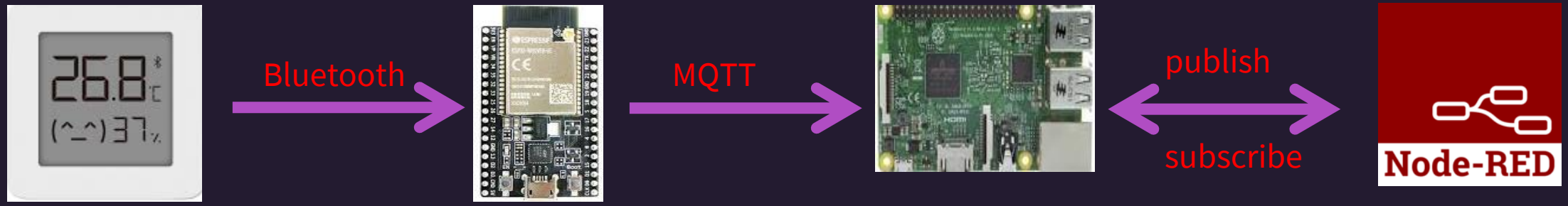
Dokumentation für den Anschluss von Xiaomi Temperatursensoren an ESP32, Übertragung an einen Raspberry Pi und Anzeige auf einem Node-RED Dashboard

Projektbeschreibung

In diesem Projekt verwenden wir zwei Xiaomi BLE (Bluetooth Low Energy) Temperatursensoren, die ihre Temperatur- und Feuchtigkeitsdaten über Bluetooth an einen ESP32 senden. Der ESP32 verbindet sich dann über WLAN mit einem MQTT-Broker auf einem Raspberry Pi und sendet die Sensordaten dorthin. Schließlich zeigen wir die Daten auf einem **Node-RED Dashboard** an, das auf dem Raspberry Pi läuft. Diese Dokumentation beschreibt den gesamten Prozess Schritt für Schritt.



Voraussetzungen

Hardware:

- Zwei Xiaomi BLE-Temperatursensoren
- ESP32 Mikrocontroller
- Raspberry Pi mit installiertem MQTT-Broker
- Node-REDWLAN-Router

Software:

- Arduino IDE zur Programmierung des ESP32
- Node-RED (auf dem Raspberry Pi) zur Visualisierung der Sensordaten
- Mosquitto MQTT-Broker auf dem Raspberry Pi

Verbindung der Sensoren mit dem ESP32 und Anzeige auf dem Seriellen Monitor

Unser erster Schritt ist es, die Xiaomi BLE-Temperatursensoren mit dem ESP32 zu verbinden und die Sensordaten auf dem Seriellen Monitor anzuzeigen. Dafür gehen wir wie folgt vor:

1. **Installation des ESP32-Boardpakets in der Arduino IDE:**
   - **Schritt 1:** Öffnen Sie die Arduino IDE und fügen Sie die ESP32-Board-URL hinzu.
   - **Schritt 2:** Installieren Sie das **ESP32-Boardpaket**, um den ESP32 programmieren zu können.

2. **Installation der benötigten Bibliotheken:**
   - **Schritt 3:** Installieren Sie die notwendigen Bibliotheken wie **NimBLEDevice**, **WiFi**, und **PubSubClient**, um BLE- und MQTT-Funktionen zu nutzen.

3. **Hardware-Setup:**
   - **Schritt 4:** Verbinden Sie den ESP32 über USB mit dem Computer.
   - **Schritt 5:** Identifizieren Sie die BLE-Adressen der Xiaomi-Sensoren (zum Beispiel durch Scannen oder mit einer App)

4. WLAN- und MQTT-Verbindung:

- **Schritt 6:** Konfigurieren Sie die WLAN-Zugangsdaten (SSID und Passwort) im Code.
- **Schritt 7:** Geben Sie die IP-Adresse des Raspberry Pi als MQTT-Broker an.

5. BLE-Verbindung und Serielle Ausgabe:

Schritt 8: Stellen Sie die Verbindung zu den Xiaomi-Sensoren her, indem Sie deren BLE-Adressen im Code angeben.

Schritt 9: Der ESP32 empfängt Temperatur- und Feuchtigkeitsdaten und zeigt sie auf dem Seriellen Monitor an.

Verbindung der Sensoren mit dem ESP32 und Anzeige auf dem Seriellen Monitor

Schritt 1: Installation des ESP32-Board-Pakets

1. Öffnen Sie die Arduino IDE und gehen Sie zu **Datei > Voreinstellungen**.

2. Fügen Sie die folgende URL unter „**Zusätzliche Boardverwalter-URLs**“ hinzu:https://dl.espressif.com/dl/package_esp32_index.json

3. Gehen Sie zu **Werkzeuge > Board > Boardverwalter** und suchen Sie nach „esp32“. Installieren Sie das **ESP32-Bordpaket**.

4. Klicke auf **OK**, um das Fenster zu schließen.

5. Gehe nun **zu Werkzeuge > Board > Boardverwalter**.

6. Suche nach **esp32** im Boardverwalter.

7. Wähle **esp32 by Espressif Systems** und klicke auf **Installieren**.

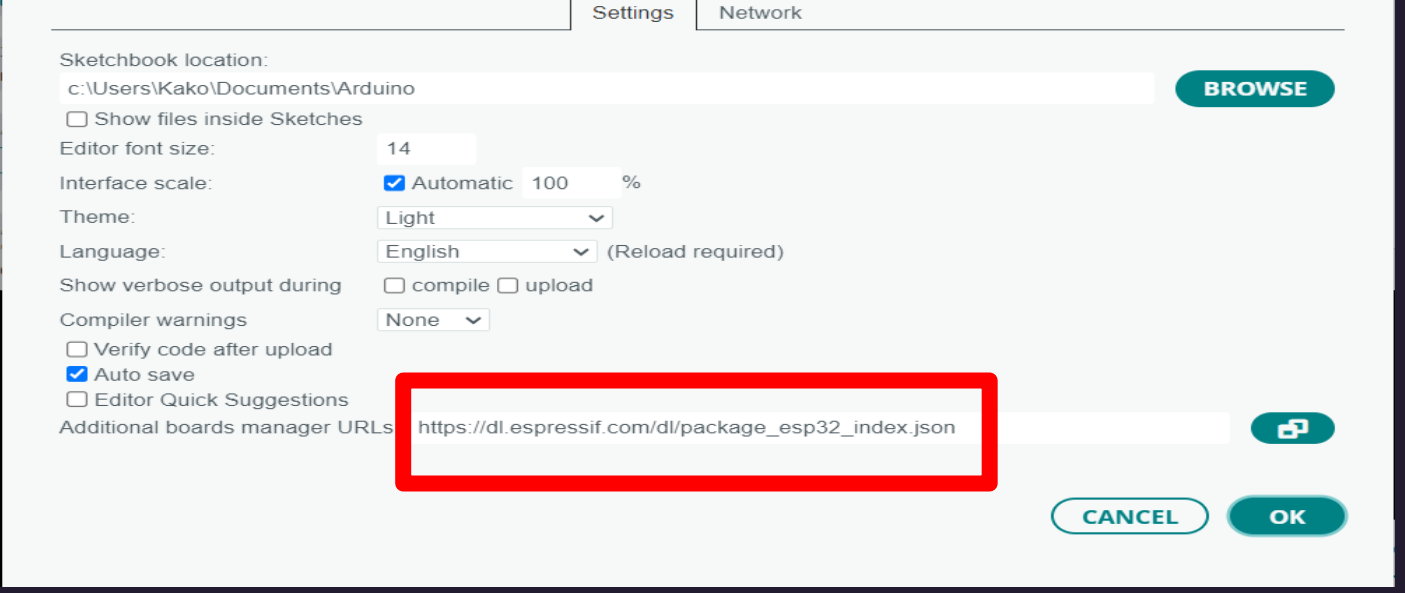
Schritt 2: ESP32-Board in der Arduino IDE auswählen

1. Verbinde das ESP32 mit einem USB-Kabel mit deinem Computer.
2. Gehe zu **Werkzeuge > Board** und wähle **ESP32 Dev Module** oder das entsprechende Board aus.
3. Gehe zu **Werkzeuge > Port** und wähle den richtigen COM-Port aus, an dem das ESP32 angeschlossen ist.

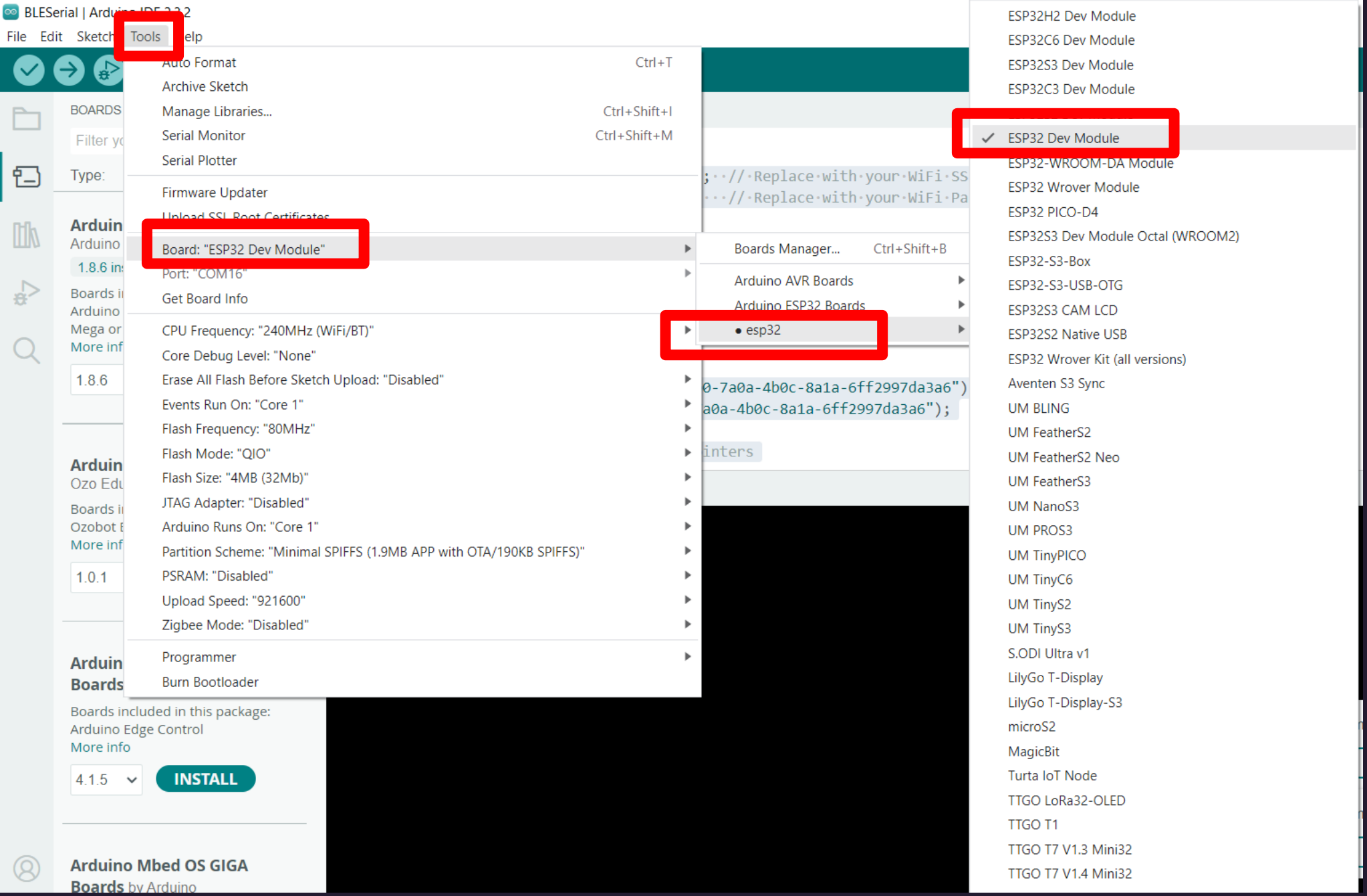
Schritt 3: Benötigte Bibliotheken installieren

1. Gehe zu **Sketch > Bibliothek einbinden > Bibliotheken** verwalten.
2. Suche und installiere die folgenden Bibliotheken:

- **NimBLEDevice** (für BLE-Funktionalität).
- **WiFi** (zum Verbinden des ESP32 mit dem WLAN).
- **PubSubClient** (für die MQTT-Kommunikation)

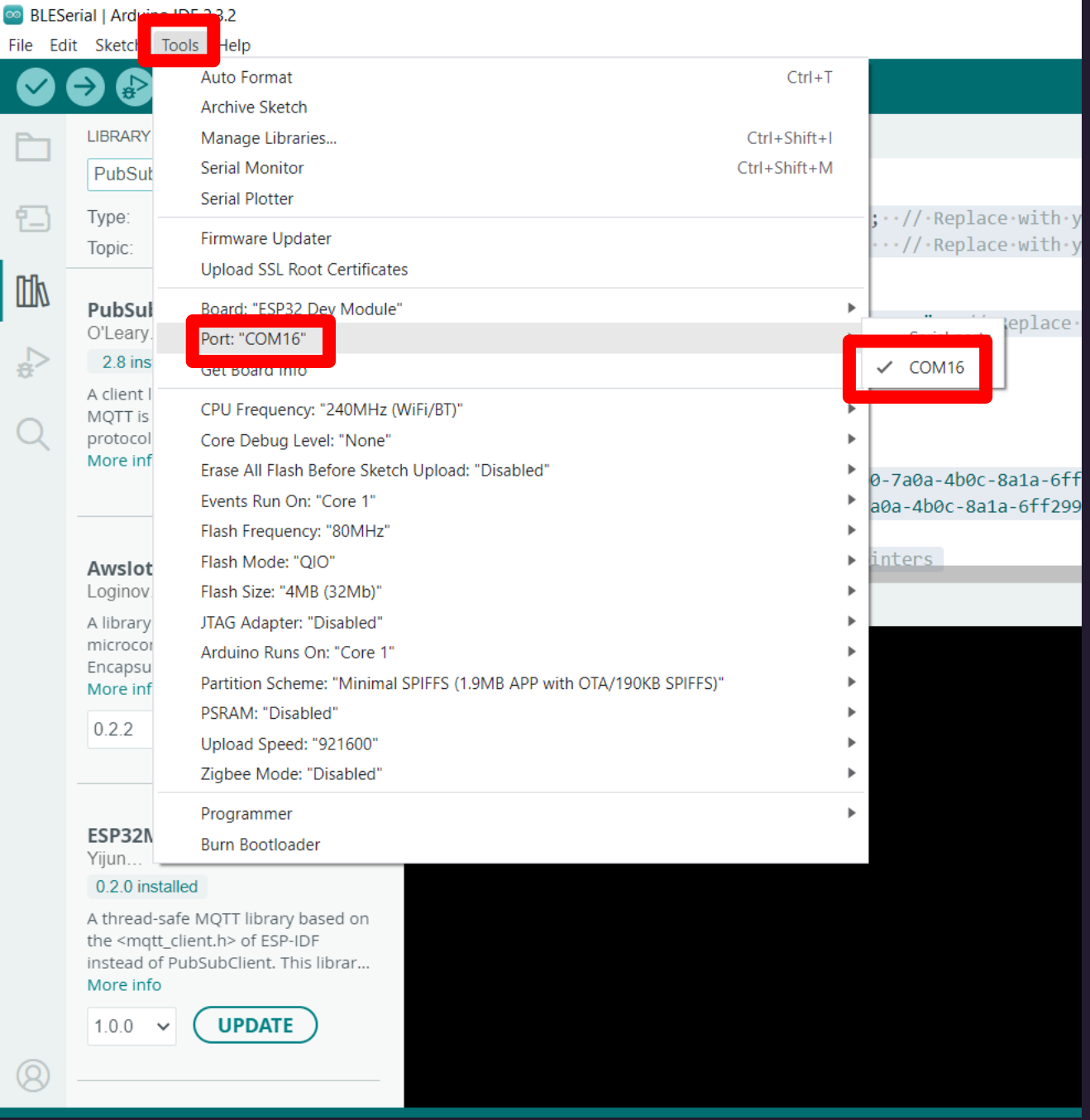


Installation des ESP32-Board-Pakets

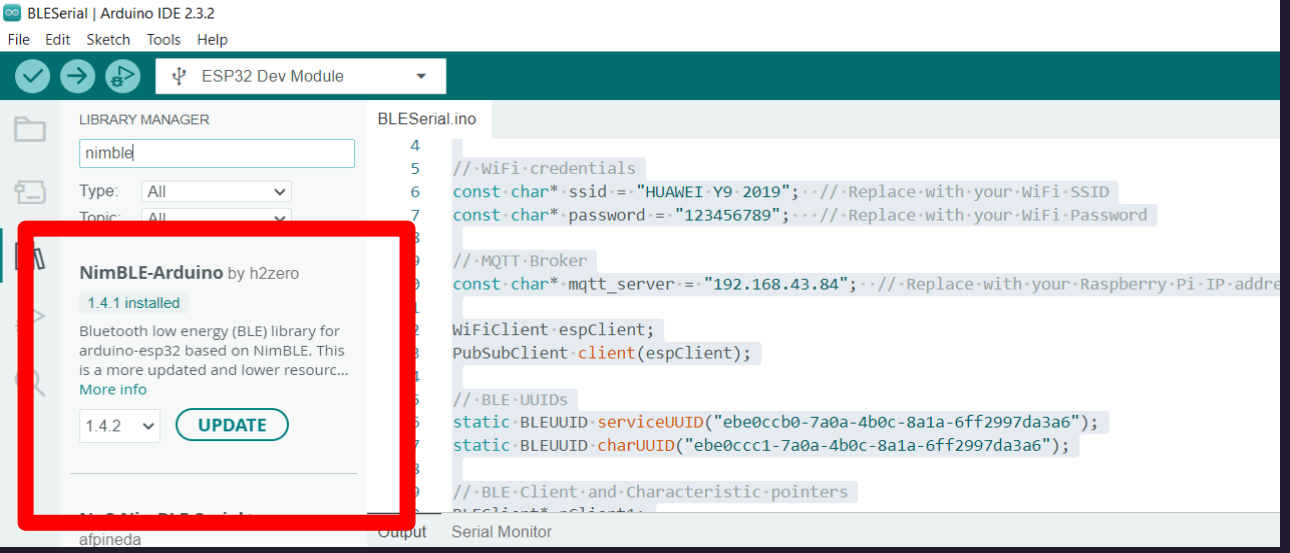


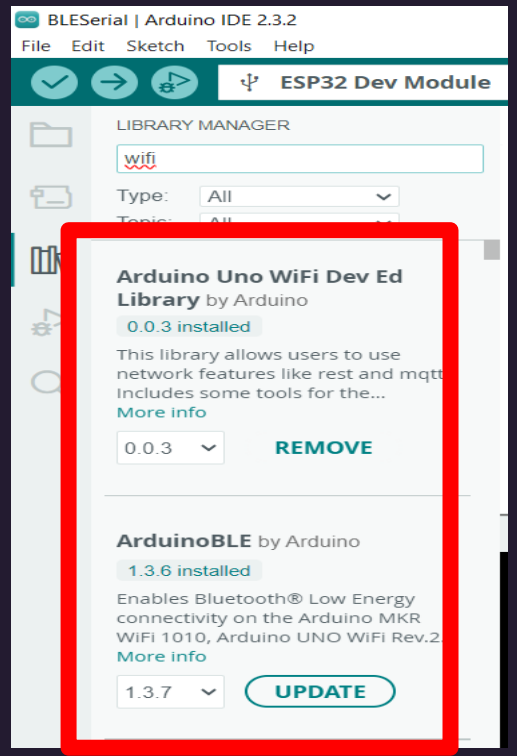
Installation des ESP32-Board-Pakets

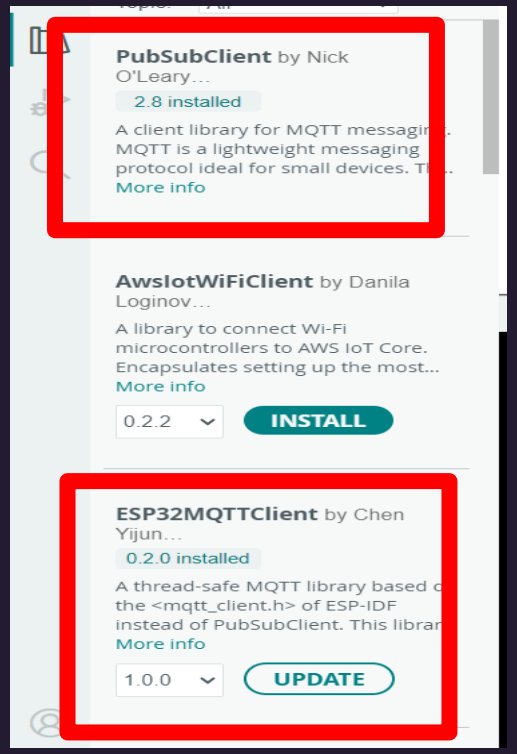
ESP32-Board in der Arduino IDE auswählen



Benötigte Bibliotheken installieren







Erläuterung des Programmiercodes

1. WiFi-Einstellungen und MQTT-Verbindung

Zu Beginn des Codes werden die **WiFi-Anmeldedaten** definiert, damit der ESP32 eine Verbindung zum WLAN herstellen kann:

```
1   #include <WiFi.h>
2   #include <PubSubClient.h>
3   #include "NimBLEDevice.h"
4
5   // WiFi credentials
6   const char* ssid = "HUAWEI Y9 2019";   // Replace with your WiFi SSID
7   const char* password = "123456789";    // Replace with your WiFi Password
8
9   // MQTT Broker
10  const char* mqtt_server = "192.168.43.84";   // Replace with your Raspberry Pi IP address
11
```

- **SSID** und **Passwort** müssen mit deinem WLAN übereinstimmen.
- Der **MQTT-Broker** ist ein Server, der Nachrichten empfängt und weiterleitet. Hier wird die IP-Adresse des Brokers (in diesem Fall ein Raspberry Pi) definiert.

2. BLE UUIDs und MAC-Adressen

BLE-Kommunikation funktioniert über **UUIDs (Universally Unique Identifiers)**, die zur Identifizierung von Diensten und Merkmalen verwendet werden. Die UUIDs für unseren BLE-Service und die Characteristic sind:

```
14
15  // BLE UUIDs
16  static BLEUUID serviceUUID("ebe0ccb0-7a0a-4b0c-8a1a-6ff2997da3a6");
17  static BLEUUID charUUID("ebe0ccc1-7a0a-4b0c-8a1a-6ff2997da3a6");
18
```

- **Service-UUID:** Identifiziert den Dienst, über den die Sensordaten gesendet werden.
- **Charakteristik-UUID:** Beschreibt das spezifische Merkmal (z.B. Temperatur und Feuchtigkeit), das die Daten enthält.

3. Definieren der BLE-MAC-Adressen der Sensoren

```
24
25  // BLE addresses
26  std::string sensorAddress1 = "A4:C1:38:3D:94:B6";
27  std::string sensorAddress2 = "a4:c1:38:1f:09:B3";
28
```

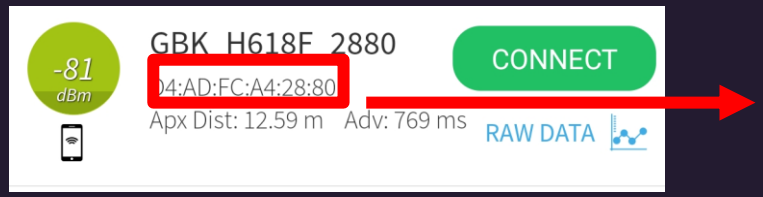
Erklärung:

•**sensorAddress1** und **sensorAddress2**: Diese beiden Adressen repräsentieren die eindeutigen **MAC-Adressen** der Xiaomi BLE-Sensoren. Die MAC-Adresse ist eine einzigartige Kennung für jedes Bluetooth-Gerät.

•**Bluetooth-Name der Xiaomi-Sensoren**: In vielen Fällen erscheinen die Xiaomi-Sensoren unter dem Namen **LYWSD03MMC** in Bluetooth-Scannern.

•**Hinweis**: Diese MAC-Adressen können mit einer App wie **BLE Scanner** gefunden werden. Diese App zeigt alle in der Nähe befindlichen Bluetooth Low Energy (BLE)-Geräte und deren MAC-Adressen an, sodass man die gewünschten Sensoren leicht identifizieren kann.

Beispiel



MAC-Address

4. Verbindungsaufbau zu den BLE-Sensoren

```
18
19  // BLE Client and Characteristic pointers
20  BLEClient* pClient1;
21  BLEClient* pClient2;
22  static BLERemoteCharacteristic* pRemoteCharacteristic1;
23  static BLERemoteCharacteristic* pRemoteCharacteristic2;
24
```

Erklärung:

•**BLEClient**: Diese Objekte (**pClient1** und **pClient2**) repräsentieren den BLE-Client (ESP32), der sich mit den Sensoren verbindet.

•**BLERemoteCharacteristic**: Diese Objekte speichern die Charakteristika der Sensoren, die die Sensordaten enthalten.

5. Initialisieren des ESP32 als BLE-Client und Verbindungsaufbau zu den Sensoren

```
36  void setup() {
37    Serial.begin(115200);
38    Serial.println();
39
40    // Initialize WiFi
41    connectToWiFi();
42
43    // Initialize MQTT client
44    client.setServer(mqtt_server, 1883);
45
46    // Initialize BLE
47    BLEDevice::init("ESP32Client");
48
49    // Connect to BLE sensors
50    connectToServer(BLEAddress(sensorAddress1), pClient1, pRemoteCharacteristic1, notifyCallback1);
51    connectToServer(BLEAddress(sensorAddress2), pClient2, pRemoteCharacteristic2, notifyCallback2);
52  }
53
```

Erklärung:

•**Serial.begin(115200)**: Initialisiert die serielle Kommunikation mit einer Baudrate von 115200, um Debug-Informationen an den Computer zu senden.
•**connectToWiFi()**: Baut die Wi-Fi-Verbindung auf, indem die zuvor definierten SSID und das Passwort verwendet werden.
•**client.setServer()**: Verbindet den ESP32 mit dem MQTT-Server über die angegebene IP-Adresse und den Standardport 1883.
•**BLEDevice::init("ESP32Client")**: Initialisiert das ESP32-Modul als BLE-Client, damit es sich mit den Sensoren verbinden kann.
•**connectToServer()**: Baut eine BLE-Verbindung zu jedem Sensor auf, indem die MAC-Adressen der Sensoren und die entsprechenden Callback-Funktionen übergeben werden.

6. Verarbeiten der Sensordaten mit Callback-Funktion

```
108  // BLE callback for Sensor 1
109  static void notifyCallback1(
110    BLERemoteCharacteristic* pBLERemoteCharacteristic,
111    uint8_t* pData,
112    size_t length,
113    bool isNotify) {
114    if (length < 5) {
115      Serial.println("Sensor 1: Received data length is too short");
116      return;
117    }
118
119    uint16_t tempRaw = (pData[0] | (pData[1] << 8));
120    float temperature = tempRaw * 0.01;
121    uint16_t humidityRaw = pData[2];
122
123    Serial.print("Sensor 1 - Temperature: ");
124    Serial.print(temperature);
125    Serial.print(" C, Humidity: ");
126    Serial.print(humidityRaw);
127    Serial.println(" %");
128
129    // Publish data to MQTT
130    String payload = "Temperature: " + String(temperature) + " C, Humidity: " + String(humidityRaw) + " %";
131    client.publish("home/sensor1", payload.c_str());
132  }
```

Erklärung:

•**notifyCallback1**: Diese Callback-Funktion wird aufgerufen, sobald der Sensor neue Daten sendet.

•**pData**: Enthält die Sensordaten in Binärform, die in der BLE-Nachricht übermittelt werden.

•**uint16_t tempRaw = (pData[0] | (pData[1] << 8))**: Diese Zeile wandelt die ersten beiden Binärwerte der Sensordaten in einen 16-Bit-Wert um, der die Temperatur in Rohform darstellt.

•**float temperature = tempRaw * 0.01**: Dieser Rohwert wird mit 0,01 multipliziert, um die tatsächliche Temperatur in Grad Celsius zu erhalten.

•**uint16_t humidityRaw = pData[2]**: Der dritte Wert von **pData** enthält die Luftfeuchtigkeit in Rohform.

•**Serial.print()**: Die Temperatur und Feuchtigkeit werden über die serielle Schnittstelle zur Anzeige ausgegeben.

•**client.publish()**: Die Daten werden über das MQTT-Protokoll an den Server gesendet, der sie weiterverarbeitet oder speichert.

Erklärung der Datenverarbeitung:

```
118
119     uint16_t tempRaw = (pData[0] | (pData[1] << 8));
120     float temperature = tempRaw * 0.01;
```

- **Rohdatenformat**: Der Sensor sendet 5 binäre Datenwerte (Bytes). Diese müssen umgewandelt werden, um die Temperatur und Feuchtigkeit zu berechnen.
- Die ersten beiden Bytes (pData[0] und pData[1]) enthalten die Temperatur in binärer Form.Diese werden zusammengeführt (|-Operator) und in einen 16-Bit-Wert (uint16_t) umgewandelt.Der Rohwert muss mit 0.01 multipliziert werden, um die tatsächliche Temperatur in Grad Celsius zu erhalten.

```
121     uint16_t humidityRaw = pData[2];
122
```

- Das dritte Byte (pData[2]) enthält die relative Feuchtigkeit, die direkt verwendet werden kann.

7. Verbindung zum BLE-Server aufbauen

```
159
160 // Function to connect to BLE server
161 bool connectToServer(BLEAddress pAddress, BLEClient*& pClient, BLERemoteCharacteristic*& pRemoteCharacteristic, void(*notifyCallback)(BLERemoteCharacteri
162   Serial.print("Forming a connection to ");
163   Serial.println(pAddress.toString().c_str());
164
165   pClient = BLEDevice::createClient();
166   Serial.println(" - Created client");
167
168   if (!pClient->connect(pAddress)) {
169     Serial.println("Failed to connect to server!");
170     return false;
171   }
172   Serial.println(" - Connected to server");
173
174   BLERemoteService* pRemoteService = pClient->getService(serviceUUID);
175   if (pRemoteService == nullptr) {
176     Serial.print("Failed to find our service UUID: ");
177     Serial.println(serviceUUID.toString().c_str());
178     return false;
179   }
180   Serial.println(" - Found our service");
181
182   pRemoteCharacteristic = pRemoteService->getCharacteristic(charUUID);
183   if (pRemoteCharacteristic == nullptr) {
184     Serial.print("Failed to find our characteristic UUID: ");
185     Serial.println(charUUID.toString().c_str());
186     return false;
187   }
188   Serial.println(" - Found our characteristic");
189
190   pRemoteCharacteristic->registerForNotify(notifyCallback);
191
192   return true;
```

Erklärung:

•**BLEDevice::createClient()**: Erstellt einen BLE-Client, der für die Kommunikation mit dem Sensor verwendet wird.

•**pClient->connect(pAddress)**: Verbindet den ESP32-Client mit der angegebenen MAC-Adresse des Sensors.

•**pRemoteService->getService(serviceUUID)**: Ruft den spezifischen BLE-Dienst vom Sensor ab, der die Temperatur- und Feuchtigkeitsdaten bereitstellt.

•**pRemoteCharacteristic->getCharacteristic(charUUID)**: Holt die Charakteristik des Sensors, die die Daten enthält.

•**pRemoteCharacteristic->registerForNotify(notifyCallback)**: Registriert die Callback-Funktion, die aufgerufen wird, wenn der Sensor neue Daten sendet.

Und schließlich sollten wir die Daten wie folgt im seriellen Monitor sehen

```
17:29:18.465 -> Sensor 1 - Temperature: 23.60 C, Humidity: 74 %
17:29:21.565 -> Sensor 2 - Temperature: 23.58 C, Humidity: 75 %
```

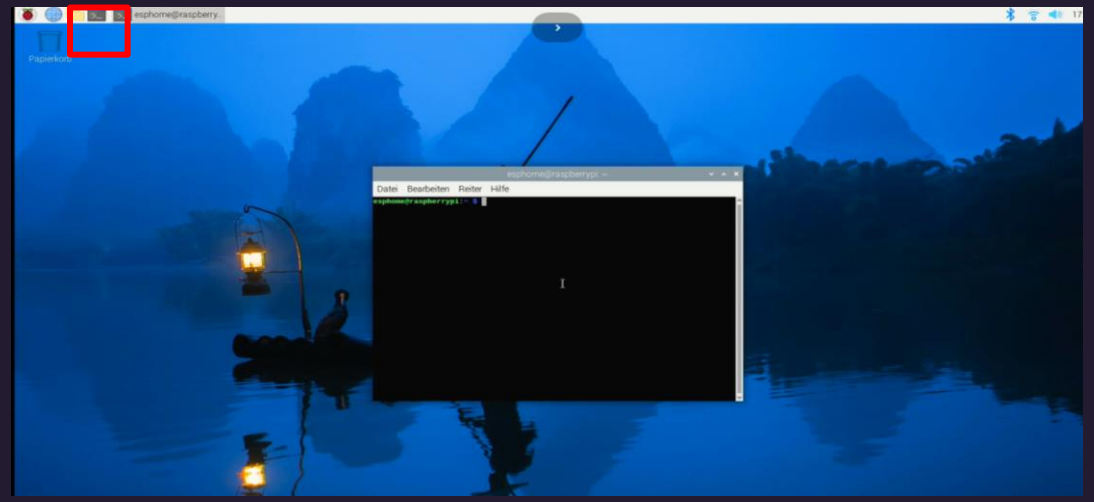
Einrichten des MQTT-Brokers auf dem Raspberry Pi mit Mosquitto

Damit der ESP32 Sensordaten an den Raspberry Pi über MQTT senden kann, richten wir den Raspberry Pi als **MQTT-Broker** ein. Dafür verwenden wir **Mosquitto**, einen weit verbreiteten MQTT-Broker.

1.1 Installation von Mosquitto auf dem Raspberry Pi

Führe diese Schritte aus, um Mosquitto auf deinem Raspberry Pi zu installieren:

1. **Öffne das Terminal auf deinem Raspberry Pi.**



2. **Aktualisiere die Paketlisten und das System**

```
sudo apt update
sudo apt upgrade
```

3. Installiere Mosquitto und die Mosquitto-Clients:

```
sudo apt install mosquitto mosquitto-clients
```

4. Starte den Mosquitto-Dienst und richte ihn so ein, dass er beim Systemstart automatisch gestartet wird:

```
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

5. Ermittlung der IP-Adresse des Raspberry Pi

Um die IP-Adresse des Raspberry Pi zu finden, kannst du den folgenden Befehl im Terminal eingeben:

```
hostname -I
```

Dies gibt die lokale IP-Adresse des Raspberry Pi zurück (z.B. 192.168.43.84). Diese IP-Adresse wird im Arduino-Sketch des ESP32 verwendet, um eine Verbindung zum MQTT-Broker herzustellen.

Installation und Einrichtung von Node-RED auf dem Raspberry Pi

Mit Node-RED kannst du die Sensordaten in einer grafischen Benutzeroberfläche anzeigen. Hier erfährst du, wie du Node-RED auf deinem Raspberry Pi installierst und ein Dashboard einrichtest.

1. Installation von Node-RED

Führe die folgenden Schritte im Terminal auf deinem Raspberry Pi aus, um Node-RED zu installieren:

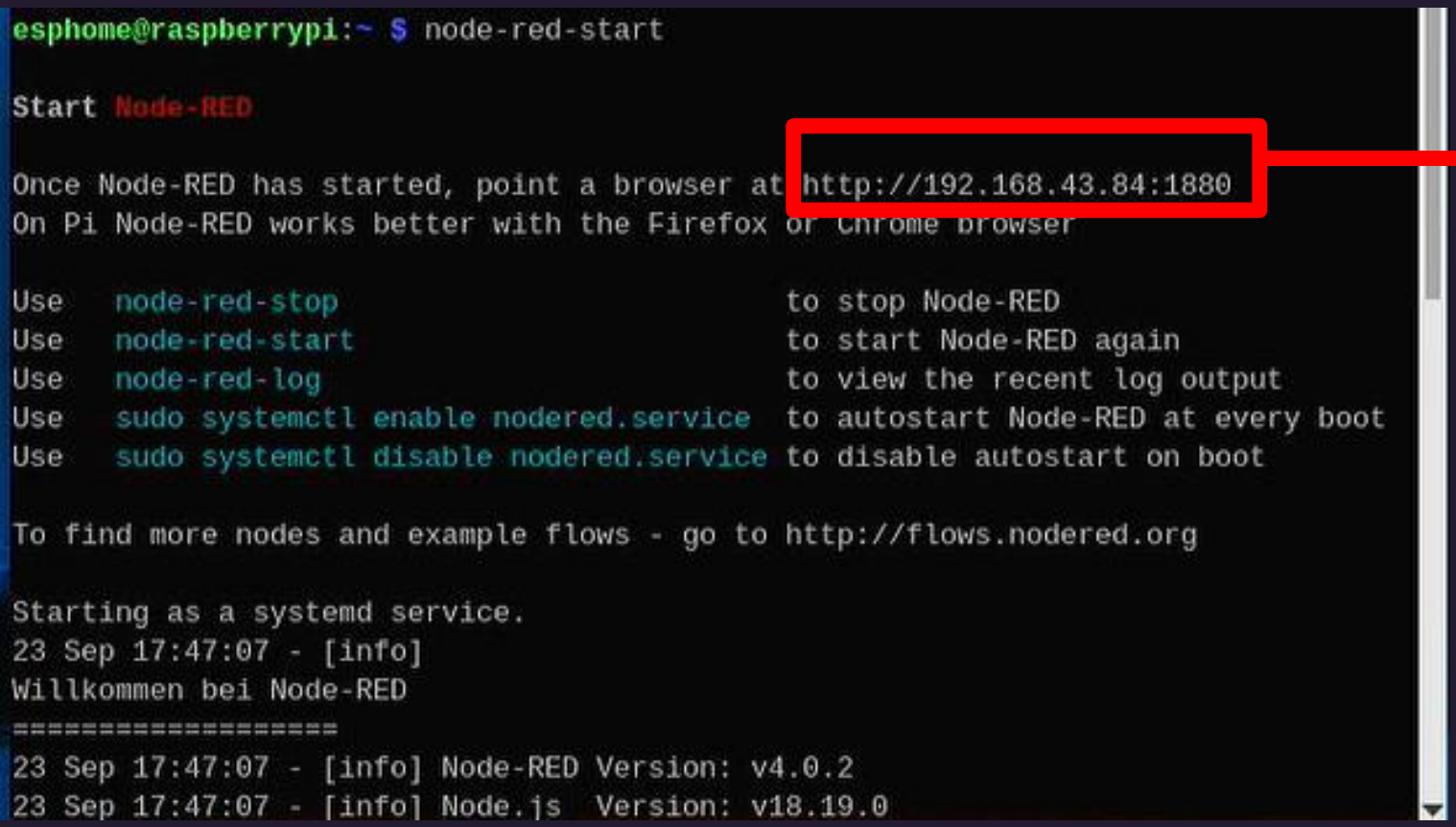
1.1. Installiere Node-RED mit dem offiziellen Installationsskript:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

2. Starte Node-RED nach der Installation:

```
node-red-start
```

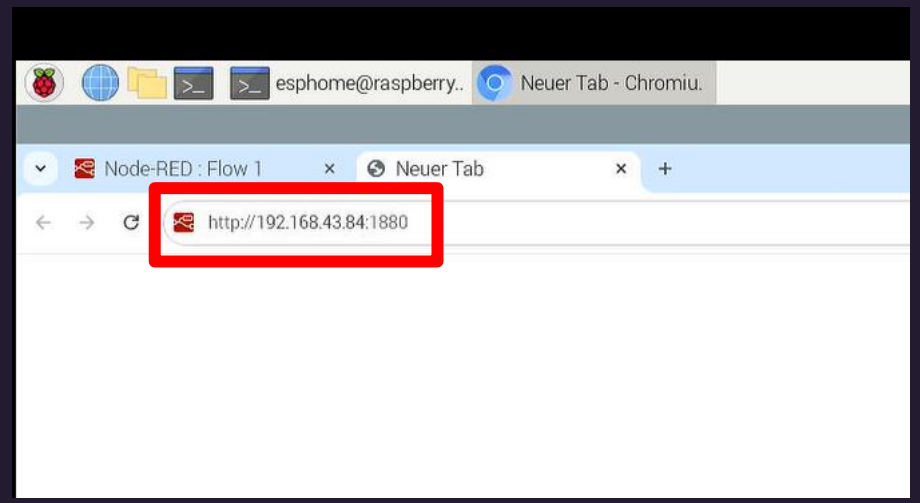
Eine Seite wie diese sollte geöffnet werden



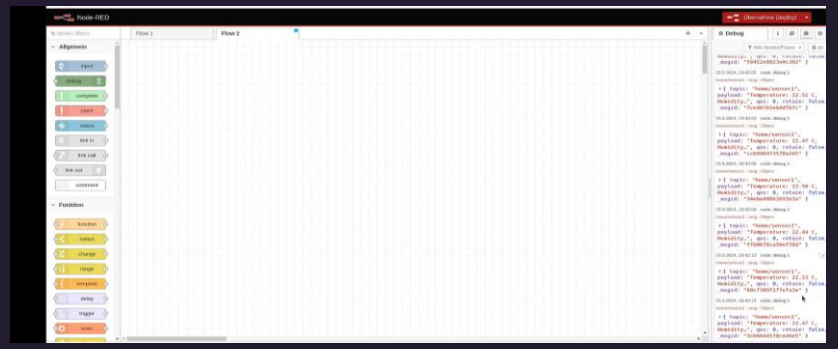
IP-Adresse

3. Aufrufen der Node-RED-Benutzeroberfläche

Node-RED läuft nun auf dem Standard-Port 1880. Um die Benutzeroberfläche zu öffnen, gehe in deinen Browser und gib die IP-Adresse deines Raspberry Pi gefolgt von :1880 ein:



Eine Seite wie diese sollte geöffnet werden



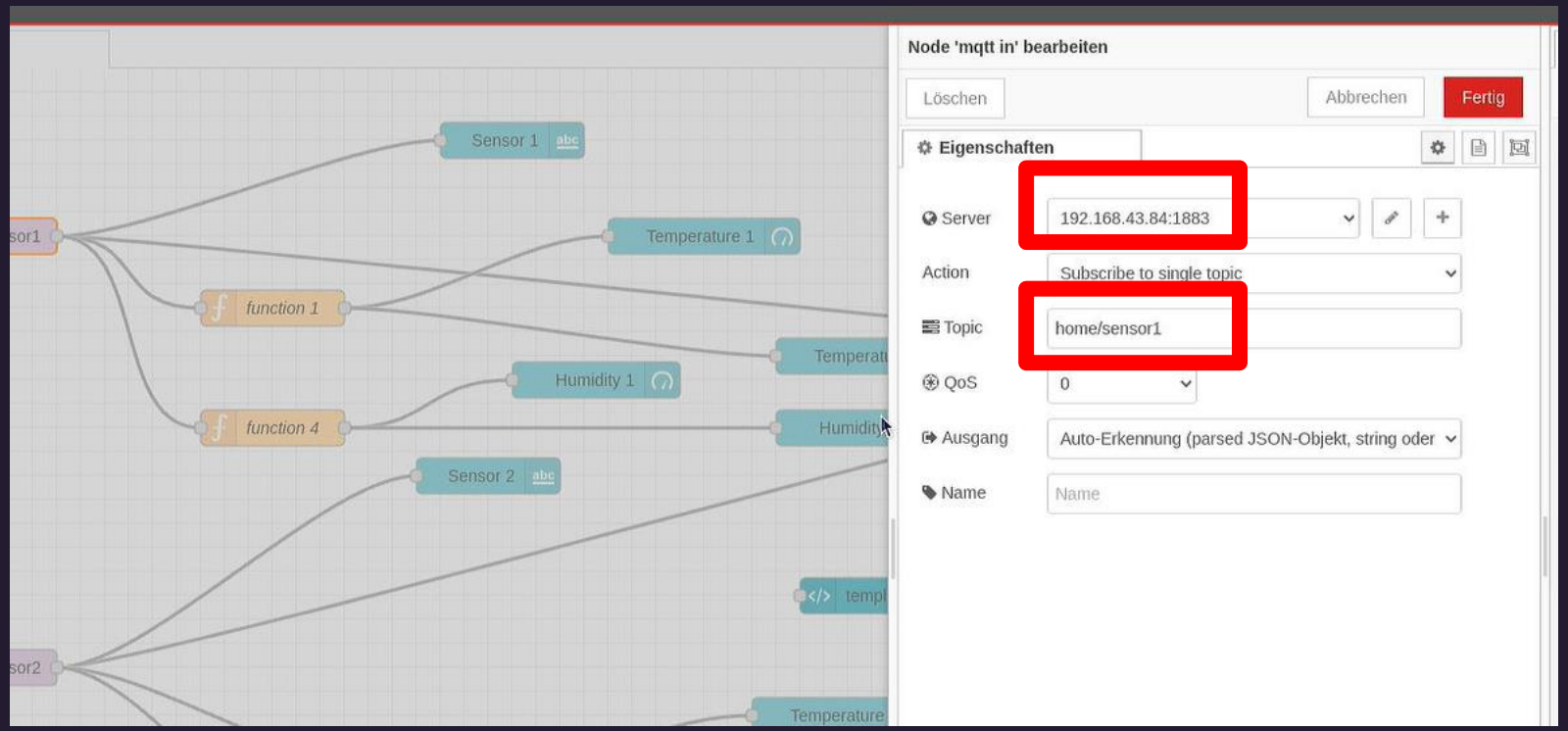
4. MQTT In Node konfigurieren, um Daten vom ESP32 zu empfangen:

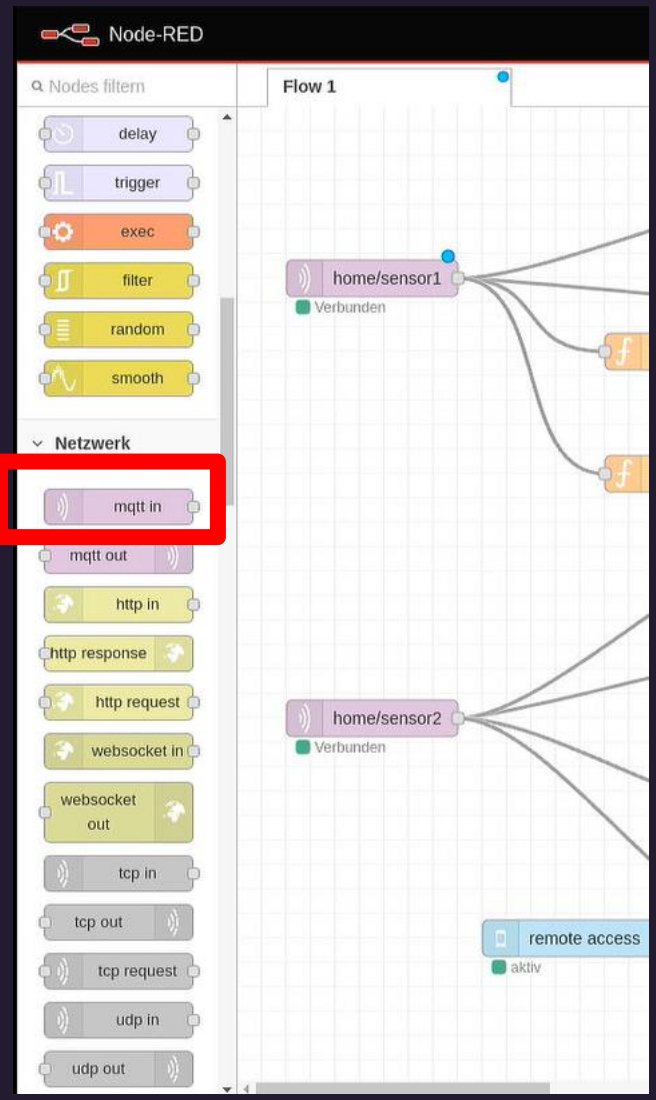
Der MQTT In Node empfängt die von deinem ESP32 gesendeten Daten über MQTT. Dieser Node wird so konfiguriert, dass er die Nachrichten, die über den MQTT-Broker des Raspberry Pi gesendet werden, abfängt.

Detaillierte Schritte:

- **MQTT In Node hinzufügen:**
- Öffne den Node-RED Editor.
- Ziehe den **MQTT In** Node aus der Palette auf dein Arbeitsfeld.
- MQTT In Node konfigurieren:
- Doppelklicke auf den Node, um ihn zu konfigurieren.
- Unter "Server":
- Klicke auf "Add new mqtt-broker…" und trage die IP-Adresse deines Raspberry Pi ein (z.B. 192.168.43.84, wie du es im ESP32-Code benutzt hast).
- Der Port sollte 1883 sein, was der Standardport für MQTT ist.

- Unter "Topic":
- Trage das MQTT-Topic ein, das der ESP32 zum Senden der Daten verwendet (z.B. home/sensor1 für den ersten Sensor und home/sensor2 für den zweiten Sensor).
- Klicke auf Done, um die Einstellungen zu speichern.





5. Debug Node konfigurieren, um die Daten zu überwachen:

Der Debug Node ist wichtig, um sicherzustellen, dass die Daten korrekt empfangen werden. Er zeigt die Daten in der Debug-Konsole von Node-RED an, was eine nützliche Überprüfung ist, bevor man sie visualisiert.

Detaillierte Schritte:

- **Debug Node hinzufügen:**
- Ziehe den Debug Node aus der Palette auf das Arbeitsfeld.
- Verbinde den Ausgang des MQTT In Nodes mit dem Eingang des Debug Nodes.
- **Debug Node konfigurieren:**
- Doppelklicke auf den Debug Node, um ihn zu konfigurieren.
- Stelle sicher, dass "msg.payload" ausgewählt ist, damit die empfangenen Sensordaten im Debug-Bereich angezeigt werden.
- Klicke auf Done, um die Konfiguration zu speichern.

- **Flow deployen:**

- Klicke auf Deploy (oben rechts), um den Flow zu aktivieren.

- Gehe auf die Debug-Konsole (rechte Seite des Node-RED Editors) und beobachte, wie die Sensordaten vom ESP32 in msg.payload angezeigt werden.

Dies bestätigt, dass die Kommunikation zwischen dem ESP32 und dem Raspberry Pi über MQTT funktioniert und die Daten erfolgreich empfangen werden.

6. Zusätzliche Nodes für die grafische Darstellung der Daten:

Nachdem du die Debug-Überprüfung durchgeführt hast, kannst du die Daten auf einer grafischen Benutzeroberfläche (Dashboard) darstellen. Dazu verwendest du Nodes wie Gauge, Chart und Text.

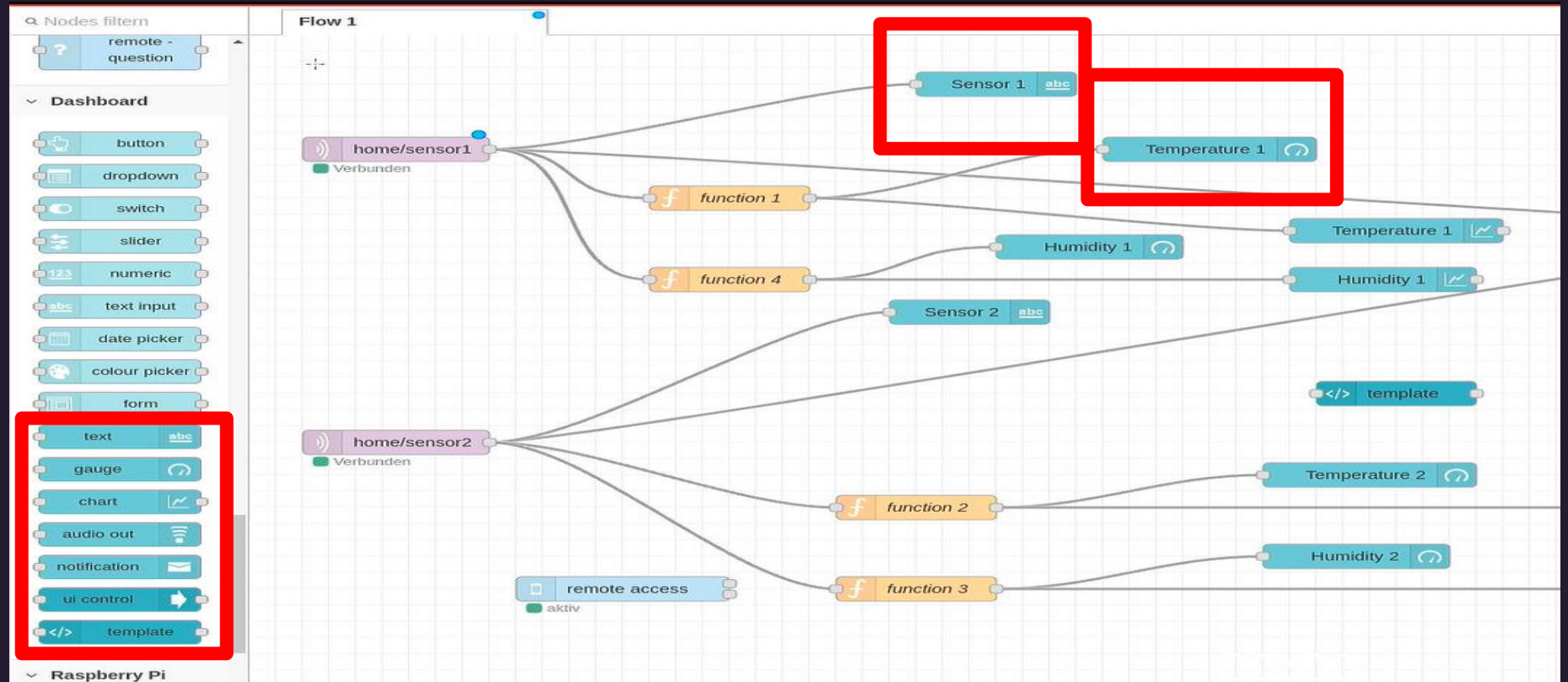
Detaillierte Schritte:

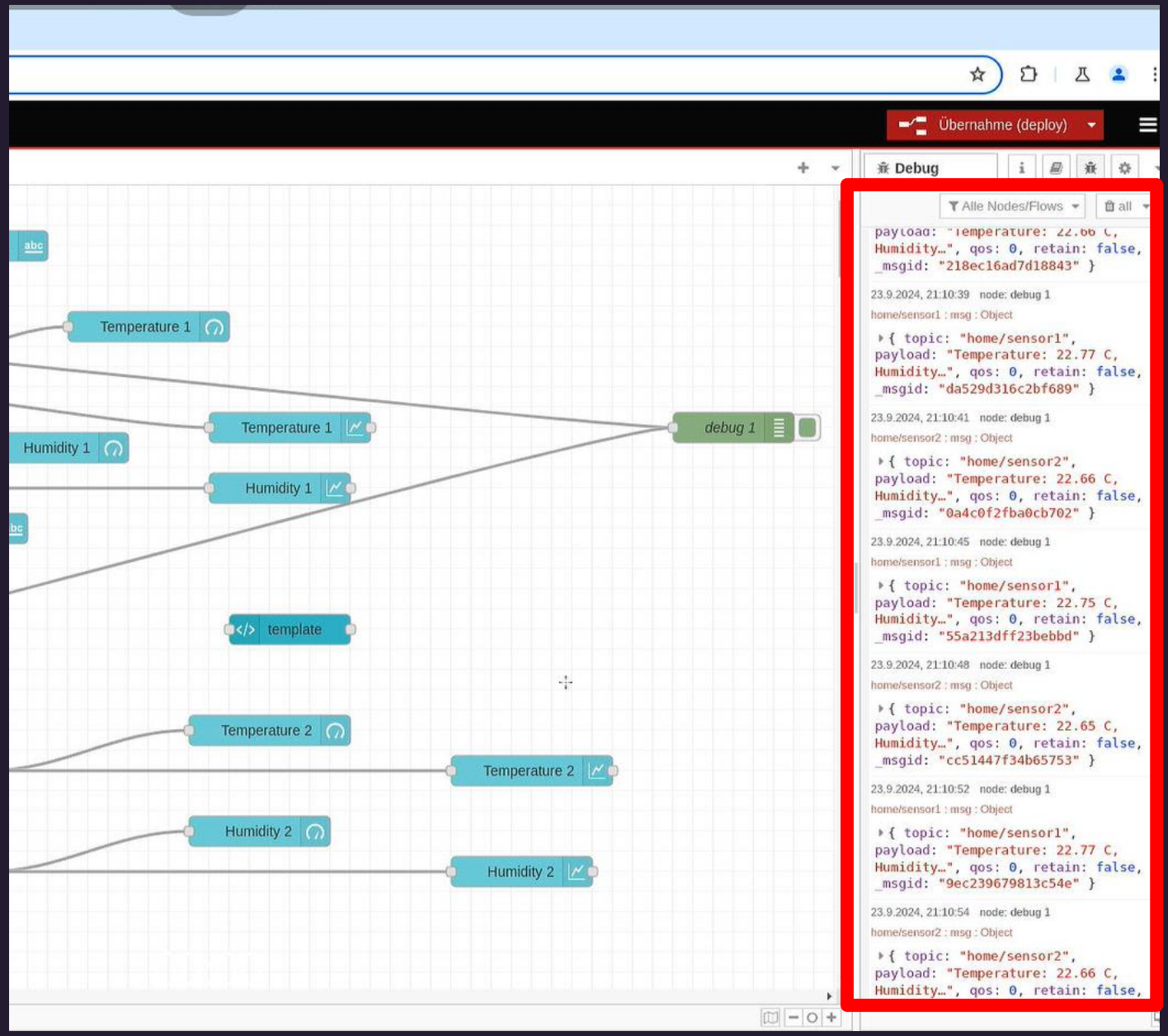
- Gauge Node:

- Ziehe den Gauge Node auf das Arbeitsfeld und verbinde ihn mit dem MQTT In Node.

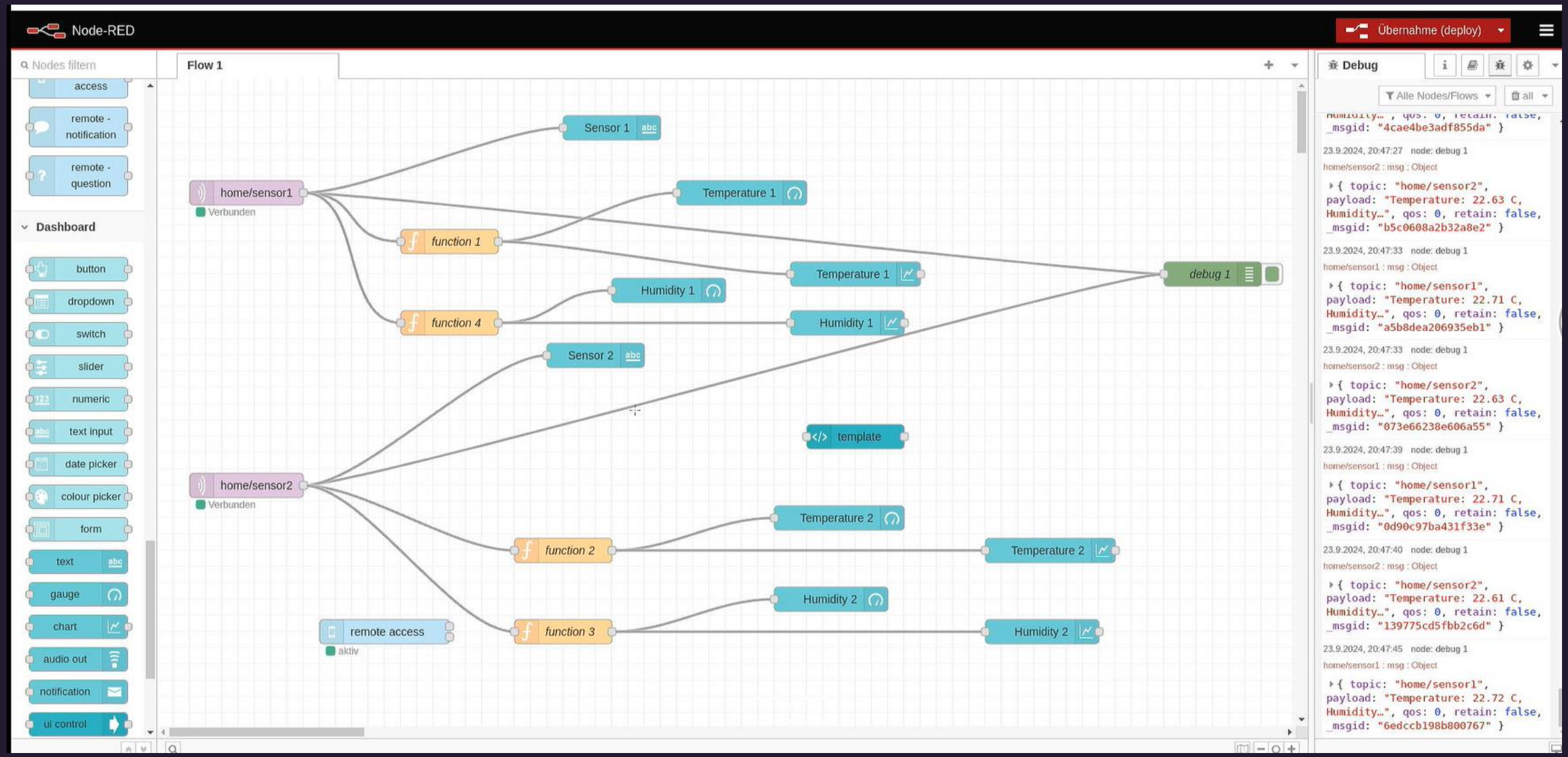
- Dieser Node zeigt die Temperatur oder Luftfeuchtigkeit als Rundinstrument an. dem MQTT In Node.

- **Chart Node:**
- Ziehe den Chart Node auf das Arbeitsfeld und verbinde ihn mit dem MQTT In Node.
- Der Chart Node visualisiert die Daten in einem Liniendiagramm, das den Verlauf der Messwerte anzeigt.
- **Text Node:**
- Der Text Node zeigt den aktuellen Wert als Text an (z.B. „Temperatur: 25°C“). Ziehe ihn ebenfalls auf das Arbeitsfeld und verbinde ihn mit dem MQTT In Node.





Nachdem du den Flow deployed hast, zeigt der Debug-Bereich alle empfangenen Daten in Echtzeit an, sodass du überprüfen kannst, ob die Datenübertragung vom ESP32 über den MQTT-Broker des Raspberry Pi erfolgreich ist.


Node-RED
Übernahme (deploy)
Nodes filtern
Flow 1
Debug
Alle Nodes/Flows
all
access
remote - notification
remote - question
Dashboard
button
dropdown
switch
slider
numeric
text input
date picker
colour picker
form
text
gauge
chart
audio out
notification
ui control
Sensor 1
home/sensor1
Verbunden
Temperature 1
function 1
Temperature 1
Humidity 1
function 4
Humidity 1
debug 1
Sensor 2
template
home/sensor2
Verbunden
Temperature 2
function 2
Temperature 2
Humidity 2
remote access
aktiv
function 3
Humidity 2
_msgid: "4cae4be3adf855da" }
23.9.2024, 20:47:27 node: debug 1
home/sensor2 : msg : Object
{ topic: "home/sensor2", payload: "Temperature: 22.63 C, Humidity…", qos: 0, retain: false, _msgid: "b5c0608a2b32a8e2" }
23.9.2024, 20:47:33 node: debug 1
home/sensor1 : msg : Object
{ topic: "home/sensor1", payload: "Temperature: 22.71 C, Humidity…", qos: 0, retain: false, _msgid: "a5b8dea206935eb1" }
23.9.2024, 20:47:33 node: debug 1
home/sensor2 : msg : Object
{ topic: "home/sensor2", payload: "Temperature: 22.63 C, Humidity…", qos: 0, retain: false, _msgid: "073e66238e606a55" }
23.9.2024, 20:47:39 node: debug 1
home/sensor1 : msg : Object
{ topic: "home/sensor1", payload: "Temperature: 22.71 C, Humidity…", qos: 0, retain: false, _msgid: "0d90c97ba431f33e" }
23.9.2024, 20:47:40 node: debug 1
home/sensor2 : msg : Object
{ topic: "home/sensor2", payload: "Temperature: 22.61 C, Humidity…", qos: 0, retain: false, _msgid: "139775cd5fbb2c6d" }
23.9.2024, 20:47:45 node: debug 1
home/sensor1 : msg : Object
{ topic: "home/sensor1", payload: "Temperature: 22.72 C, Humidity…", qos: 0, retain: false, _msgid: "6edccb198b800767" }

UI-Visualisierung:

- Gehe in die Dashboard-Einstellungen (über die Dashboard-Seitenleiste in Node-RED) und erstelle eine Benutzeroberfläche.
- Hier kannst du verschiedene Gruppen und Panels definieren, um die Daten in einem übersichtlichen Layout darzustellen.

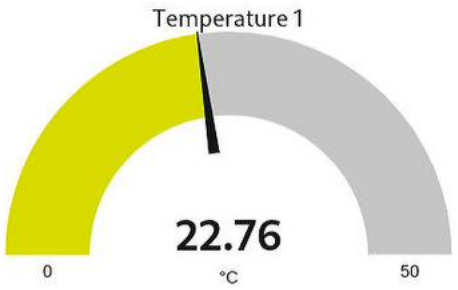
Dashboard anzeigen:

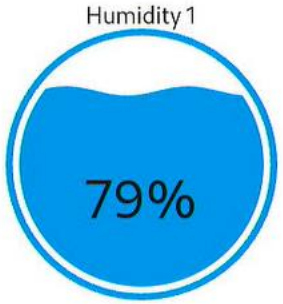
- Nachdem du das Layout konfiguriert und den Flow erneut deployed hast, kannst du das Node-RED Dashboard aufrufen, indem du im Browser die URL http://<Raspberry-Pi-IP>:1880/ui eingibst.
- Hier siehst du dann die Gauge, Charts und Textfelder, die die Sensordaten anzeigen

Gesamt

Sensor 1

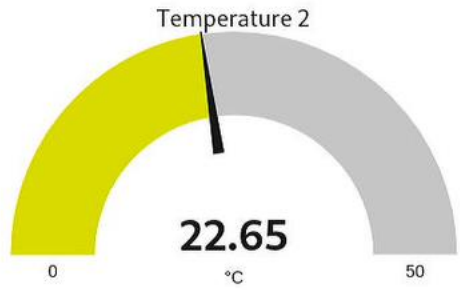
Temperature: 22.76 C, Humidity: 79 %

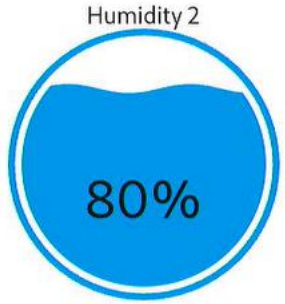




Sensor 2

Temperature: 22.65 C, Humidity: 80 %





Node-RED : Flow 1 | Node-RED Dashboard

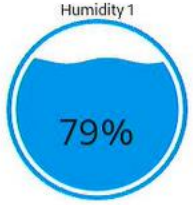
Nicht sicher 192.168.43.84:1880/ui/#!/0?socketid=2dVgUGDHawreEiT9AAAB

Gesamt

Sensor 1
Temperature: 22.75 C, Humidity: 79 %

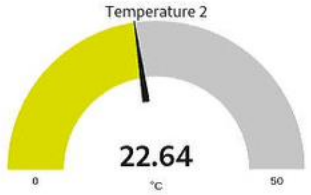
Temperature 1

Humidity 1



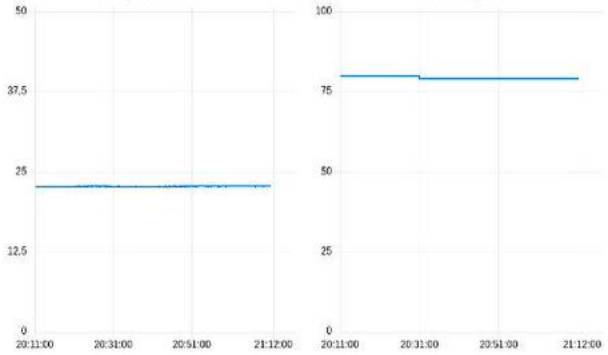
Sensor 2
Temperature: 22.64 C, Humidity: 79 %

Temperature 2

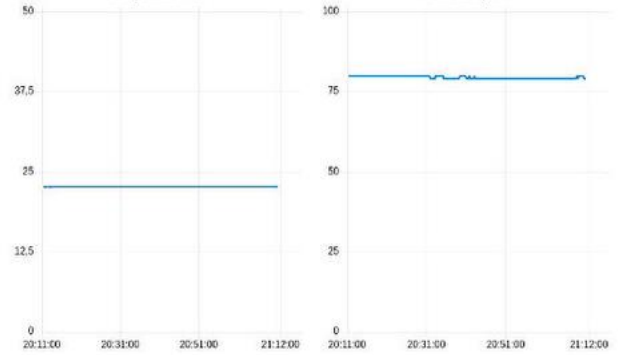


Humidity 2

Temperature 1 | Humidity 1



Temperature 2 | Humidity 2



Professor:Hartmut Opperskalski
Studenten:Amirhossein Kafilzadeh,Vaishnav Chitte,Kasra Sharifinejad,[illegible]

Codebeschreibung

Hier ist eine detaillierte Erklärung des Codes in Deutsch, die beschreibt, wie die Daten vom ESP32 an den Raspberry Pi gesendet werden:

1. Eingebundene Bibliotheken

- WiFi.h: Diese Bibliothek verwaltet die WiFi-Verbindung des ESP32.
- PubSubClient.h: Dient der MQTT-Kommunikation. Sie ermöglicht es dem ESP32, Daten an den MQTT-Broker (den Raspberry Pi) zu senden.
- NimBLEDevice.h: Diese Bibliothek ist für die Bluetooth Low Energy (BLE) Kommunikation. Damit kann der ESP32 als BLE-Client agieren und sich mit BLE-Geräten verbinden.

2. WiFi-Zugangsdaten und MQTT-Broker-Konfiguration

```
const char* ssid = "HUAWEI Y9 2019";
const char* password = "123456789";

const char* mqtt_server = "192.168.43.84";  // IP-Adresse des Raspberry Pi
```

- ssid und password enthalten die Zugangsdaten für das WiFi-Netzwerk, mit dem sich der ESP32 verbindet.
- mqtt_server speichert die IP-Adresse des MQTT-Brokers, der auf dem Raspberry Pi läuft.

3. WiFi- und MQTT-Clients

```
WiFiClient espClient;
PubSubClient client(espClient);
```

- WiFiClient espClient: Dieser Client kümmert sich um die Netzwerkverbindung mit dem Raspberry Pi über WiFi.
- PubSubClient client: Der MQTT-Client, der die Verbindung mit dem MQTT-Broker herstellt und für den Versand von Nachrichten verantwortlich ist.

4. WiFi-Verbindung zum Netzwerk herstellen

Die Funktion connectToWiFi() stellt die Verbindung her, indem sie den ESP32 mit dem angegebenen Netzwerk verbindet und die Verbindung überwacht:

```
void connectToWiFi() {
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 20) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("");
    Serial.println("WiFi connected successfully");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("");
    Serial.println("Failed to connect to WiFi");
  }
}
```

- WiFi.begin(ssid, password): Startet den Verbindungsprozess.
- WiFi.status(): Überprüft, ob die Verbindung erfolgreich war.
- Wenn der ESP32 verbunden ist, gibt er die IP-Adresse aus.

5. MQTT-Verbindung zum Raspberry Pi

Nach der WLAN-Verbindung verbindet sich der ESP32 mit einem MQTT-Broker, der auf dem Raspberry Pi läuft. Dieser Broker verwendet das MQTT-Protokoll, um Nachrichten zwischen Geräten zu vermitteln. Die IP-Adresse des Raspberry Pi wird hier festgelegt:

```
const char* mqtt_server = "192.168.43.84";  // IP-Adresse des Raspberry Pi (MQTT-Broker)
```

Diese IP muss die des Raspberry Pi in deinem Netzwerk sein.

6. MQTT-Client initialisieren

```
client.setServer(mqtt_server, 1883);
```

7. MQTT-Verbindung wiederherstellen

Falls die Verbindung zwischen dem ESP32 und dem MQTT-Broker unterbrochen wird, stellt die reconnect()-Funktion sicher, dass der ESP32 die Verbindung wieder aufnimmt. Diese Funktion wird wiederholt aufgerufen, bis die Verbindung erfolgreich ist:

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client")) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" (try again in 5 seconds)");
      delay(5000);  // Warte 5 Sekunden und versuche erneut zu verbinden
    }
  }
}
```

- client.connected(): Prüft, ob die MQTT-Verbindung besteht.
- client.connect("ESP32Client"): Verbindet den ESP32 mit dem Broker. Der ESP32 erhält die ID "ESP32Client".
- Bei Fehlschlag wartet der ESP32 5 Sekunden und versucht es erneut.

8. MQTT-Daten senden

Der ESP32 sendet Daten über MQTT an den Raspberry Pi mit client.publish(). In deinem Code werden Sensordaten als MQTT-Nachricht an den Broker gesendet. Ein Beispiel hierfür ist das Senden von Daten für Sensor 1:

```
client.publish("home/sensor1", payload.c_str());
```

- "home/sensor1": Dies ist das Topic, unter dem die Daten veröffentlicht werden. Ein Topic ist eine Art "Adresse", unter der der Broker die Nachrichten empfängt.
- payload: Die Daten, die in der Nachricht gesendet werden (z.B. Temperatur und Feuchtigkeit). Diese werden als String formatiert und dann an das Topic gesendet.

Für Sensor 2 wird ein anderes MQTT-Topic verwendet:

```
client.publish("home/sensor2", payload.c_str());
```

Die Nachricht wird hier über das Topic home/sensor2 veröffentlicht.

9. MQTT-Client aufrechterhalten

Damit die MQTT-Verbindung dauerhaft aktiv bleibt und der ESP32 neue Daten senden und empfangen kann, wird client.loop() in der loop()-Funktion aufgerufen. Dies stellt sicher, dass der ESP32 weiterhin mit dem Broker kommunizieren kann:

```
client.loop();
```

Zusammenfassung der MQTT-Kommunikation:

Der ESP32 verbindet sich über die connectToWiFi()-Funktion mit dem lokalen WLAN.Der MQTT-Client wird mit der client.setServer()-Funktion konfiguriert, um den Broker (auf dem Raspberry Pi) zu erreichen.Bei Verbindungsverlust stellt die reconnect()-Funktion sicher, dass der ESP32 versucht, die Verbindung wiederherzustellen.Der ESP32 sendet Daten über client.publish() an den Raspberry Pi, der diese Daten unter einem bestimmten MQTT-Topic empfängt.client.loop() stellt sicher, dass die MQTT-Verbindung aktiv bleibt und der ESP32 weiter Nachrichten senden und empfangen kann.Der Raspberry Pi fungiert in diesem Szenario als MQTT-Broker und empfängt die veröffentlichten Nachrichten des ESP32.