**Dokumentation MQTT Projekt - Florian Diehle**

**08.02.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

- Begriffsdefinition und Erklärungen

- Versuch Zugriff auf Raspberry Pi per SSH

- MQTT Broker Mosquito 2.0.18 installiert

- MQTT Fx installiert

- Versuch Verbindungsaufbau (fehlgeschlagen)

- Broker Dienst gestartet

- publish Nachricht erstellt

neue Topic erstellt: "Test"

Daten auf dem Topic gepublished: 25

- subscribe Nachricht erstellt

Topic "Test" subscribed

neue Daten gesendet: 25

Daten auf Client empfangen: 25

- Broker auf öffentlich gestellt und mit anderen Clients aus der Gruppe getestet

-schreiben Dokumentation

**11.04.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

-ESP-Sketch geschrieben

-testen des publishen über den ESP

-auf Mosquito gepublished als Test

-Node-JS installiert für Node-Red

-install von Node-Red

-hat nicht funktioniert (Server startet nicht) -> erneuter Ablauf des installs

-Versuch bei Cornelius zu installieren -> Abgebrochen

-Update des Schaubildes für MQTT Broker

-schreiben Dokumentation

**18.04.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

-Neue Installation von MQTT wegen Abwesenheit von Jonas Wolf

-Vebindungsaufbau (erfolgreich)

-Test publish

-Installation von Node Red (erfolgreich)

-Update Node Red auf neuste Version

-Start Node Red

-Verbindung mit Node Red über localhost

-Node Red Verbindung mit MQTT Server

-Installation Node Red Dashboard

-Erstellen von Textfeld für die gesendete Temperatur | Grad Celsius

-Erstellen von einem Diagramm für die Ausgabe der Temperatur | Grad Celsius

-Erstellen von Textfeld für die gesendete Temperatur | Grad Fahrenheit

-Erstellen Diagramm für die Ausgabe der Temperatur | Grad Fahrenheit

-Anzeigen von Temperatur im Dashboard (erfolgreich)

-schreiben Dokumentation

**16.05.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

-Wiederaufbau Temperatursenor und Esp32

-Start MQTT auf Laptop Jonas

-Start Node Red auf Laptop Cornelius

-Start Mosquito auf Laptop Jonas

-Test Publish (nicht erfolgreich)

-Neuer Test (erfolgreich)

auf Topic Jonas Message: Test -> empfangen auf Cornelius Laptop in Topic Jonas

-Code umgeschrieben, weil Jonas seinen eigenen Sensor der vorher verwendet wurde vergessen hat

-Erstellen von Dashboard in NodeRed für Ausgabe von Temperaturen

-Erstellen von Textfeld für die gesendete Temperatur vom Esp32 | Grad Celsius

-Erstellen von einem Diagramm für die gesendete Temperatur vom Esp32 | Grad Celsius

-Anpassen von Code für Celsius und Fahrenheit auf den Topics Temperature\_Celsius und Temperature\_Fahrenheit

-Erstellen von Textfeld für die gesendete Temperatur vom Esp32 | Grad Fahrenheit

-Erstellen von einem Diagramm für die gesendete Temperatur vom Esp32 | Grad Fahrenheit

-Node Red MySql Modul gesucht

-node-red-mysql-r2 installiert

-Erstellen neuer Datenbank in Heidi SQL mit Namen NodeRed (fehlgeschlagen)

-installieren von Postgresql

-schreiben Dokumentation

## Begriffe, Definitionen usw.

MQTT (Message Queuing Telemetry Protocol) ist ein Netzwerkprotokoll für die Kommunikation zwischen Maschinen bzw. von Machine zu Machine (M2M).

Es wurde 1999 von Dr. Andy Stanford-Clark und Arlen Nipper erfunden. Das Protokoll wurde für minimalen Batterieverlust

und minimale Bandbreite, bei der Kommunikation zwischen Satellit und Ölpipelines entwickelt und später wurde der primäre

Fokus auf das Internet of Things und dessen Anwendungsfälle gelegt.

MQTT-Broker/Server = Aufgabe des Brokers ist das Filtern aller eingehenden Nachrichten, sowie

das verteilen an die Abonnenten.

MQTT-Client = jedes Gerät, vom Server bis zum Microcontroller,

auf dem eine MQTT-Bibliothek läuft und das über ein Netzwerk kommuniziert.

Subscribe-Nachricht = ein Client sendet eine SUBSCRIBE-Nachricht an den MQTT-Broker

um über Themen von Interesse informiert zu werden.

Diese Nachricht besteht aus einem eindeutigen Bezeichner und einer Liste aus Abonnements.

Das Topic übersetzt Thema, wird vom Broker zum Filtern von Nachrichten aller Clients verwendet.

Das Topic besteht aus einer oder mehreren Themenebenen. Jede Themenebene wird durch einen Schrägstrich (/) getrennt.

Wenn ein Client ein Topic abonniert, kann er genau das Topic einer veröffentlichten Nachricht abonnieren oder er kann Platzhalter verwenden, um mehrere Topics gleichzeitig zu abonnieren.

Publish-Nachricht bezeichnet eine Nachricht, die ein MQTT-Client an einen MQTT-Broker sendet, um sie an alle Clients zu verteilen, die sich für dieses spezifische Thema (Topic) abonniert haben+

Payload:

Jeder Broker verteilt Nachrichten die einen Topic besitzen und, logischerweise, einen Nachrichteninhalt. Dies ist der Payload.

Obwohl dem Payload keine Struktur vorgeschrieben ist, sollte man sich an eine gewisse Struktur binden damit dieser von anderen Geräten und Software gelesen werden kann.

Mögliche Optionen wären unter anderem JSON, XML oder OPC UA. Falls die Struktur festgelegt wurde hat dies sen Vorteil das es auch zu einer Reibungslosen Kommunikation unter den Geräten kommt.

Quality of Service

Der Quality of Service gibt an, wie sicher eine Nachricht schlussendlich zugestellt werden soll. Insgesamt gibt es drei verschiedene Stufen, von 0 bis 2

Stufe 0:

Es wird höchstens einmal übertragen.

Stufe 1:

Es wird mindestens einmal übertragen.

Stufe 2:

Es wird genau einmal übertragen.

JSON:

JSON (JavaScript Object Notation) ist ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen (Analysieren von Datenstrukturen) und zu generieren ist.

**Code für den Esp32**

// Es werden diverse Biblotheken eingebunden

// für z.B WLAN-Kommunikation, Sensordaten auslesen

#include <WiFi.h>

#include <PubSubClient.h>

#include <Wire.h>

#include <OneWire.h>

#include <DallasTemperature.h>

// Sensor Setup

#define ONE\_WIRE\_BUS 4

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

// Hier werden einige Daten festgelegt die für die WLAN bzw MQTT verbindungen wichtig sind

const char\* ssid = "WLAN-ESP";

const char\* password = "agsesp32";

const char\* mqtt\_server = "10.0.42.12";

// WLAN, MQTT Setup

WiFiClient wifi\_client;

PubSubClient client(wifi\_client);

void setup() {

// Die Serielle Kommunikation wird zum debuggen verwendet

Serial.begin(115200);

while (!Serial);

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

// Die WLAN-Verbindung wird aufgebaut

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

// Die MQTT-Broker Informationen werden gesetzt

client.setServer(mqtt\_server, 1883);

// Der Sensorbus wird gestartet

sensors.begin();

}

void loop() {

// hier wird geschaut ob die Verbindung zum MQTT-Broker

// noch besteht

if (!client.connected()) {

reconnect();

}

// es werden die neuen Temperaturen gemessen

sensors.requestTemperatures();

// in entsprechende variablen gespeichert

// und in zeichenketten zum übertragen umgewandelt

float tempC = sensors.getTempCByIndex(0);

char tempCString[8];

dtostrf(tempC, 6, 2, tempCString);

Serial.println(tempCString);

float tempF = sensors.getTempFByIndex(0);

char tempFString[8];

dtostrf(tempF, 6, 2, tempFString);

Serial.println(tempFString);

// Die Daten werden auf den jeweiligen Topics veröffentlicht

client.publish("Temperature\_Celsius", tempCString);

client.publish("Temperature\_Fahrenheit", tempFString);

// Der Messvorgang wird um 2s verzögert

delay(2000);

}

// Diese Methode baut die Verbindung zum MQTT-Broker auf

void reconnect() {

while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

// Der Client erhält bei jedem Verbindungsversuch eine neue einzigartige ID damit keine Konflikte

// mit bereits bestehenden Clients auftreten

String clientId = "ESP32Client-";

clientId += String(random(0xffff), HEX);

if (client.connect(clientId.c\_str())) {

Serial.println("connected");

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

delay(5000);

}

}

}

