Function description for:
ESP8266 NodeMCU
ESP32 DevKitC V4
Mosquitto MQTT broker
NodeRed
ESP32 / SolarMax4200S
V1.00, 2023-04-01, Armin Rehberger

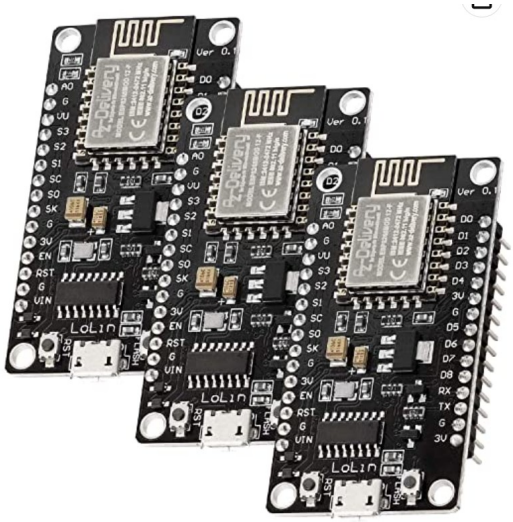
Table of content

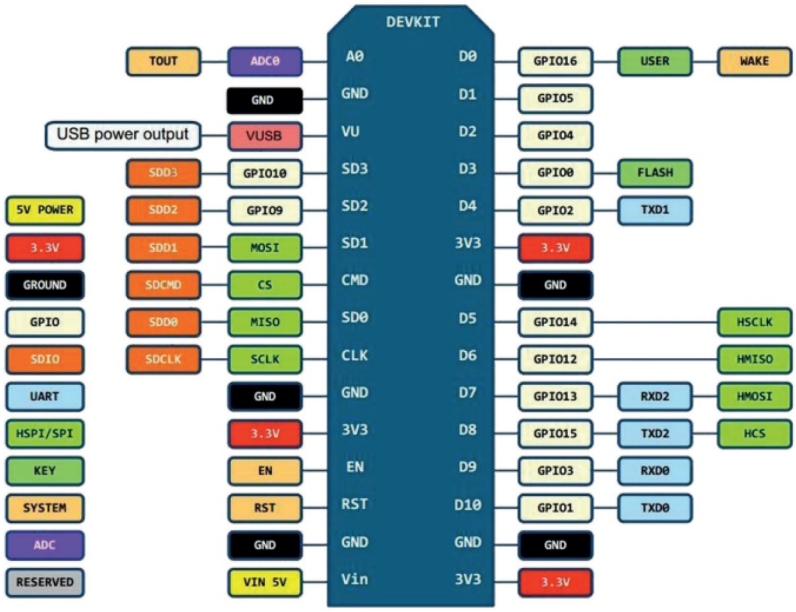
General description

MQTT broker mosquitto, running on a Raspberry PI (Linux) or Windows.
ESP8266 NodeMCU or ESP32, connected to WIFI network, send PubSub messages to the MQTT broker.
GUI programmed with Python, run on the Raspberry PI (Linux), to virtualize the data send from the ESP8266 NodeMCU or ESP32 to the MQTT broker.
GUI NodeRed running on a Raspberry PI (Linux) or Windows to virtualize the data send from the ESP8266 NodeMCU or ESP32 to the MQTT broker.

Hardware NodeMCU Lua Lolin V3 ESP8266MOD 12-F

AZDelivery 3 x NodeMCU Lolin V3 Module ESP8266 ESP-12F WiFi WiFi Development Board mit CH340 kompatibel mit Arduino
https://www.amazon.de/gp/product/B074Q2WM1Y/ref=ox_sc_act_image_1?smid=A1X7QLRQH87QA3&th=1

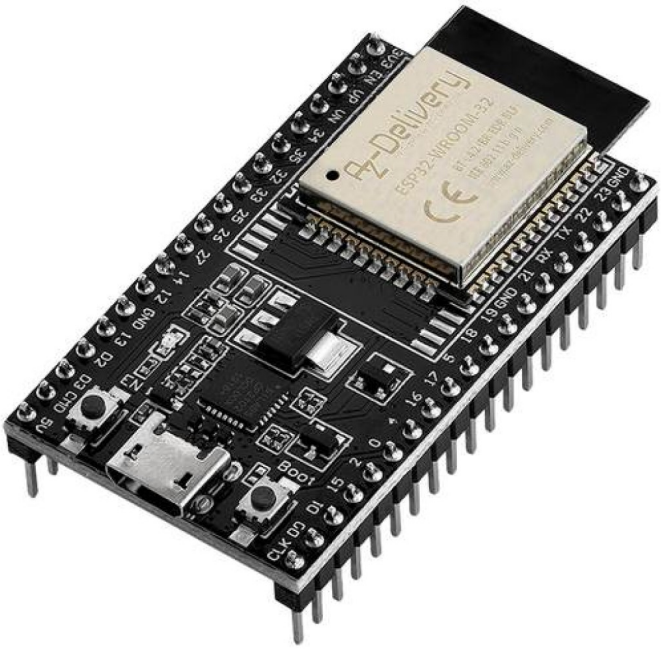


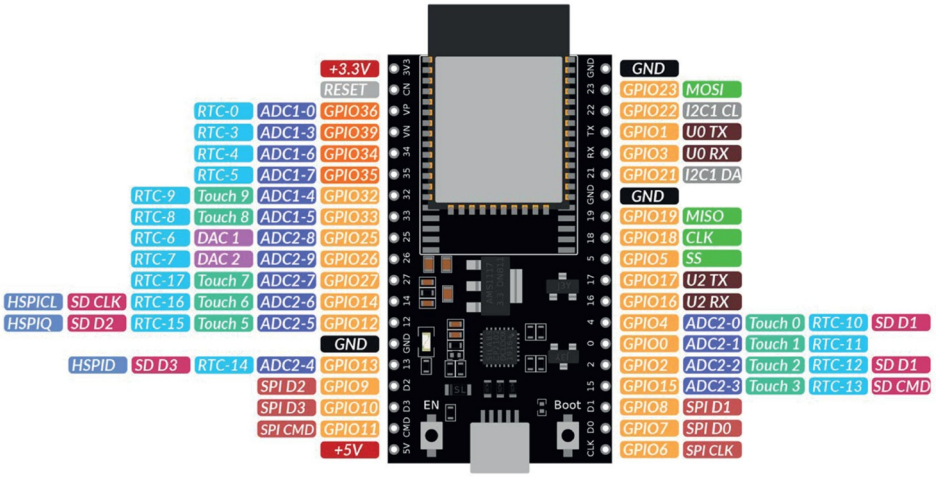


Hardware ESP32 DevKitC V4

AZDelivery 3er Set ESP32 Dev Kit C V4 NodeMCU WLAN/WiFi Development Board unverlötet kompatibel mit Arduino inklusive E-Book! (Nachfolger Modul von ESP32 Dev Kit C)
https://www.amazon.de/dp/B08BZGC22Q?psc=1&ref=ppx_yo2ov_dt_b_product_details

ESP32 has no LED on board. To download a program the button „BOOT“ has to be pressed.



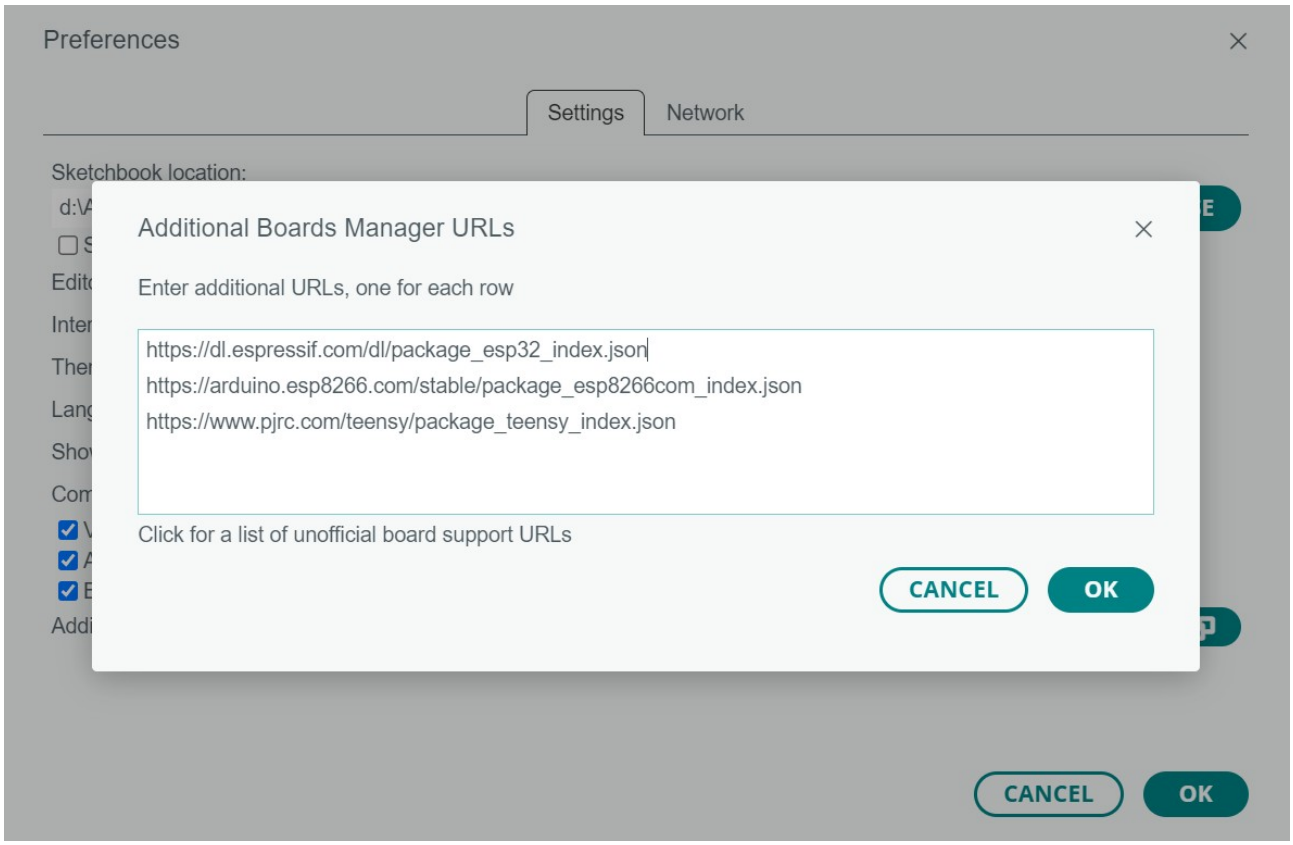


Digital In/Out ports (all support PWM)
Digital Input ports
Analog Input 12 bits, 0 to 3.3V
Analog Output 8 bits, 0 - 3.3V
Capacitive Touch Sensor ports
I/O -pins from RTC ultra low power processor, usable in deep sleep mode
SD card interface
SPI bus for Flash-memory, do not use

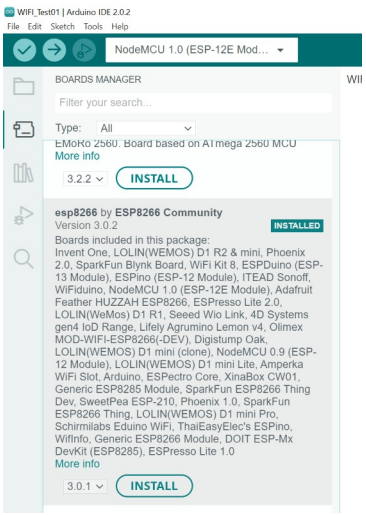
The following pins show the default assignment. All these signals can be changed to any In/Out port.
This applies also to UART0 and UART1, which cannot be accessed in the default assignment.
I2C bus (Wire)
VSPI bus
Serial interfaces
HSPI bus

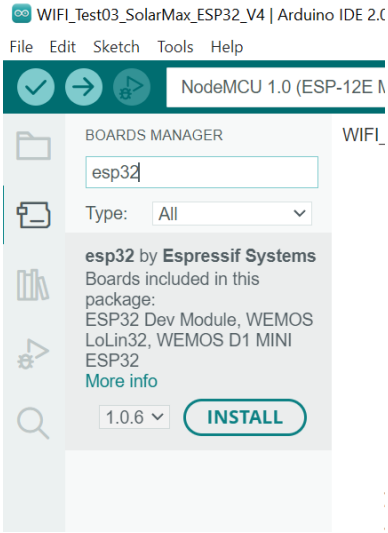
Arduino IDE for ESP8266 or ESP32

1. Preferences / Board Manager
ESP8266, add: https://arduino.esp8266.com/stable/package_esp8266com_index.json
ESP32 add: https://dl.espressif.com/dl/package_esp32_index.json

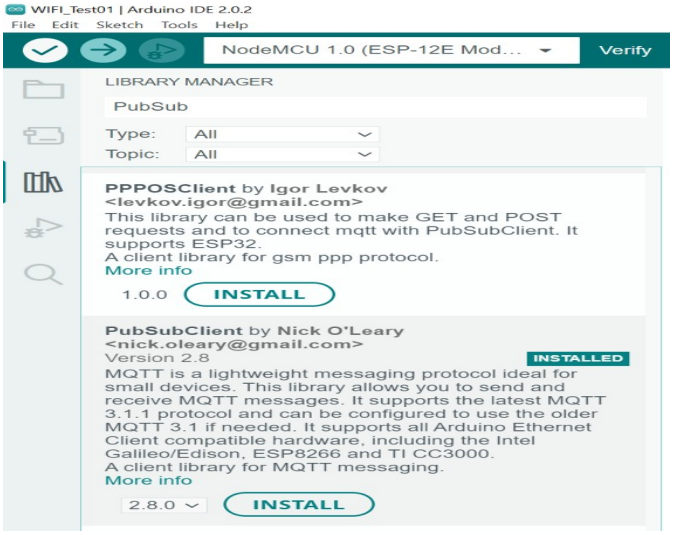


2. Boards Manager:
Install esp8266
Install esp32

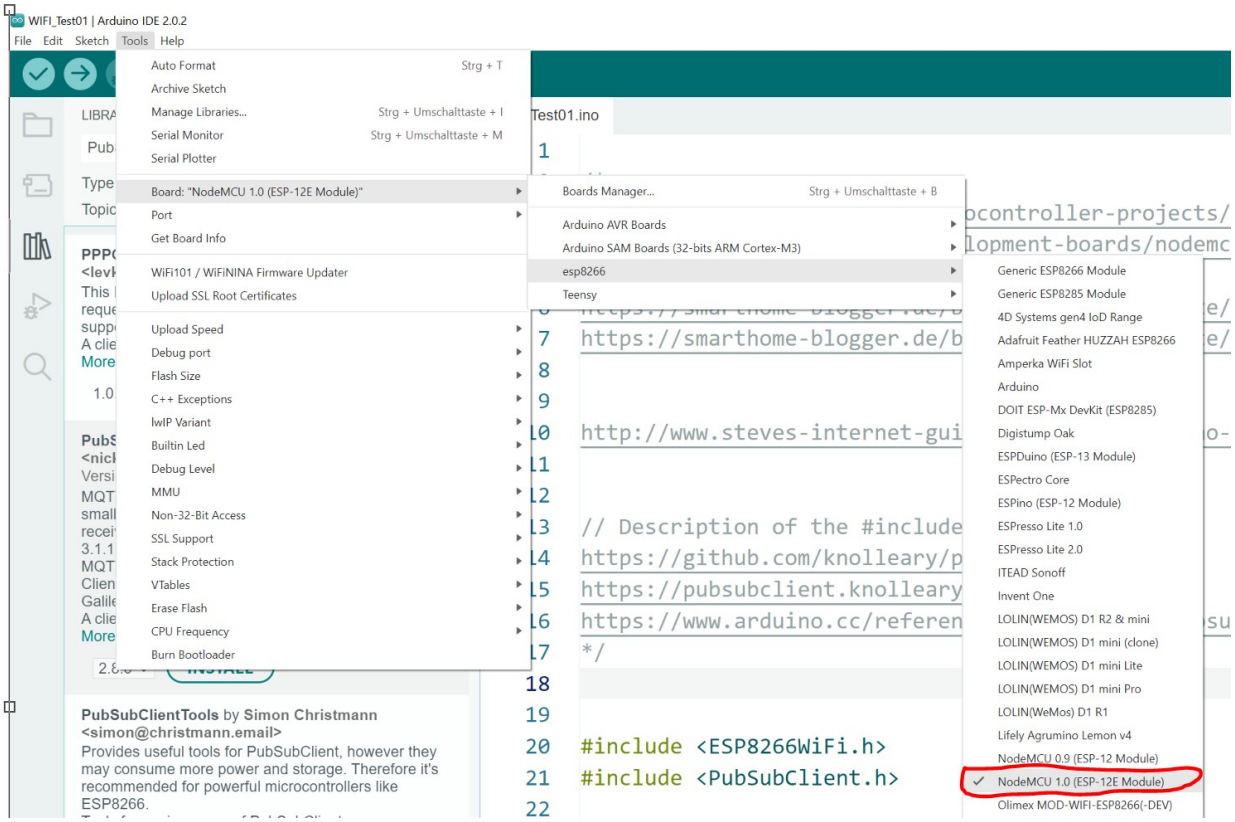




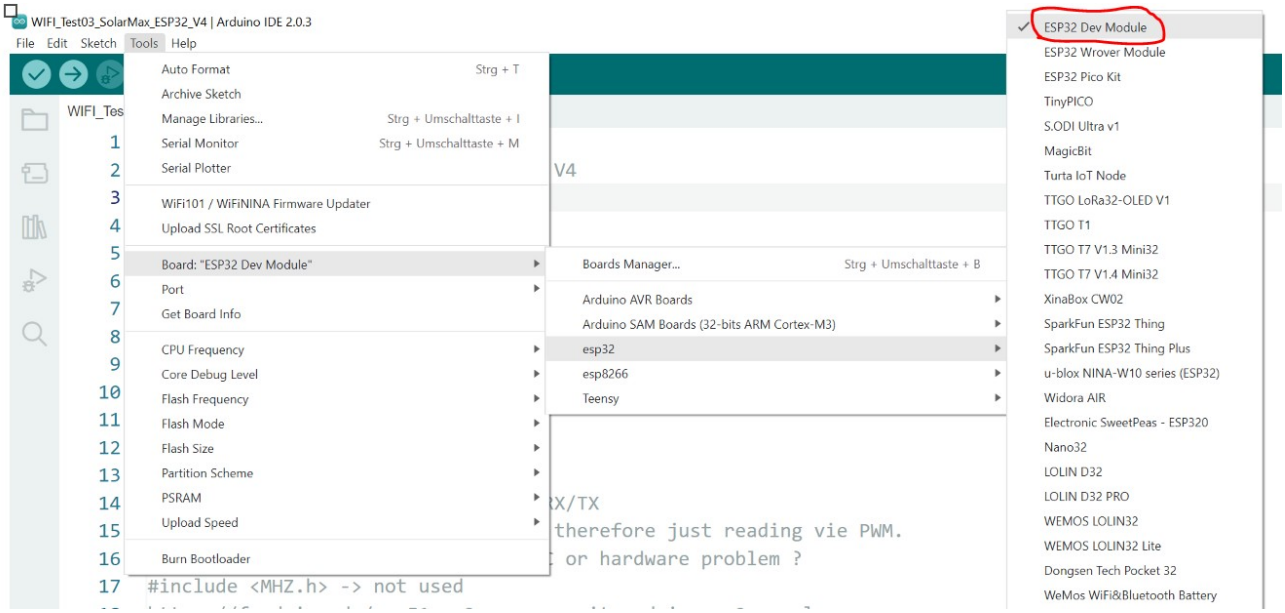
3. Library Manager: Install PubSubClient by Nick O'Leary



4. Select board:
ESP8266 NodeMCU 1.0 (ESP-12E Module)

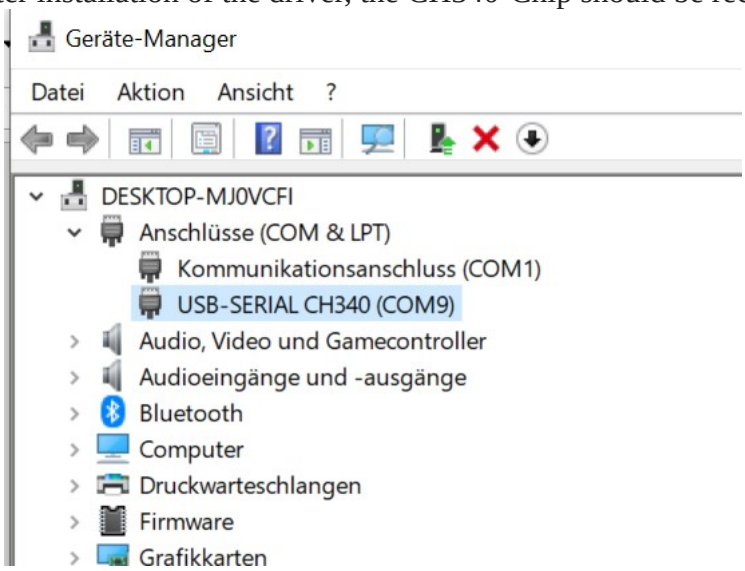


ESP32 DevModule:



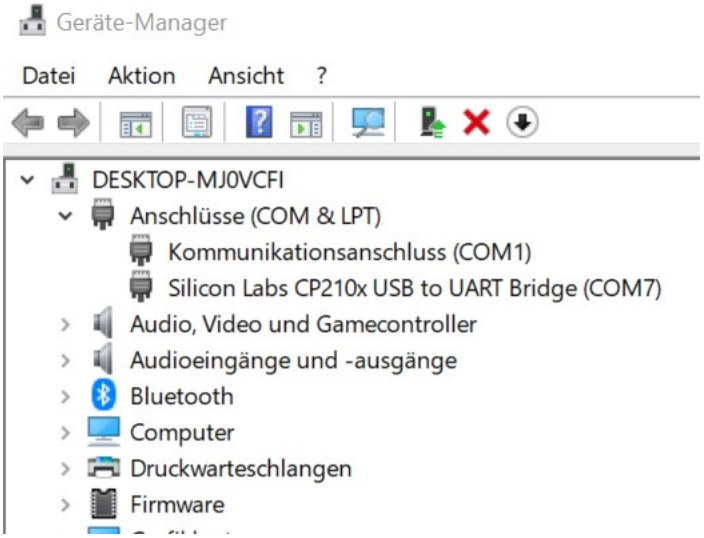
5.0 ESP8266: Communication via USB, driver CH340-Chip
If your Windows doesn’t automatically recognize your board, then install the following driver:
http://www.wch.cn/download/CH341SER_ZIP.html

After installation of the driver, the CH340-Chip should be recognized.

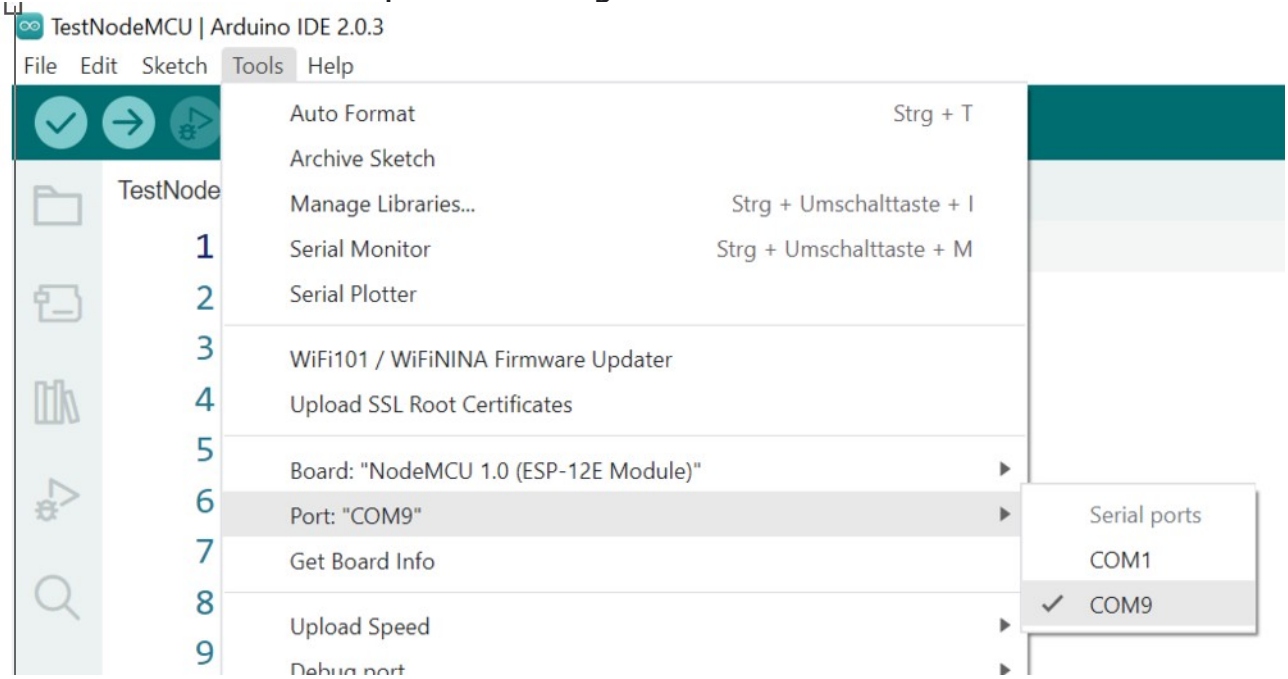


5.1 ESP32: Communication via USB driver CP2102-Chip
If your Windows doesn’t automatically recognize your board, then install the following driver:
https://www.silabs.com/interface/usb-bridges/classic/device.cp2102?tab=specs

After installation of the driver, the CH210x-Chip should be recognized.



6. Arduino IDE, select USB port: Tools / e.g. Port “COM9”



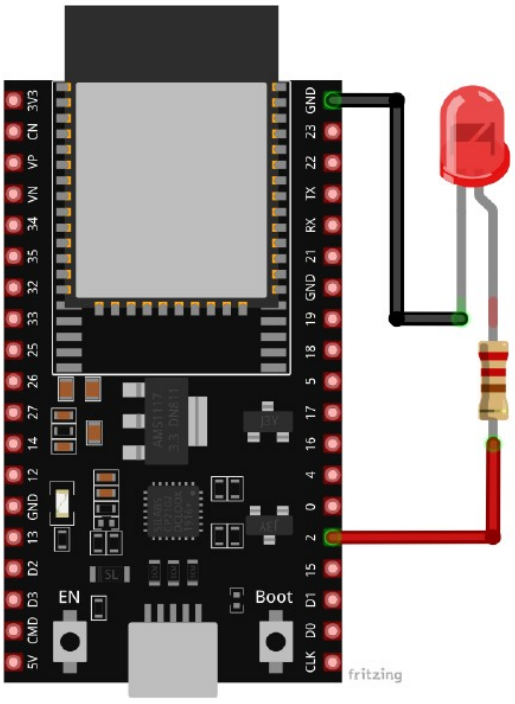
7. Upload testprogram

```
#define led_built_in_ESP 2

void setup()
{
 pinMode(led_built_in_ESP, OUTPUT);
}

void loop()
{
 digitalWrite(led_built_in_ESP, HIGH);
 delay(1000);
 digitalWrite(led_built_in_ESP, LOW);
 delay(1000);
}
```

ESP8266: Has a LED on board, Pin GPIO02
ESP32: There is no LED on board, Pin GPIO02 needs a separate LED and 330 Ohm resistor.



Raspberry, Mosquitto MQTT broker

1. Install Mosquitto broker and Mosquitto Client:
sudo apt install -y mosquitto mosquitto-clients
-y = install with higher rights

2. Start system service mosquitto:
sudo systemctl start mosquitto
sudo systemctl enable mosquitto.service

Get status with:
service mosquitto status

List of system service mosquitto:

Systemd-Service

Der Mosquitto-Server wird automatisch mit der Installation gestartet und als Systemd-Service registriert, so dass er beim Reboot automatisch wieder mit startet. Mit den folgenden Kommandos kannst du ihn starten, stoppen, neustarten und den Autostart deaktivieren bzw. aktivieren.

```
sudo systemctl start mosquitto    # Starten
sudo systemctl stop mosquitto     # Stoppen
sudo systemctl restart mosquitto  # Neustarten
sudo systemctl disable mosquitto  # Autostart deaktivieren
sudo systemctl enable mosquitto   # Autostart aktivieren
```

3. To enable remote access so that we can communicate with other IoT devices, we need to edit/create a configuration file.
Create configuration file /etc/mosquitto/conf.d/my.conf
sudo nano /etc/mosquitto/conf.d/my.conf

Insert the following two lines:
listener 1883
allow_anonymous true
Then, press CTRL-X to exit and save the file. Press Y and Enter.

This configuration file is called from the mosquitto configuration file:
/etc/mosquitto/mosquitto.conf
With this command line:
include_dir /etc/mosquitto/conf.d

Restart Raspberry

Test mosquitto broker on localhost Raspberry:

Raspberry, terminal 1:
View broker (server) with topic test/topic
mosquitto_sub -h localhost -t /test/topic

Raspberry terminal 2:
Send Data to broker, topic: test/topic
mosquitto_pub -h localhost -t /test/topic -m "Hello from Terminal 2!"

View broker with wildcard:
View broker (server) with topic /Groundfloor/Livingroom/…
is used as wildcard
-v is used to see the wholly topic (verbose logging)
mosquitto_sub -h localhost -t /Groundfloor/Livingroom/# -v

View broker (server) with topic /...
mosquitto_sub -h localhost -t /# -v

View broker (server) with topics ems-esp/...
mosquitto_sub -h localhost -t ems-esp/# -v

View all connected clients:
netstat -ntp | grep ESTABLISHED.*mosquitto

Files mosquitto:
/etc/mosquitto/mosquitto.conf
/etc/mosquitto/conf.d/my.conf
/run/mosquitto/mosquitto.pid
/var/log/mosquitto/mosquitto.log
/*var*/lib/mosquitto/mosquitto.db

ESP8266 or ESP32, Connect to WIFI and publish message to MQTT broker

Get IP-address of raspberry:
Command: ip a

ESP8266 program:

```
#include <ESP8266WiFi.h>                  #include <WiFi.h> for ESP32
#include <PubSubClient.h>

// Setup WIFI Network, SSID and password
const char* SSID = "MagentaWLAN-5DHZ";
const char* PSK = "12345";
WiFiClient espClient; // WiFi network client for MQTT

// Setup MQTT
const char* MQTT_BROKER = "192.168.2.51"; // MQTT broker (server) IP-Address
PubSubClient client(espClient); // Creates a partially initialised MQTT client
instance. Pass WiFi network client
const char* ClientID = "ESP8266Client"; //  MQTT client ID when connecting to the MQTT
broker

// LED on board
#define led_built_in_ESP 2

// ########## Setup
void setup()
{
 // LED on board
 pinMode(led_built_in_ESP, OUTPUT);
 digitalWrite(led_built_in_ESP, LOW);

 // Serial
 Serial.begin(9600);

 // Setup and connect to WIFI network
 setup_wifi();

 // MQTT, PubSub client
 client.setServer(MQTT_BROKER, 1883); // Configure broker, IPAdress and port
}

// ########## Connect to WIFI network
void setup_wifi()
{
 // Print: Connecting to WIFI networkname
 Serial.println();
 Serial.print("Connecting to ");
```

```
 Serial.println(SSID);

 // Connect to WIFI network
 // Station (STA) mode is used to get the ESP module connected
 // to a Wi-Fi network established by an access point.
 WiFi.mode(WIFI_STA); // Mode = WIFI station (STA)
 WiFi.begin(SSID, PSK); // Connect to WIFI network
 WiFi.printDiag(Serial); // Print WIFI diagnostic information

 // Wait until WIFI connection is etablished
 /*
 WiFi.status:
 0 : WL_IDLE_STATUS when Wi-Fi is in process of changing between statuses
 1 : WL_NO_SSID_AVAILin case configured SSID cannot be reached
 2 : WL_SCAN_COMPLETED
 3 : WL_CONNECTED after successful connection is established
 4 : WL_CONNECT_FAILED if connection failed
 5 : WL_CONNECTION_LOST
 6 : WL_CONNECT_WRONG_PASSWORD if password is incorrect
 7 : WL_DISCONNECTED if module is not configured in station mode
 */
 int WIFIStatus = 7;
 char outStr[100];
 while (WIFIStatus != WL_CONNECTED)
 {
  WIFIStatus = WiFi.status();
  sprintf(outStr, "WIFI Status during connecting to network: %d", WIFIStatus);
  Serial.println(outStr);
  delay(500);
 }

 // WIFI is connected, print IP-Address
 Serial.println("");
 Serial.println("WiFi connected");
 Serial.print("IP address: ");
 Serial.println(WiFi.localIP());
}

// ########## Loop
void loop()
{
 char topic[50];
 char msg[50];
 static int Counter = 0;
 static bool ledon = false;

 // MQTT check connection to broker
 if (!client.connected()) // Returns true when connected to the broker
 {
  connect(); // Call function Connect to MQTT broker (server)
```

```
 }
 client.loop();

 // MQTT publish message
 sprintf(topic, "/test/topic");
 sprintf(msg, "Message from %s %d", ClientID, Counter);

 Serial.print("Publish message: ");
 Serial.print(topic);
 Serial.print(" ");
 Serial.println(msg);

 Counter++;

 client.publish(topic, msg);

 // Toggle LED built in
 ledon = ! ledon;
 digitalWrite(led_built_in_ESP, ledon);

 // Delaytime bevore next message
 delay(5000);
}

// ########## Connect to MQTT broker (server)
/*
-4 : MQTT_CONNECTION_TIMEOUT - the server didn't respond within the keepalive time
-3 : MQTT_CONNECTION_LOST - the network connection was broken
-2 : MQTT_CONNECT_FAILED - the network connection failed
-1 : MQTT_DISCONNECTED - the client is disconnected cleanly
0 : MQTT_CONNECTED - the client is connected
1 : MQTT_CONNECT_BAD_PROTOCOL - the server doesn't support the requested version of
MQTT
2 : MQTT_CONNECT_BAD_CLIENT_ID - the server rejected the client identifier
3 : MQTT_CONNECT_UNAVAILABLE - the server was unable to accept the connection
4 : MQTT_CONNECT_BAD_CREDENTIALS - the username/password were rejected
5 : MQTT_CONNECT_UNAUTHORIZED - the client was not authorized to connect

-2 means it's failing to create a network connection to the broker.
*/
void connect()
{
 while (!client.connected())
 {
  Serial.println("Connecting to MQTT broker...");
  if (!client.connect(ClientID))
  {
   Serial.print("failed, rc=");
   Serial.print(client.state());
   Serial.println(" retrying in 5 seconds");
   delay(5000);
```

```
    }
  }
}
```

Raspberry, Python program to virtualize the data (GUI)

Install MQTT Python client library on raspberry
sudo pip install paho-mqtt (will install paho-mqtt 1.6.1 for Python 3.x)

Python program

```
import paho.mqtt.client as mqtt

BROKER_ADDRESS = "localhost" # broker address, IP of the raspberry PI or just localhost
MQTT_PATH = "/test/topic" #this is the name of topic

""" The callback for when the client receives a CONNACK response from the server. """
def on_message(client, userdata, message):
    msg = str(message.payload.decode("utf-8"))
    print("message received: ", msg)
    print("message topic: ", message.topic)

""" The callback for when the client receives a CONNACK response from the server. """
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe(MQTT_PATH) # Subscribing in on_connect() means that if we lose the
connection and reconnect then subscriptions will be renewed.

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(BROKER_ADDRESS)
print("Connected to MQTT Broker: " + BROKER_ADDRESS)

client.loop_forever() # use this line if you don't want to write any further code. It blocks the code
forever to check for data
#client.loop_start()  #use this line if you want to write any more code here
```

Raspberry, NodeRed

########## Installation:
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
Installieret
Node.js 16.19.0
Npm 8.19.3
Node-red core 3.0.2

Installieren einer älteren Version:
sudo apt install nodered

Welche Version ist installiert:
node-red -h

######### Service Starten:
node-red-start
oder Menü / Programming / Node-RED
oder Taskleiste / Node-RED

Service Stoppen:
node-red-stop

Service Restarten:
node-red restart

Service automatisch beim Hochlauf starten:
sudo systemctl enable nodered.service

########## Start Node-Red / Dashboard
Node Red GUI am Raspberry starten
http://localhost:1880

Node Red GUI an anderem Rechner über Netzwerk starten:
http://raspberrypi:1880/
http://192.168.2.77:1880/

Node Red Dashboard an anderem Rechner über Netzwerk starten:
http://raspberrypi:1880/ui
http://192.168.2.77:1880/ui

########## Node-RED Konfiguration
Organisation:
Tab GroundFloor
Group GroundFloor LivingRoom (mehrere Gruppen pro Tab sind möglich, sie werden untereinander angeordnet und können minimiert werden)
Label Temperature
Label Humidity

Label Pressure

Nodes installieren, e.g. Dashboard, DWD_Weather, Compass, Calculator, ...:
Menü / Palette verwalten
Reiter Installieren, Suchfeld:
node-red-dashboard
node-red-contrib-dwd-local-weather
node-red-contrib-compass
node-red-contrib-ui-led // Statusanzeige über LED
node-red-contrib-calc -->> wird nicht benötigt, da ersetzt durch Funktion

Konfiguration exportieren
Menü / Export / Alle Flows / Download

Layout Dashboard einrichten
Menü Dashboard (im Menü Pfeil nach unten) / MouseOver Tab (Groundfloor) / Button Layout
Zunächst die breite einstellen (oben rechts)

Erscheinungsbild Dashboard einrichten
Menü Dashboard (im Menü Pfeil nach unten) / Theme / Stil Benutzerdefiniert
Menü Dashboard (im Menü Pfeil nach unten) / Site / Seitenmenü immer anzeigen
Die Reihenfolge des Seitenmenüs ist die der Erstellung im FlowFenster

Reihenfolge der Menüs
Menü Dashboard (im Menü Pfeil nach unten)
Tabs können von oben nach unten verschoben werden. Reihenfolge ist Anordnung im Dashboard

Run application
Button Übernahme (deploy)
Button Run (Kleines Viereck mit Pfeil nach oben rechts). Es wird ein neues Browserfenster geöffnet

Weather data
https://flows.nodered.org/node/node-red-contrib-dwd-local-weather
https://www.dwd.de/DE/leistungen/met_verfahren_mosmix/mosmix_stationskatalog.cfg
https://www.brunweb.de/wetterstation-berechnungen/
Node: node-red-contrib-dwd-local-weather

Links Node-Red
https://nodered.org/docs/user-guide/writing-functions
https://www.digikey.de/en/maker/blogs/2022/how-to-install-and-get-started-with-node-red
https://www.digikey.de/en/maker/blogs/2022/how-to-use-variables-in-node-red

Windows, Mosquitto MQTT broker

Installation Windows, V2.0.15, 64Bit:
https://mosquitto.org/download/

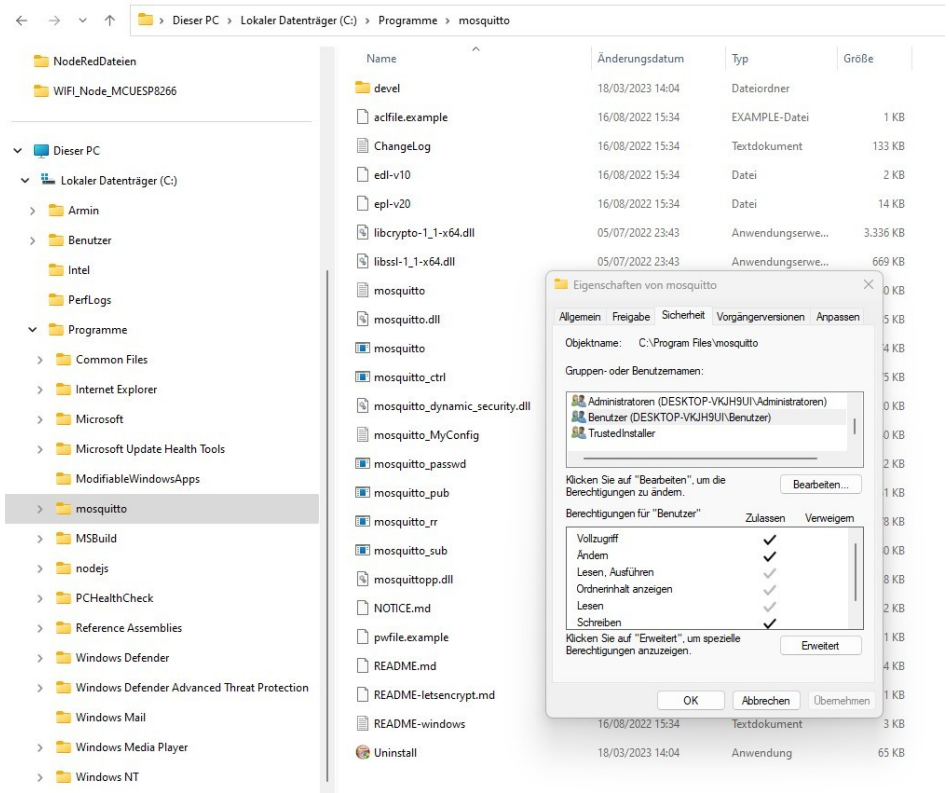
All componens selected (Files, Visual Studio Runtime, Service)
→ Service = start possible as service (Dienst)
C:\Program Files\mosquitto

ConfigFile erstellen als Kopie von mosquitto.conf
C:\Program Files\mosquitto
mosquitto_MyConfig.conf (CONF-Datei, Kopie von mosquitto.conf)

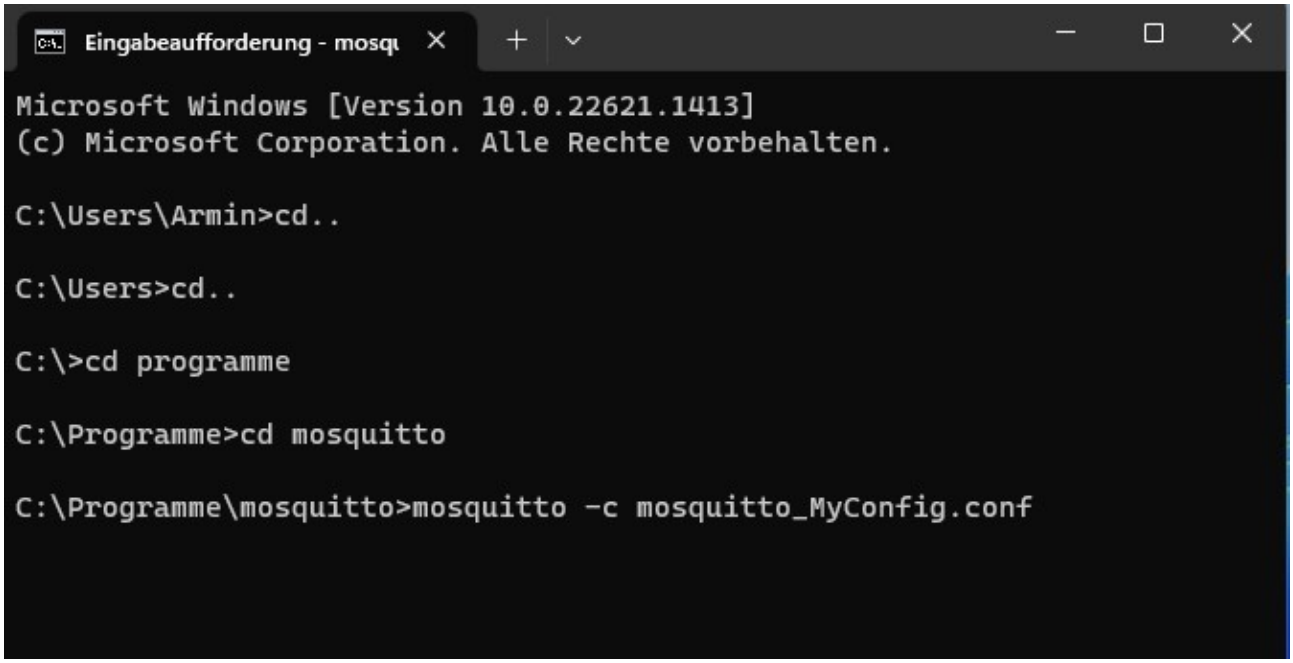
Bei Eigenschaften/Sicherheit für diese Datei alle freigeben, ansonsten kann nicht editiert werden.
Diese Zeilen einfügen:
==
Enable remote access, 2023-01-29, are
listener 1883
allow_anonymous true
==

Datei mosquitto.conf bleibt unverändert.

Einstellungen Benutzerfreigaben Mosquitto:



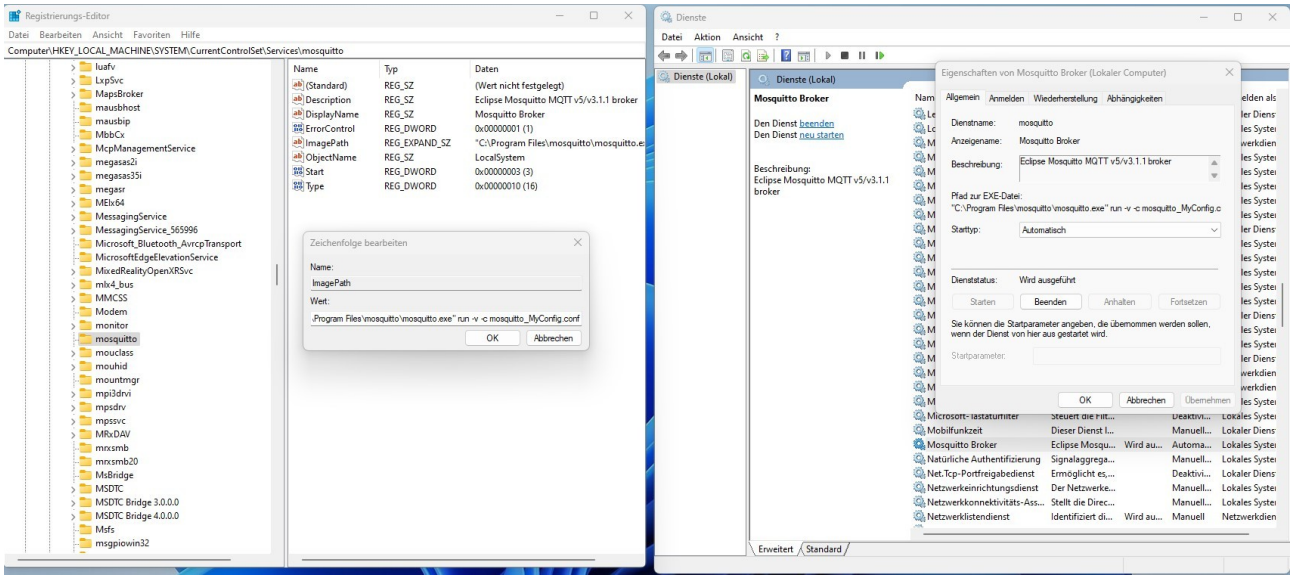
cmd zum Starten:
C:\Programme\mosquitto\mosquitto -c mosquitto_MyConfig.conf (Start with own config file without logging)
C:\Programme\mosquitto\mosquitto -v -c mosquitto_MyConfig.conf (Start with own config file with logging)
C:\Programme\mosquitto\mosquitto -v -c mosquitto.conf (Start with default config file with logging)
-c = Konfigfile angeben
-v = verbose logging



cmd für Netzstatus:
netstat -a
TCP 1883 mit Status Abhören/Hergestellt muss als aktive Verbindung vorhanden sein
e.g. TCP 192.168.2.77:1883

Dienst zum automatischen Starten:
Im Suchen Fenster Dienste (oder Englisch: services) eingeben. Applikation Dienste starten.
Mosquitto Broker / Eigenschaften
Starttyp auf Automatisch
Starten mit Startparameter: -v -c mosquitto_MyConfig.conf

Registry mit Startparameter und Dienst auf Automatisch:
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\mosquitto
ImagePath
"C:\Program Files\mosquitto\mosquitto.exe" run -v -c mosquitto_MyConfig.conf



Start über Dienst bereitete Probleme, deshalb über cmd gestartet.

Windows, NodeRed

Documentation:
https://nodered.org/docs/getting-started/windows
https://nodered.org/docs/getting-started/windows#running-on-windows

1. Install Node.js
Installation Windows:
https://nodejs.org/en/
Aktual version: V18.13.0

CMD commands:
node --version
npm --version

2. Install Node-RED
CMD command:
npm install -g --unsafe-perm node-red

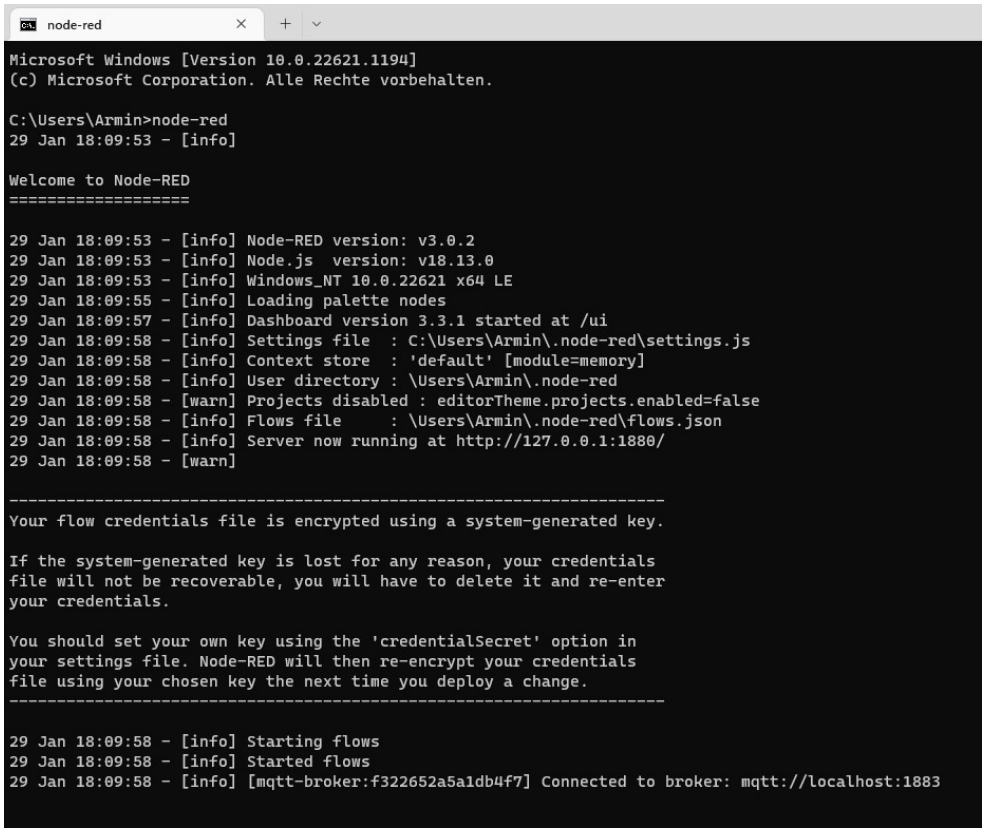
3. Run Node-RED
CMD command:
node-red Startet Node-Red

4. Editor Localhost (beim installierten Rechner)
http://localhost:1880 Zugang NodeRed über Browser

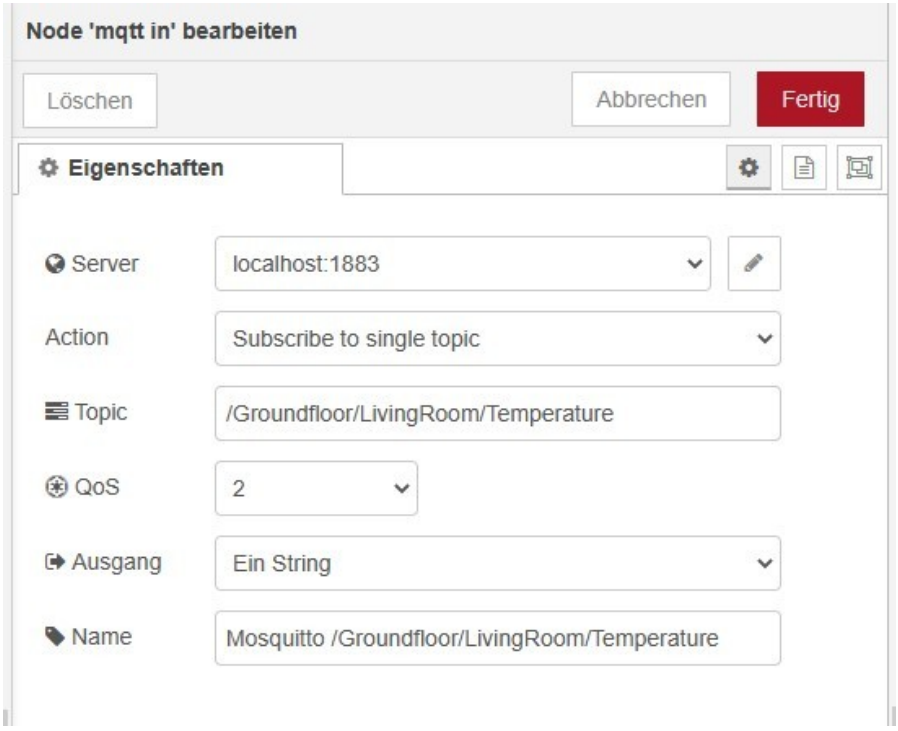
5. Dashboard remote Rechner
http://192.168.2.77:1880/ui

Einstellung in NodeRed, MQTT Server (Broker)
localhost:1883

NodeRed CMD command on Windows



NodeRed MQTT server setting



ESP8266, Power supply

One of these opportunities can be choosen:

1.	5V	via USB	Internal voltage regulator AMS1117, 5V-12V → 3.3V
2.	5V...7V / GND	via pins VIN 5V / GND.	Internal voltage regulator AMS1117, 5V-12V → 3.3V
3.	3.3V / GND	via pins 3V3 / GND	

DeepSleep

```
ESP.deepSleep(5 * 1000000); // Time in us, go to deep sleep for 5s
```

After DeepSleep setup is executed again.
GPIO 16 (D0) must be connected to reset (RST) pin so the ESP8266 is able to wake up.
If you set a deep sleep timer with the ESP8266, once the timer ends, GPIO 16 sends a LOW signal. That means that GPIO 16, when connected to the RST pin, can wake up the ESP8266 after a set period of time.

Current consumption NodeMCU Lua Lolin V3 ESP8266MOD 12-F:
80mA during normal operating mode with WiFi
20uA during DeepSleep

Current consumption sensor BME280:
1mA during measuring mode
5uA during idle mode

Example PeriodTime 5min (300s). Send values each 5min to broker:
10s/5min WakeUp, connect to WiFi, read values BME280, send values to MQTT broker:
290s/5min DeepSleep
--> 120s/h * 81mA = 9720mAs
--> 3480s/h * 0.000025mA = 0.087mAs

--> 9720.087mAs / 3600s = 2.7mAh --> Energy per hour

Battery 6V, connected to pins VIN 5V / GND (power supply oppertunity 2):
With two CR123A batteries in series connected. Each battery has 3V and 1550mAhV.
1550mAh *2 = 3100mAh
--> 3100mAh / 2.7mAh = 1148h
--> 1148h / 24 = 47 days durability

Battery 3V, connected to pins 3.3V / GND (power supply oppertunity 3):
With one LiFePO4 battery, battery has 3.2V and up to 6000mAh
--> 6000mAh / 2.7mAh = 2222h
--> 2222h / 24 = 92 days durability

https://www.eremit.de/c/lifepo4-akkus/3-2v-lifepo4

ESP8266, Sleep modes

https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/

Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	ON	PAUSE	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC memory and RTC peripherals	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 14 dBm ~ 19.5 dBm	Please refer to Table 10 for details.
	Wi-Fi / BT Tx packet 0 dBm	
	Wi-Fi / BT Rx and listening	
Modem-sleep	The CPU is powered on.	Max speed 240 MHz: 30 mA ~ 50 mA
		Normal speed 80 MHz: 20 mA ~ 25 mA
		Slow speed 2 MHz: 2 mA ~ 4 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	150 μA
	ULP sensor-monitored pattern	100 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	5 μA
Power off	CHIP_PU is set to low level, the chip is powered off	0.1 μA

ESP8266, Temperature, humidity and pressure sensor BME280

BME280 with address 0x76
VIN 3.3V
GND GND
SCL GPIO5 (D1)
SDA GPIO4 (D2)

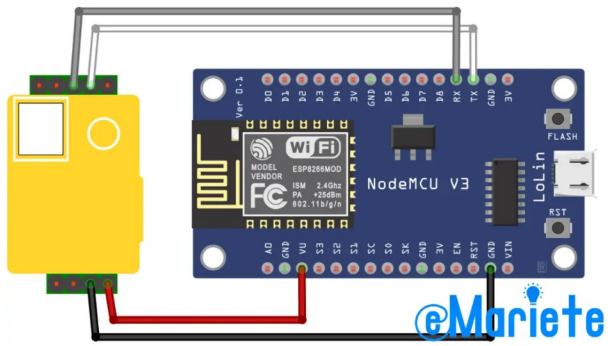
https://randomnerdtutorials.com/esp8266-bme280-arduino-ide/

ESP8266, CO2 sensor MH-Z19C

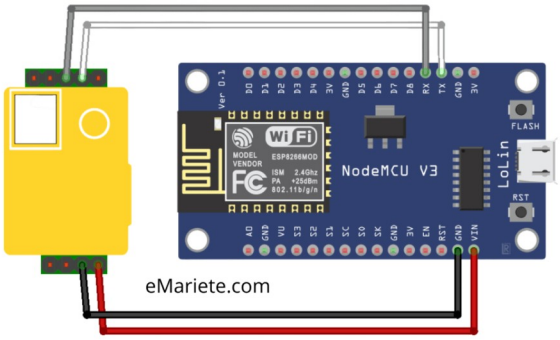
MH-Z19C	**NodeMCU**
Vin	VU (5V USB power output) or VIN (5V)
GND	GND
RX	D10 GPIO1 TXD0
TX	D9 GPIO3 RXD0
PWM	D8 GPIO15

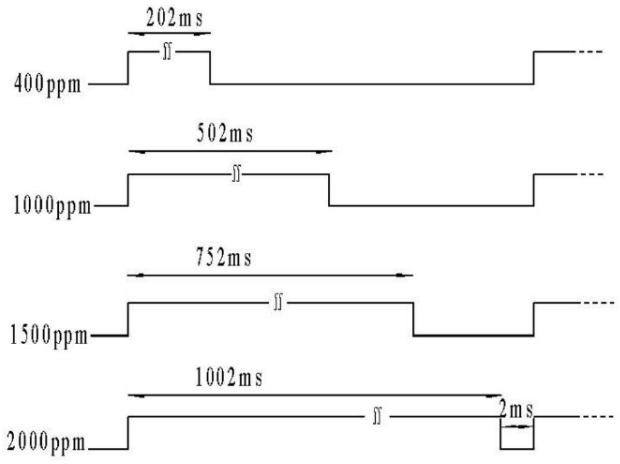
Reading CO2 value via PWM
Pulse duration = 1004ms
CO2 concentration: Cppm=2000×(TH-2ms)/(TH+TL-4ms)
 400ppm = 202ms high, 802ms low
2500ppm = 502ms high, 502ms low
3750ppm = 752ms high, 252ms low
5000ppm = 1002ms high, 2ms low

If your board has a pin marked VU connect the positive of the sensor to that pin.



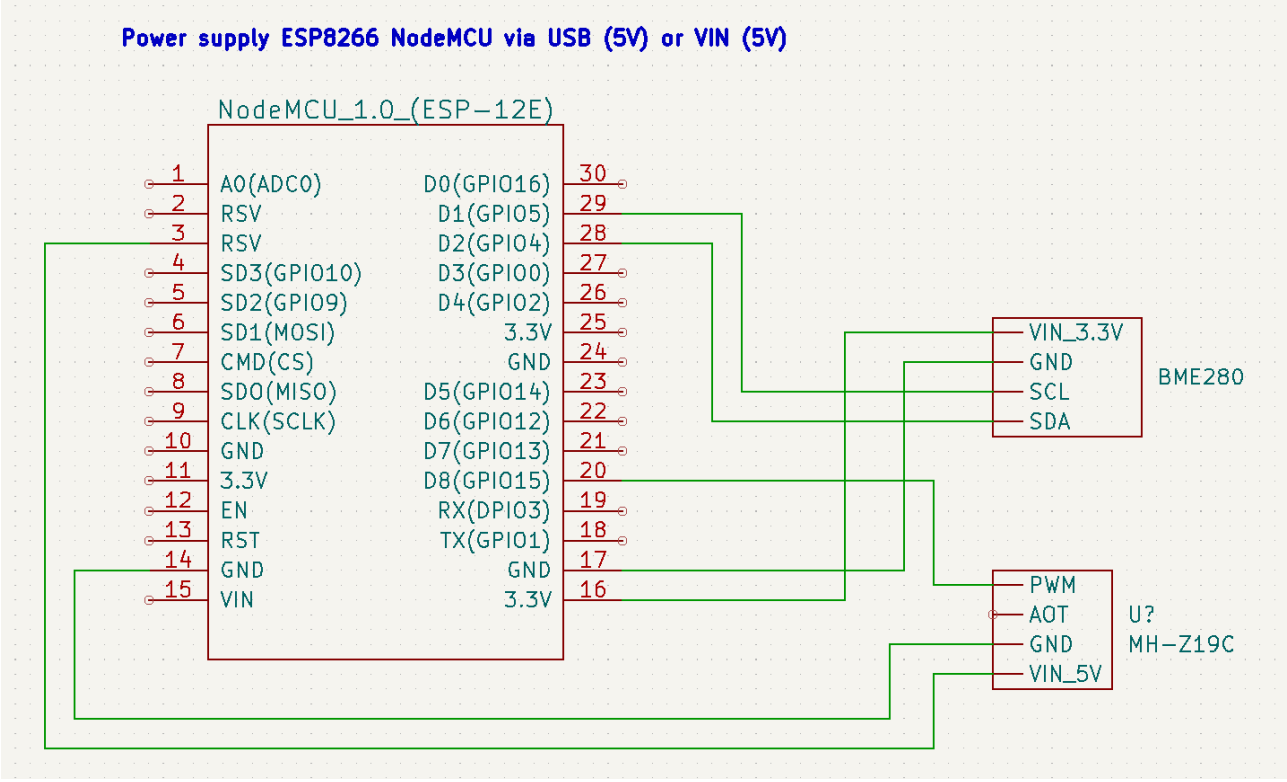
If your badge **DOES NOT have a VU pin**connect it like this (positive of the sensor to the Vin pin):



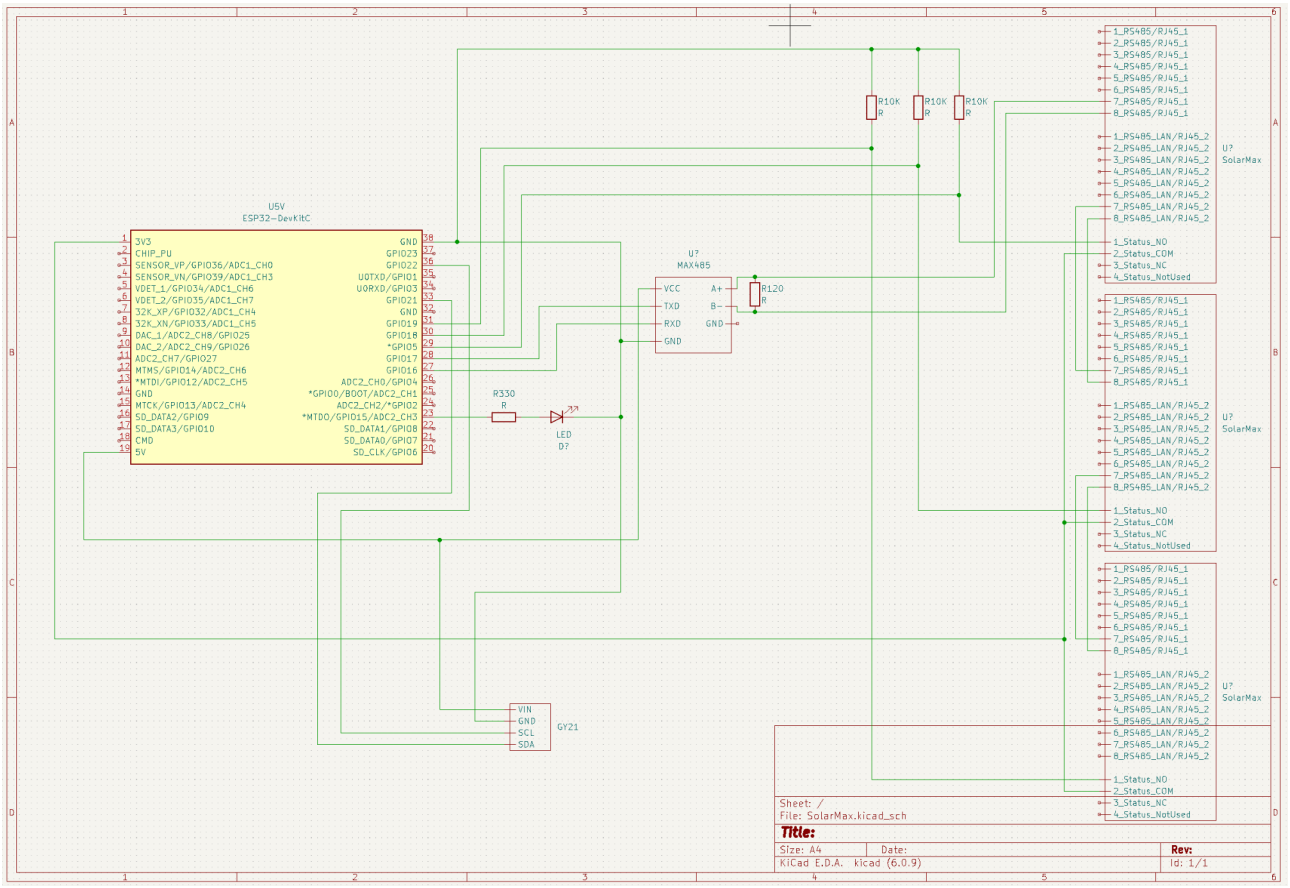
PWM output	
Take 400~2000ppm for example	
CO2 output range	400~2000ppm
Cycle	1004ms±5%
Cycle start high level output	2ms(theoretical value)
The middle cycle	1000ms±5%
cycle end low level output	2ms(theoretical value)
CO2 concentration: Cppm=2000×(TH-2ms)/(TH+TL-4ms)	
C_{ppm}: CO2 concentration could be calculated by PWM output TH high level output time during cycle TL low level output time during cycle	
	

https://funduino.de/nr-51-co2-messung-mit-arduino-co2-ampel
https://emariete.com/en/diy-co2-monitor-wifi/

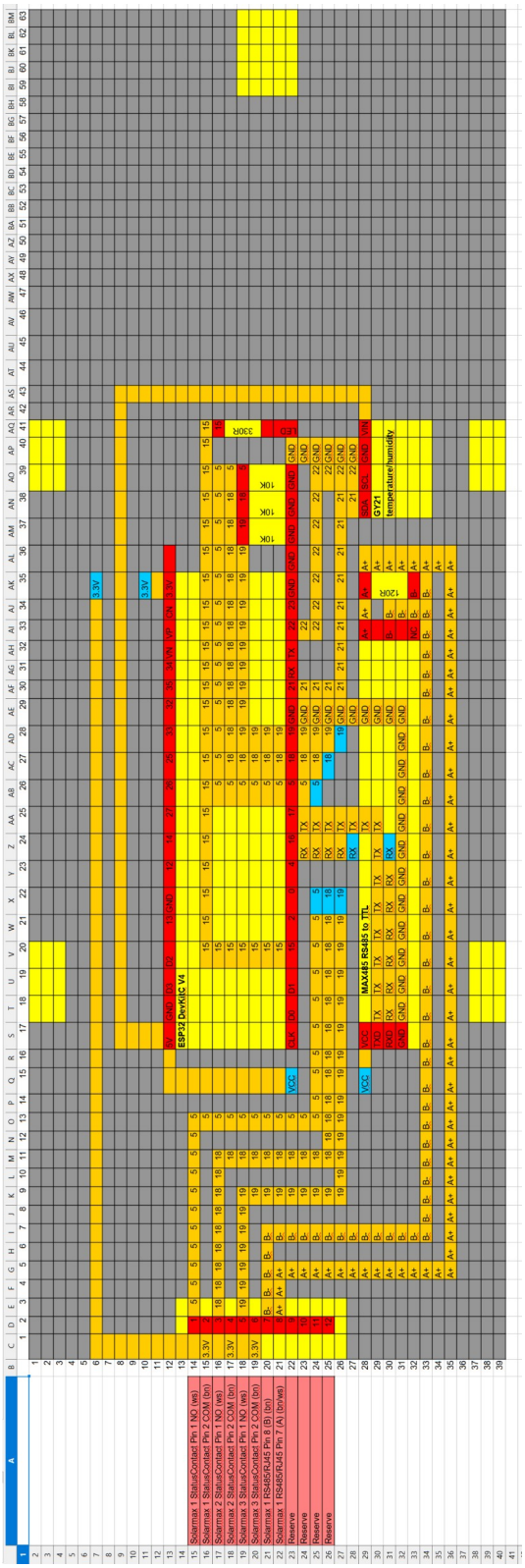
Schematic ESP8266 NodeMCU with BME280 and MH-Z19C



ESP32 SolarMax 4200S



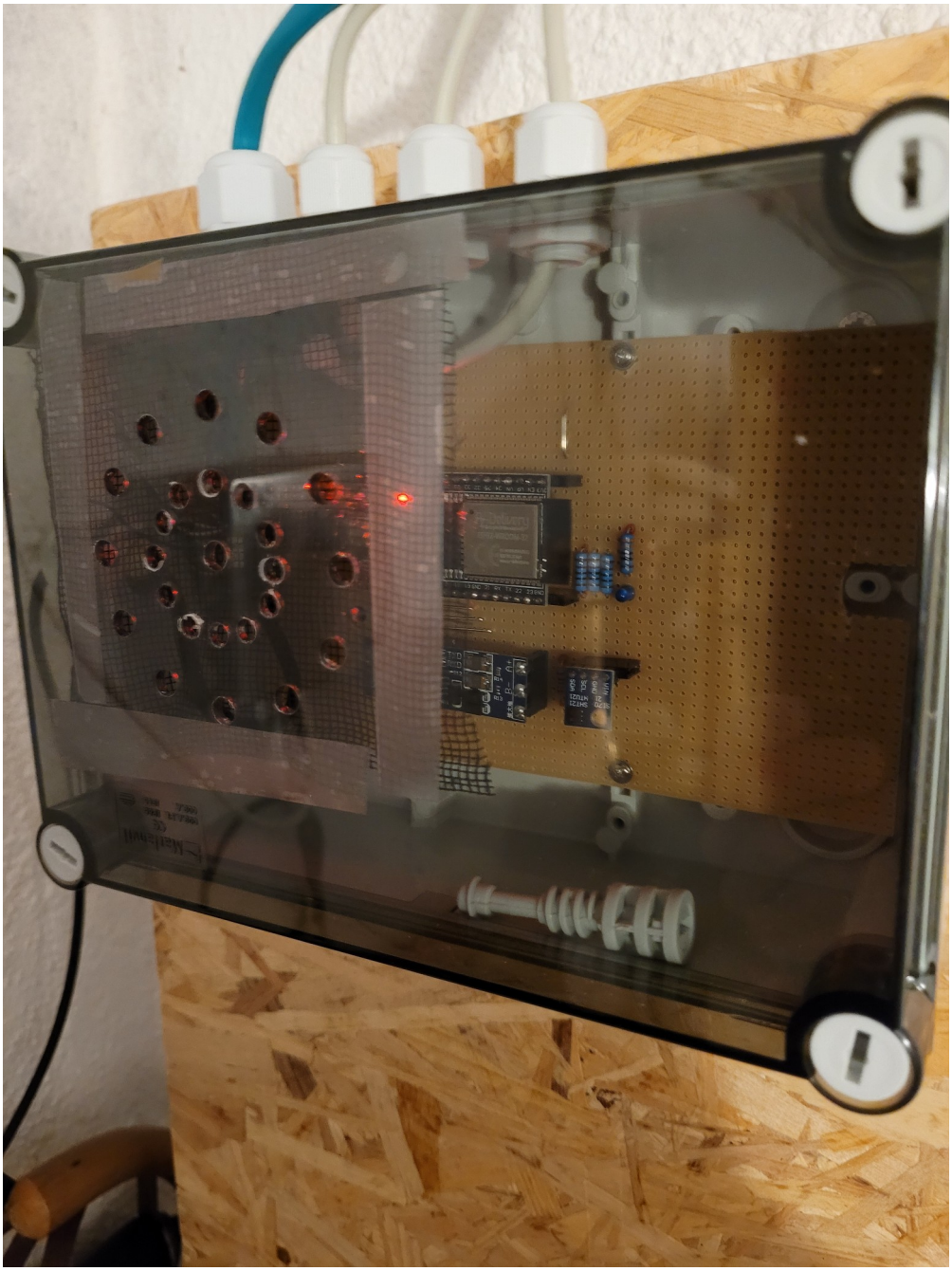
ESP32 SolarMax 4200S



NodeRed SolarMax 4200S

SolarMax

ESP32 platine



Useful links

ESP8266 NodeMCU
https://circuitdigest.com/microcontroller-projects/getting-started-with-nodemcu-esp12

https://components101.com/development-boards/nodemcu-esp8266-pinout-features-and-datasheet

https://smarthome-blogger.de/blog/esp8266-projekte/nodemcu-esp8266-einfuehrung

https://smarthome-blogger.de/blog/esp8266-projekte/esp8266-mqtt-tutorial

https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html

https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/

Raspberry
https://smarthome-blogger.de/blog/raspberry-pi-smart-home/mqtt-raspberry-pi-einfuehrung

https://myhomethings.eu/de/mosquitto-mqtt-broker-installation-auf-raspberry-pi/

https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/

https://mosquitto.org/documentation/

https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/mqtt-auf-dem-raspberry-pi-und-mit-micropython-auf-der-esp-familie-teil-1

https://www.az-delivery.de/en/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/mqtt-auf-dem-raspberry-pi-und-mit-micropython-auf-der-esp-familie-teil-3

https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/mqtt-auf-dem-raspberry-pi-und-mit-micropython-auf-der-esp-familie-teil-2

Description of the #include <PubSubClient.h>
https://github.com/knolleary/pubsubclient

https://pubsubclient.knolleary.net/api

https://www.arduino.cc/reference/en/libraries/pubsubclient/

http://www.steves-internet-guide.com/using-arduino-pubsub-mqtt-client/