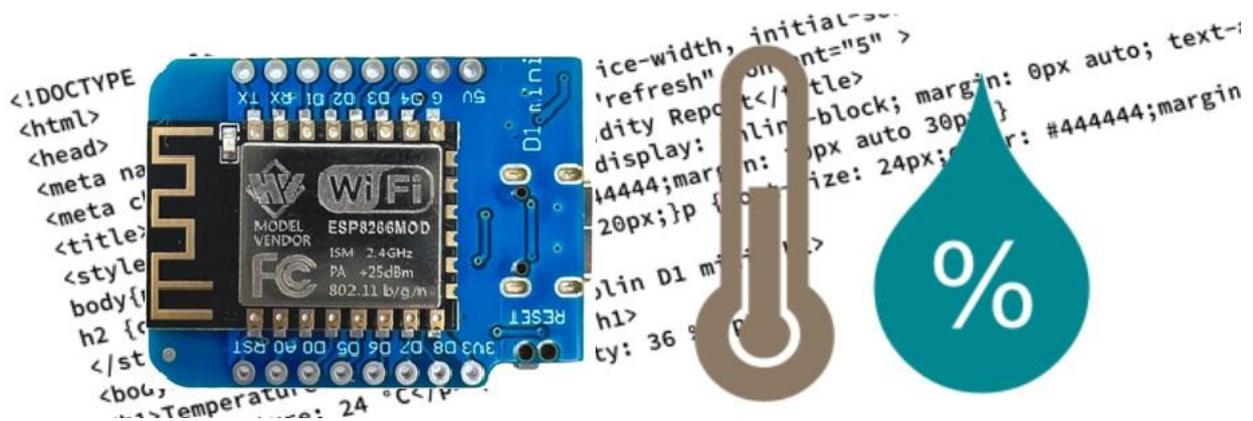


# Arduino Projekt

## ESP8266-D1mini HTU OLED



Klasse	
Fach/Gegenstand	Arduino
Lehrer / Ersteller	G.Ehrenberger
Datum	Juli 2025

## Inhaltsverzeichnis

<b>1. Vorwort .....</b>	<b>6</b>
<b>2. Aufgabenstellung .....</b>	<b>6</b>
2.1. Funktionsbeschreibung .....	6
2.2. Quellenbeschreibung .....	7
<b>3. Schaltungsaufbau.....</b>	<b>7</b>
3.1. Bauteilliste.....	7
3.2. Bauteilbeschreibung .....	7
3.2.1. ESP8266 Wemos D1 mini .....	7
3.2.2. HTU21, I2C Temperatur, Luftfeuchtigkeit .....	8
3.2.3. OLED 0,96“ I2C Display 128x64 Pixel, monochrom.....	8
3.3. Schaltplan.....	9
3.4. Aufbau .....	9
<b>4. System-Start.....</b>	<b>10</b>
4.1. Software - Installation .....	10
4.1.1. Treiberinstallation testen.....	10
4.1.2. Arduino IDE download.....	10
4.1.3. Zusätzliche hilfreiche Software .....	11
4.2. Software – Start und Einstellungen .....	11
<b>5. System-Check .....</b>	<b>16</b>
5.1. Hardware – Check .....	16
5.2. Beispielprogramm - Check .....	16
5.3. Aufgabe .....	17
<b>6. Sensor HTU21 Test.....</b>	<b>19</b>
6.1. Funktionsbeschreibung .....	19
6.2. Neuer Programmcode .....	19
6.3. Bibliothek einbinden/installieren .....	19
6.4. Programmcode Ausgabe .....	21
6.5. Fehlerbehandlung.....	21
6.6. Aufgabe .....	21
<b>7. Anzeige OLED 0,96“ Test .....</b>	<b>22</b>
7.1. Funktionsbeschreibung .....	22
7.2. Neuer Programmcode .....	22
7.3. Bibliothek einbinden/installieren .....	22
7.4. Programmcode Ausgabe .....	23
7.5. Fehlerbehandlung.....	23
7.6. Aufgabe .....	23
<b>8. HTU21 – Sensor mit OLED-Ausgabe.....</b>	<b>24</b>
8.1. Funktionsbeschreibung .....	24
8.2. Neuer Programmcode .....	24

8.3.	Bibliothek einbinden/installieren .....	24
8.4.	Programmcode Ausgabe .....	25
8.5.	Fehlerbehandlung.....	25
8.6.	Aufgabe .....	25
<b>9.</b>	<b>Datum und Uhrzeit vom Internet per NTP .....</b>	<b>26</b>
9.1.	Funktionsbeschreibung .....	26
9.2.	Neuer Programmcode .....	26
9.3.	Bibliothek einbinden/installieren .....	26
9.4.	Programmcode Ausgabe .....	26
9.5.	Fehlerbehandlung.....	27
9.6.	Aufgabe .....	27
<b>10.</b>	<b>NTP-Zeit mit OLED-Ausgabe .....</b>	<b>28</b>
10.1.	Funktionsbeschreibung .....	28
10.2.	Neuer Programmcode .....	28
10.3.	Bibliothek einbinden/installieren .....	28
10.4.	Programmcode Ausgabe .....	29
10.5.	Fehlerbehandlung.....	29
10.6.	Aufgabe .....	29
<b>11.</b>	<b>ESP8266 und WebServer .....</b>	<b>30</b>
11.1.	WebServer und WiFi-Manager.....	30
11.2.	WebServer - Beschreibung von DeepSeek:.....	30
<b>12.</b>	<b>Internet of Things (IoT) WiFi-Manager .....</b>	<b>34</b>
12.1.	Funktionsbeschreibung .....	34
12.2.	Neuer Programmcode .....	34
12.3.	Bibliothek einbinden/installieren .....	34
12.4.	Programmcode Ausgabe .....	34
12.5.	Fehlerbehandlung.....	36
12.6.	Aufgabe .....	36
<b>13.</b>	<b>HTU – OLED – NTP - WiFi-Manager.....</b>	<b>37</b>
13.1.	Funktionsbeschreibung .....	37
13.2.	Neuer Programmcode .....	37
13.3.	Bibliothek einbinden/installieren .....	37
13.4.	Programmcode Ausgabe .....	38
13.5.	Fehlerbehandlung.....	38
13.6.	Aufgabe .....	38
<b>14.</b>	<b>HTU mit MQTT-Broker Verbindung .....</b>	<b>39</b>
14.1.	Funktionsbeschreibung .....	39
14.2.	Neuer Programmcode .....	39
14.3.	Bibliothek einbinden/installieren .....	39
14.4.	Programmcode Ausgabe .....	40
14.5.	Fehlerbehandlung.....	40
14.6.	Aufgabe .....	41

<b>15. HTU mit MQTT-Broker Auslagerung .....</b>	<b>42</b>
15.1. Funktionsbeschreibung .....	42
15.2. Neuer Programmcode .....	42
15.3. Bibliothek einbinden/installieren .....	42
15.4. Programmcode Ausgabe .....	42
15.5. Fehlerbehandlung.....	43
15.6. Aufgabe .....	43
<b>16. OLED mit MQTT-Broker und WiFi-Manager mit Erweiterung.....</b>	<b>44</b>
16.1. Funktionsbeschreibung .....	44
16.2. Neuer Programmcode .....	45
16.3. Bibliothek einbinden/installieren .....	45
16.4. Programmcode Ausgabe .....	46
16.5. Fehlerbehandlung.....	47
16.6. Aufgabe .....	47
16.6.1. Siehe 13_HTU_OLED_NTP_WiFi-Manager2.ino .....	47
<b>17. HTU mit Web-Server einfach.....</b>	<b>49</b>
17.1. Webserver Start .....	49
17.2. Funktionsbeschreibung .....	50
17.3. Neuer Programmcode .....	50
17.4. Bibliothek einbinden/installieren .....	50
17.5. Programmcode Ausgabe .....	51
17.6. Fehlerbehandlung.....	52
17.7. Aufgabe .....	52
<b>18. HTU mit Web-Server mit „Gauges“ .....</b>	<b>53</b>
18.1. Funktionsbeschreibung .....	53
18.2. Neuer Programmcode .....	53
18.3. Bibliothek einbinden/installieren .....	54
18.4. Programmcode Ausgabe .....	56
18.5. Fehlerbehandlung.....	56
18.6. Aufgabe .....	56
<b>19. HTU mit Web-Server + WebSocket (in Arbeit) .....</b>	<b>57</b>
19.1. Funktionsbeschreibung .....	57
19.2. Neuer Programmcode .....	57
19.3. Bibliothek einbinden/installieren .....	58
19.4. Programmcode Ausgabe .....	58
19.5. Fehlerbehandlung.....	58
19.6. Aufgabe .....	58
<b>20. HTU mit Async-Web-Server und WiFi-Manager (in Arbeit) .....</b>	<b>59</b>
20.1. Funktionsbeschreibung .....	59
20.2. Neuer Programmcode .....	59
20.3. Bibliothek einbinden/installieren .....	60
20.4. Programmcode Ausgabe .....	60

20.5. Fehlerbehandlung.....	60
20.6. Aufgabe .....	60
<b>21. Vorlage (in Arbeit).....</b>	<b>61</b>
21.1. Funktionsbeschreibung .....	61
21.2. Neuer Programmcode .....	61
21.3. Bibliothek einbinden/installieren .....	61
21.4. Programmcode Ausgabe .....	61
21.5. Fehlerbehandlung.....	61
21.6. Aufgabe .....	61
<b>22. Over The Air update (OTA) (in Arbeit).....</b>	<b>62</b>
22.1. Funktionsbeschreibung .....	62
22.2. Neuer Programmcode .....	62
22.3. Bibliothek einbinden/installieren .....	62
22.4. Programmcode Ausgabe .....	62
22.5. Fehlerbehandlung.....	62
22.6. Aufgabe .....	62
<b>23. Vorschau .....</b>	<b>63</b>
<b>24. Anhang: .....</b>	<b>64</b>
24.1. Abbildungsverzeichnis .....	64

## 1. Vorwort

**Es gibt beim Schaltungsbau und beim Programmieren keine Abkürzung.**

**Selbermachen ist der Schlüssel zum Erfolg, um mit den eigenen Fehlern zu lernen.**

**Geduld und Ausdauer sind notwendig.**

**Der Weg ist das Ziel, dann wird es Spaß machen, sich der Herausforderung zu stellen und zu meistern.**

**Viel Erfolg mit folgenden Projekten.**

Beachte bitte, dass sich in dieser Beschreibung trotz sorgfältiger Arbeit auch noch Fehler befinden könnten.

## 2. Aufgabenstellung

Es soll ein Projekt mit einem ESP8266 (ESP D1-mini) einem Display (OLED) und einem Sensor (HTU) aufgebaut werden. Bei dieser schrittweisen Anleitung soll die Vorgehensweise bei komplexen Projekten herausgearbeitet werden.

Grundlagenkenntnisse in der Programmierung eines Arduino mit der Arduino IDE werden benötigt. Ebenso sind Strategien zur Fehlersuche von Vorteil. Das umfassendste Starterkit für Arduino UNO R3 Mega2560 ist ein sehr guter Einstieg.

### 2.1. Funktionsbeschreibung

Ein HTU21 Temperatur- und Luftfeuchte-Sensor wird von einem ESP8266 D1-mini ausgelesen und die Anzeige erfolgt auf einem OLED 0,96“.

- Sensor testen mit Ausgabe auf dem Seriellen Monitor
- Sensor auslesen und Anzeige am OLED
- ... mit Real-Time-Clock DS1307
- ... mit WLAN (WiFi)
- ... mit Uhrzeit (NTP) aus dem Internet
- ... mit WLAN + WiFi-Manager
- ... mit Datenaustausch über MQTT
- ... mit WLAN + MQTT + WiFi-Manager
- ... mit Web-Server für Sensordaten
- ... mit WLAN + Web-Server + WiFi-Manager
- ... mit WLAN + Web-Server + MQTT + WiFi-Manager

### 2.2. Quellenbeschreibung

- Für die Grundlagen und Einzelaufgaben:
  - „The Most Complete Starter Kit for Mega V1.0.2021.05.13-Deutsch“
  - [www.randomnerdtutorial.com](http://www.randomnerdtutorial.com)
- Für die Kombinationen und Details:
  - DeepSeek
  - [www.randomnerdtutorial.com](http://www.randomnerdtutorial.com)
- Für die Bauteile:
  - [AZ-Delivery.de](http://AZ-Delivery.de)

## 3. Schaltungsaufbau

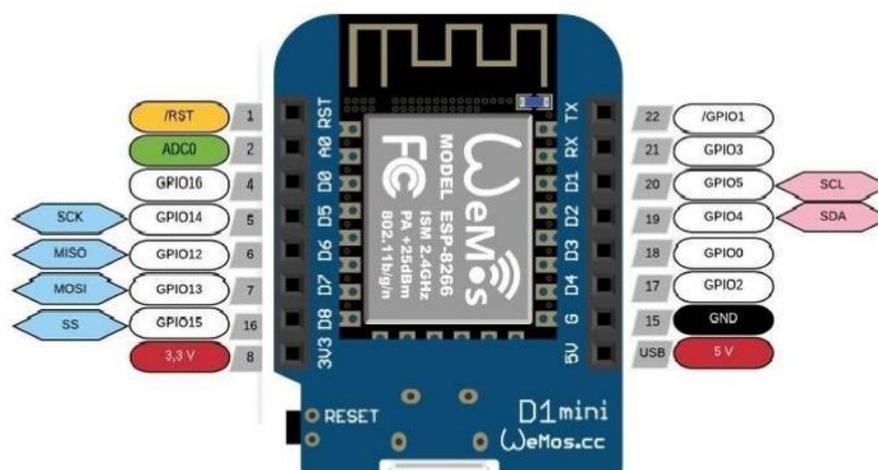
### 3.1. Bauteilliste

- Grundausrüstung für das Elektronik Basteln (Steckboard, Widerstände, LED, Steck-Draht, ...)
- NodeMCU: ESP8266 Wemos D1 mini
- Sensor: HTU21, I2C Temperatur, Luftfeuchtigkeit
- OLED 0,96“ I2C Display 128x64 Pixel, monochrom

### 3.2. Bauteilbeschreibung

#### 3.2.1. ESP8266 Wemos D1 mini

##### Pin-Belegung D1 mini



**Achtung: ESP ist nur 3,3V verträglich.**

Abbildung 1: ESP8266 Wemos D1-mini

## Schaltungsaufbau

Datenblätter von AZ-Delivery.de herunterladen und beachten.

- Stromversorgung über USB-micro Stecker
- 9 digitale I/O Pins (maximal 3,3V!)
- 1 analoger I/O Pins (maximal 3,0V!)
- ESP8266 Variante ESP-12F
- USB-Schnittstelle CH340G

### 3.2.2. HTU21, I2C Temperatur, Luftfeuchtigkeit

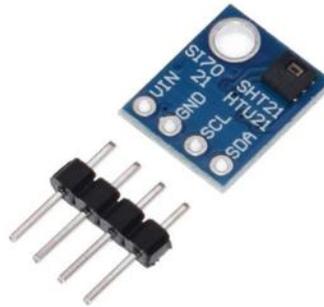


Abbildung 2: HTU21 Temperatur- und Feuchte-Sensor

- Stromversorgung 3,3V bis 5V
- Schnittstelle i2c: Vin, GND, SCL, SDA
- Typische Messgenauigkeit Temperatur: +/-1°C Abweichung zwischen -30°C und 90°C
- Typische Messgenauigkeit Feuchtigkeit: +/-2% RL zwischen 5% und 95% RL

### 3.2.3. OLED 0,96" I2C Display 128x64 Pixel, monochrom

- Stromversorgung 3,3V bis 5V (0,04W)
- Schnittstelle i2c: Vin, GND, SCL, SDA
- Auflösung 128 x 64 Pixel
- Chipsatz SSD1306



Abbildung 3: OLED 0,96" i2c monochrom

### 3.3. Schaltplan

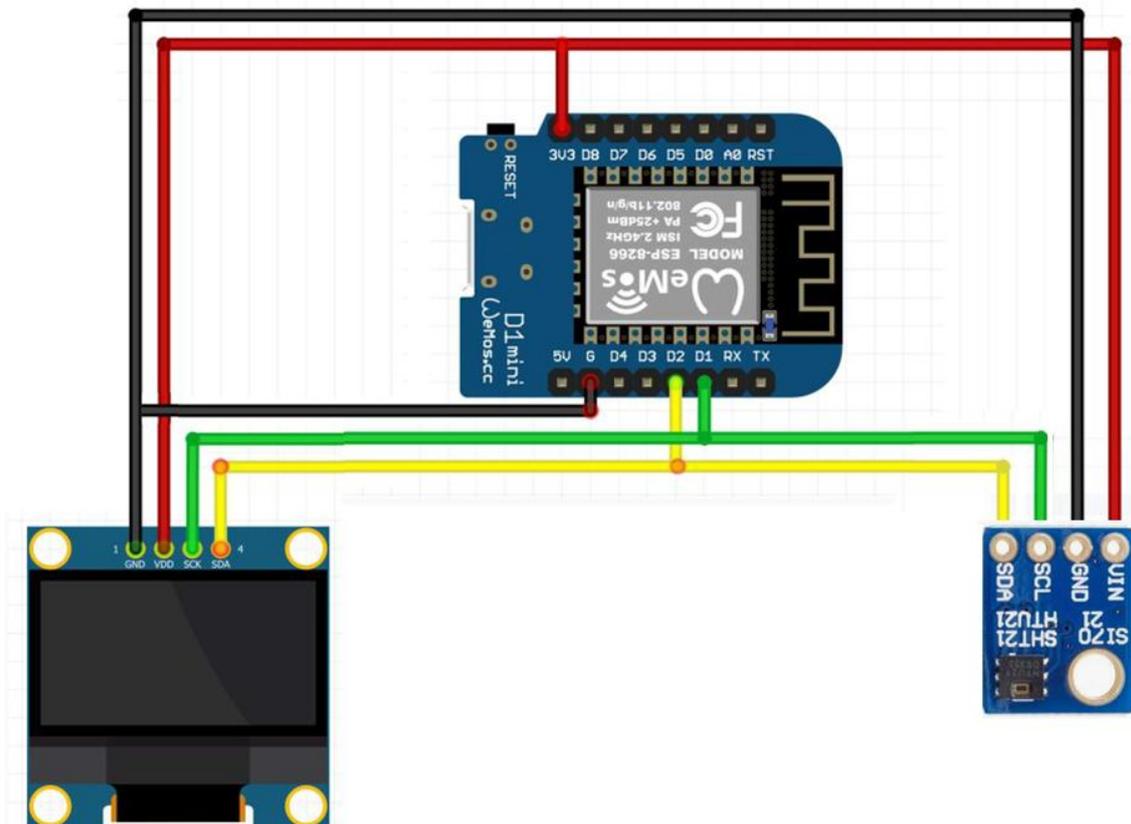


Abbildung 4: Schaltplan

### 3.4. Aufbau

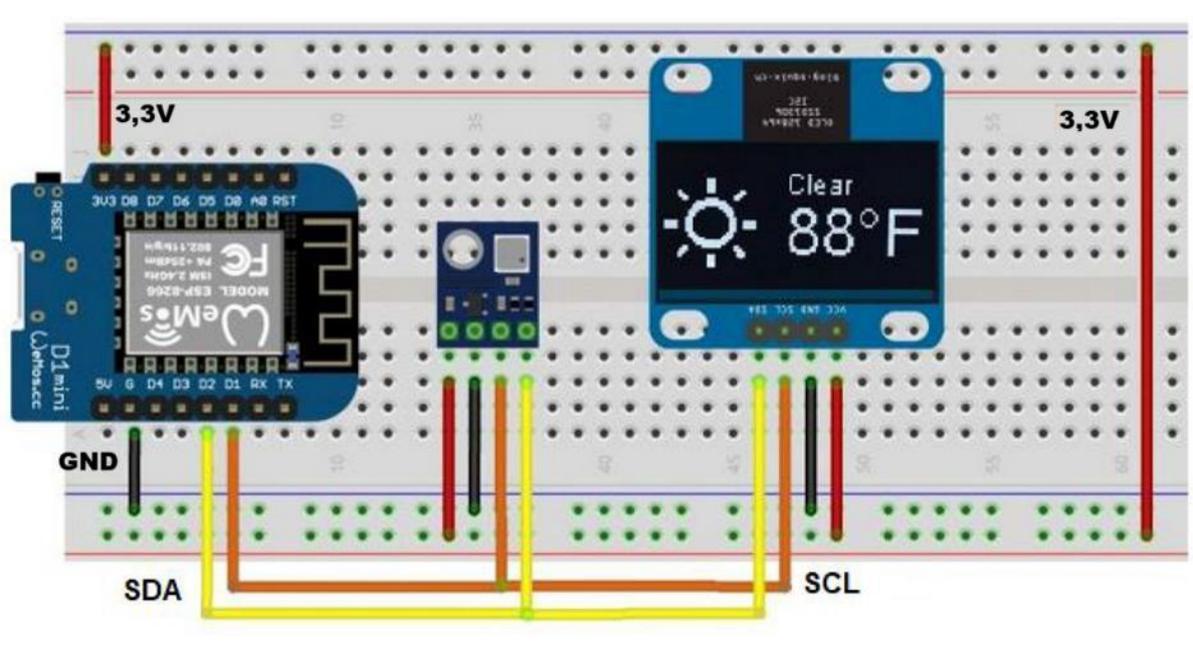


Abbildung 5: Schaltungsaufbau auf dem Steckboard

## 4. System-Start

### 4.1. Software - Installation

#### 4.1.1. Treiberinstallation testen

- Mit der rechten Maustaste auf das Windows-Symbol und im Menü auswählen.
- Den „Geräte-Manager“ unter Windows öffnen.
- Das ESP-Board an den USB anstecken. Es muss ein Eintrag wie folgender auftauchen. Hier ist alles in Ordnung

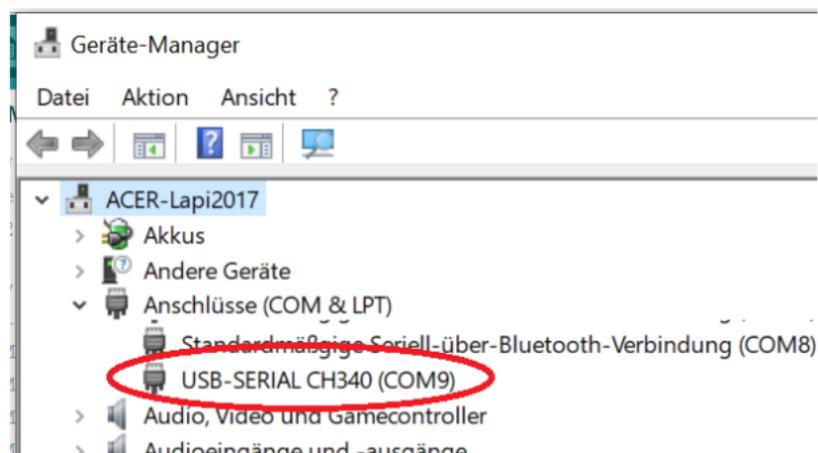


Abbildung 6: Gerätemanager Anschlüsse

- Wenn dieser Eintrag nicht erscheint, dann muss zuerst der USB-SERIAL CH340 Treiber geladen werden. Treiber herunterladen:
  - Windows: [http://www.wch.cn/download/CH341SER\\_ZIP.html](http://www.wch.cn/download/CH341SER_ZIP.html)
  - Mac: [http://www.wch.cn/download/CH341SER\\_MAC\\_ZIP.html](http://www.wch.cn/download/CH341SER_MAC_ZIP.html)

Unter Windows installierst du ihn einfach durch das Ausführen der "SETUP.EXE" im Ordner "CH341SER".

Mac-Nutzer folgen am besten den Installationsanweisungen, die dem Treiberpaket beiliegen.

Nach dem erneuten Anschließen des MCU sollte dieser als "USB-SERIAL CH340"-Gerät erkannt werden und der verwendete COM-Port angezeigt werden.

#### 4.1.2. Arduino IDE download

- Software-Download unter <https://www.arduino.cc/en/software/>
- Download Options Windows Win 10 and newer, 64 bit

## System-Start

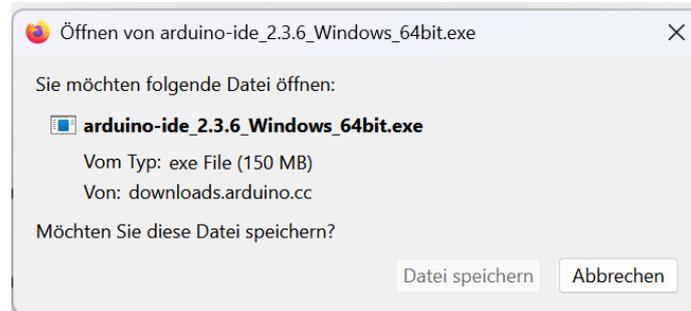


Abbildung 7: Arduino Software Download

- Verwende Arduino Dokumentationen unter <https://www.arduino.cc/> und zusätzlichen Internet – Quellen.
- Die randomnerdtutorials.com ist auch sehr empfehlenswert bei der Umsetzung von Projekten:  
<https://randomnerdtutorials.com/getting-started-with-esp8266-wifi-transceiver-review/>

### 4.1.3. Zusätzliche hilfreiche Software

Für die Programmierung könnten zusätzlich zur Arduino Software, auch noch folgende Programme hilfreich sein:

- Notepad++: <https://notepad-plus-plus.org/downloads/v7.0/>
- Visual Studio Code und Plattform IO: <https://code.visualstudio.com/> (nur für Profis)

## 4.2. Software – Start und Einstellungen



Abbildung 8: Arduino IDE

- Menü: Datei → Einstellungen  
 Erstelle ein **eigenes Verzeichnis** für die Arduino Projekte und stelle das Verzeichnis unter „Pfad für Sketchbook:“ ein. zB: C:\users\DEINNAME\Documents\Arduino
- In diesem Fall werden die Bibliotheken gespeichert unter:  
 C:\users\DEINNAME\Documents\Arduino\libraries

## System-Start

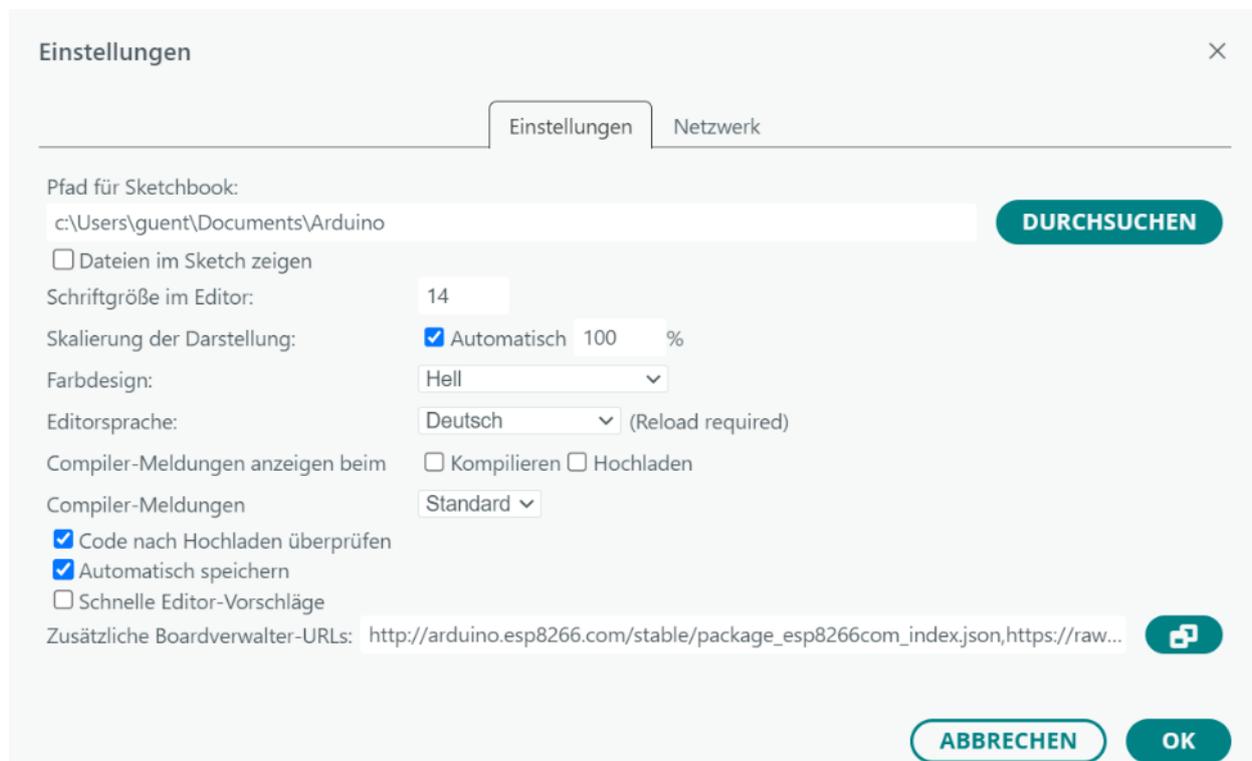


Abbildung 9: Einstellungen

Für den ESP8266 und den ESP32 werden zusätzliche Boardverwalter benötigt. Dazu sind die URLs einzutragen:

[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

[https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json)

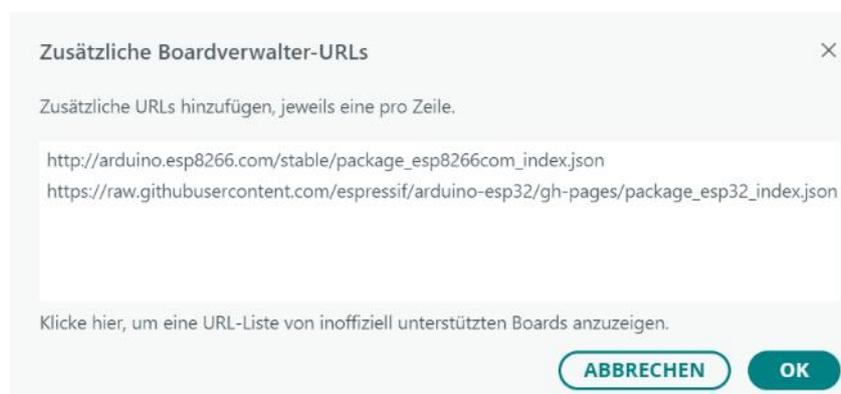


Abbildung 10: Zusätzliche Boardverwalter für die WLAN-Chips

- Sobald hier etwas geändert wurde, ist ein Arduino „Neustart“ empfehlenswert.
- Boardverwalter starten und zusätzliche Boards installieren mit Menü:  
„Werkzeuge“ → Board → „Boardverwaltung“
- Beim Filter „espress“ eingeben

## System-Start

- Installieren von **esp8266** und **esp32**

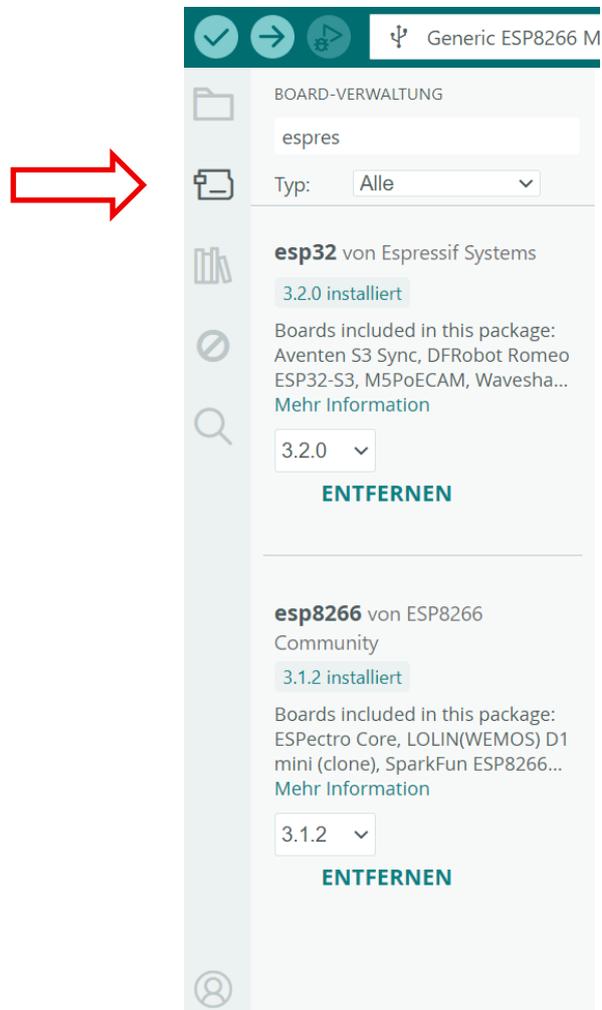


Abbildung 11: esp8266 und esp32 installiert

- Überprüfung der Installation von **esp8266** und **esp32**  
Wenn beide Boards **esp8266** und **esp32** installiert sind, dann sieht es so aus:

## System-Start

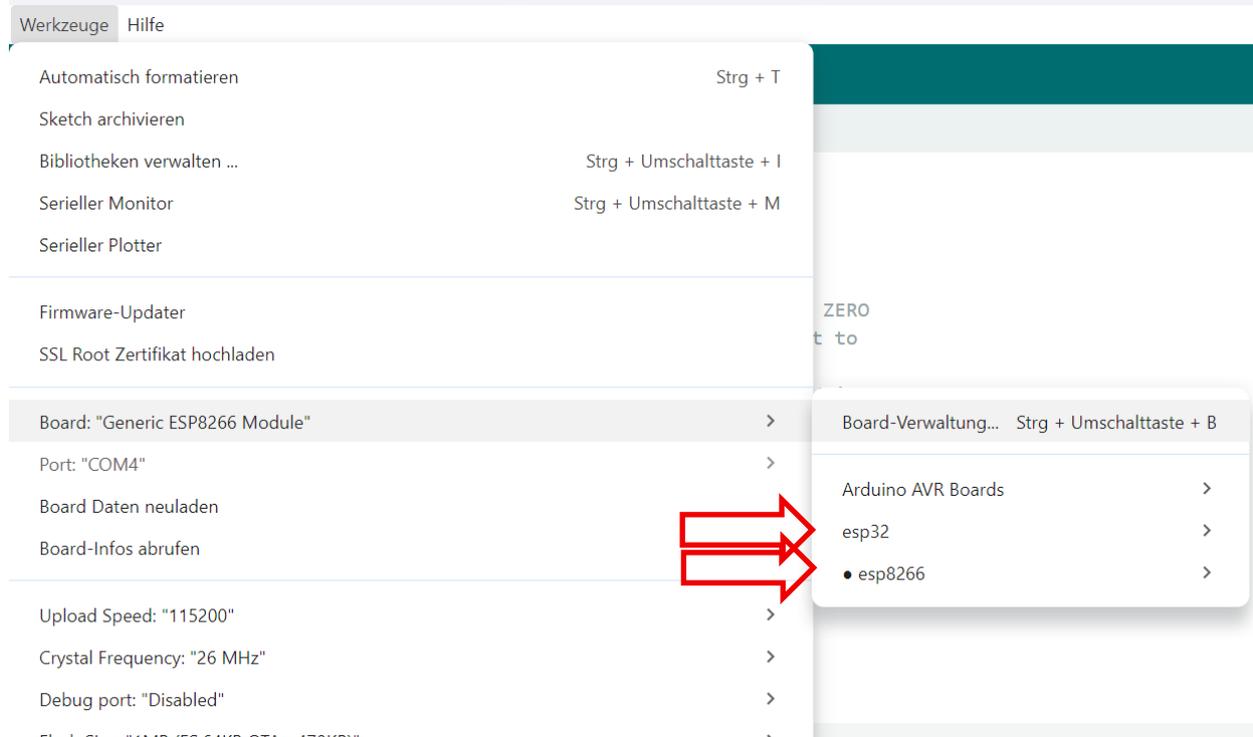


Abbildung 12: Beide neuen Boards installiert

- Einstellen vom Board ESP8266 Wemos D1 mini mit Menü:  
„Werkzeuge“ → Board → „esp8266“

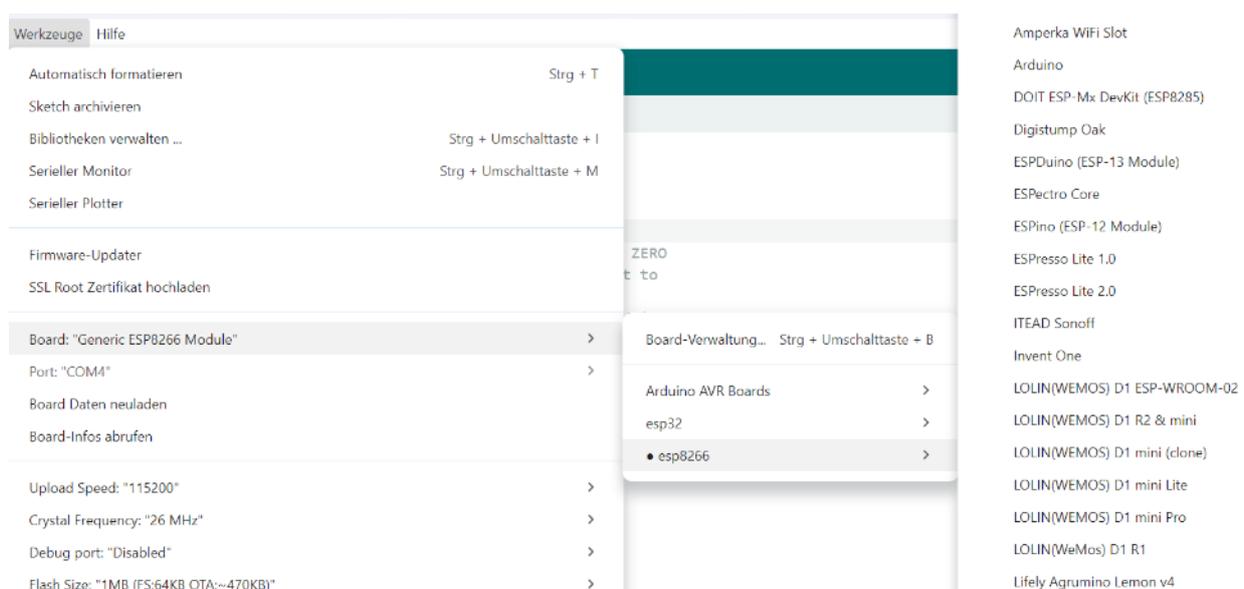


Abbildung 13: Board auswählen

- Auswählen von: Generic ESP8266 Module oder LOLIN(WeMos) D1 R1

## System-Start

- COM-Port auswählen. (Prüfe das Ergebnis im Geräte-Manager. Siehe vorher)

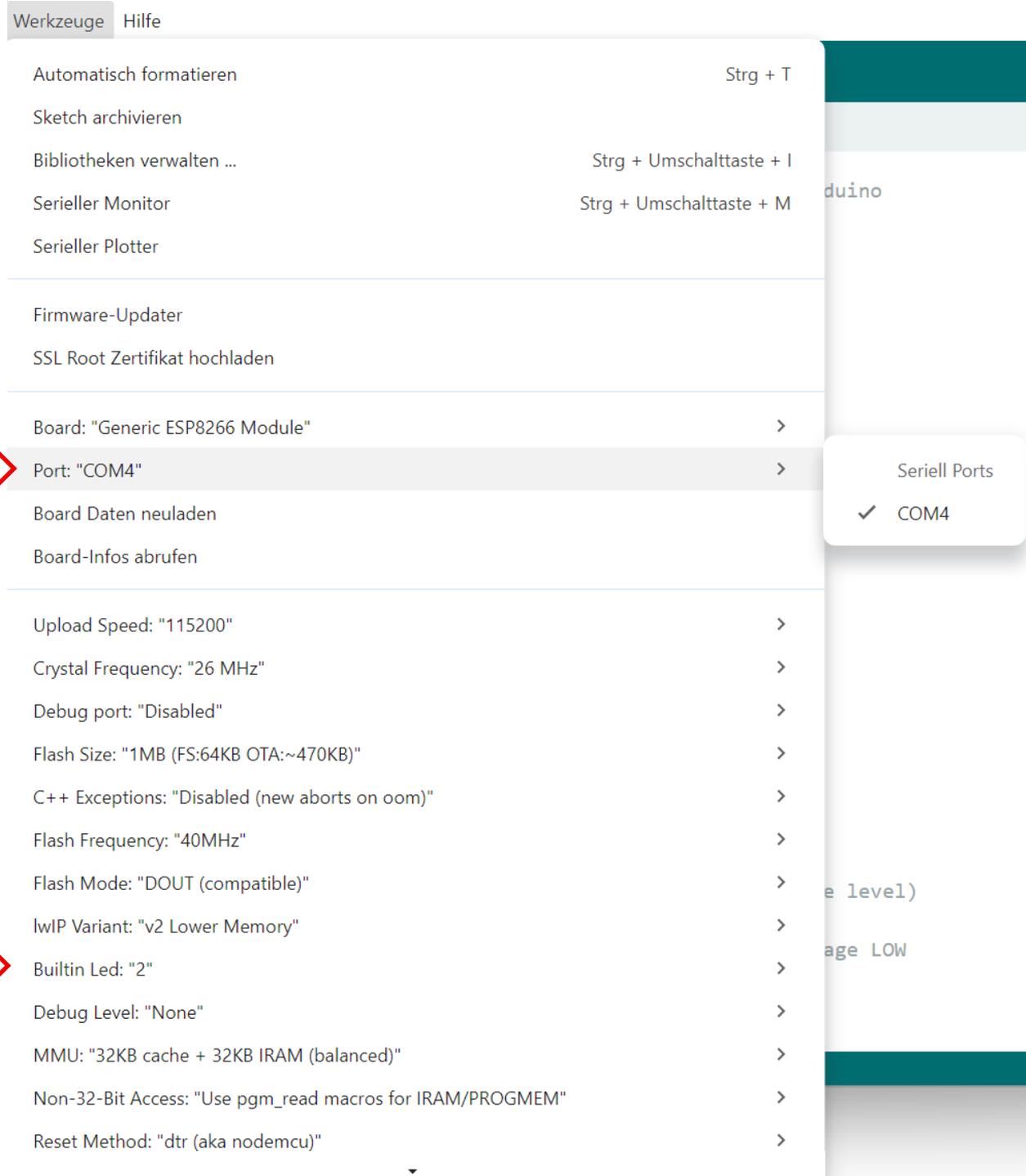


Abbildung 14: Board Parameter einstellen

Sonst alle Einstellungen „Standard“

## 5. System-Check

### 5.1. Hardware – Check

Nach dem Schaltungsaufbau jede Verbindung nochmals prüfen.

- Schaltung aufbauen
- Schaltung prüfen
- Nur 3,3V in der Schaltung

### 5.2. Beispielprogramm - Check

- Das Beispielprogramm „Blink“ öffnen

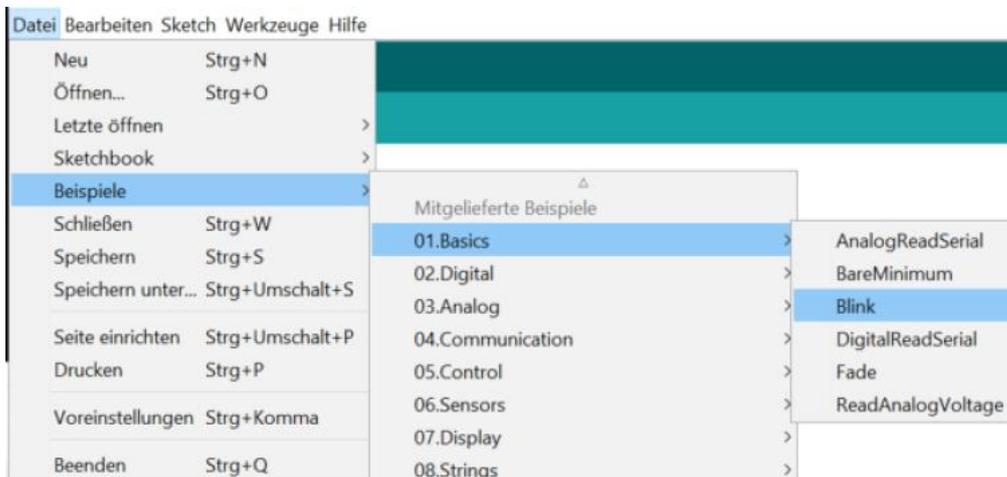


Abbildung 15: Beispiel-Programm Blink

- Das Beispielprogramm „Blink“ durchsehen und eventuelle Änderungen durchführen (Optional)

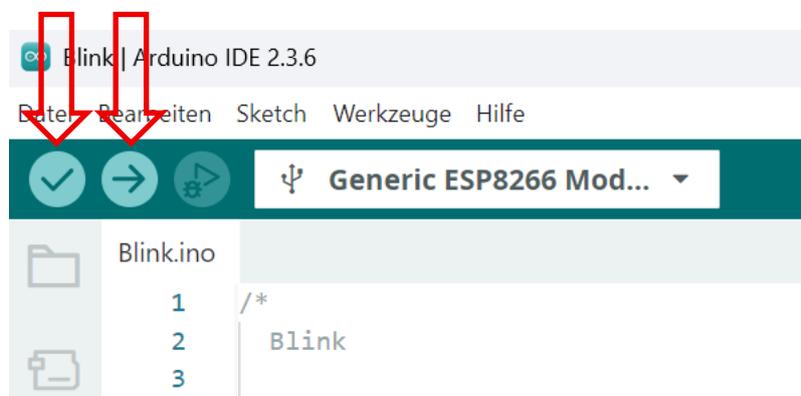
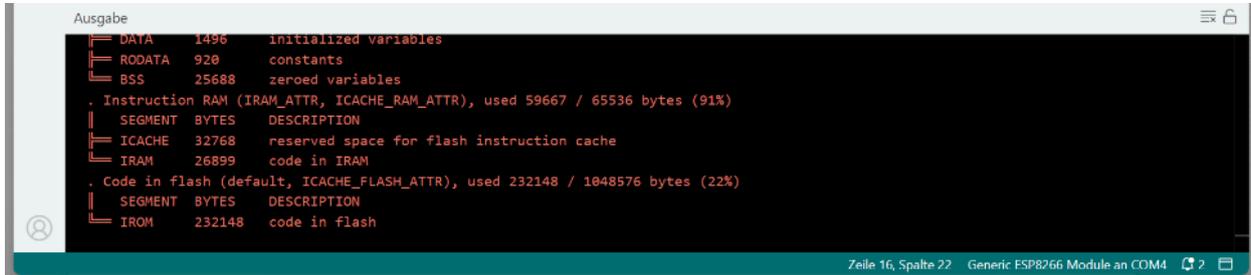


Abbildung 16: Programm prüfen und hochladen

## System-Check

- Das Beispielprogramm „Blink“ prüfen



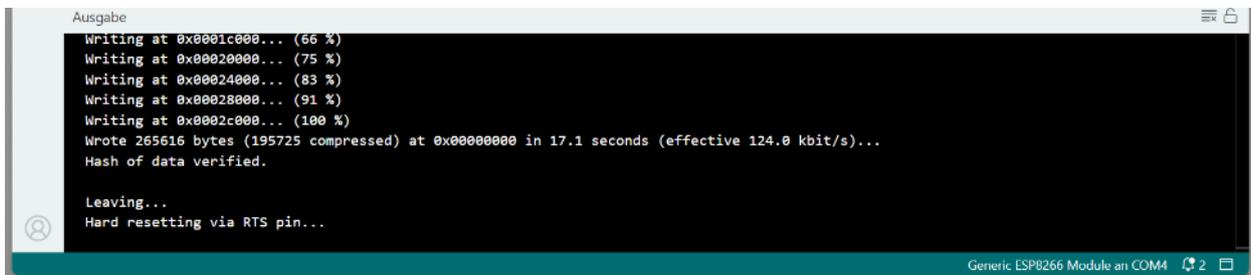
```

Ausgabe
┌─ DATA 1496 initialized variables
┌─ RODATA 920 constants
┌─ BSS 25688 zeroed variables
. Instruction RAM (IRAM_ATTR, ICACHE_RAM_ATTR), used 59667 / 65536 bytes (91%)
┌─ SEGMENT BYTES DESCRIPTION
┌─ ICACHE 32768 reserved space for flash instruction cache
┌─ IRAM 26899 code in IRAM
. Code in flash (default, ICACHE_FLASH_ATTR), used 232148 / 1048576 bytes (22%)
┌─ SEGMENT BYTES DESCRIPTION
┌─ IROM 232148 code in flash
  
```

Zeile 16, Spalte 22 Generic ESP8266 Module an COM4

Abbildung 17: Ausgabe nach dem Prüfen ohne Fehler

- Das Beispielprogramm „Blink“ hochladen



```

Ausgabe
Writing at 0x0001c000... (66 %)
Writing at 0x00020000... (75 %)
Writing at 0x00024000... (83 %)
Writing at 0x00028000... (91 %)
Writing at 0x0002c000... (100 %)
Wrote 265616 bytes (195725 compressed) at 0x00000000 in 17.1 seconds (effective 124.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
  
```

Generic ESP8266 Module an COM4

Abbildung 18: Ausgabe nach dem erfolgreichen Hochladen

Sollten Fehler im Programm entdeckt werden, dann werden die Zeilen rot markiert und eine Meldung ausgegeben.

Nur Übung und Erfahrung macht den Meister.

**Es wird empfohlen beim Beginnen mit einer Schaltung jedes Mal mit dem Blink-Programm zu starten. Dieses Programm muss funktionieren. Wenn nicht, dann gibt es ein grundlegendes Problem: Kabel, Stromversorgung, Bauteile defekt, usw.**

### 5.3. Aufgabe

- Verändere die Blink-Geschwindigkeit.
- Speichere das Programm unter einem eigenen Namen „myBlink.ino“ ab.
- Suche das Verzeichnis und Programm im Datei-Explorer.

## System-Check

```
Blink.ino
1  /*
2   Blink
3
4   Turns an LED on for one second, then off for one second, repeatedly.
5
6   Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
7   it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
8   the correct LED pin independent of which board is used.
9   If you want to know what pin the on-board LED is connected to on your Arduino
10  model, check the Technical Specs of your board at:
11  https://docs.arduino.cc/hardware/
12
13  modified 8 May 2014
14  by Scott Fitzgerald
15  modified 2 Sep 2016
16  by Arturo Guadalupi
17  modified 8 Sep 2016
18  by Colby Newman
19
20  This example code is in the public domain.
21
22  https://docs.arduino.cc/built-in-examples/basics/Blink/
23  */
24
25  // the setup function runs once when you press reset or power the board
26  void setup() {
27    // initialize digital pin LED_BUILTIN as an output.
28    pinMode(LED_BUILTIN, OUTPUT);
29  }
30
31  // the loop function runs over and over again forever
32  void loop() {
33    digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
34    delay(1000); // wait for a second
35    digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
36    delay(1000); // wait for a second
37  }
38
```

Abbildung 19: Beispiel Programm Blink

## 6. Sensor HTU21 Test

### 6.1. Funktionsbeschreibung

Der HTU21 Sensor wird ausgelesen und die Daten werden in bestimmten Zeitabständen am seriellen Monitor angezeigt. Dieses Programm dient zum Testen des Sensors.

Der Quellcode ist aus dem Beispielverzeichnis der Bibliothek „Adafruit\_HTU21DF\_Library“

### 6.2. Neuer Programmcode

#### 01\_myHTU21DF\_Test.ino

- Menü: Datei → Neu
- Menü: Datei → Speichern unter

### 6.3. Bibliothek einbinden/installieren

Für die Installation von Bibliotheken gibt es bei der Arduino Software mehrere Möglichkeiten.

- Bibliotheksmanager
- Github – Download einer ZIP-Datei

**Hier wird nicht auf die konkrete Anleitung eingegangen!**

- Wir benötigen die Bibliothek Adafruit HTU21DF Library  
 Wähle links das Icon „Bibliothek“ und tipe „htu“ ein oder lade die Bibliothek von **github**.

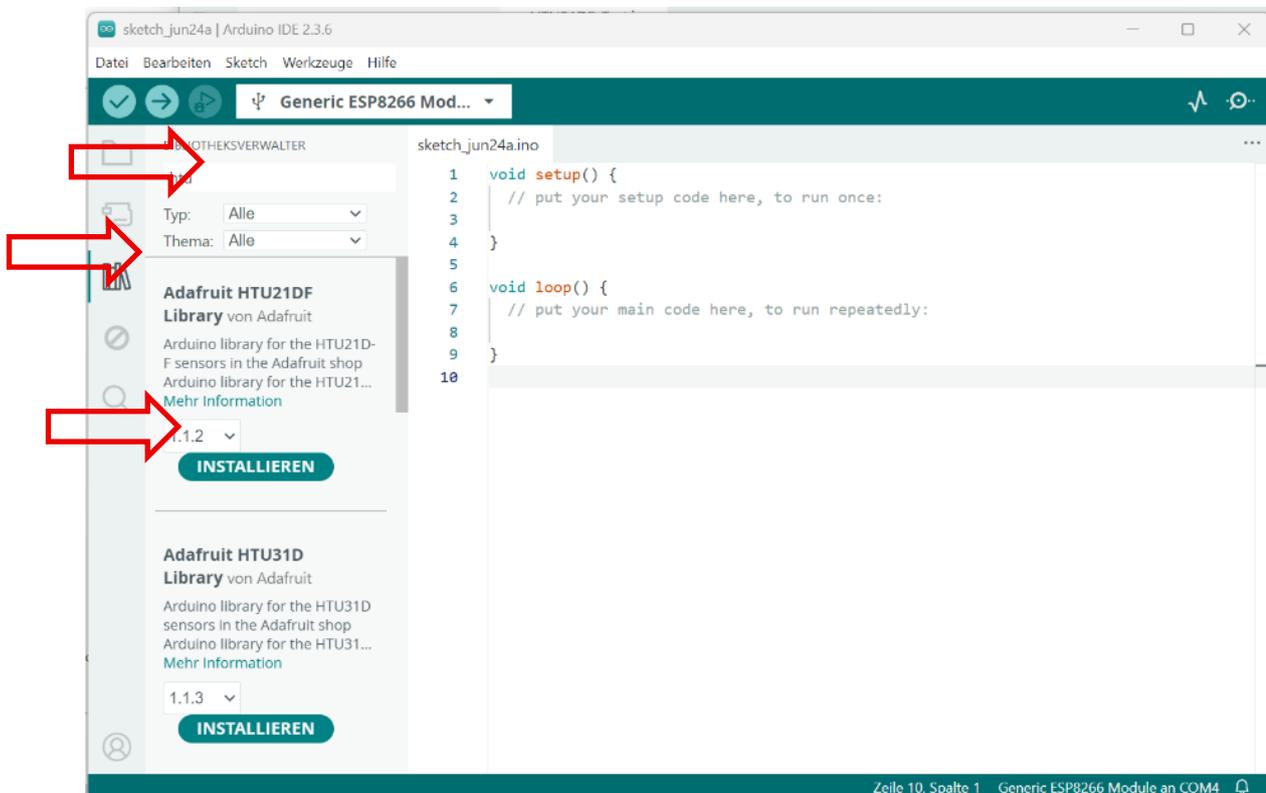


Abbildung 20: Neuer Programmcode und Bibliothek installieren

## Sensor HTU21 Test

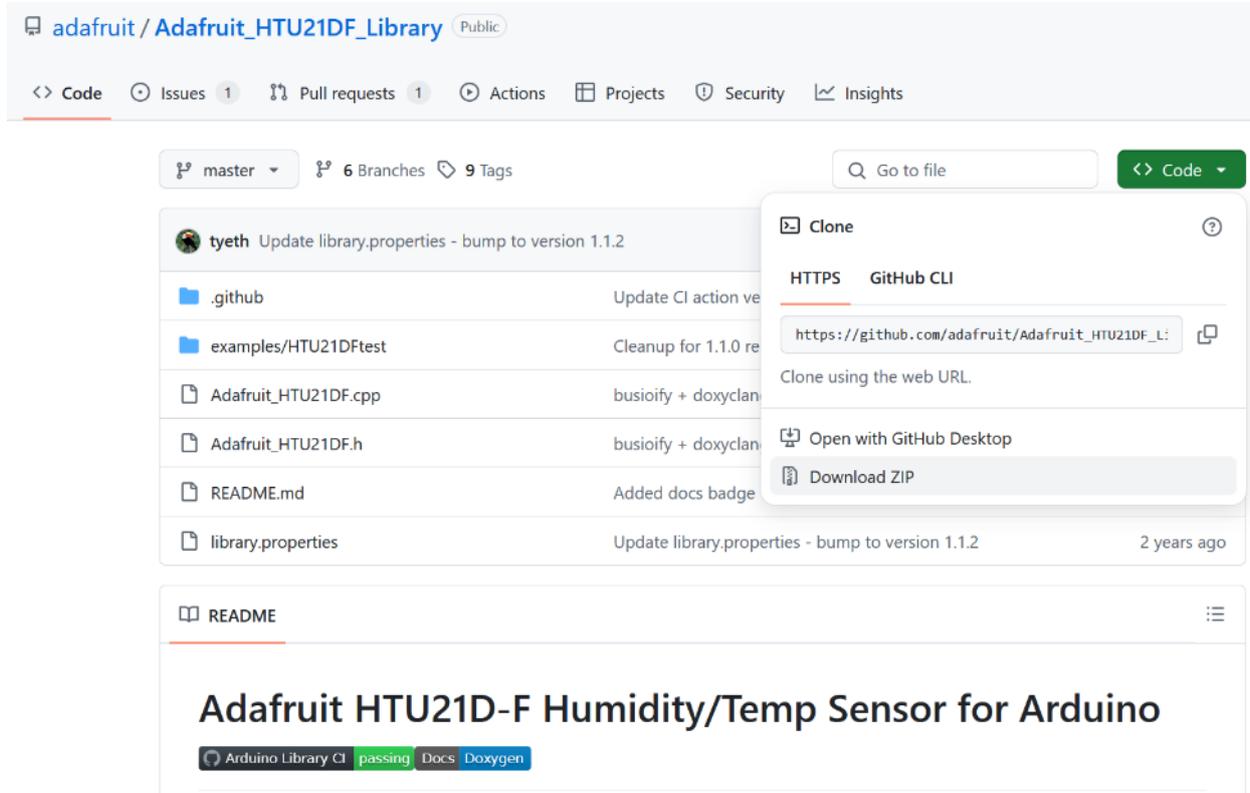


Abbildung 21: Bibliothek von github

- Das entpackte Verzeichnis in das Arduino Verzeichnis „C:\Program Files\Arduino IDE\libraries“ laden und das Wort „master“ löschen
- Im Verzeichnis der Library ist ein Verzeichnis „examples“ zu finden. Hier öffnen wir die Ino-Datei mit der Arduino IDE.

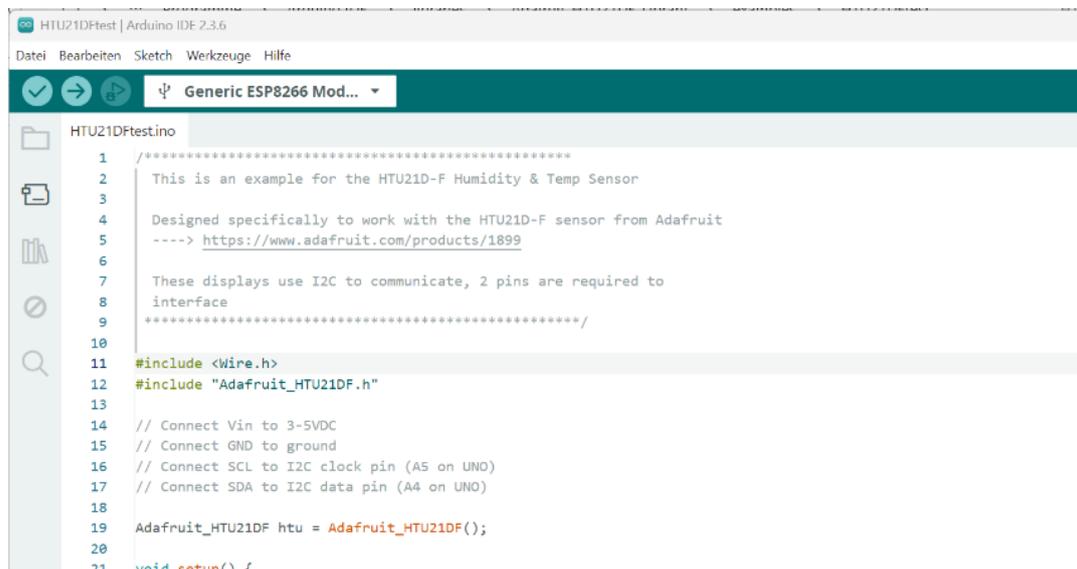
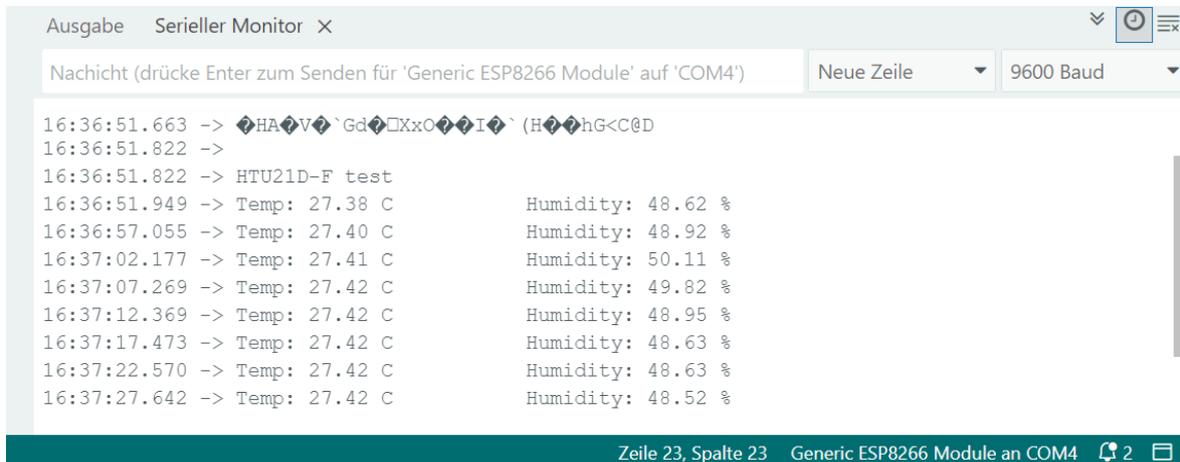


Abbildung 22: Beispiel Programm HTU21DF

## 6.4. Programmcode Ausgabe

- Board auswählen „Generic ESP8266 Module“
- Port auswählen (siehe Geräte-Manager)
- Code überprüfen
- Code hochladen
- Prüfen ob Fehler aufgetreten sind und gegebenenfalls korrigieren.
- Seriellen Monitor starten



```

16:36:51.663 -> HA V `Gd XxO I` (H hG<C@D
16:36:51.822 ->
16:36:51.822 -> HTU21D-F test
16:36:51.949 -> Temp: 27.38 C           Humidity: 48.62 %
16:36:57.055 -> Temp: 27.40 C           Humidity: 48.92 %
16:37:02.177 -> Temp: 27.41 C           Humidity: 50.11 %
16:37:07.269 -> Temp: 27.42 C           Humidity: 49.82 %
16:37:12.369 -> Temp: 27.42 C           Humidity: 48.95 %
16:37:17.473 -> Temp: 27.42 C           Humidity: 48.63 %
16:37:22.570 -> Temp: 27.42 C           Humidity: 48.63 %
16:37:27.642 -> Temp: 27.42 C           Humidity: 48.52 %
  
```

Zeile 23, Spalte 23    Generic ESP8266 Module an COM4    2    ☰

Abbildung 23: Ausgabe am seriellen Monitor

- **Kommentar** nach eigenen Bedürfnissen ändern: Datum, Funktion, Quellen, Beschreibung, ...  
Kommentare sind wichtig, damit man im Nachhinein weiß, was gemacht wurde.
- Code-Dokumentation im Quelltext.

## 6.5. Fehlerbehandlung

- Blink Programm hat funktioniert?
- Richtige Beschaltung vom Sensor?
- Serieller Monitor richtige Geschwindigkeitseinstellung „xxx Baud“
- Seriellen Monitor zur Fehlersuche verwenden, um an anderer Stelle Text auszugeben.

## 6.6. Aufgabe

- Ändere den Kommentar nach eigenen Vorstellungen
- Ändere die Auslesegeschwindigkeit und die Anzeige

## 7. Anzeige OLED 0,96“ Test

### 7.1. Funktionsbeschreibung

Auf dem OLED wird „Hallo World“ ausgegeben.

### 7.2. Neuer Programmcode

#### 02\_myOLED\_Test.ino

- Menü: Datei → Neu
- Datei -> Speichern unter ...  
Verzeichnis auswählen und speichern

### 7.3. Bibliothek einbinden/installieren

- Wir benötigen die Bibliothek U8g2 von oliver

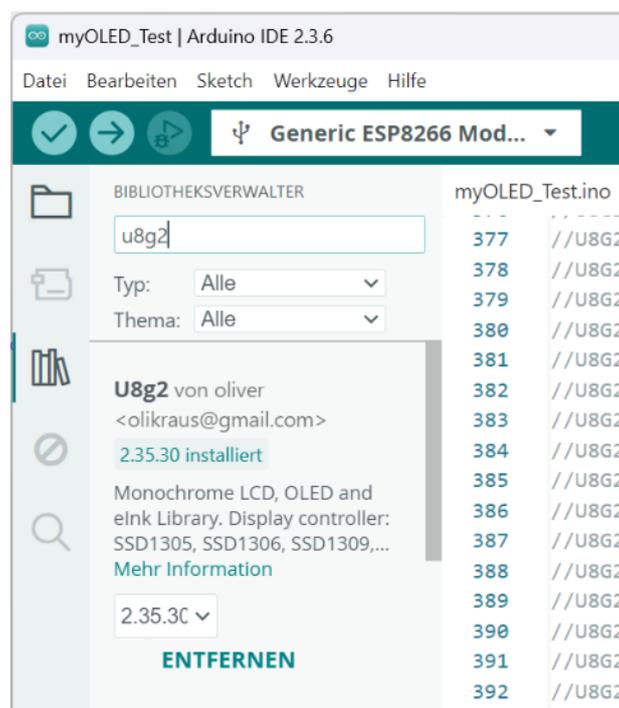


Abbildung 24: Bibliothek U8g2 von oliver installieren

- Im Programmcode wird verwendet  
`#include <Arduino.h>`  
`#include <U8g2lib.h>`

## 7.4. Programmcode Ausgabe



Abbildung 25: OLED-Ausgabe

## 7.5. Fehlerbehandlung

- Kopiere die Fehler und analysiere diese mit ChatGPT oder DeepSeek.

## 7.6. Aufgabe

- Verändere den Text und schreibe eine zweite Zeile
- Starte ein weiteres Beispiel-Programm: GraphicsTest.ino  
Beispiele – U8g2 – u8x8 – GraphicsTest
- Verwende in einem eigenen Programm graphische Elemente mit Text und verwende unterschiedliche Schriften und Textgrößen.

## 8. HTU21 – Sensor mit OLED-Ausgabe

### 8.1. Funktionsbeschreibung

- Der HTU21 – Sensor wird ausgelesen
- Die Werte werden am seriellen Monitor angezeigt
- Die Werte werden am OLED angezeigt.
- Es wird ein Logo angezeigt

Die Funktionen vom OLED werden in einer eigenen Datei ausgelagert, ähnlich wie bei einer Bibliothek. Das Einbinden erfolgt mit `#include „xxxx.h“` in Anführungszeichen. Dadurch bleibt der Hauptcode übersichtlicher.

### 8.2. Neuer Programmcode

#### 03\_HTU\_OLED.ino

- Es wird eine Auslagerung von inline-Funktionen für das OLED in eine h-Datei erstellt. Dadurch wird die Wiederverwendbarkeit vereinfacht.
- Ein neuer Programmcode wird aus dem Code „Sensor-HTU21-Test“ erstellt. Wir kombinieren nun die vorherigen Programme zu einem neuen Projekt.
- Es gibt ein HTU\_OLED.ino, myLogo\_u8g2.h, myOLED\_uu8g2.h  
Alle erfaßten Programme werden in Reiter aufgelistet



```

03_HTU_OLED.ino  myLogo_u8g2.h  myOLED_u8g2.h
1  /*****
2  Juni 2025 03_HTU_OLED.ino
3  This is an example for the HTU21D-F Humidity & Temp Sensor including OLED 0,96"
4
5  These displays use I2C to communicate, 2 pins are required to interface
6  *****/
  
```

Abbildung 26: Programme Reiter

### 8.3. Bibliothek einbinden/installieren

- `#include "myOLED_u8g2.h"`
- `#include <Adafruit_HTU21DF.h>`

In myOLED\_u8g2.h

- `#include <Wire.h>`
- `#include <Arduino.h>`
- `#include <U8g2lib.h>`
- `#include "myLogo_u8g2.h"`

In manchen Fällen kann die Reihenfolge der `#include ...` Anweisungen von Bedeutung sein. Insbesondere dann, wenn eigene Auslagerungsdateien erstellt werden und globale Variablen und Definitionen verwendet werden.

## 8.4. Programmcode Ausgabe

```

Ausgabe  Serieller Monitor  X
Nachricht (drücke Enter zum Senden für 'Generic ESP8266 Module' auf 'COM4')

11:33:44.289 -> s10100|00d0|000000d0c|??0?0?s?c?0b?0oc?log??0
11:33:44.436 ->
11:33:44.436 -> HTU21D-F test mit OLED-Ausgabe
11:33:44.436 -> Start Sensor Test HTU21
11:33:44.436 ->
11:33:54.927 -> Temp: 24.29 C           Humidity: 48.45 %
11:34:00.083 -> Temp: 24.27 C           Humidity: 48.44 %
11:34:05.230 -> Temp: 24.29 C           Humidity: 48.50 %

```

Abbildung 27: Ausgabe auf dem seriellen Monitor

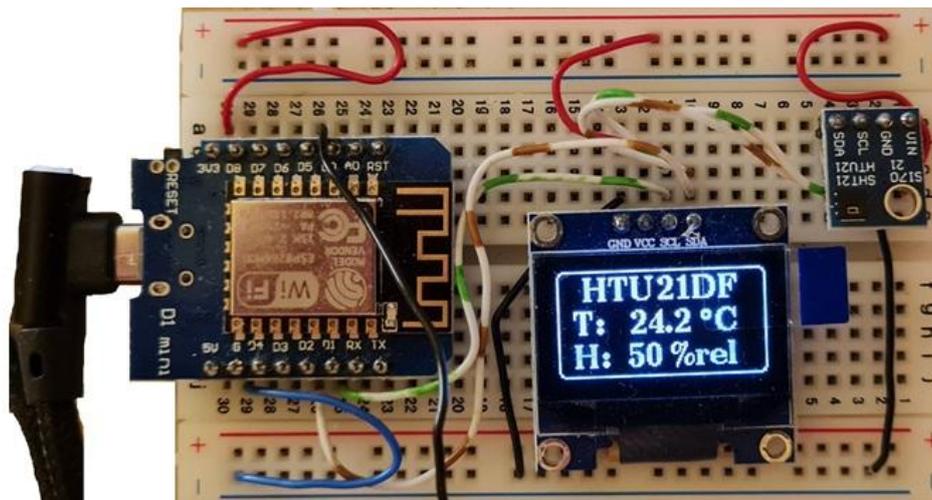


Abbildung 28: OLED-Ausgabe

## 8.5. Fehlerbehandlung

Achte darauf, dass der Sensor nicht zu oft abgefragt wird, da er sich sonst selbst erwärmt und die Daten verfälscht werden. Jede Minute einmal abfragen ist bei Umgebungswerten ausreichend, da sich die Werte nicht so rasch ändern werden.

## 8.6. Aufgabe

- Steigere das Ausleseintervall auf >1 Minute.

## 9. Datum und Uhrzeit vom Internet per NTP

### 9.1. Funktionsbeschreibung

Datum und Uhrzeit werden aus dem Internet per NTP abgefragt und am seriellen Monitor angezeigt. Es ist dazu eine WLAN-Verbindung notwendig. Das Programm funktioniert für den ESP8266 ebenso für den ESP32.

### 9.2. Neuer Programmcode

#### 04\_NTP\_Date\_Time.ino

Zuerst wird ein einfaches Minimal-Programm erstellt, welches nur das Datum und Uhrzeit aus dem Internet holt und am seriellen Monitor anzeigt.

Mit `configTime(MY_TZ, MY_NTP_SERVER)` wird die Zeitzone und der ntp-Server eingestellt, danach wird die Erreichbarkeit geprüft, bis die Verbindung funktioniert.

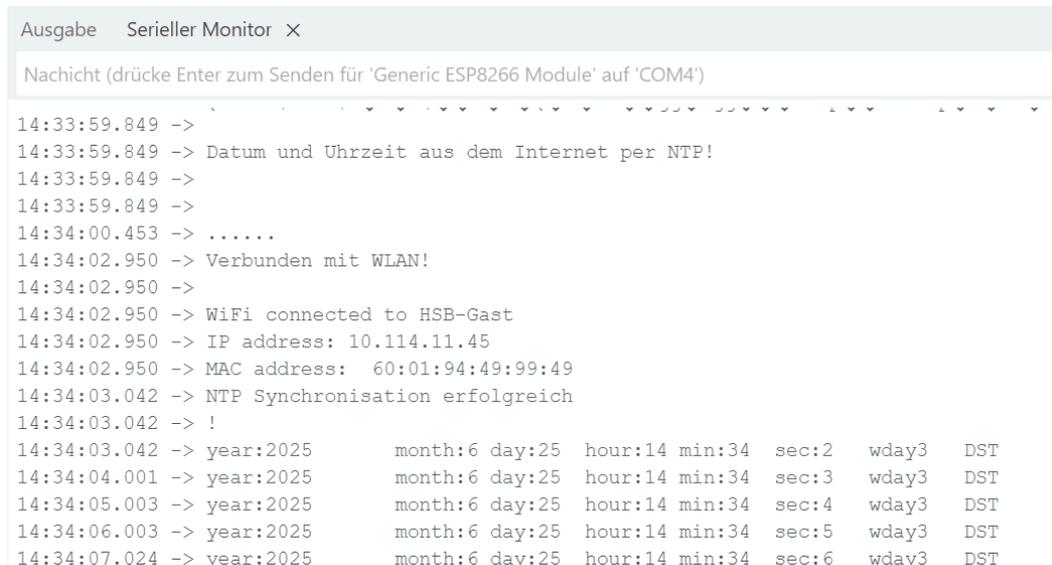
### 9.3. Bibliothek einbinden/installieren

```

// #include <WiFi.h>           // Für ESP32
#include <ESP8266WiFi.h>      // Für ESP8266 (statt WiFi.h)
#include <time.h>             // Für Zeitabfrage und Format
  
```

In manchen Fällen kann die Reihenfolge der `#include ...` von Bedeutung sein. Insbesondere dann, wenn eigene Auslagerungsdateien erstellt werden und globale Variablen und Definitionen benötigt werden.

### 9.4. Programmcode Ausgabe



```

Ausgabe Serieller Monitor X
Nachricht (drücke Enter zum Senden für 'Generic ESP8266 Module' auf 'COM4')
14:33:59.849 ->
14:33:59.849 -> Datum und Uhrzeit aus dem Internet per NTP!
14:33:59.849 ->
14:33:59.849 ->
14:34:00.453 -> .....
14:34:02.950 -> Verbunden mit WLAN!
14:34:02.950 ->
14:34:02.950 -> WiFi connected to HSB-Gast
14:34:02.950 -> IP address: 10.114.11.45
14:34:02.950 -> MAC address: 60:01:94:49:99:49
14:34:03.042 -> NTP Synchronisation erfolgreich
14:34:03.042 -> !
14:34:03.042 -> year:2025      month:6 day:25  hour:14 min:34  sec:2  wday3  DST
14:34:04.001 -> year:2025      month:6 day:25  hour:14 min:34  sec:3  wday3  DST
14:34:05.003 -> year:2025      month:6 day:25  hour:14 min:34  sec:4  wday3  DST
14:34:06.003 -> year:2025      month:6 day:25  hour:14 min:34  sec:5  wday3  DST
14:34:07.024 -> year:2025      month:6 day:25  hour:14 min:34  sec:6  wday3  DST
  
```

Abbildung 29: Ausgabe am seriellen Monitor

## 9.5. Fehlerbehandlung

## 9.6. Aufgabe

- Erstelle eine Funktion zur formatierten Anzeige vom Datum (Deutsch / Englisch)
- Erstelle eine Funktion zur formatierten Anzeige der Zeit
- Erstelle eine Auslagerungsdatei „**myNTP\_Time.h**“ mit allen zeitrelevanten Einstellungen und Funktionen, damit der Code einfach wiederverwendet werden kann und der Hauptcode übersichtlich bleibt.
- Erstelle eine Auslagerungsdatei „**myWiFi.h**“ mit allen WLAN relevanten Einstellungen und Funktionen, damit der Code einfach wiederverwendet werden kann und der Hauptcode übersichtlich bleibt.
- Erstelle ein Verzeichnis mit allen eigenen Auslagerungsdateien, um eine Sammlung zu erstellen.
- Erstelle eine Ausgabe auf das OLED.

## 10. NTP-Zeit mit OLED-Ausgabe

### 10.1. Funktionsbeschreibung

Es wird die vorher erstellte Auslagerungsdatei *myOLED\_u8g2.h* verwendet, um die Uhrzeit und das Datum auf dem OLED anzuzeigen.

### 10.2. Neuer Programmcode

*05\_NTP\_Date\_Time\_OLED.ino* mit vier Auslagerungsdateien.

```
#include "myNTP_Time.h" // für NTP Internet Zeit-Datum
#include "myOLED_u8g2.h" // für Anzeige am OLED
#include "mLogo_u8g2.h" // für Logo Bitmap Muster zurAnzeige am OLED
#include "myWiFi.h" // für WLAN-Verbindung
```



Abbildung 30: Hauptcode mit Auslagerungsdateien

### 10.3. Bibliothek einbinden/installieren

Siehe vorherige Projekte

## 10.4. Programmcode Ausgabe

```

Ausgabe  Serieller Monitor  X
Nachricht (drücke Enter zum Senden für 'Generic ESP8266 Module' auf 'COM4')
18:43:57.706 -> Datum und Uhrzeit aus dem Internet per NTP!
18:44:10.218 ->
18:44:10.218 ->
18:44:10.218 -> Start connecting to WiFi .....
18:44:13.198 -> WiFi connected to HSB-Gast
18:44:13.198 -> IP address: 10.114.11.45
18:44:13.245 -> MAC address: 60:01:94:49:99:49
18:44:13.284 -> NTP Synchronisation erfolgreich
18:44:13.284 -> !
18:44:13.284 -> 25.06.2025      18:44:13      Mittwoch
18:44:14.310 -> 25.06.2025      18:44:14      Mittwoch
18:44:15.344 -> 25.06.2025      18:44:15      Mittwoch

```

Abbildung 31: Ausgabe am seriellen Monitor

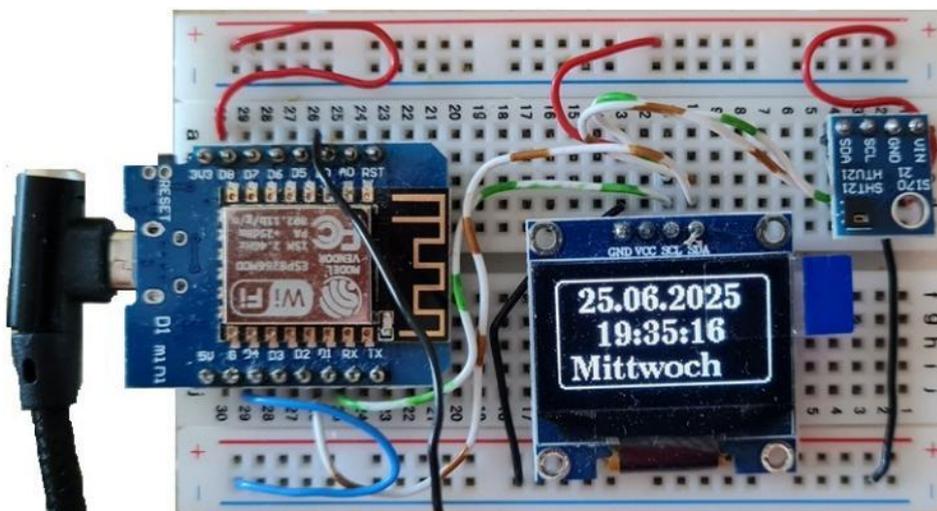


Abbildung 32: Ausgabe auf OLED

## 10.5. Fehlerbehandlung

### 10.6. Aufgabe

- Ergänze die Ausgabe auf dem seriellen Monitor mit den HTU-Sensorwerten
- Ergänze die Ausgabe auf dem OLED mit den HTU-Sensorwerten
- Die OLED-Anzeige schaltet zwischen Datum-Uhrzeit und HTU-Werten hin und her.

Siehe **06\_HTU\_NTP\_OLED.ino**

## 11. ESP8266 und WebServer

### 11.1. WebServer und WiFi-Manager

Beachte bitte, dass die Bibliotheken

```
#include <ESPAsyncTCP.h>      // Für ESP8266
// #include <AsyncTCP.h>     // Für ESP32
#include <ESPAsyncWebServer.h>
```

... nicht gleichzeitig mit

```
#include <WiFiManager.h>
#include <WebServer.h> // Standard-WebServer (nicht async!)
```

... verwendet werden können.

Suche dazu ein Beispiel aus RandomNerdTutorial:

<https://randomnerdtutorials.com/esp8266-nodemcu-wi-fi-manager-asyncwebserver/>

### 11.2. WebServer - Beschreibung von DeepSeek:

Mit dem ESP32 oder ESP8266 kannst du verschiedene Arten von Webservern aufbauen, die sich in Funktionsweise und Einsatzgebiet unterscheiden. Hier sind die wichtigsten Unterschiede:

#### 1. Einfacher HTTP-Server (AsyncWebServer / ESP8266WebServer)

- **Funktion:**
  - Stellt statische oder dynamische Webseiten bereit.
  - Verarbeitet HTTP-Anfragen (GET, POST) für Steuerung und Datenabfrage.
- **Einsatz:**
  - Lokale Steuerung eines ESP über das Netzwerk (z. B. Schalten von GPIOs).
  - Anzeige von Sensordaten in einem Webbrowser.
- **Beispiel:**

cpp

 Copy  Download

```
#include <ESP8266WebServer.h>
ESP8266WebServer server(80);
server.on("/", []() { server.send(200, "text/plain", "Hello World!"); });
server.begin();
```

## 2. WebSocket-Server

- **Funktion:**
  - Ermöglicht **bidirektionale Echtzeit-Kommunikation** zwischen Client und Server.
  - Kein ständiges Neuladen der Seite nötig (z. B. für Live-Daten-Updates).
- **Einsatz:**
  - Dashboards mit Live-Grafiken (z. B. Sensorwerte in Echtzeit).
  - Interaktive Steuerung ohne Seitenrefresh.
- **Bibliothek:**

cpp

[Copy](#) [Download](#)

```
#include <WebSocketsServer.h>  
WebSocketsServer websocket(81);
```

## 3. OTA-Webserver (Over-The-Air Updates)

- **Funktion:**
  - Ermöglicht das **Firmware-Update** des ESP über das Netzwerk.
  - Meist kombiniert mit einem einfachen HTTP-Server für ein Upload-Formular.
- **Einsatz:**
  - Fernwartung ohne physischen Zugriff auf den ESP.
- **Bibliothek:**

cpp

[Copy](#) [Download](#)

```
#include <ESP8266HTTPUpdateServer.h>  
ESP8266HTTPUpdateServer httpUpdater;
```

#### 4. MQTT + WebServer (IoT-Gateway)

- **Funktion:**
  - Der ESP dient als **Bridge** zwischen HTTP und MQTT.
  - Empfängt Befehle über HTTP und leitet sie an ein MQTT-Netzwerk weiter (oder umgekehrt).
- **Einsatz:**
  - Integration in Smart-Home-Systeme (z. B. Home Assistant, Node-RED).

- **Bibliothek:**

```
cpp Copy Download  
  
#include <PubSubClient.h>  
PubSubClient mqttClient;
```

#### 5. REST-API-Server

- **Funktion:**
  - Stellt eine **REST-Schnittstelle** für externe Clients bereit (z. B. mobile Apps).
  - Nutzt JSON für die Datenübertragung.
- **Einsatz:**
  - Anbindung an Cloud-Dienste oder Smartphone-Apps.
- **Beispiel-Endpoint:**

```
cpp Copy Download  
  
server.on("/api/sensor", HTTP_GET, []() {  
    String json = "{\"temp\": 23.5}";  
    server.send(200, "application/json", json);  
});
```

#### 6. Captive Portal

- **Funktion:**
  - Erzwingt eine Weiterleitung auf eine Webseite (z. B. für WLAN-Konfiguration).
  - Wird oft in Kombination mit einem DNS-Server genutzt.
- **Einsatz:**
  - Erstkonfiguration von IoT-Geräten (z. B. WiFi-Credentials eingeben).
- **Bibliothek:**

```
cpp Copy Download  
  
#include <DNSServer.h>  
DNSServer dnsServer;
```

Vergleichstabelle:

Typ	Protokoll	Echtzeit	Typischer Einsatz
HTTP-Server	HTTP/HTTPS	Nein	GPIO-Steuerung, einfache Dashboards
WebSocket-Server	WebSocket	Ja	Live-Daten, interaktive Steuerung
OTA-Webserver	HTTP	Nein	Firmware-Updates
MQTT + WebServer	HTTP + MQTT	Teilweise	IoT-Gateway (z. B. Home Assistant)
REST-API	HTTP/HTTPS	Nein	Cloud-Anbindung, mobile Apps
Captive Portal	HTTP + DNS	Nein	WLAN-Konfiguration

Welcher Server ist der richtige?

- **Für einfache Steuerung:** HTTP-Server (z. B. Licht ein/aus).
- **Für Echtzeit-Daten:** WebSocket (z. B. Live-Temperaturdiagramm).
- **Für Updates:** OTA-Webserver.
- **Für IoT-Integration:** MQTT + REST-API.

## 12. Internet of Things (IoT) WiFi-Manager

### 12.1. Funktionsbeschreibung

Wenn in einem Projekt eine WLAN-Verbindung benötigt wird, dann können die Einstellungen im Klartext in den Code fixiert werden. Sollten die WLAN-Einstellungen zur Laufzeit verändert werden können, dann ist ein WiFi-Manager notwendig.

Der WiFi-Manager stellt einen Web-Service auf 192.168.4.1 zur Verfügung, wenn keine WLAN-Verbindung hergestellt werden kann. Die SSID und das PASSWORT können eingestellt und abgespeichert werden. Die Daten werden im nichtflüchtigen Speicher abgelegt und beim neuerlichen Start zur Initialisierung verwendet.

Mit Tastendruck, während dem Reset-Start wird der WiFi-Manager gezwungen zu starten und eine neue WiFi-Verbindung kann eingestellt werden.

Auf dem seriellen Monitor kann der Status des Verbindungsaufbaus mit dem WLAN mitverfolgt werden.

In einem bestimmten Zeitintervall wird die WiFi-Verbindung geprüft und bei Bedarf wieder hergestellt.

### 12.2. Neuer Programmcode

#### 07\_IoT\_WiFi-Manager.ino

Die Funktionen des WiFi-Managers werden in einer Auslagerungsdatei "myWiFi\_Manager.h" zusammengefasst. In diesem Projekt wird nur der WiFi-Manager getestet.

### 12.3. Bibliothek einbinden/installieren

```
#include <WiFiManager.h> // von Tzapu
```

### 12.4. Programmcode Ausgabe

- Mit einem Laptop eine WLAN-Verbindung mit dem Access-Point des ESP8266 herstellen.
- Im Internet-Explorer (Firefox) die IP-Adresse des Webserver 192.168.4.1 eingeben.
- Configure WiFi
  - SSID eingeben
  - Passwort eingeben
  - Update

## Internet of Things (IoT) WiFi-Manager

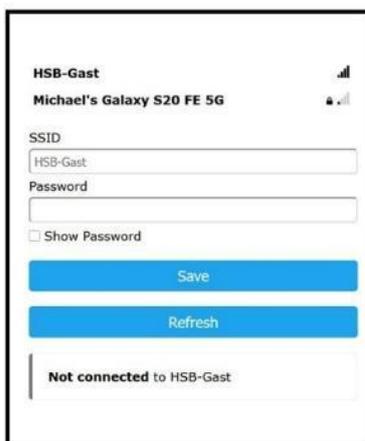
```

21:57:50.181 -> WiFi-Manager für IoT ohne WebServer.
21:57:50.218 -> Start mit AP und 192.168.4.1 wenn kein WLAN verfügbar.
21:57:50.218 -> Wenn das verfügbare WLAN Probleme macht, dann mit der Taste an GGPI014 (D5) den WiFi-Manager erzwingen
21:57:50.218 ->
21:57:50.599 -> *wm:AutoConnect
21:57:50.720 -> *wm:Connecting to SAVED AP: HSB-Gast
21:57:51.214 -> *wm:connectTimeout not set, ESP waitForConnectResult...
21:57:51.366 -> *wm:AutoConnect: SUCCESS
21:57:51.366 -> *wm:STA IP Address: 10.114.11.45
21:57:51.366 -> Erfolgreich verbunden mit: HSB-Gast
21:57:51.411 ->
21:57:51.411 -> Start Loop.
21:57:51.411 ->
21:57:51.411 -> Fertig
21:57:56.376 -> Fertig
21:58:01.398 -> Fertig
  
```

Abbildung 33: Ausgabe am seriellen Monitor



Startet, wenn kein WLAN gespeichert ist, oder wenn beim RESET die Taste GPIO\_14 = (D5) gedrückt wird.



Mögliche WLAN-Netze werden angezeigt und können ausgewählt werden.

Gespeichertes Netz wird angezeigt

Saving Credentials  
 Trying to connect ESP to network.  
 If it fails reconnect to AP to try again

Abbildung 34: WiFi - Manager Ausgabe

## 12.5. Fehlerbehandlung

## 12.6. Aufgabe

- Den WiFi-Manager beim HTU – NTP – OLD – Projekt einsetzen.  
Kopiere das Projekt NTP-Zeit mit OLED-Ausgabe  
Kopiere die Auslagerungsdatei vom Wifi-Manager in das Verzeichnis  
Passe die Ausgaben am OLED und im seriellen Monitor
- WiFi-Status Anzeige auf dem OLED

Board Einstellung „Flash Size“ prüfen, auf 4MB (FS:2MB OTA 1019KB) einstellen

Board: "Generic ESP8266 Module"	>	1MB (FS:256KB OTA:~374KB)
Port: "COM4"	>	1MB (FS:512KB OTA:~246KB)
Board Daten neuladen		1MB (FS:none OTA:~502KB)
Board-Infos abrufen		2MB (FS:64KB OTA:~992KB)
		2MB (FS:128KB OTA:~960KB)
Upload Speed: "115200"	>	2MB (FS:256KB OTA:~896KB)
Crystal Frequency: "26 MHz"	>	2MB (FS:512KB OTA:~768KB)
Debug port: "Disabled"	>	2MB (FS:1MB OTA:~512KB)
Flash Size: "4MB (FS:2MB OTA:~1019KB)"	>	2MB (FS:none OTA:~1019KB)
C++ Exceptions: "Disabled (new aborts on oom)"	>	✓ 4MB (FS:2MB OTA:~1019KB)
Flash Frequency: "40MHz"	>	4MB (FS:3MB OTA:~512KB)
Flash Mode: "DOUT (compatible)"	>	4MB (FS:1MB OTA:~1019KB)
lwIP Variant: "v2 Lower Memory"	>	4MB (FS:none OTA:~1019KB)
Built-in Led: "2"	>	8MB (FS:6MB OTA:~1019KB)
Debug Level: "None"	>	8MB (FS:7MB OTA:~512KB)
MMU: "32KB cache + 32KB IRAM (balanced)"	>	8MB (FS:none OTA:~1019KB)
Non-32-Bit Access: "Use pgm_read macros for IRAM/PROGMEM"	>	16MB (FS:14MB OTA:~1019KB)
Reset Method: "dtr (aka nodemcu)"	>	16MB (FS:15MB OTA:~512KB)
NONOS SDK Version: "nonos-sdk 2.2.1+100 (190703)"	>	16MB (FS:none OTA:~1019KB)
SSL Support: "All SSL ciphers (most compatible)"	>	512KB (FS:32KB OTA:~230KB)
Stack Protection: "Disabled"	>	512KB (FS:64KB OTA:~214KB)
		512KB (FS:128KB OTA:~182KB)

Abbildung 35: Board-Einstellung Flash Size

## 13. HTU – OLED – NTP - WiFi-Manager

### 13.1. Funktionsbeschreibung

Es wird das Projekt **06\_HTU\_NTP\_OLED** mit dem Sensor HTU21 mit NTP-Zeit aus dem Internet und der OLED-Anzeige kopiert und angepasst.

In diesem Fall übernimmt der WiFi-Manager die Funktionen der WLAN-Einstellungen und Verbindungsaufbau, somit wird die **myWiFi.h** in diesem Projekt nicht mehr benötigt. Wegen der Nachvollziehbarkeit wird es in diesem Projekt nicht gelöscht, sondern auskommentiert.

Wir kopieren **myWiFi\_Manager.h** in unser Verzeichnis und ändern die **setup()** Funktion.

Die Hauptschleife **loop()** ergänzen wir mit der Button-Abfrage zum Start des WiFi-Managers und mit einem WiFi-Check.

Anzeige für HTU-Sensor und NTP-Zeit bleiben gleich.

### 13.2. Neuer Programmcode

#### **08\_HTU\_NTP\_OLED\_WiFi.ino**

- Kopiere das Verzeichnis Projekt **06\_HTU\_NTP\_OLED**
- Vergib dem Verzeichnis und der **ino-Datei** einen anderen Namen.
- Kopiere **myWiFi\_Manager.h**
- **myWiFi.h** benötigen wir nicht mehr.
- *Passe den Programmcode in der ino-Datei an*
- Führe Anpassungen in **myWiFi\_Manager.h** bei Bedarf durch.

### 13.3. Bibliothek einbinden/installieren

```
#include "myNTP_Time.h"           // für NTP Internet Zeit-Datum
#include "myOLED_u8g2.h"         // für Anzeige am OLED
#include "myWiFi_Manager.h"     // WiFi-Manager
#include <Adafruit_HTU21DF.h>    // Sensor
```

In **myNTP\_Time.h** wird das **#include "myWiFi.h"** auskommentiert da diese Funktionen der WiFi-Manager übernimmt.

In diesem Fall ist die Reihenfolge der **#include ...** von Bedeutung. Insbesondere dann, wenn OLED-Ausgaben bei den WiFi-Manager Funktionen gemacht werden.

### 13.4. Programmcode Ausgabe

```

08:06:00.121 -> Datum und Uhrzeit aus dem Internet per NTP!
08:06:00.121 -> WiFi-Manager für IoT ohne WebServer.
08:06:00.121 -> Start mit AP und 192.168.4.1 wenn kein WLAN verfügbar.
08:06:00.121 -> Wenn das verfügbare WLAN Probleme macht, dann mit der Taste an GPIO14 (D5) den WiFi-Manager erzwingen
08:06:00.154 ->
08:06:12.969 -> *wm:AutoConnect
08:06:13.044 -> *wm:Connecting to SAVED AP: HSB-Gast
08:06:13.572 -> *wm:connectTimeout not set, ESP waitForConnectResult...
08:06:14.678 -> *wm:AutoConnect: SUCCESS
08:06:14.678 -> *wm:STA IP Address: 10.114.11.45
08:06:14.723 -> Erfolgreich verbunden mit: HSB-Gast
08:06:14.723 -> WiFi-Signal 31 dB
08:06:14.756 -> NTP Synchronisation erfolgreich
08:06:14.799 -> !
08:06:14.799 ->
08:06:14.799 -> 26.06.2025      08:06:15      Donnerstag
08:06:15.826 -> 26.06.2025      08:06:16      Donnerstag
08:06:16.836 -> 26.06.2025      08:06:17      Donnerstag
08:06:17.907 -> 26.06.2025      08:06:18      Donnerstag
08:06:18.942 -> 26.06.2025      08:06:19      Donnerstag
08:06:19.987 -> 26.06.2025      08:06:20      Donnerstag
08:06:21.018 -> 26.06.2025      08:06:21      Donnerstag
08:06:22.027 -> 26.06.2025      08:06:22      Donnerstag
08:06:22.923 ->
08:06:22.923 -> Temp: 26.17 °C      Humidity: 50.08 %
08:06:22.923 ->
08:06:31.102 -> 26.06.2025      08:06:31      Donnerstag
08:06:32.113 -> 26.06.2025      08:06:32      Donnerstag
08:06:33.124 -> 26.06.2025      08:06:33      Donnerstag
08:07:59.810 ->
08:08:00.052 -> Checking WiFi... WiFi ok!
08:08:07.959 -> 26.06.2025      08:08:08      Donnerstag
08:08:09.026 -> 26.06.2025      08:08:09      Donnerstag

```

Abbildung 36 Ausgabe am seriellen Monitor

Und siehe Web-Server Anzeige vom WiFi-Manager.

### 13.5. Fehlerbehandlung

### 13.6. Aufgabe

- Erweitere die OLED – Ausgabe um die Status – Anzeige vom WLAN
  - SSID-Verbindung
  - IP-Adresse, MAC-Adresse
  - Verbindung erfolgreich/fehlgeschlagen
  - Wifi check

Siehe **09\_HTU\_NTP\_OLED\_WiFi\_2.ino**

## 14. HTU mit MQTT-Broker Verbindung

### 14.1. Funktionsbeschreibung

Das Projekt mit dem Sensor HTU21 wird erweitert um den Datenaustausch mit einem MQTT-Broker.

Es wird das Projekt **01\_myHTU21DF\_Test.ino** mit dem Sensor HTU21 kopiert und angepasst.

Im *setup()* wird der Sensor gestartet, die WLAN – Verbindung hergestellt und der MQTT-Broker eingestellt und die MQTT-Verbindung aufgebaut. Ausgaben am seriellen Monitor informieren über den aktuellen Stand im Programm.

Es werden alle ausgesendeten Daten auch über den MQTT-Broker wieder zurückgeschickt weil das Haupttopic abonniert wurde.

Im *loop()* wird in bestimmten Zeitintervallen die MQTT-Verbindung geprüft und bei Bedarf wieder hergestellt. Der HTU-Sensor wird in bestimmten Zeitintervallen ausgelesen und die Werte an den MQTT-Broker gesendet. Ausgaben am seriellen Monitor informieren über den aktuellen Stand im Programm.

Bekommt der MQTT-Broker die Daten vom Client, dann schickt er diese weiter an den Client, welche diese Daten (Topic) abonniert haben. Dies kann der gleich ESP oder ein anderer sein. Mit dieser Möglichkeit ist es möglich einen anderen ESP fernzusteuern.

### 14.2. Neuer Programmcode

#### 10\_HTU\_MQTT.ino

- Kopiere das Projekt „01\_myHTU21DF\_Test.ino“
- Ergänze mit den WLAN-Definitionen und Einstellungen
- Ergänze mit den MQTT-Definitionen und Einstellungen

### 14.3. Bibliothek einbinden/installieren

```
#include <ESP8266WiFi.h>    // für WLAN-Verbindung
#include <PubSubClient.h>    // von Nick für MQTT-Datenaustausch
```



## 14.6. Aufgabe

- Erstelle eine Auslagerungsdatei ***myMQTT.h*** mit den relevanten Programmteilen für die Verbindung mit dem MQTT-Broker. Dazu kopiere das Projekt in einen neuen Ordner, benenne diesen um und schiebe die Programmteile für den MQTT – Datenaustausch in die neue Auslagerungsdatei.
- Ergänze das Projekt um die Anzeige am OLED und verwenden dabei die Auslagerungsdatei ***myOLED\_u8g2.h***

## 15. HTU mit MQTT-Broker Auslagerung

### 15.1. Funktionsbeschreibung

Gleich wie bei Projekt vorher, nur mit Auslagerungsdatei für die Wiederverwendung der MQTT-Funktionen.

### 15.2. Neuer Programmcode

#### 11\_HTU\_myMQTT.ino

- Kopiere das Projekt „*HTU mit MQTT-Broker Verbindung*“ in einen neuen Ordner und benenne diesen um.
- Erstelle eine Auslagerungsdatei *myMQTT.h* mit den relevanten Programmteilen für die Verbindung mit dem MQTT-Broker.
- Eine Auslagerungsdatei *myWiFi.h* für die WLAN relevanten Funktionen wäre auch denkbar.

### 15.3. Bibliothek einbinden/installieren

Keine zusätzlichen Bibliotheken notwendig.

### 15.4. Programmcode Ausgabe

```
20:34:49.804 -> ***** HTU21D-F Test mit MQTT - Verbindung
20:34:49.804 ->
20:34:49.804 -> WiFi try to connect.....
20:34:52.919 -> WiFi connected to HSB-Gast
20:34:52.919 -> IP address: 10.114.11.45
20:34:52.919 -> MAC address: 60:01:94:49:99:49
20:34:52.919 ->
20:34:52.919 -> MQTT-Verbindung herstellen mit ESP-4823369-31d3
20:34:53.912 -> MQTT not connected, try to connect...
20:34:54.152 -> MQTT reconnected tomqtt.eclipseprojects.io on port: 1883
20:34:54.198 -> Start loop...
20:34:54.198 ->
20:34:54.635 -> MQTT received: HTU-Temperatur = 26.6 °C
20:34:59.803 -> Temp: 26.60 °C          Humidity: 49.52 %rel
20:34:59.914 -> MQTT received: HTU-Temperatur = 26.6 °C
20:35:09.736 -> MQTT still connected.
20:35:09.907 -> Temp: 26.58 °C          Humidity: 49.47 %rel
20:35:10.309 -> MQTT received: HTU-Temperatur = 26.6 °C
20:35:20.035 -> Temp: 26.57 °C          Humidity: 49.50 %rel
20:35:20.203 -> MQTT received: HTU-Temperatur = 26.6 °C
20:35:29.721 -> MQTT still connected.
20:35:30.118 -> Temp: 26.58 °C          Humidity: 49.53 %rel
```

Abbildung 39: Ausgabe auf dem seriellen Monitor

## 15.5. Fehlerbehandlung

## 15.6. Aufgabe

- Erstelle einen Zähler, welcher die Verbindungsabbrüche zählt und sende den Wert an den MQTT-Broker.
- Erstelle eine Auslagerungsdatei ***myWiFi.h*** für die Wiederverwendung von WiFi-Funktionen
- Erweitere um die NTP-Zeit aus dem Internet mit der Auslagerungsdatei ***myNTP.h***.
- Erweitere um die Anzeige auf dem OLED mit der Auslagerungsdatei ***myOLED\_u8g2.h*** und ***myLogo\_u8g2.h***

## 16. OLED mit MQTT-Broker und WiFi-Manager mit Erweiterung

### 16.1. Funktionsbeschreibung

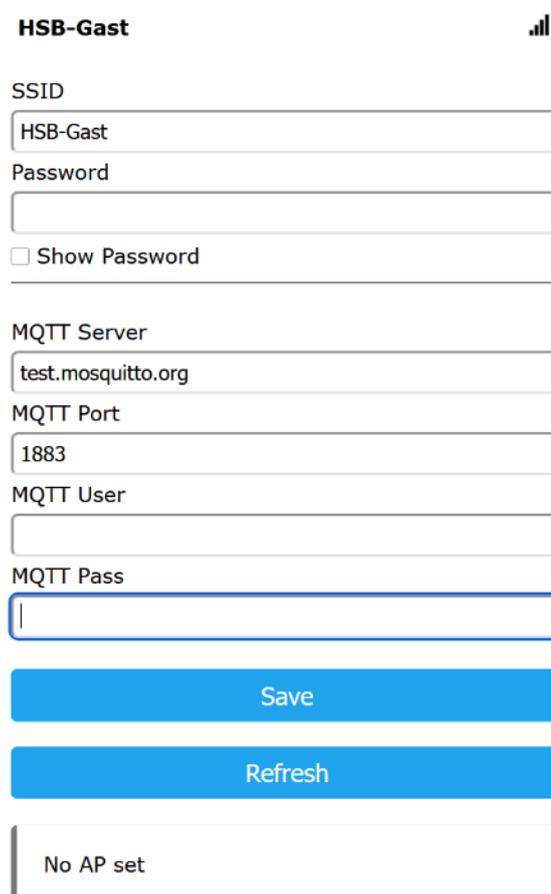
Die WLAN und die MQTT-Verbindungsdaten werden in einem nichtflüchtigen Speicher (LittleFS) im ESP abgelegt. Mit diesen Daten werden eine WLAN-Verbindung und eine MQTT-Verbindung aufgebaut. Scheitert der Verbindungsaufbau, dann wird ein Accesspoint (AP) vom ESP gestartet. Mit dem Computer oder dem Handy kann man sich mit diesem WLAN-Accesspoint **ESP\_MQTT\_Config** verbinden. Mit einem Internet-Explorer und der IP-Adresse **192.168.4.1** wird der WiFi-Manager erreicht und die Verbindungsdaten von WiFi und MQTT können eingegeben werden.

Auf dem OLED wird der aktuelle Status der Verbindungen angezeigt.

Im Hauptprogramm wird nur überprüft, ob die WLAN-Verbindung und die MQTT-Verbindung in Ordnung sind.

Es gibt keine Programminhalte. Datenaustausch mit dem MQTT-Broker erfolgt bereits mit dem Austausch von Verbindungsdaten.

Wird der Accesspoint **ESP\_MQTT\_Config** gestartet dann ist dieser mit einem Internet-Explorer über die IP-Adresse **192.168.4.1** erreichbar.



The screenshot shows a web interface for configuring WiFi and MQTT settings. At the top, it displays the network name 'HSB-Gast' and a signal strength indicator. Below this are several input fields: 'SSID' (containing 'HSB-Gast'), 'Password' (empty), and a 'Show Password' checkbox. A horizontal line separates the WiFi settings from the MQTT settings. The MQTT section includes fields for 'MQTT Server' (containing 'test.mosquitto.org'), 'MQTT Port' (containing '1883'), 'MQTT User' (empty), and 'MQTT Pass' (empty). At the bottom of the form are two blue buttons labeled 'Save' and 'Refresh'. Below the buttons is a status box that currently displays 'No AP set'.

Abbildung 40: WiFi-Manager mit MQTT-Verbindungsdaten

Hier werden die Verbindungsdaten für WLAN und MQTT eingegeben und gespeichert. Danach wird mit den neuen Daten der Verbindungsaufbau durchgeführt und geprüft.

Mit einer Taste an D5 (GPIO\_14) gegen Masse kann beim Hochstarten nach dem Reset der Start des WiFi-Managers erzwungen werden, um neue Daten einzugeben.

## 16.2. Neuer Programmcode

### 12\_OLED\_WiFi-Manager2.ino

Es ist darauf zu achten, dass der MQTT-Client einen anderen Namen hat wie der WiFi-Client.

Achtung: Beim erstmaligen Starten auf einem neuen ESP ist eine Zeile in *myWiFi\_Manager.h* durchzuführen:

```
„LittleFS.format(); // Nur beim ersten Mal oder bei Problemen“
```

## 16.3. Bibliothek einbinden/installieren

Im Hauptprogramm .ino

```
#include "myOLED_u8g2.h" // Auslagerung der OLED-Funktionen  
#include "myWiFi_Manager.h" // WiFi-Manager mit MQTT-Erweiterung
```

In *myWiFi\_Manager.h*

```
#include <WiFiManager.h> // https://github.com/tzapu/WiFiManager  
#include <PubSubClient.h> // Für MQTT  
#include <ArduinoJson.h> // https://arduinojson.org/  
#include <LittleFS.h> // nichtflüchtiger Speicher
```

Nach „*PubSubClient mqttClient(espClient);*“ wird ...

```
#include "myMQTT_Data.h" // für den MQTT-Datenaustausch
```

In *myMQTT\_Data.h* sind enthalten die Topics und die Funktionen zum Empfangen und Versenden der MQTT-Daten.

In *myOLED\_u8g2.h*

```
#include <Wire.h> // für i2c  
#include <Arduino.h>  
#include <U8g2lib.h>  
#include "myLogo_u8g2.h"
```

## 16.4. Programmcode Ausgabe

```

20:07:05.746 -> WiFi Manager für IoT ohne WebServer.
20:07:05.746 -> Start mit AP und 192.168.4.1 wenn kein WLAN verfügbar.
20:07:05.746 -> Mit MQTT Einstellungen
20:07:05.746 -> Wenn das verfügbare WLAN Probleme macht, dann mit der Taste an D5 (GPIO_14) den WiFi-Manager erzwingen
20:07:05.746 ->
20:07:24.171 -> *wm:AutoConnect
20:07:24.295 -> *wm:Connecting to SAVED AP: HSB-Gast
20:07:24.784 -> *wm:connectTimeout not set, ESP waitForConnectResult...
20:07:24.946 -> *wm:AutoConnect: SUCCESS
20:07:24.946 -> *wm:STA IP Address: 10.114.11.45
20:07:25.026 -> Saved config: test.mosquitto.org:1883
20:07:25.026 -> MQTT verbunden mit: MQTT Broker: test.mosquitto.org:1883
20:07:25.026 -> Erfolgreich verbunden mit: HSB-Gast
20:07:25.061 ->
20:07:25.061 -> Start Loop.
20:07:25.061 ->
20:07:25.094 -> Versuche MQTT-Verbindung... Domain-Name erkannt, Erreichbarkeit wird geprüft...
20:07:25.128 -> Erfolgreich aufgelöst: test.mosquitto.org -> 5.196.78.28
20:07:29.676 -> MQTT fehlgeschlagen, Fehler=Nicht autorisiert
20:07:29.713 -> ...try again in 5 seconds
20:07:34.735 -> Versuche MQTT-Verbindung... Domain-Name erkannt, Erreichbarkeit wird geprüft...
20:07:34.735 -> Erfolgreich aufgelöst: test.mosquitto.org -> 5.196.78.28
20:07:37.504 -> MQTT fehlgeschlagen, Fehler=Nicht autorisiert
20:07:37.541 -> ...try again in 5 seconds
20:07:42.523 -> Versuche MQTT-Verbindung... Domain-Name erkannt, Erreichbarkeit wird geprüft...
20:07:42.523 -> Erfolgreich aufgelöst: test.mosquitto.org -> 5.196.78.28

```

Abbildung 41: Ausgabe am seriellen Monitor bei MQTT-Fehler

```

20:26:53.425 -> WiFi Manager für IoT ohne WebServer.
20:26:53.460 -> Start mit AP und 192.168.4.1 wenn kein WLAN verfügbar.
20:26:53.460 -> Mit MQTT Einstellungen
20:26:53.460 -> Wenn das verfügbare WLAN Probleme macht, dann mit der Taste an D5 (GPIO_14) den WiFi-Manager erzwingen
20:26:53.460 ->
20:27:11.935 -> *wm:AutoConnect
20:27:12.014 -> *wm:Connecting to SAVED AP: HSB-Gast
20:27:12.592 -> *wm:connectTimeout not set, ESP waitForConnectResult...
20:27:13.700 -> *wm:AutoConnect: SUCCESS
20:27:13.700 -> *wm:STA IP Address: 10.114.11.45
20:27:13.769 -> Saved config: test.mosquitto.org:1883
20:27:13.769 -> MQTT verbunden mit: MQTT Broker: test.mosquitto.org:1883
20:27:13.769 -> Erfolgreich verbunden mit: HSB-Gast
20:27:13.802 ->
20:27:13.802 -> Start Loop.
20:27:13.802 ->
20:27:13.873 -> Versuche MQTT-Verbindung... Domain-Name erkannt, Erreichbarkeit wird geprüft...
20:27:13.921 -> Erfolgreich aufgelöst: test.mosquitto.org -> 5.196.78.28
20:27:20.196 -> MQTT verbunden!
20:27:20.196 -> test.mosquitto.org on port: 1883Fertig
20:27:30.259 -> Fertig

```

Abbildung 42: Ausgabe am seriellen Monitor bei erfolgreichem Verbindungsaufbau

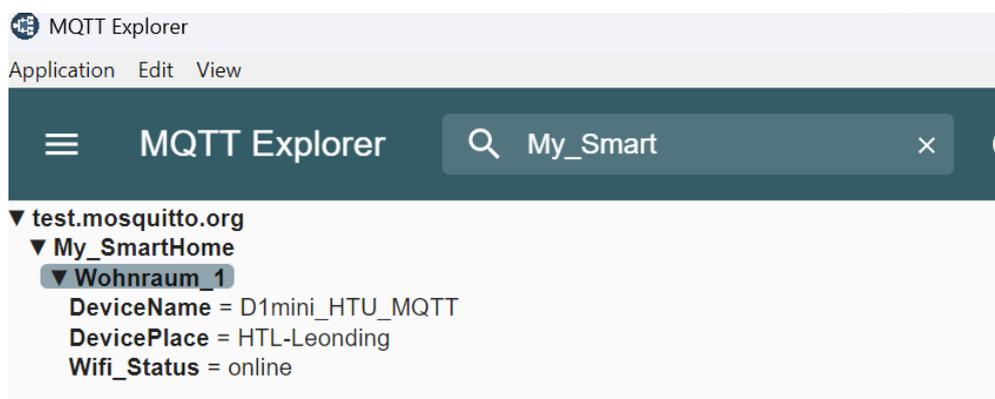


Abbildung 43: Anzeige am MQTT Explorer

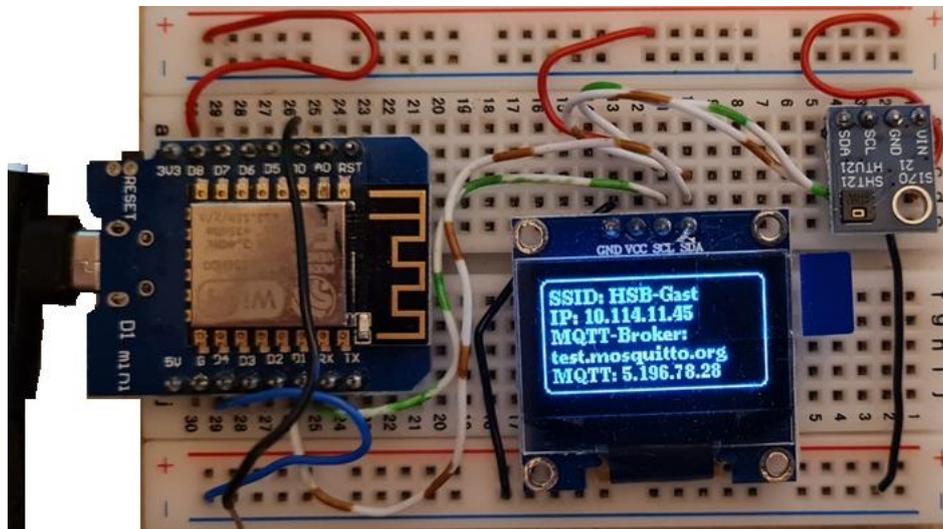


Abbildung 44: Ausgabe auf dem OLED

## 16.5. Fehlerbehandlung

## 16.6. Aufgabe

- Erweitere das Projekt mit dem HTU-Sensor
- Erweitere um die NTP-Zeit aus dem Internet mit der Auslagerungsdatei *myNTP.h*.
- Tausche die HTU-Sensor-Daten mit dem MQTT-Broker aus.
- Erzeuge einen Zeitstempel für den MQTT-Datenaustausch.

### 16.6.1. Siehe 13\_HTU\_OLED\_NTP\_WiFi-Manager2.ino

Es wird mit der Kopie von *12\_OLED\_WiFi-Manager2.ino* begonnen und die HTU und NTP relevanten Funktionen aus *06\_HTU\_NTP\_OLED.ino* hinzugefügt.

Anpassen von *myNTP\_Time.h*, da die WiFi-Funktionen von *myWiFi\_Manager.h* übernommen werden.

Ergänzen der MQTT-Daten um die WLAN-Verbindungsdaten.

## OLED mit MQTT-Broker und WiFi-Manager mit Erweiterung

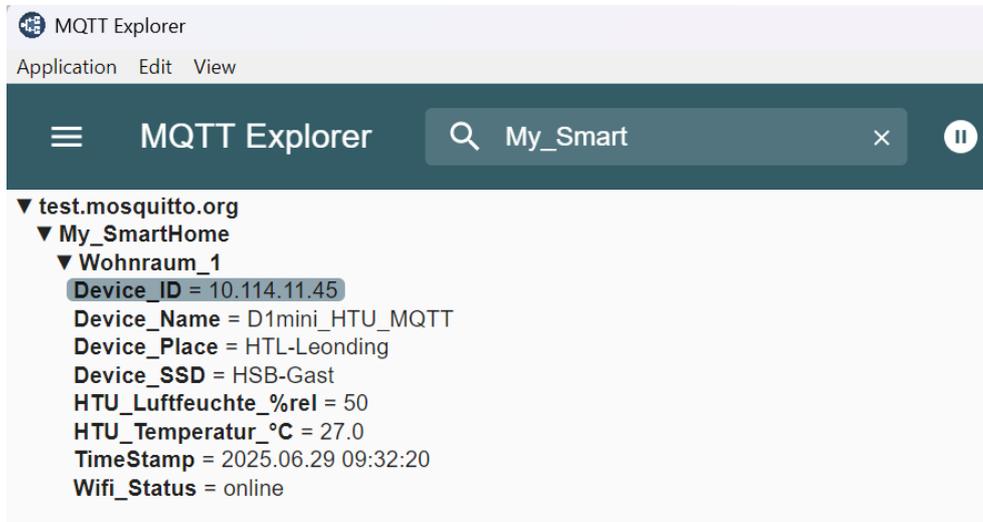


Abbildung 45: Ausgabe am MQTT-Explorer

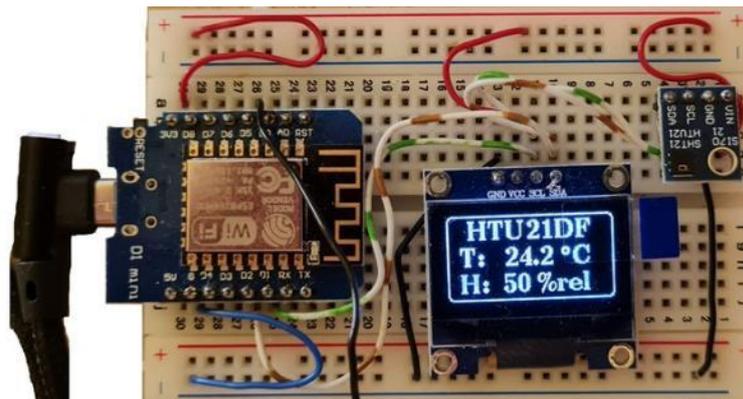


Abbildung 46: Ausgabe der HTU-Sensor Daten auf dem OLED

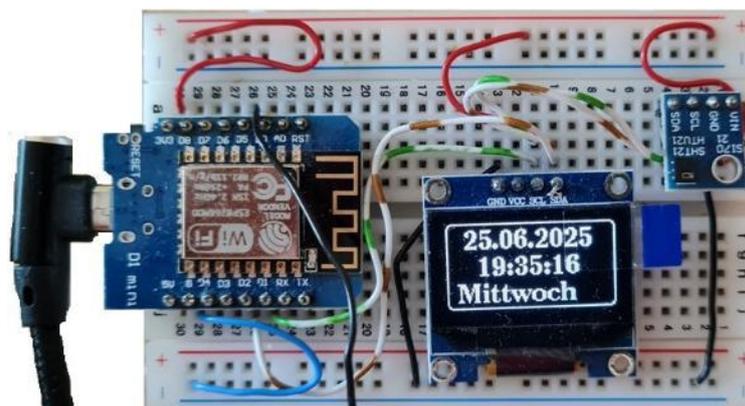


Abbildung 47: Ausgabe von Datum und Uhrzeit auf dem OLED

## 17. HTU mit Web-Server einfach

### 17.1. Webserver Start

<https://randomnerdtutorials.com/esp32-web-server-beginners-guide/>

Zum Einstieg ist die Anleitung von Randomnerdtutorials sehr hilfreich. Dort gibt es auch kostenfreie und kostenpflichtige Bücher zum Download.

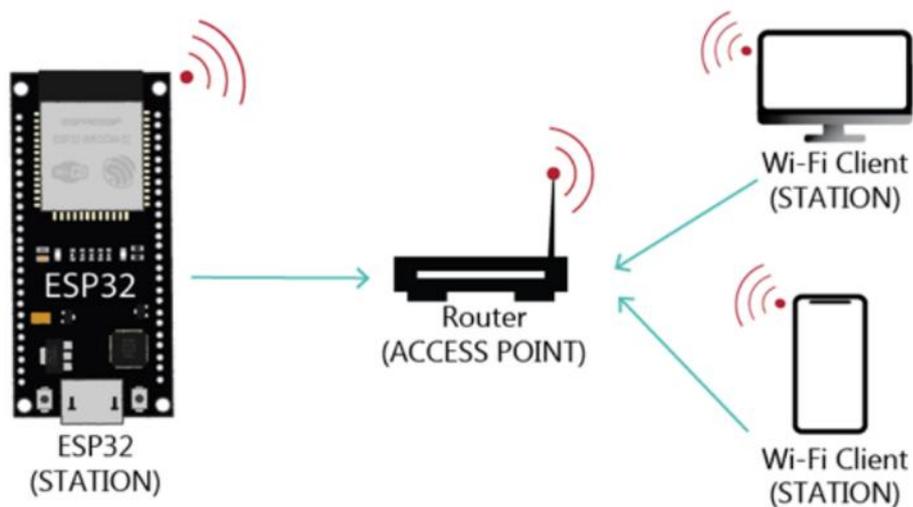


Abbildung 48: Der ESP im Station Mode

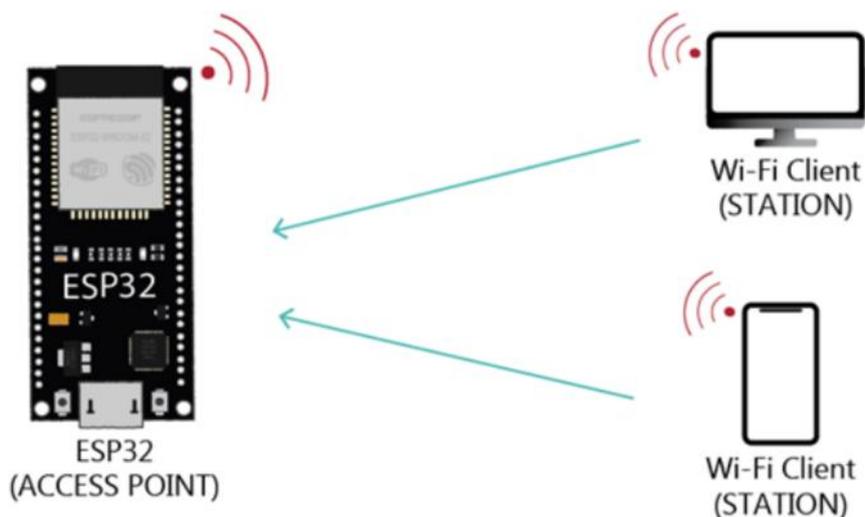


Abbildung 49: Der ESP als Access Point

## 17.2. Funktionsbeschreibung

Zuerst wird mit einem einfachen Webserver für die HTU-Sensor-Daten gestartet. Dazu bedienen wir uns der Anleitung von **RandomNerdTutorial**.

Im Setup wird eine Event-Handler eingesetzt, sodass die Sensorwerte auf der Webseite auch im gleichen Intervall wie die serielle Ausgabe aktualisiert werden.

In Loop werden die Sensorwerte in einem bestimmten Zeitintervall ausgelesen und aktualisiert.

## 17.3. Neuer Programmcode

Es wird eine Anleitung von **RandomNerdTutorial** verwendet und kopieren den Code von:

<https://randomnerdtutorials.com/esp8266-nodemcu-bme680-web-server-arduino/>

in ein neues Projektverzeichnis und geben dem Code einen Namen, wie zum Beispiel:

**`\14_HTU_WebServer`** **`\14_HTU_WebServer.ino`**

- Erstellen von einem neuen Ordner: **`\14_HTU_WebServer`**
- Erstellen einer neuen Textdatei: **`14_HTU_WebServer.ino`**
- Code von **RandomNerdTutorial** kopieren.
- Code anpassen:
  - Zur besseren Übersicht eine Auslagerungsdatei von **`const char index_html[]`** erstellen. „**`myHTML.h`**“
  - HTU-Sensordaten verwenden anstatt BME-Sensor
  - Die seriellen Ausgaben anpassen.
  - Sonst keine Anpassungen, da der WebServer sehr empfindlich auf Fehler reagiert.

## 17.4. Bibliothek einbinden/installieren

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HTU21DF.h>
#include <ESP8266WiFi.h>
#include "ESPASyncWebServer.h" // für ESP8266 und ESP32
```

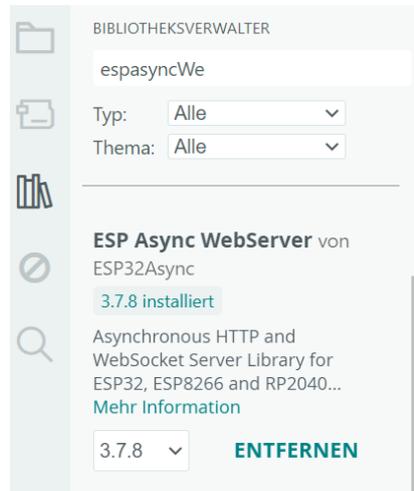


Abbildung 50: Bibliothek ESPAsyncWebServer

## 17.5. Programmcode Ausgabe

```
09:29:52.025 -> s10100|00d0|0000d00c|000000r0b0c0?0?nn0?doc0?0?c0x01{d{dp0o0000?10?
09:29:52.140 ->
09:29:52.140 -> 14 - HTU-Senosr - WebServer !
09:29:52.140 ->
09:29:52.140 ->
09:29:52.265 -> Setting as a Wi-Fi Station.....
09:29:55.276 -> WiFi connected to HSB-Gast      Station IP Address: 10.114.11.45
09:29:55.276 ->
09:29:58.168 -> Temperature = 25.52 °C
09:29:58.168 -> Humidity = 53.72
09:29:58.168 -> Pressure = 0.00 hPa
09:29:58.168 ->
09:30:04.248 -> Temperature = 25.52 °C
09:30:04.248 -> Humidity = 51.11
```

Abbildung 51: Ausgabe am seriellen Monitor

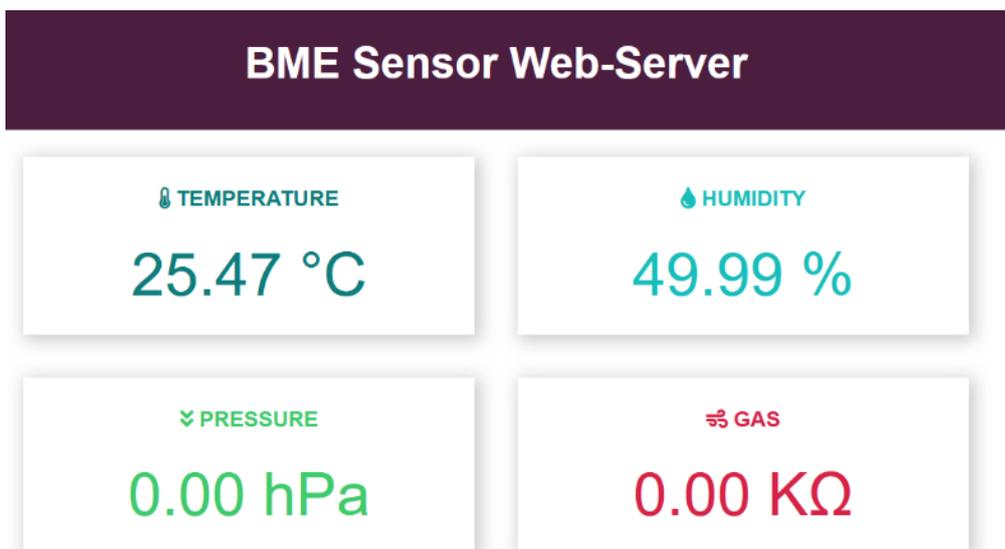


Abbildung 52: Ausgabe vom Web-Server durch Eingabe der IP-Adresse

Wir haben bei unserem Sensor keine Werte für Luftdruck und Gaszusammensetzung, darum sind die Werte 0.00.

### 17.6. Fehlerbehandlung

Achtung: Fehler in der Webseite werden in der Arduino IDE nicht angezeigt und sind somit sehr schwer zu finden.

### 17.7. Aufgabe

- Anpassen der Software und Webseite auf die richtige Ausgabewerte.
  - Code kopieren und umbenennen, danach anpassen  
**14\_HTU\_WebServer2.ino**

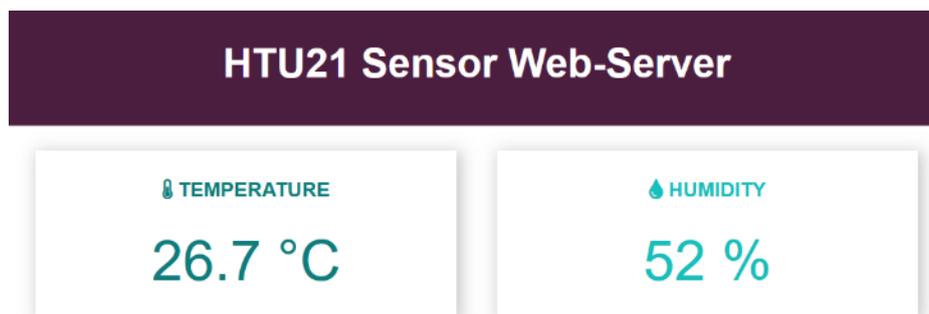


Abbildung 53: Web-Server Anzeige mit HTU-Sensordaten

- Anpassen der Software und Webseite mit
  - OLED-Ausgabe
  - MQTT-Datenaustausch.
  - NTP-Datum und Uhrzeit mit Ausgabe auf seriellen Monitor und OLED.

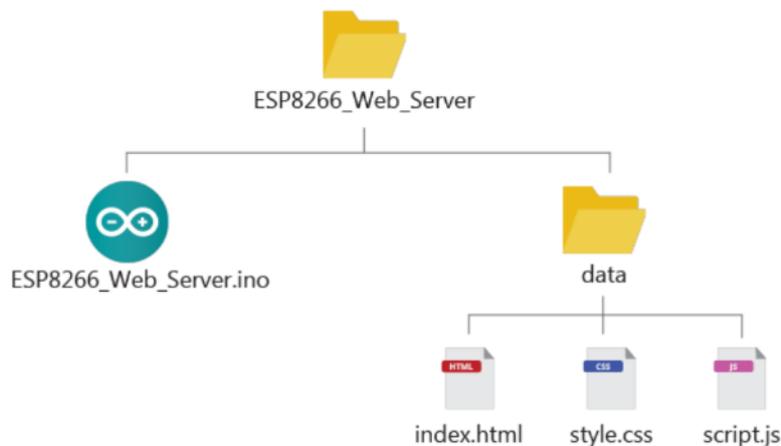
## 18. HTU mit Web-Server mit „Gauges“

### 18.1. Funktionsbeschreibung

Ein aufwändiger Web-Server wird erstellt, dazu bedienen wir uns der Anleitung von **RandomNerdTutorial**. <https://randomnerdtutorials.com/esp8266-web-server-gauges/>

Das Tutorial erklärt alle Details, darum wird hier nicht darauf eingegangen. Es werden nur die Anpassungen gemeinsam durchgeführt. Vorerst laden wir den Original-Code und testen diesen. Bei diesem Projekt wird die Webseite im Filesystem ausgelagert, wobei **LittleFS Filesystem Uploader** benötigt wird. <https://randomnerdtutorials.com/arduino-ide-2-install-esp8266-littlefs/>

- **Arduino sketch** that handles the web server;
- **index.html**: to define the content of the web page;
- **style.css**: to style the web page;
- **script.js**: to program the behavior of the web page—handle web server responses, events, create the gauges, etc.



### 18.2. Neuer Programmcode

Es wird eine Anleitung von **RandomNerdTutorial** verwendet und kopieren den Code von:

<https://randomnerdtutorials.com/esp8266-web-server-gauges/>

in ein neues Projektverzeichnis und geben dem Code einen Namen, wie zum Beispiel:

**`15_HTU_WebServer1\15_HTU_WebServer1.ino`**

- Erstellen von einem neuen Ordner: **`15_HTU_WebServer1`**
- Erstellen einer neuen Textdatei: **`15_HTU_WebServer1.ino`**
- Code von **RandomNerdTutorial** kopieren, dazu wählen wir auf der Webseite [Download All the Arduino Project Files](#)
- Code testen,
- Code anpassen: HTU-Sensor Library einbinden, HTU definieren und starten, Sensor auslesen und Werte anzeigen.

- HTU-Sensordaten verwenden anstatt BME-Sensor
- Die seriellen Ausgaben anpassen, wenn gewünscht.
- Keine Anpassungen an der Webseite. (vorerst)

### 18.3. Bibliothek einbinden/installieren

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include "LittleFS.h"
#include <Arduino_JSON.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HTU21DF.h>
```

Neue Bibliotheken installieren:

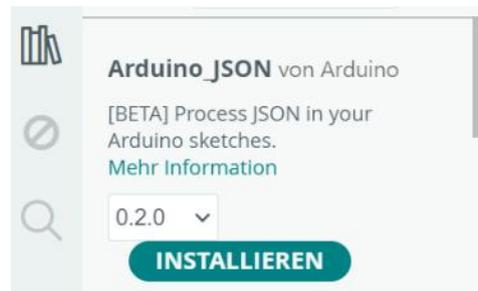


Abbildung 54: Neue Bibliothek installieren

Installieren von **LittleFS Filesystem Uploader**:

- Gehe auf die Webseite:  
<https://github.com/earlephilhower/arduino-littlefs-upload/releases>
- Download von: [arduino-littlefs-upload-1.5.4.vsix](#)
- Erzeuge ein Verzeichnis unter `C:\Users\<username>\.arduino\IDE`
- Erstelle ein Verzeichnis „*plugins*“
- Verschiebe den ...*vsix* Download in dieses Verzeichnis.
- Starte die Arduino IDE 2.xx Software neu. (alle Fenster)
- Es gibt kein Menü unter Werkzeuge zur Durchführung des **Upload LittleFS to ESP**.
- Stelle sicher, dass der serielle Monitor ausgeschaltet/entfernt ist.
- Starte das Command-Fenster
- Drücke [**Ctrl**] + [**Shift**] + [**P**] um ein Command-Fenster zu öffnen und tippe „upload“



Abbildung 55: Command Fenster: Upload LittleFS to ....

- **Starte Upload LittleFS to ESP**

```
MAC: 60:01:94:49:99:49
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 2072576 bytes to 5530...
Writing at 0x00200000... (100 %)
Wrote 2072576 bytes (5530 compressed) at 0x00200000 in 0.5 seconds (effective 34068.5 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

Completed upload.
```

Abbildung 56: Upload LittleFS completed

- Einstellen der Flash Size auf 4MB (FS2MB, OTA 1019kB)
- Erst jetzt den Sketch hochladen
- Jetzt kann bei Bedarf auch wieder der serielle Monitor gestartet werden

Der Upload von LittleFS braucht jetzt nur mehr dann durchgeführt werden, wenn die Daten im Verzeichnis data geändert werden. Das Hochladen von einem neuen Sketch beeinflusst den LittleFS nicht. Ebenso eine Stromunterbrechung macht kein Problem

## 18.4. Programmcode Ausgabe

### ESP WEB SERVER GAUGES

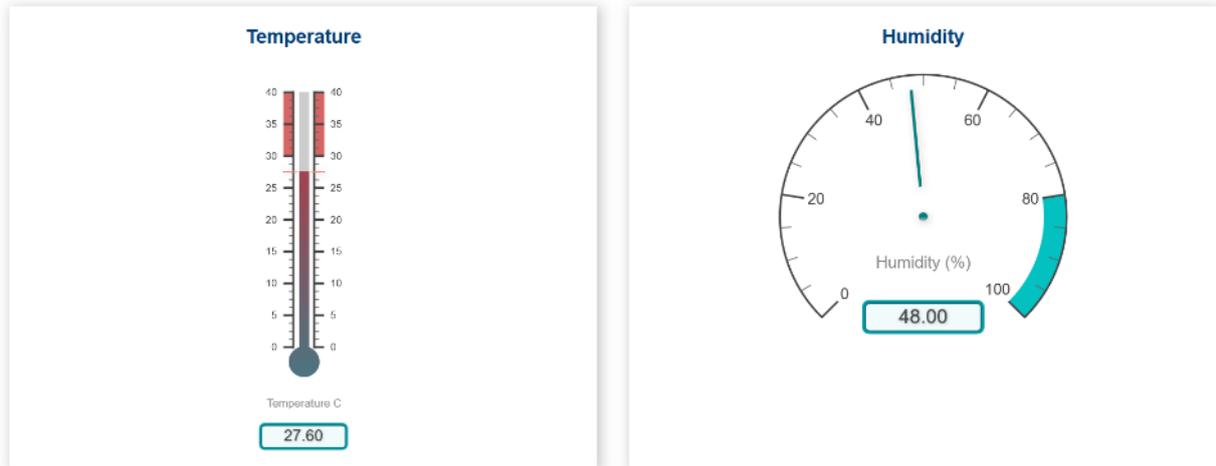


Abbildung 57: Webserver Ausgabe am Internet-Explorer

Diese Webseite ist nur von Geräten innerhalb des eigenen Netzwerkes erreichbar. Von außerhalb kann diese Seite nicht erreicht werden.

## 18.5. Fehlerbehandlung

### 18.6. Aufgabe

- Versuche die Webseite mit dem Smartphone zu erreichen.
- Füge eine Ausgabe der Sensordaten am seriellen Monitor hinzu.
- Ergänze mit OLED - Ausgabe
- Ergänze mit der NTP-Uhrzeit
- Ergänze mit MQTT
- Ergänze mit WiFi-Manager
- Verwende einen BMP280 Sensor mit zusätzlichem Luftdruck und passe die Webseite an.

## 19. HTU mit Web-Server + WebSocket (in Arbeit)

### 19.1. Funktionsbeschreibung

Es ist eventuell aufgefallen, dass bei manchen Web-Server Projekten in manchen Fällen die Webseite nicht automatisch aktualisiert wird, ohne dass man die Seite nicht erneut aktualisiert. Um dieses Problem zu beheben, werden das WebSocket – Protokoll verwendet. Alle Web-Klienten werden bei einer Änderung automatisch aktualisiert.

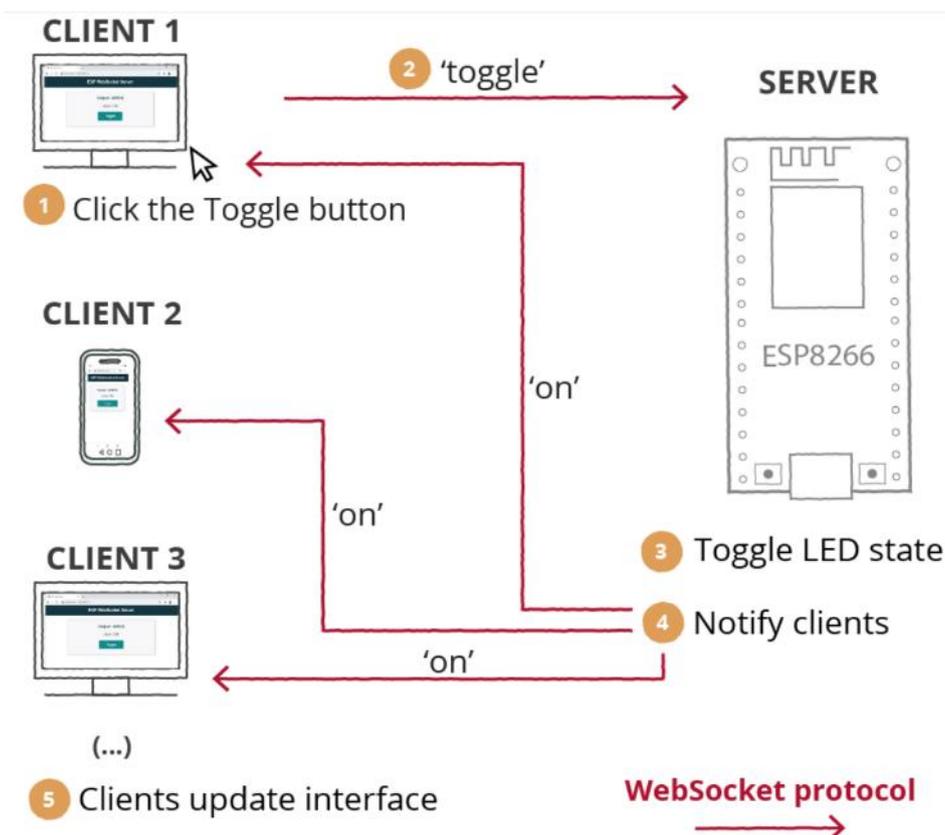


Abbildung 58: Übersicht WebSocket Funktion

### 19.2. Neuer Programmcode

#### 17\_HTU\_WebServer\_WebSocket.ino

Siehe <https://randomnerdtutorials.com/esp8266-nodemcu-websocket-server-arduino/> als Vorlage.

Die Webseite lagern wir wieder in eine myHTML.h aus, damit die Übersichtlichkeit besser wird.

### **19.3. Bibliothek einbinden/installieren**

### **19.4. Programmcode Ausgabe**

### **19.5. Fehlerbehandlung**

### **19.6. Aufgabe**

## 20. HTU mit Async-Web-Server und WiFi-Manager (in Arbeit)

### 20.1. Funktionsbeschreibung

- Es wird ein Webserver erstellt mit den Daten des HTU21 Sensors und zusätzlich dem Status einer LED (Built\_In\_LED).
- Die Daten werden auch mit einem MQTT-Broker ausgetauscht.
- Die Verbindungsdaten für WLAN und MQTT werden über einen WiFi-Manager eingegeben.
- Auf dem seriellen Monitor können die Ausgaben mitverfolgt werden.
- Die Webseiten werden im Filesystem abgelegt (LittleFS)
- Die Daten werden ebenso in einem File abgelegt und im nichtflüchtigen Speicher aufbewahrt (LittleFS)

### 20.2. Neuer Programmcode

#### 16\_HTU\_OLED\_NTP\_Async\_WebServer\_WiFi-Manager.ino

Vorlage ist ein Beispiel von RandomNerdTutorial:

<https://randomnerdtutorials.com/esp8266-nodemcu-wi-fi-manager-asyncwebserver/>

- [Download All the Arduino Project Files](#)
- Entpacken der Zip-Datei. Es gibt wieder ein Verzeichnis `\data`
- Benennen die ino-Datei um in unseren Projektnamen: 16\_HTU....
- Kopieren unserer Auslagerungsdateien für OLED, NTP, MQTT, myWiFi aus den Vorprojekten in dieses Verzeichnis.

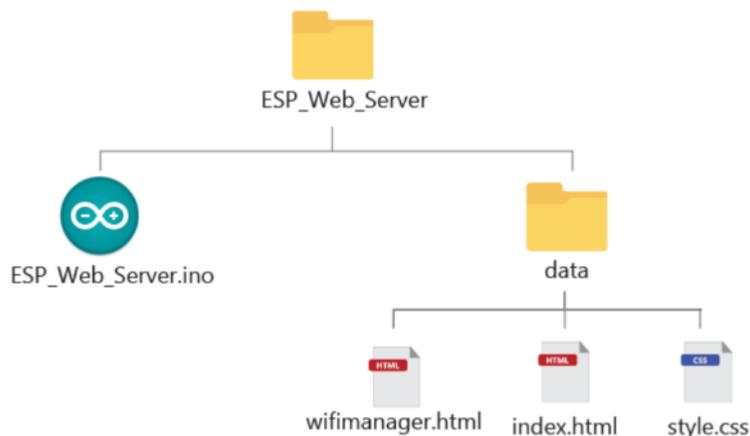


Abbildung 59: Verzeichnis-Struktur vom Beispiel

- Als erstes wird die myWiFi angepasst, da nun die WiFi-Daten vom Speicher vorliegen und nicht mehr „Hardcoded“ sind.

### 20.3. Bibliothek einbinden/installieren

Auf die Reihenfolge der **#include ...** achten, da gegenseitige Abhängigkeiten möglich sind.

- Drücke [**Ctrl**] + [**Shift**] + [**P**] um ein Command-Fenster zu öffnen und tippe „upload“



Abbildung 60: Command Fenster: Upload LittleFS to ....

- Starte **Upload LittleFS to ESP**

### 20.4. Programmcode Ausgabe

### 20.5. Fehlerbehandlung

### 20.6. Aufgabe

## **21. Vorlage (in Arbeit)**

### **21.1. Funktionsbeschreibung**

### **21.2. Neuer Programmcode**

### **21.3. Bibliothek einbinden/installieren**

### **21.4. Programmcode Ausgabe**

### **21.5. Fehlerbehandlung**

### **21.6. Aufgabe**

## **22. Over The Air update (OTA) (in Arbeit)**

### **22.1. Funktionsbeschreibung**

Durch eine Erweiterung kann das Upload einer neuen Software über die WLAN-Verbindung erfolgen.

Nur beim ersten Upload ist eine USB-Verbindung nötig.

### **22.2. Neuer Programmcode**

### **22.3. Bibliothek einbinden/installieren**

### **22.4. Programmcode Ausgabe**

### **22.5. Fehlerbehandlung**

### **22.6. Aufgabe**

## 23. Vorschau

- Ein Webserver mit WiFi-Manager zur Eingabe der WLAN- und MQTT-Verbindungsdaten und Austausch der Sensor-Daten über MQTT
- Anzeige der Sensor-Daten auf seriellem Monitor und OLED oder TFT-Display.
- Einbau eines Tasters wobei beim Tastendruck eine LED ein-/ausgeschaltet wird. Dabei ist die Herausforderung, dass immer der richtige Zustand der LED angezeigt wird.
- Einbau eines Schalters, wobei immer der richtige Zustand des Schalters angezeigt wird.
- Bauen einer Senderstation und einer Empfängerstation. (MQTT, HTML, ...)
  
- Eine Wetterstation mit OLED-Anzeige von:  
<https://randomnerdtutorials.com/esp32-weather-station-pcb/>

Viel Erfolg und Spaß bei den Experimenten



## 24. Anhang:

### 24.1. Abbildungsverzeichnis

Abbildung 1: ESP8266 Wemos D1-mini.....	7
Abbildung 2: HTU21 Temperatur- und Feuchte-Sensor .....	8
Abbildung 3: OLED 0,96" i2c monochrom.....	8
Abbildung 4: Schaltplan.....	9
Abbildung 5: Schaltungsaufbau auf dem Steckboard .....	9
Abbildung 6: Gerätemanager Anschlüsse.....	10
Abbildung 7: Arduino Software Download .....	11
Abbildung 8: Arduino IDE .....	11
Abbildung 9: Einstellungen .....	12
Abbildung 10: Zusätzliche Boardverwalter für die WLAN-Chips .....	12
Abbildung 11: esp8266 und esp32 installiert.....	13
Abbildung 12: Beide neuen Boards installiert.....	14
Abbildung 13: Board auswählen.....	14
Abbildung 14: Board Parameter einstellen.....	15
Abbildung 15: Beispiel-Programm Blink .....	16
Abbildung 16: Programm prüfen und hochladen.....	16
Abbildung 17: Ausgabe nach dem Prüfen ohne Fehler.....	17
Abbildung 18: Ausgabe nach dem erfolgreichen Hochladen .....	17
Abbildung 19: Beispiel Programm Blink .....	18
Abbildung 20: Neuer Programmcode und Bibliothek installieren .....	19
Abbildung 21: Bibliothek von github .....	20
Abbildung 22: Beispiel Programm HTU21DF.....	20
Abbildung 23: Ausgabe am seriellen Monitor.....	21
Abbildung 24: Bibliothek U8g2 von oliver installieren.....	22
Abbildung 25: OLED-Ausgabe .....	23
Abbildung 26: Programme Reiter .....	24
Abbildung 27: Ausgabe auf dem seriellen Monitor.....	25
Abbildung 28: OLED-Ausgabe .....	25
Abbildung 29: Ausgabe am seriellen Monitor.....	26
Abbildung 30: Hauptcode mit Auslagerungsdateien .....	28
Abbildung 31: Ausgabe am seriellen Monitor.....	29
Abbildung 32: Ausgabe auf OLED.....	29
Abbildung 33: Ausgabe am seriellen Monitor.....	35
Abbildung 34: WiFi - Manager Ausgabe.....	35
Abbildung 35: Board-Einstellung Flash Size .....	36
Abbildung 36 Ausgabe am seriellen Monitor.....	38
Abbildung 37: Ausgabe am seriellen Monitor.....	40
Abbildung 38: Kontrolle mit dem MQTT Explorer .....	40
Abbildung 39: Ausgabe auf dem seriellen Monitor.....	42
Abbildung 40: WiFi-Manager mit MQTT-Verbindungsdaten .....	44
Abbildung 41: Ausgabe am seriellen Monitor bei MQTT-Fehler .....	46
Abbildung 42: Ausgabe am seriellen Monitor bei erfolgreichem Verbindungsaufbau.....	46
Abbildung 43: Anzeige am MQTT Explorer.....	46
Abbildung 44: Ausgabe auf dem OLED .....	47
Abbildung 45: Ausgabe am MQTT-Explorer .....	48
Abbildung 46: Ausgabe der HTU-Sensor Daten auf dem OLED .....	48
Abbildung 47: Ausgabe von Datum und Uhrzeit auf dem OLED .....	48
Abbildung 48: Der ESP im Station Mode .....	49
Abbildung 49: Der ESP als Access Point .....	49
Abbildung 50: Bibliothek ESPAsyncWebServer.....	51
Abbildung 51: Ausgabe am seriellen Monitor.....	51
Abbildung 52: Ausgabe vom Web-Server durch Eingabe der IP-Adresse.....	51
Abbildung 53: Web-Server Anzeige mit HTU-Sensordaten .....	52

**Anhang:**

Abbildung 54: Neue Bibliothek installieren .....	54
Abbildung 55: Command Fenster: Upload LittleFS to .....	55
Abbildung 56: Upload LittleFS completed .....	55
Abbildung 57: Webserver Ausgabe am Internet-Explorer .....	56
Abbildung 58: Übersicht WebSocket Funktion .....	57
Abbildung 59: Verzeichnis-Struktur vom Beispiel .....	59
Abbildung 55: Command Fenster: Upload LittleFS to .....	60