

Raspberry Pi Webserver mit Flask



Hier auf Pollux Labs findest du bereits ein [Tutorial für einen ESP8266 Webserver](#) – in diesem Projekt programmierst du jedoch einen Raspberry Pi Webserver. **Dieser wird regelmäßig Messdaten von einem ESP8266 empfangen und auf einer Webseite anzeigen.** Neben einem Raspberry Pi benötigst du für dieses Tutorial nur noch einen Sensor – ich verwende im Folgenden den BMP180, um Temperatur und Luftfeuchtigkeit zu messen.

Außerdem verwende ich das **Python-Modul Flask**, um den Webserver zu programmieren. Damit kannst du sowohl einfache Webseiten erstellen als auch komplexere Applikationen, mit denen du, wie in unserem Fall, Hardware steuern und Messdaten anzeigen kannst.

Eine erste Webseite mit Flask

Bevor du dich einem Webserver widmest, lass uns mit den Grundlagen anfangen. Falls du Flask noch nicht auf deinem Raspberry Pi installiert hast, hole das im Terminal nach:

```
pip install Flask
```

Öffne nun einen Editor (z.B. Visual Studio Code) und erstelle ein Python-Script mit folgendem Inhalt:

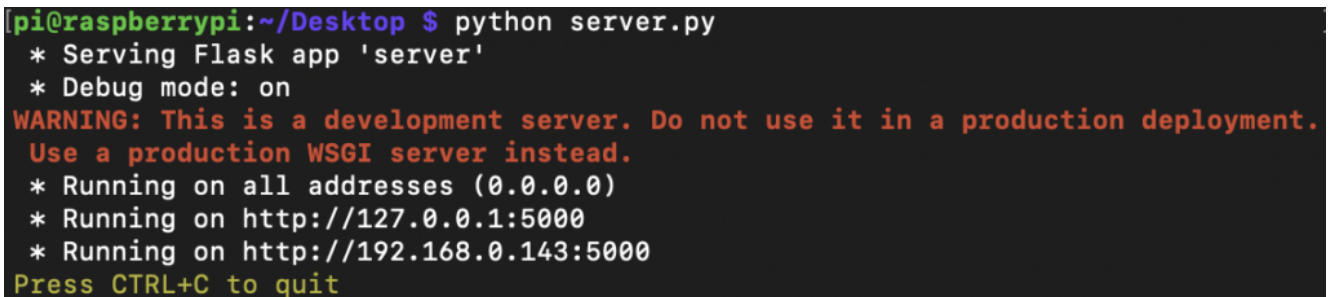
```
from flask import Flask
```

```
app = Flask(__name__)

@app.route('/')
def home():
    return "Hallo, Raspberry Pi!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Speichere es anschließend ab unter einem Namen deiner Wahl und starte es. Solltest du im Terminal daraufhin die Fehlermeldung „Port 5000 is in use by another program.“ erhalten, wähle hinter **port=** eine andere Zahl, zum Beispiel 5010. Sobald das Script ordnungsgemäß auf einem freien Port läuft, siehst du im Terminal die IP-Adresse deines Raspberry Pis. Zum Beispiel: „Running on http://127.0.0.1:5010“

A terminal window on a Raspberry Pi showing the output of running 'python server.py'. The output includes: '* Serving Flask app 'server'', '* Debug mode: on', a red warning message 'WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.', and '* Running on all addresses (0.0.0.0)', '* Running on http://127.0.0.1:5000', and '* Running on http://192.168.0.143:5000'. It ends with 'Press CTRL+C to quit'.

```
pi@raspberrypi:~/Desktop $ python server.py
* Serving Flask app 'server'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.0.143:5000
Press CTRL+C to quit
```

Wenn du die IP-Adresse (127.0.0.1:5010) kopierst und sie im Browser öffnest, solltest du den kleinen Gruß sehen, der im Code hinterlegt ist. Neben dieser IP-Adresse erhältst du auch eine Alternative – hier 192.168.0.143:5000. **Falls sich die erste Adresse nicht öffnen lässt, probiere es einmal mit dieser.**

So funktioniert der code

Lass uns nun einen genaueren Blick auf das Python-Script werfen, um zu verstehen, was dort passiert. Zunächst importierst du die Klasse **Flask** ([Mehr über Flask](#)) aus dem gleichnamigen Modul:

```
from flask import Flask
```

Anschließend erstellst du eine Instanz dieser Klasse namens **app**. Mit (`__name__`) dahinter teilst du Python mit, dass alle etwaigen zusätzlichen Dateien im selben Ordner liegen, wie das Script selbst. Dazu später mehr.

Die folgenden drei Zeilen

```
@app.route('/')
def home():
    return "Hallo, Raspberry Pi!"
```

stellen nun die Webseite bereit – in unserem Fall ist das nur eine einzelne Startseite. Mit dem sogenannten Dekorator **@app.route(, '/')** bestimmst du, was passiert, wenn diese **Startseite** aufgerufen wird – also einfach nur die IP-Adresse. Hierfür dient der Schrägstrich `/`.

Wenn das nun also passiert (so wie du es vorhin im Browser getan hast), dann wird die darauf folgende Funktion **def home():** aufgerufen. Darin befindet sich einfach nur die Anweisung, den String **Hallo, Raspberry Pi!** wiederzugeben – den du dann im Browser siehst. Später folgen nun noch weitere Dekoratoren und Funktionen, die deinem Raspberry Pi Webserver mehr Leben einhauchen werden.

Zuletzt die beiden Zeilen

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Dieser Code wird ausgeführt, wenn wir dieses Skript direkt ausführen (anstatt es als Modul in ein anderes Script zu importieren). In dem Fall startet der Webserver von Flask, der auf Anfragen von jedem Gerät im gleichen WLAN-Netzwerk am Port 5000 reagiert.

Sensordaten empfangen und anzeigen

So ein kleiner Gruß ist ja nett, aber eine wirklich praktische Anwendung ist besser. Wie wäre es, wenn dein Raspberry Pi Webserver regelmäßig Daten von einem ESP8266 empfängt und diese auf einer Webseite anzeigt?

Im Folgenden lernst du, wie du mit dem Sensor BMP180 an einem ESP8266 die **Temperatur und Luftfeuchtigkeit misst und die Messdaten an den Raspberry Pi sendest**. Du selbst kannst die aktuellen Daten dann auf einer Webseite einsehen.



BMP180 Sensordaten

Temperatur: 21.40°C

Luftdruck: 1008.65 hPa

Letzte Aktualisierung: 2024-09-30 10:37:24

Hierfür benötigst du zweimal Code – einmal für den Raspberry Pi Webserver und einen Sketch für den ESP8266.

Das Python-Script für den Server

Zunächst kümmern wir uns um den Server. Damit dein Raspberry Pi die Daten in Empfang nehmen und auf einer Webseite anzeigen kann, ist nicht viel Code nötig. Du benötigst im Prinzip eine Funktion, die die gesendeten Messdaten entgegennimmt und eine

weitere, um diese auf einer Webseite anzuzeigen. Hier das vollständige Script:

```
//Raspberry Pi Webserver
//polluxlabs.net

from flask import Flask, request, render_template_string
from datetime import datetime

app = Flask(__name__)

temperature = 0
pressure = 0
last_update = "Noch keine Daten empfangen"

@app.route('/update-sensor', methods=['POST'])
def update_sensor():
    global temperature, pressure, last_update
    temperature = request.form.get('temp')
    pressure = request.form.get('pressure')
    last_update = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    return "Daten aktualisiert", 200

@app.route('/')
def index():
    html = """
    <!DOCTYPE html>
    <html lang="de">
    <head>
        <meta charset="UTF-8">
        <title>BMP180 Sensordaten</title>
    </head>
    <body>
        <h1>BMP180 Sensordaten</h1>
        <p>Temperatur: {{ temperature }}°C</p>
        <p>Luftdruck: {{ pressure }} hPa</p>
        <p>Letzte Aktualisierung: {{ last_update }}</p>
    </body>
    </html>
    """

    return render_template_string(html,
```

```
temperature=temperature,                pressure=pressure,
last_update=last_update)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Kopiere den obigen Code und speichere ihn in einer Datei, die du z.B. **webserver.py** nennst.

So funktioniert das Script

Ein kurzer Blick auf die Funktionsweise des Raspberry Pi Webservers:

Daten empfangen

Der Teil des Scripts, der neue Daten empfängt, sieht so aus:

```
@app.route('/update-sensor', methods=['POST'])
def update_sensor():
    global temperature, pressure, last_update
    temperature = request.form.get('temp')
    pressure = request.form.get('pressure')
    last_update = datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
    return "Daten aktualisiert", 200
```

- Dieser Code wartet auf neue Messwerte vom Sensor.
- Wenn neue Daten ankommen, werden sie in den Variablen `temperature` und `pressure` gespeichert.
- `last_update` speichert den Zeitpunkt, an dem die Daten ankamen.

Daten anzeigen

Die Webseite, die die Daten anzeigt, wird hier erstellt:

```
@app.route('/')
def index():
```

```

html = """
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>BMP180 Sensordaten</title>
</head>
<body>
    <h1>BMP180 Sensordaten</h1>
    <p>Temperatur: {{ temperature }}°C</p>
    <p>Luftdruck: {{ pressure }} hPa</p>
    <p>Letzte Aktualisierung: {{ last_update }}</p>
</body>
</html>
"""

    return render_template_string(html,
temperature=temperature,          pressure=pressure,
last_update=last_update)

```

- Dieser Code erstellt eine einfache Webseite mithilfe von HTML.
- Die Seite zeigt die aktuelle Temperatur, den Luftdruck und die Zeit der letzten Aktualisierung an.

Webseite einrichten

Der Server wird hier gestartet:

```

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

```

- Diese Zeilen starten den Webserver.
- `host='0.0.0.0'` bedeutet, dass der Server von anderen Geräten im Netzwerk erreichbar ist.
- `port=5000` legt fest, dass die Webseite über Port 5000 erreichbar ist.

So arbeiten alle Teile zusammen: Der Sensor sendet Daten, das Script empfängt und speichert sie, und die Webseite zeigt sie

an.

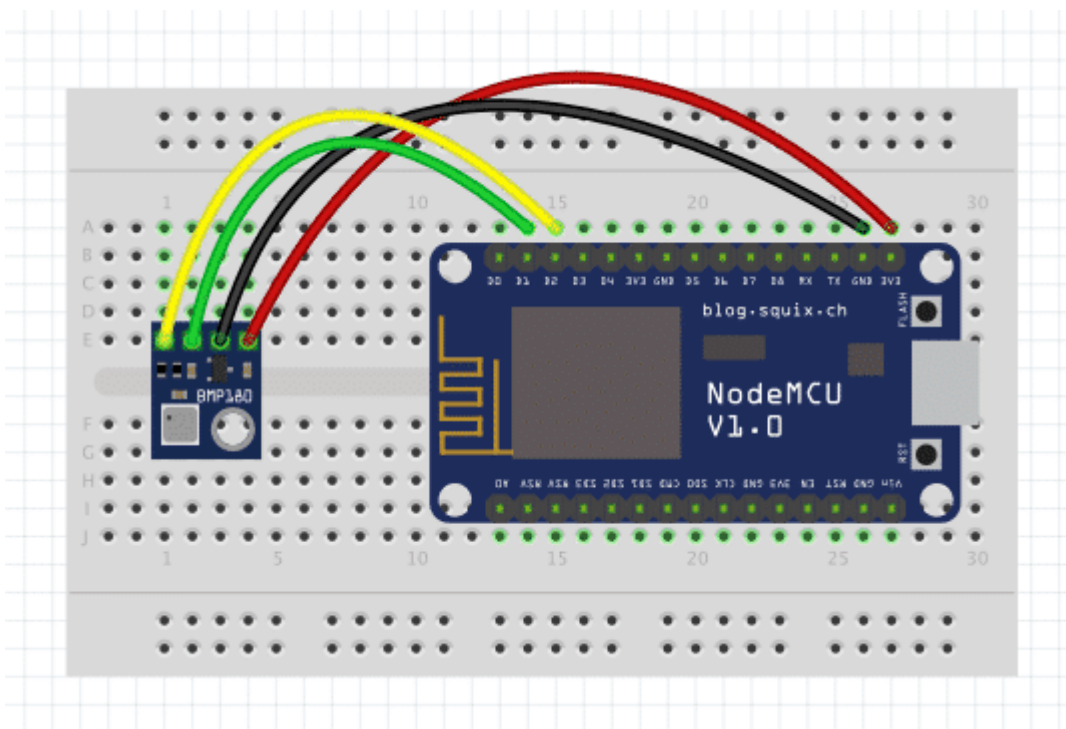
Starte den Raspberry Pi WebServer

Öffne nun auf deinem Raspberry Pi das Terminal und öffne dein Script, wobei du natürlich den Script-Namen verwendest, den du vergeben hast:

```
python3 webserver.py
```

Anschluss des BMP180 und der Sketch für den ESP8266

Nun zum Sender, dem ESP8266, der mit dem Sensor BMP180 die aktuelle Temperatur und Luftfeuchtigkeit misst und an den Raspberry Pi weiterleitet. Bevor wir zum Code kommen, hier eine Skizze, wie du den Sensor am ESP8266 anschließt:



Wenn du den Sensor angeschlossen hast, kann es direkt mit dem

Sketch weitergehen. **Nur ein Hinweis vorab:** Solltest du noch nie ein Projekt mit dem BMP180 gebaut haben, fehlt dir vermutlich noch die zugehörige Bibliothek. Öffne in diesem Fall den Bibliotheksmanager in der Arduino IDE und installiere die Bibliothek **Adafruit BMP085 Library**. Falls du gefragt wirst, ob du weitere benötigte Bibliotheken installieren möchtest, antworte bitte mit Ja.

Doch nun zum Sketch:

```
//Sending data to the Raspberry Pi Webserver
//polluxlabs.net
```

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <Wire.h>
#include <Adafruit_BMP085.h>
```

```
const char* ssid = "DEIN NETZWERK";
const char* password = "DEIN PASSWORT";
const char* serverName = "http://SERVER-IP/update-sensor";
```

```
Adafruit_BMP085 bmp;
```

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  Serial.println("Verbinde mit WLAN");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Verbunden mit IP-Adresse: ");
  Serial.println(WiFi.localIP());

  if (!bmp.begin()) {
    Serial.println("BMP180 nicht gefunden, überprüfen Sie die Verkabelung!");
    while (1) {}
  }
}
```

```

}

void loop() {
  if(WiFi.status() == WL_CONNECTED) {
    WiFiClient client;
    HTTPClient http;

    float temperature = bmp.readTemperature();
    float pressure = bmp.readPressure() / 100.0F;

    // Daten für den POST-Request vorbereiten
    String httpRequestData = "temp=" + String(temperature) +
"&pressure=" + String(pressure);

    // HTTP POST Request senden
    http.begin(client, serverName);
    http.addHeader("Content-Type", "application/x-www-form-
urlencoded");
    int httpResponseCode = http.POST(httpRequestData);

    if (httpResponseCode > 0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
    }
    else {
      Serial.print("Fehler code: ");
      Serial.println(httpResponseCode);
    }
    http.end();
  }
  else {
    Serial.println("WiFi getrennt");
  }

  delay(30000); // Alle 30 Sekunden senden
}

```

So funktioniert der Sketch

Werfen wir nun einen Blick auf die einzelnen Bestandteile des Sketchs.

Einbindung der Bibliotheken

Am Anfang werden verschiedene Bibliotheken eingebunden, die für die WiFi-Verbindung, HTTP-Anfragen und die Kommunikation mit dem BMP180 Sensor benötigt werden.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <Wire.h>
#include <Adafruit_BMP085.h>
```

Konfiguration

Anschließend legst du die WLAN-Zugangsdaten (SSID und Passwort) fest und hinterlegst die Adresse des Raspberry Pi Webservers. Ersetze hierbei **SERVER-IP** zum Beispiel durch **192.168.0.143:5000**.

```
const char* ssid = "NETZWERK";
const char* password = "PASSWORT";
const char* serverName = "http://SERVER-IP/update-sensor";
```

```
Adafruit_BMP085 bmp;
```

Setup-Funktion

Hier stellst du die Verbindung zum WLAN her und initialisierst den Sensor:

```
void setup() {
  Serial.begin(115200);
```

```

WiFi.begin(ssid, password);
Serial.println("Verbinde mit WLAN");
while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.print("Verbunden mit IP-Adresse: ");
Serial.println(WiFi.localIP());

if (!bmp.begin()) {
    Serial.println("BMP180 nicht gefunden, überprüfen Sie die
Verkabelung!");
    while (1) {}
}
}

```

Loop-Funktion

Nun zum Kern, hier prüfst du zunächst, ob die Internetverbindung steht, liest Temperatur und Luftdruck ein und sendest die Messdaten an den Raspberry Pi Webserver per HTTP POST. Anschließend prüfst du die Antwort des Servers, kappst die Verbindung zum WLAN und wartest 30 Sekunden bis zur nächsten Messung.

```

void loop() {
    if(WiFi.status() == WL_CONNECTED) {
        WiFiClient client;
        HTTPClient http;

        float temperature = bmp.readTemperature();
        float pressure = bmp.readPressure() / 100.0F;

        // Daten für den POST-Request vorbereiten
        String httpRequestData = "temp=" + String(temperature) +
"&pressure=" + String(pressure);

        // HTTP POST Request senden
    }
}

```

```

    http.begin(client, serverName);
    http.addHeader("Content-Type", "application/x-www-form-
urlencoded");

    int httpResponseCode = http.POST(httpRequestData);

    // Überprüfung der Antwort
    if (httpResponseCode > 0) {
        Serial.print("HTTP Response code: ");
        Serial.println(httpResponseCode);
    }
    else {
        Serial.print("Fehler code: ");
        Serial.println(httpResponseCode);
    }
    http.end();
}
else {
    Serial.println("WiFi getrennt");
}

delay(30000); // Alle 30 Sekunden senden
}

```

Lade den Sketch nun auf deinen ESP8266. Wenn du nun die IP-Adresse des Raspberry Pi Webserver in einem Browser öffnest, solltest du bald darauf die aktuellen Messdaten des ESP8266 darauf sehen.

Hübsche die Webseite etwas auf

Klappt alles? Dann wäre vielleicht eine etwas ansprechendere Webseite eine gute Idee. Im Prinzip sind dir hier keine Grenzen gesetzt, du kannst hier mit HTML und CSS schalten und walten wie du möchtest. Eine weiteres Layout **inklusive eines Graphen für den Verlauf der Messdaten** könnte z.B. dieses hier sein:

BMP180 Sensordaten Dashboard



Temperatur

22.6

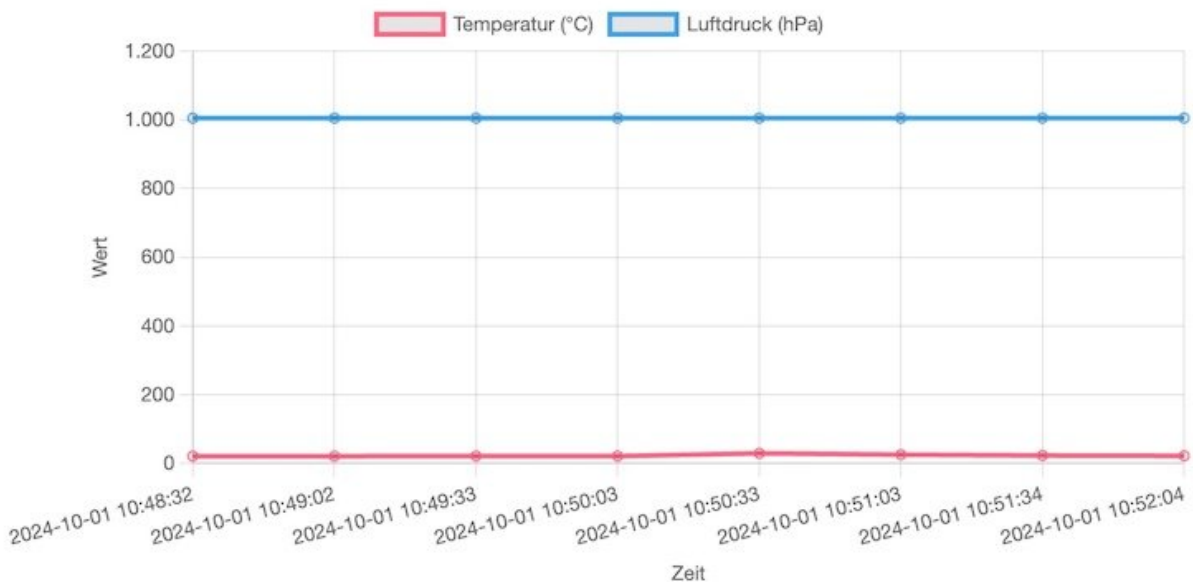
°C



Luftdruck

1004.92

hPa



Letzte Aktualisierung: 2024-10-01 10:52:04

Das zugehörige Python-Script sieht folgendermaßen aus:

```
//Raspberry Pi Webserver  
//polluxlabs.net
```

```
from flask import Flask, request, jsonify,  
render_template_string  
from datetime import datetime  
import json
```

```

app = Flask(__name__)

temperature = 20.0
pressure = 1013.25
last_update = "Noch keine Daten empfangen"
history = []

@app.route('/update-sensor', methods=['POST'])
def update_sensor():
    global temperature, pressure, last_update, history
    temperature = float(request.form.get('temp'))
    pressure = float(request.form.get('pressure'))
    last_update = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    history.append({"time": last_update, "temperature":
temperature, "pressure": pressure})
    if len(history) > 10:
        history.pop(0)
    return "Daten aktualisiert", 200

@app.route('/get-data')
def get_data():
    return jsonify({
        "temperature": temperature,
        "pressure": pressure,
        "last_update": last_update,
        "history": history
    })

@app.route('/')
def index():
    html = """
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>BMP180 Sensordaten Dashboard</title>
    <link
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15
.3/css/all.min.css" rel="stylesheet">

```

```
src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<style>
  :root {
    --primary-color: #3498db;
    --secondary-color: #2c3e50;
    --background-color: #ecf0f1;
    --card-background: #ffffff;
  }
  body {
    line-height: 1.6;
    color: var(--secondary-color);
    background-color: var(--background-color);
    margin: 0;
    padding: 0;
  }
  .container {
    max-width: 1200px;
    margin: 0 auto;
    padding: 20px;
  }
  header {
    background-color: var(--primary-color);
    color: white;
    text-align: center;
    padding: 1rem;
    margin-bottom: 2rem;
  }
  h1 {
    margin: 0;
  }
  .dashboard {
    display: grid;
    grid-template-columns: repeat(auto-fit,
minmax(300px, 1fr));
    gap: 20px;
  }
  .card {
    background-color: var(--card-background);
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  }
```



```

        padding: 20px;
        text-align: center;
    }
    .card-title {
        font-size: 1.2rem;
        color: var(--secondary-color);
        margin-bottom: 10px;
    }
    .card-value {
        font-size: 2.5rem;
        font-weight: bold;
        color: var(--primary-color);
    }
    .card-icon {
        font-size: 3rem;
        margin-bottom: 10px;
        color: var(--primary-color);
    }
    #updateTime {
        text-align: center;
        margin-top: 20px;
        font-style: italic;
    }
    #chart {
        width: 100%;
        height: 300px;
    }
</style>
</head>
<body>
    <header>
        <h1>BMP180 Sensordaten Dashboard</h1>
    </header>
    <div class="container">
        <div class="dashboard">
            <div class="card">
                <i class="fas fa-thermometer-half card-
icon"></i>
                <div class="card-title">Temperatur</div>
                <div class="card-value" id="temperature">--
</div>

```

```

        <div>°C</div>
    </div>
    <div class="card">
        <i class="fas fa-tachometer-alt card-
icon"></i>
        <div class="card-title">Luftdruck</div>
        <div class="card-value" id="pressure">--</div>
        <div>hPa</div>
    </div>
</div>
<div class="card" style="margin-top: 20px;">
    <canvas id="chart"></canvas>
</div>
    <div id="updateTime">Letzte Aktualisierung: <span
id="lastUpdate">--</span></div>
</div>

<script>
    let chart;

    function updateData() {
        fetch('/get-data')
            .then(response => response.json())
            .then(data => {
document.getElementById('temperature').textContent    =
data.temperature.toFixed(1);
document.getElementById('pressure').textContent        =
data.pressure.toFixed(2);
document.getElementById('lastUpdate').textContent      =
data.last_update;
                updateChart(data.history);
            });
    }

    function updateChart(history) {
        const ctx =
document.getElementById('chart').getContext('2d');
        if (chart) {
            chart.destroy();
        }
        chart = new Chart(ctx, {

```

```

        type: 'line',
        data: {
            labels: history.map(entry => entry.time),
            datasets: [{
                label: 'Temperatur (°C)',
                data: history.map(entry =>
entry.temperature),
                borderColor: 'rgb(255, 99, 132)',
                tension: 0.1
            }, {
                label: 'Luftdruck (hPa)',
                data: history.map(entry =>
entry.pressure),
                borderColor: 'rgb(54, 162, 235)',
                tension: 0.1
            }]
        },
        options: {
            responsive: true,
            scales: {
                x: {
                    display: true,
                    title: {
                        display: true,
                        text: 'Zeit'
                    }
                },
                y: {
                    display: true,
                    title: {
                        display: true,
                        text: 'Wert'
                    }
                }
            }
        }
    });
}

```

```

// Initialer Datenabruf
updateData();

```

```

        // Aktualisiere Daten alle 30 Sekunden
        setInterval(updateData, 30000);
    </script>
</body>
</html>
"""
    return render_template_string(html)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)

```

Wie geht es weiter?

Du hast nun einen Raspberry Pi Webserver, der Daten empfangen und auf einer Webseite visualisieren kann. Der nächste Schritt wäre eine Möglichkeit, über diese Webseite auch Geräte zu steuern, die wiederum z.B. an einem ESP8266 hängen und darüber gesteuert werden.