an den Browser gesendet.

Teil 4: LED-Steuerung und Temperatur zusammen

Wir haben gelernt, wie man eine LED schaltet und wie man Sensordaten anzeigt. Jetzt bringen wir beides zusammen! Unser Ziel ist eine einzige Webseite, auf der wir die aktuelle Raumtemperatur sehen und gleichzeitig eine LED ein- und ausschalten können. Das ist wie eine kleine, feine Smart-Home-Zentrale.

Auf dem Breadboard musst du hierfür nichts verändern – du wirst nur den Sketch anpassen:

```
//ESP32 Webserver mit LED-Steuerung und Temperaturmessung
//polluxlabs.net

#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

// Deine WLAN-Zugangsdaten
const char* ssid = "DEIN_WLAN_NAME";
const char* password = "DEIN_WLAN_PASSWORT";

WiFiServer server(80);
```

```
// --- Konfiguration für die LED ---
const int ledPin = 2;
String ledStatus = "aus";

// --- Konfiguration für den DHT Sensor ---
#define DHTPIN 4       // Pin, an dem der DHT11 angeschlossen
ist
#define DHTTYPE DHT11  // Sensortyp
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  while (!Serial) { ; }
  // --- Initialisierung für die LED ---
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // --- Initialisierung für den DHT Sensor ---
  dht.begin();
  // Mit dem WLAN verbinden (Code aus den vorherigen Teilen)
  Serial.println();
  Serial.print("Verbinde mit ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WLAN verbunden.");
  Serial.println("IP-Adresse: ");
  Serial.println(WiFi.localIP());

  server.begin();
}

void loop() {
  WiFiClient client = server.available();

  if (client) {
    Serial.println("Neuer Client verbunden.");
```

```
    String header = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        header += c;

         // Wenn die Anfrage des Browsers komplett empfangen
wurde
        if (c == '\n' && header.endsWith("\r\n\r\n")) {

          // --- URL auswerten für die LED-Steuerung ---
          if (header.indexOf("GET /led/an") >= 0) {
            digitalWrite(ledPin, HIGH);
            ledStatus = "an";
          } else if (header.indexOf("GET /led/aus") >= 0) {
            digitalWrite(ledPin, LOW);
            ledStatus = "aus";
          }

          // --- Temperatur vom DHT-Sensor auslesen ---
          float temperatur = dht.readTemperature();
          if (isnan(temperatur)) {
              Serial.println("Fehler beim Auslesen des DHT-
Sensors!");
            temperatur = 0; // Standardwert bei Fehler
          }

          // --- HTTP-Antwort & kombinierte Webseite senden --
-
          client.println("HTTP/1.1 200 OK");
               client.println("Content-type:text/html;
charset=UTF-8");
           // Wichtiger Hinweis: Die folgende Zeile sorgt für
einen Auto-Refresh.
          // Das ist super für die Temperatur, setzt aber auch
den angezeigten LED-Status
          // alle 10 Sekunden auf den echten Zustand zurück.
          client.println("Refresh: 10");
          client.println(); // Leere Zeile nach den Headern

          // Die eigentliche Webseite
```

```
                                    client.print("<!DOCTYPE
html><html><head><title>ESP32 Kombi-Steuerung</title>");
           client.print("<style> body {  } button { padding:
10px; font-size: 16px; } </style>");
          client.print("</head><body>");
          client.print("<h1>ESP32 Kombi-Steuerung</h1>");
          // Abschnitt für den Sensor
          client.print("<h2>Mini-Wetterstation</h2>");
           client.print("<p>Aktuelle Temperatur: <strong>" +
String(temperatur) + " &deg;C</strong></p>");

          // Abschnitt für die LED
          client.print("<h2>LED Steuerung</h2>");
           client.print("<p>Der Zustand der LED ist: <strong>"
+ ledStatus + "</strong></p>");
             client.print("<a href=\"/led/an\"><button>LED
AN</button></a>");
            client.print("<a href=\"/led/aus\"><button>LED
AUS</button></a>");

          client.print("</body></html>");
          client.println();
            break; // Schleife verlassen, da die Antwort
gesendet wurde
        }
      }
    }
    client.stop();
    Serial.println("Client getrennt.");
  }
}
```

Was ist neu?

1. **Variablen zusammengeführt:** Wir haben die Deklarationen für den ledPin und den ledStatus ganz oben zu denen für den DHT-Sensor hinzugefügt.
2. **setup() erweitert:** Im setup() wird jetzt nicht nur der

Sensor, sondern auch der LED-Pin initialisiert.

3. **Der loop():** Das ist der wichtigste Teil. Bevor die Webseite zusammengebaut wird, passiert jetzt beides:
    - Zuerst wird wie in Teil 2 geprüft, ob die aufgerufene URL /led/an oder /led/aus enthält, und die LED wird entsprechend geschaltet.
    - *Danach* wird wie in Teil 3 die Temperatur vom Sensor ausgelesen.

4. **HTML kombiniert:** Der HTML-Code, der an den Browser gesendet wird, enthält jetzt einfach beide Abschnitte – die Temperaturanzeige und die Schaltflächen für die LED. Es wurde auch ein paar simple <h2>-Überschriften zur Strukturierung und ein winziges bisschen CSS für die Buttons hinzugefügt.

Wenn du diesen Code nun hochlädst und die IP-Adresse im Browser aufrufst, siehst du eine Webseite, die dir die Temperatur anzeigt und dir gleichzeitig erlaubt, das Licht ein- und auszuschalten.

Teil 5: Der Feinschliff – Eine moderne und responsive Oberfläche

Dein ESP32 Webserver funktioniert also – nur die Webseite könnte etwas moderner sein. Hier kommt etwas CSS (Cascading Style Sheets) zum Einsatz. Die Schaltung bleibt hingegen genau dieselbe wie in Teil 4.

Der finale Code mit schicker Oberfläche: Lade diesen Code auf deinen ESP32. Das Design der Webseite passiert in den langen client.print()-Zeilen, die den HTML- und CSS-Code enthalten.

```
//ESP32 Webserver mit LED-Steuerung und
```

```
Temperaturmessung//polluxlabs.net

#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

// Deine WLAN-Zugangsdaten
const char* ssid = "DEIN_WLAN_NAME";
const char* password = "DEIN_WLAN_PASSWORT";

WiFiServer server(80);

// --- Konfiguration für die LED ---
const int ledPin = 2;

// --- Konfiguration für den DHT Sensor ---
#define DHTPIN 4       // Pin, an dem der DHT11 angeschlossen
ist
#define DHTTYPE DHT11  // Sensortyp
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
  dht.begin();
  // Mit dem WLAN verbinden
  Serial.println();
  Serial.print("Verbinde mit ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WLAN verbunden.");
  Serial.println("IP-Adresse: ");
  Serial.println(WiFi.localIP());

  server.begin();
```

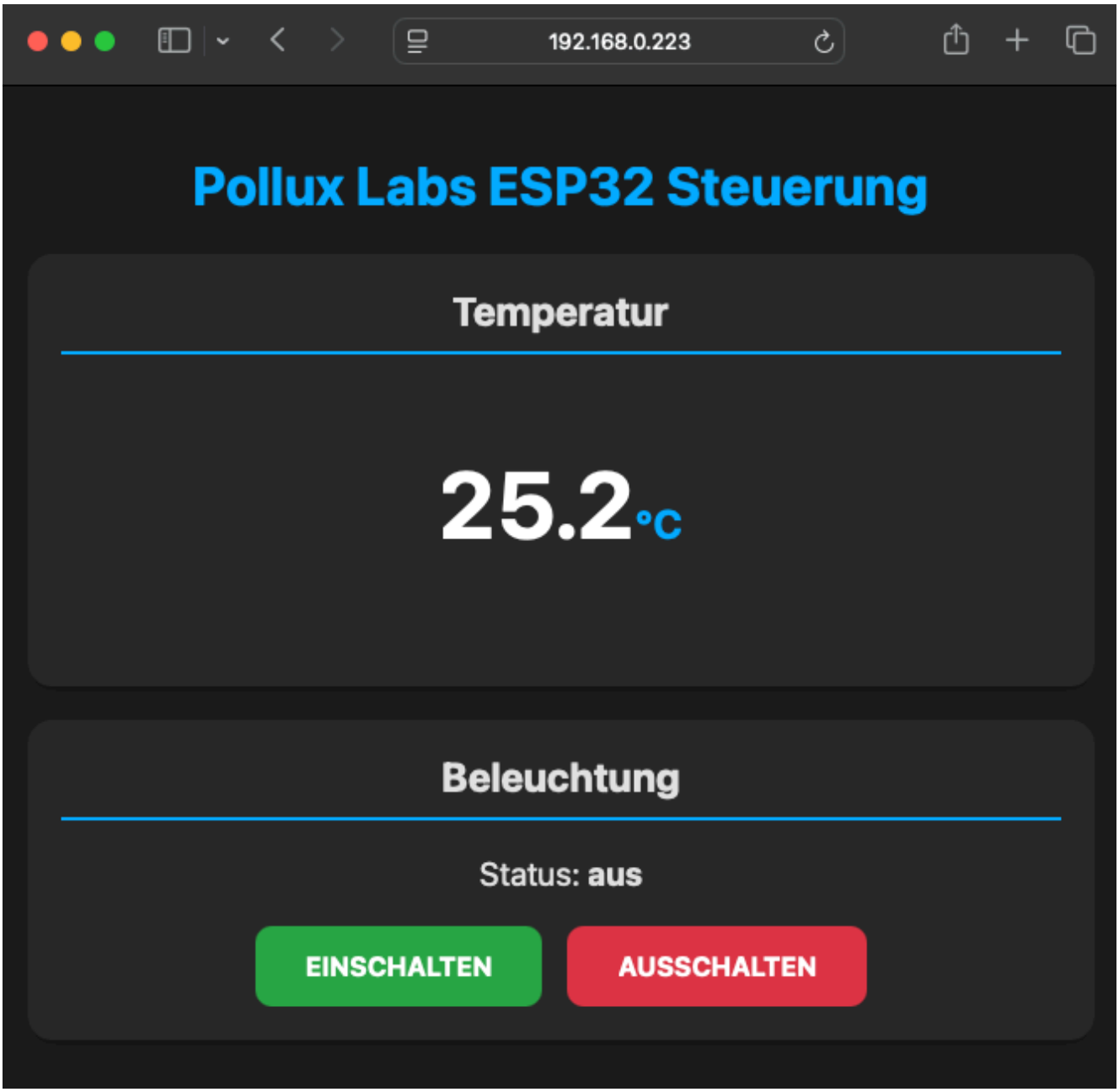
```
}

void loop() {
  WiFiClient client = server.available();

  if (client) {
    String header = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        header += c;

        if (c == '\n' && header.endsWith("\r\n\r\n")) {
          // --- URL auswerten für die LED-Steuerung ---
          if (header.indexOf("GET /led/an") >= 0) {
            digitalWrite(ledPin, HIGH);
          } else if (header.indexOf("GET /led/aus") >= 0) {
            digitalWrite(ledPin, LOW);
          }

          // --- Aktuellen Status von LED und Sensor auslesen
---
           String ledStatus = digitalRead(ledPin) ? "an" :
"aus";
          float temperatur = dht.readTemperature();

          if (isnan(temperatur)) {
             Serial.println("Fehler beim Auslesen des DHT-
Sensors!");
            temperatur = 0.0;
          }

          // --- HTTP-Antwort & moderne Webseite senden ---
          client.println("HTTP/1.1 200 OK");
               client.println("Content-type:text/html;
charset=UTF-8");
          client.println();

          client.print("<!DOCTYPE html>");
          client.print("<html lang='de'>");
          client.print("<head>");
```

```
          client.print("<meta charset='UTF-8'>");
           // Diese Zeile ist ENTSCHEIDEND für die korrekte
Darstellung auf Smartphones!
                  client.print("<meta  name='viewport'
content='width=device-width, initial-scale=1.0'>");
             client.print("<title>Pollux Labs - ESP32
Steuerung</title>");
              client.print("<meta http-equiv='refresh'
content='10'>");
          // --- HIER BEGINNT DAS CSS STYLING ---
          client.print("<style>");
                        client.print("body{background-
color:#1e1e1e;color:#e0e0e0;text-
align:center;margin:0;padding:20px;}");
                         client.print(".container{max-
width:800px;margin:auto;display:flex;flex-wrap:wrap;justify-
content:center;gap:20px;}");
          client.print(".card{background-color:#2a2a2a;border-
radius:15px;padding:20px;box-shadow:0    4px    8px
rgba(0,0,0,0.2);flex-basis:300px;flex-grow:1;}");
          client.print("h1{color:#00aaff;}");
              client.print("h2{border-bottom:2px solid
#00aaff;padding-bottom:10px;margin-top:0;}");
           client.print(".sensor-value{font-size:3.5rem;font-
weight:bold;color:#fff;}");
                               client.print(".unit{font-
size:1.5rem;color:#00aaff;}");
           client.print(".led-status{font-size:1.2rem;margin-
bottom:20px;}");
client.print(".buttons{display:flex;gap:15px;justify-
content:center;}");
                           client.print("a.button{text-
decoration:none;color:#fff;padding:15px    30px;border-
radius:10px;font-weight:bold;transition:transform    0.2s
ease;}");
                  client.print(".on-button{background-
color:#28a745;}"); // Grün
                 client.print(".off-button{background-
color:#dc3545;}"); // Rot
client.print("a.button:active{transform:scale(0.95);}");
          client.print("footer{color:#555;margin-top:40px;}");
```

```
          client.print("</style>");
          // --- HIER ENDET DAS CSS STYLING ---
          client.print("</head>");
          client.print("<body>");
                  client.print("<h1>Pollux Labs ESP32
Steuerung</h1>");
          client.print("<div class='container'>");
          // Karte für den Temperatursensor
          client.print("<div class='card'>");
          client.print("<h2>Temperatur</h2>");
              client.print("<p class='sensor-value'>" +
String(temperatur,         1)         +         "<span
class='unit'>&deg;C</span></p>");
          client.print("</div>");
          // Karte für die LED-Steuerung
          client.print("<div class='card'>");
          client.print("<h2>Beleuchtung</h2>");
             client.print("<p class='led-status'>Status:
<strong>" + ledStatus + "</strong></p>");
          client.print("<div class='buttons'>");
           client.print("<a href='/led/an' class='button on-
button'>EINSCHALTEN</a>");
            client.print("<a href='/led/aus' class='button off-
button'>AUSSCHALTEN</a>");
          client.print("</div>");
          client.print("</div>");
          client.print("</div>"); // Ende .container
            client.print("<footer>Diese Seite wird alle 10
Sekunden automatisch aktualisiert.</footer>");
          client.print("</body></html>");
          client.println();
          break;
        }
      }
    }
    client.stop();
  }
}
```

Das Ergebnis

Wenn du jetzt die IP-Adresse deines ESP32 aufrufst, wirst du mit einer modernen Oberfläche begrüßt, die in etwa so aussieht:



Was ist neu und warum?

- **<meta name='viewport'...>:** Diese eine Zeile im <head> ist der Schlüssel für responsives Design. Sie sagt dem

Smartphone-Browser: „Hey, betrachte die Webseite nicht als winzige Desktop-Seite, sondern passe die Breite an den Bildschirm an und starte ohne Zoom.“

- **.card:** Jede Funktion hat ihre eigene „Karte“. Das schafft eine tolle optische Trennung. Durch display:flex im .container ordnen sich die Karten auf großen Bildschirmen nebeneinander an und auf schmalen Bildschirmen automatisch untereinander.
- **.sensor-value:** Wir geben der reinen Temperaturzahl eine eigene Klasse, um sie mit font-size riesig und fett darzustellen – so sticht der wichtigste Wert sofort ins Auge.
- **Farbige Buttons:** Die Buttons sind jetzt nicht nur größer und rund, sondern auch farbig. Grün für „AN“, Rot für „AUS“. Das ist intuitiv und sieht professionell aus.

Fühl dich frei, mit den Farbwerten (#1e1e1e, #00aaff, etc.) im CSS-Teil zu experimentieren und der Seite deinen ganz persönlichen Anstrich zu geben. Viel Spaß mit deinem neuen, schicken Web-Interface!

Fazit

Herzlichen Glückwunsch! Du hast dich vom einfachen „Hallo Welt“ über die LED-Steuerung bis hin zur Anzeige von Live-Sensordaten hochgearbeitet. Du hast jetzt eine solide Grundlage, um deine eigenen, viel komplexeren IoT-Projekte zu realisieren. Stell dir vor, du könntest die Heizung fernsteuern, die Rollläden automatisieren oder eine E-Mail erhalten, wenn die Pflanzen Wasser braucht – all das ist mit dem Wissen, das du heute gelernt hast, nicht weit entfernt.