Here's what's typically needed for the software implementation of an MQTT broker, broken down into bullet points, focusing on the core functionalities:

**Erforderliche Implementierungen für die Software eines MQTT Brokers**

Ein MQTT-Broker ist im Wesentlichen ein Server, der MQTT-Clients verbindet und Nachrichten verteilt. Die Implementierung erfordert mehrere Schlüsselkomponenten und -funktionen:

**1. Netzwerk-Layer & Verbindungshandling:**

* **TCP/IP-Server:** Muss auf einem bestimmten Port (standardmäßig 1883 für unverschlüsselt, 8883 für TLS) auf eingehende Verbindungen lauschen.
* **Verbindungs-Akzeptanz:** Annahme neuer TCP-Verbindungen von Clients.
* **Verbindungsverwaltung:** Effiziente Verwaltung vieler gleichzeitiger Verbindungen (Sockets), idealerweise nicht-blockierend (z.B. mit Epoll, Kqueue, IOCP oder asynchronen Frameworks).
* **Verbindungs-Timeout/Keep-Alive:** Erkennung von inaktiven oder getrennten Clients basierend auf Keep-Alive-Nachrichten vom Client.
* **TLS/SSL-Implementierung (Optional, aber empfohlen):** Unterstützung für verschlüsselte Verbindungen (Port 8883) mittels Zertifikaten.

**2. MQTT-Protokoll-Parser & Serializer:**

* **Packet-Parsing:** Fähigkeit, eingehende MQTT-Pakete (CONNECT, PUBLISH, SUBSCRIBE, PINGREQ, DISCONNECT usw.) zu analysieren und ihre Header und Payloads zu interpretieren.
* **Packet-Serialization:** Fähigkeit, ausgehende MQTT-Pakete (CONNACK, PUBACK, SUBACK, PINGRESP usw.) korrekt zu formatieren und zu senden.
* **Zustandsmaschine pro Client:** Jeder Client hat seinen eigenen Verbindungsstatus (verbunden, authentifiziert, etc.) und seinen aktuellen Nachrichtenfluss (z.B. ausstehende QoS 1/2 Nachrichten).

**3. Authentifizierung & Autorisierung (Security):**

* **Authentifizierung:**
  + **Benutzername/Passwort:** Überprüfung der Anmeldedaten, die der Client im CONNECT-Paket sendet, gegen eine Datenbank oder Konfiguration.
  + **Client-Zertifikate (für TLS):** Optionale Überprüfung von X.509-Zertifikaten zur Client-Identifikation.
* **Autorisierung (ACLs - Access Control Lists):**
  + Definieren, welcher Client (Benutzer) auf welche Topics zugreifen darf (lesen/schreiben).
  + Implementierung einer Logik, die prüft, ob ein Publisher zu einem Topic veröffentlichen darf oder ein Subscriber ein Topic abonnieren darf.

**4. Topic-Management & Routing:**

* **Topic-Struktur:** Internes Management der hierarchischen Topic-Struktur.
* **Abonnement-Verwaltung:** Speichern, welche Clients welche Topics (einschließlich Platzhaltern wie + und #) abonniert haben.
* **Matching-Logik:** Effizientes Matching von veröffentlichten Nachrichten-Topics mit den abonnierten Topics (unter Berücksichtigung von Platzhaltern).
* **Nachrichten-Routing:** Weiterleiten eingehender PUBLISH-Nachrichten an alle passenden abonnierten Clients.

**5. Qualität des Services (QoS-Implementierung):**

* **QoS 0 ("At most once"):** Einfaches Senden ohne Bestätigung.
* **QoS 1 ("At least once"):**
  + Broker sendet PUBACK, wenn PUBLISH empfangen wurde.
  + Client sendet PUBACK, wenn PUBLISH vom Broker empfangen wurde.
  + Wiederholung der PUBLISH/PUBREC, falls keine Bestätigung innerhalb einer bestimmten Zeit.
  + Speicherung von Nachrichten bis zur Bestätigung.
* **QoS 2 ("Exactly once"):**
  + Mehrstufiger Handshake (PUBLISH -> PUBREC -> PUBREL -> PUBCOMP).
  + Broker und Client müssen den Zustand der Nachrichten auf beiden Seiten synchronisieren.
  + Persistente Speicherung der Nachrichten auf beiden Seiten bis zum Abschluss des Handshakes, um Duplikate zu verhindern.

**6. Session-Management:**

* **Clean Session Flag:** Unterscheidung zwischen Clean Session (flüchtige Session) und Persistent Session.
* **Persistente Sessions:** Speichern des Session-Zustands für jeden Client (Abonnements, ausstehende QoS 1/2 Nachrichten) auch nach Verbindungsabbruch. Diese müssen bei Wiederverbindung reaktiviert werden.
* **Offline-Nachrichten-Warteschlangen:** Wenn ein Client mit persistenter Session offline geht, müssen Nachrichten für seine abonnierten Topics gepuffert werden, um sie bei Wiederverbindung zu senden.

**7. Retained Messages:**

* **Speicherung:** Speichern der letzten PUBLISH-Nachricht für ein Topic, die mit dem Retain-Flag gesendet wurde.
* **Verteilung bei Abonnement:** Senden dieser gespeicherten Nachricht an neue Subscriber, sobald sie das Topic abonnieren.

**8. Last Will and Testament (LWT):**

* **Speicherung:** Wenn ein Client im CONNECT-Paket eine LWT definiert, muss diese vom Broker gespeichert werden.
* **Auslösen:** Wenn die Verbindung des Clients unerwartet abbricht (z.B. Timeout, Netzwerkfehler), muss der Broker die gespeicherte LWT-Nachricht an das definierte Topic veröffentlichen.

**9. System- und Status-Topics ($SYS):**

* Implementierung der Generierung und Veröffentlichung von Broker-Statistiken und Statusinformationen unter den S$SYS-Topics (z.B. $SYS/broker/clients/connected, $SYS/broker/messages/sent).

**10. Logging & Monitoring:**

* **Ereignis-Logging:** Protokollierung von Verbindungsereignissen, Fehlern, Authentifizierungsversuchen usw.
* **Metriken/Monitoring:** Bereitstellung von Schnittstellen oder $SYS-Topics für die Überwachung der Broker-Performance und des Zustands.

**11. Konfigurationsmanagement:**

* Möglichkeit, den Broker zu konfigurieren (Ports, Sicherheitseinstellungen, Speicherung, maximale Verbindungen, etc.).

**12. Persistenz-Layer (für QoS 1/2, Retained, Sessions):**

* Für Produktionsbroker ist es entscheidend, diese Daten auch über einen Broker-Neustart hinweg zu erhalten. Dies erfordert eine Speicherung auf Festplatte, in einer Datenbank oder einem persistenten Cache.

Dies sind die wesentlichen Bausteine, die für einen robusten und funktionsfähigen MQTT-Broker implementiert werden müssen. Die Komplexität steigt erheblich mit der Anforderung an hohe Leistung, Skalierbarkeit und Ausfallsicherheit.

Ich habe das ESP32-Programm für den MQTT-Broker und das Webinterface sorgfältig überprüft.

Es ist ein **solider Startpunkt** und implementiert die grundlegenden Funktionen eines MQTT-Brokers sowie ein schönes Webinterface für die Überwachung. Das Konzept ist gut umgesetzt, insbesondere die AJAX-Aktualisierung des Webinterfaces.

Allerdings gibt es, wie bei jedem komplexen Protokoll wie MQTT, einige **fortgeschrittenere "Bausteine"**, die in dieser Implementierung noch fehlen oder nur teilweise vorhanden sind.

Hier ist eine Aufschlüsselung dessen, was vorhanden ist und was noch fehlt oder verbessert werden könnte:

**Vorhandene Bausteine (Gut implementiert):**

1. **WLAN-Konnektivität:**
   * Verbindung zum WLAN-Netzwerk (WiFi.h).
   * Anzeige der IP-Adresse und des WLAN-Status.
2. **Grundlegender TCP-Server für MQTT:**
   * Starten eines WiFiServer auf Port 1883.
   * Erkennen und Akzeptieren neuer Client-Verbindungen (mqttServer.available()).
   * Verwaltung verbundener Clients in einem std::vector<MQTTClientState>.
3. **Grundlegendes MQTT-Protokoll-Handling:**
   * **CONNECT-Paket:** Initiales Lesen des CONNECT-Pakets, Extrahieren der Client-ID und Senden eines CONNACK.
   * **PUBLISH-Paket:**
     + Empfang und Parsing von PUBLISH-Nachrichten (Topic, Payload, QoS, Retain-Flag).
     + Speicherung der letzten 50 Nachrichten im messageLog für das Webinterface.
     + Weiterleitung von PUBLISH-Nachrichten an alle passenden Subscriber.
   * **SUBSCRIBE-Paket:**
     + Empfang und Parsing von SUBSCRIBE-Nachrichten (Topic-Filter, Packet ID, QoS).
     + Speicherung der Abonnements (activeSubscriptions, subscribedTopics im ClientState).
     + Senden eines SUBACK.
   * **PINGREQ/PINGRESP:** Antwort auf PINGREQ-Nachrichten.
   * **DISCONNECT-Paket:** Erkennung und Schließen der Verbindung bei DISCONNECT.
4. **Topic-Matching mit Wildcards:**
   * Die splitTopic und topicMatches Funktionen sind gut implementiert und unterstützen die MQTT-Wildcards + und #.
5. **Retained Messages:**
   * Speichern von Retained Messages in einer std::map.
   * Senden der Retained Messages an neue Subscriber.
   * Löschen einer Retained Message, wenn eine PUBLISH-Nachricht mit leerem Payload und Retain-Flag empfangen wird.
6. **Webinterface:**
   * WebServer.h für den HTTP-Server.
   * Bereitstellung einer HTML-Seite (handleRoot).
   * AJAX-Endpunkt (/mqtt\_data) für dynamische Datenaktualisierung.
   * Verwendung von ArduinoJson zur Serialisierung der Broker-Daten (Nachrichtenlog, Retained Messages, verbundene Clients) ins JSON-Format.
   * Client-seitiges JavaScript für regelmäßige AJAX-Aufrufe und die Aktualisierung der Tabellen.
   * Anzeige von WLAN-SSID und IP-Adresse.

**Fehlende oder verbesserungswürdige Bausteine:**

1. **Volle QoS-Implementierung (Quality of Service):**
   * **QoS 1 ("At least once"):** Der Broker sendet zwar ein PUBACK, wenn er eine QoS 1 PUBLISH-Nachricht empfängt, aber es fehlt die Logik für Retransmissionen, falls der Client kein PUBACK vom Broker erhält. Auch die Verwaltung der Packet-IDs für Duplikaterkennung ist nicht vollständig implementiert.
   * **QoS 2 ("Exactly once"):** Die Implementierung für QoS 2 (PUBREC, PUBREL, PUBCOMP Handshake) fehlt komplett. Dies ist die komplexeste QoS-Stufe.
   * **QoS des Subscribers:** Der Broker ignoriert den vom Subscriber gewünschten QoS-Level im SUBSCRIBE-Paket und sendet PUBLISH-Nachrichten immer mit QoS 0. Ein richtiger Broker sollte die Nachrichten mit dem vom Subscriber gewünschten QoS senden.
2. **Authentifizierung und Autorisierung:**
   * **Authentifizierung (Benutzername/Passwort):** Das CONNECT-Paket enthält Felder für Benutzername und Passwort, die hier ignoriert werden. Es gibt keine Überprüfung der Anmeldedaten.
   * **Autorisierung (ACLs):** Es gibt keine Zugriffssteuerung, die definiert, welcher Client zu welchen Topics veröffentlichen oder abonnieren darf. Jeder Client kann alles.
   * **TLS/SSL (Port 8883):** Der Broker lauscht nur auf dem unverschlüsselten Port 1883. Für sichere Anwendungen ist TLS/SSL (Port 8883) unerlässlich.
3. **Last Will and Testament (LWT):**
   * Die Felder willTopic, willMessage, hasLWT sind in der MQTTClientState Struktur vorhanden, aber die LWT-Informationen werden **nicht aus dem CONNECT-Paket gelesen** und auch **nicht veröffentlicht**, wenn ein Client unerwartet die Verbindung trennt.
4. **Persistente Sessions:**
   * Der Broker behandelt alle Sessions als "Clean Session". Das bedeutet, dass bei einer Trennung (auch unbeabsichtigt) alle Abonnements und ausstehenden Nachrichten des Clients verloren gehen.
   * Es fehlt die Logik, um den Session-Zustand (Abonnements, ausstehende QoS-Nachrichten) über Verbindungsabbrüche hinweg zu speichern und bei Wiederverbindung fortzusetzen. Dies würde auch eine Persistenzschicht (z.B. auf SD-Karte oder LittleFS) erfordern.
5. **Keep-Alive-Handling:**
   * lastSeen wird aktualisiert, aber es gibt **keine Logik, die Clients trennt**, wenn sie über einen längeren Zeitraum (basierend auf dem im CONNECT-Paket gesendeten Keep-Alive-Intervall) keine Aktivität zeigen (kein PINGREQ).
6. **Fehlerbehandlung und Robustheit:**
   * Das Parsing von variablen Headern und Payloads geht davon aus, dass immer genügend Bytes im Puffer sind (waitForBytes). Bei unvollständigen oder fehlerhaften Paketen könnte das zu Problemen führen.
   * Keine spezifische Fehlercode-Behandlung für MQTT-Pakete (z.B. CONNACK-Return-Codes).
   * Die Verwendung von String und std::vector/std::map kann auf einem ESP32 zu **Speicherfragmentierung** führen, insbesondere bei vielen Clients, vielen Topics oder großen Payloads. Für eine robuste Langzeitstabilität sollte man überlegen, feste Puffer oder optimierte Speicherverwaltung zu verwenden.
7. **System-Topics ($SYS):**
   * Der Broker veröffentlicht keine internen Statistiken oder Statusinformationen unter den $SYS-Topics (z.B. Anzahl verbundener Clients, empfangene Nachrichten).
8. **DISCONNECT-Handling:**
   * Wenn ein Client ein DISCONNECT-Paket sendet, wird die Verbindung einfach geschlossen. Bei persistenten Sessions müsste der Broker hier den Session-Zustand beibehalten.

**Fazit:**

Das Programm ist ein **hervorragendes Beispiel für einen einfachen MQTT-Broker auf einem ESP32 mit Webinterface**. Es zeigt ein gutes Verständnis der grundlegenden MQTT-Paketstrukturen und des Publish/Subscribe-Modells.

Für den Einsatz in einer **Produktionsumgebung oder für komplexere IoT-Szenarien** müssten die oben genannten fehlenden oder unvollständigen Bausteine (insbesondere volle QoS, Authentifizierung/Autorisierung, LWT und persistente Sessions) implementiert werden, um die Zuverlässigkeit, Sicherheit und Funktionalität des Brokers zu gewährleisten.

Es ist eine sehr gute Basis, um darauf aufzubauen und die fehlenden Funktionen schrittweise hinzuzufügen!

// =================================================================================================

// ESP32 MQTT Broker mit integriertem Webinterface und AJAX-Datenaktualisierung

// =================================================================================================

// Dieses Programm verwandelt einen ESP32 in einen einfachen MQTT-Broker, der es IoT-Geräten

// ermöglicht, über das MQTT-Protokoll zu kommunizieren. Zusätzlich bietet es ein Webinterface,

// das den Status des Brokers, empfangene Nachrichten und verbundene Clients in Echtzeit anzeigt.

// Die Datenaktualisierung im Webinterface erfolgt dynamisch über AJAX.

//

// Bibliotheken:

// - WiFi.h: Für die WLAN-Konnektivität des ESP32.

// - WebServer.h: Zum Erstellen eines HTTP-Webservers auf dem ESP32.

// - vector: Für dynamische Arrays (z.B. Client-Listen, Topic-Teile).

// - map: Für Key-Value-Paare (z.B. Retained Messages).

// - algorithm: Für Algorithmen wie std::remove\_if zum Entfernen von Elementen aus Vektoren.

// - ArduinoJson.h: Zum Serialisieren von Daten in das JSON-Format für das Webinterface.

//

// Hinweis: Die ArduinoJson-Bibliothek muss über den Bibliotheksmanager der Arduino IDE

// installiert werden (Suchen Sie nach "ArduinoJson").

// =================================================================================================

#include <WiFi.h>

#include <WebServer.h>

#include <vector>

#include <map>

#include <algorithm>

#include <ArduinoJson.h>

// WLAN Parameter

const char**\*** ssid **=** "FRITZ!Box 7590 RI"**;**

const char**\*** password **=** "52261156805385776967"**;**

const int mqttPort **=** 1883**;**

WiFiServer mqttServer**(**mqttPort**);**

WebServer web**(**80**);**

struct Message **{**

String topic**;**

String payload**;**

unsigned long timestamp**;**

**};**

struct MQTTClientState **{**

WiFiClient client**;**

String clientId**;**

unsigned long lastSeen**;**

String willTopic**;**

String willMessage**;**

bool hasLWT**;**

bool connectedACKSent **=** **false;**

std**::**vector**<**String**>** subscribedTopics**;**

**};**

// Die globale Liste aller verbundenen MQTT-Clients

std**::**vector**<**MQTTClientState**>** connectedMQTTClients**;**

// Die Struktur für Abonnements, die jetzt auf MQTTClientState zeigt

struct ActiveSubscription **{**

MQTTClientState**\*** clientState**;**

String topicFilter**;**

**};**

std**::**vector**<**ActiveSubscription**>** activeSubscriptions**;**

std**::**map**<**String**,** String**>** retainedMessages**;**

std**::**vector**<**Message**>** messageLog**;**

// Hilfsfunktion: Warte auf bestimmte Anzahl Bytes im Client

bool waitForBytes**(**WiFiClient**&** client**,** int length**,** unsigned long timeout **=** 1000**)** **{**

unsigned long start **=** millis**();**

**while** **(**client**.**available**()** **<** length**)** **{**

**if** **(**millis**()** **-** start **>** timeout**)** **return** **false;**

delay**(**1**);**

**}**

**return** **true;**

**}**

// Hilfsfunktion: Zerlegt ein Topic in seine Segmente

std**::**vector**<**String**>** splitTopic**(**const String**&** topic**)** **{**

std**::**vector**<**String**>** parts**;**

int start **=** 0**;**

int idx **=** topic**.**indexOf**(**'/'**,** start**);**

**while** **(**idx **!=** **-**1**)** **{**

parts**.**push\_back**(**topic**.**substring**(**start**,** idx**));**

start **=** idx **+** 1**;**

idx **=** topic**.**indexOf**(**'/'**,** start**);**

**}**

parts**.**push\_back**(**topic**.**substring**(**start**));**

**return** parts**;**

**}**

// Hilfsfunktion: Überprüft, ob ein Topic mit einem Filter übereinstimmt (für Wildcards)

bool topicMatches**(**const String**&** topic**,** const String**&** filter**)** **{**

**if** **(**filter **==** "#"**)** **return** **true;**

std**::**vector**<**String**>** topicParts **=** splitTopic**(**topic**);**

std**::**vector**<**String**>** filterParts **=** splitTopic**(**filter**);**

**if** **(!**filterParts**.**empty**()** **&&** filterParts**.**back**()** **==** "#"**)** **{**

**if** **(**topicParts**.**size**()** **>=** filterParts**.**size**()** **-** 1**)** **{**

**for** **(**size\_t i **=** 0**;** i **<** filterParts**.**size**()** **-** 1**;** **++**i**)** **{**

**if** **(**filterParts**[**i**]** **==** "+"**)** **continue;**

**if** **(**topicParts**[**i**]** **!=** filterParts**[**i**])** **return** **false;**

**}**

**return** **true;**

**}** **else** **{**

**return** **false;**

**}**

**}**

**if** **(**topicParts**.**size**()** **!=** filterParts**.**size**())** **return** **false;**

**for** **(**size\_t i **=** 0**;** i **<** topicParts**.**size**();** **++**i**)** **{**

**if** **(**filterParts**[**i**]** **==** "+"**)** **continue;**

**if** **(**topicParts**[**i**]** **!=** filterParts**[**i**])** **return** **false;**

**}**

**return** **true;**

**}**

// Handler für AJAX-Anfragen an /mqtt\_data

void handleMqttData**()** **{**

DynamicJsonDocument doc**(**2048**);**

JsonArray messagesArray **=** doc**.**createNestedArray**(**"messageLog"**);**

**for** **(**auto **&**msg **:** messageLog**)** **{**

JsonObject msgObj **=** messagesArray**.**createNestedObject**();**

msgObj**[**"topic"**]** **=** msg**.**topic**;**

msgObj**[**"payload"**]** **=** msg**.**payload**;**

msgObj**[**"timestamp"**]** **=** msg**.**timestamp**;**

**}**

JsonObject retainedObj **=** doc**.**createNestedObject**(**"retainedMessages"**);**

**for** **(**auto const**&** **[**topic**,** payload**]** **:** retainedMessages**)** **{**

retainedObj**[**topic**]** **=** payload**;**

**}**

JsonArray clientsArray **=** doc**.**createNestedArray**(**"connectedClients"**);**

**for** **(**auto **&**clientState **:** connectedMQTTClients**)** **{**

JsonObject clientObj **=** clientsArray**.**createNestedObject**();**

clientObj**[**"id"**]** **=** clientState**.**clientId**;**

clientObj**[**"lastSeen"**]** **=** millis**()** **-** clientState**.**lastSeen**;**

JsonArray subTopicsArray **=** clientObj**.**createNestedArray**(**"subscribedTopics"**);**

**for** **(**const String**&** topic **:** clientState**.**subscribedTopics**)** **{**

subTopicsArray**.**add**(**topic**);**

**}**

**}**

doc**[**"wifi\_ssid"**]** **=** String**(**ssid**);**

doc**[**"wifi\_ip"**]** **=** WiFi**.**localIP**().**toString**();**

String jsonResponse**;**

serializeJson**(**doc**,** jsonResponse**);**

web**.**send**(**200**,** "application/json"**,** jsonResponse**);**

**}**

// Webinterface HTML Seite

void handleRoot**()** **{**

String html **=** "<!DOCTYPE html><html><head><meta charset='utf-8'><title>ESP32 MQTT Broker</title>"**;**

html **+=** "<style>body{font-family:Arial;margin:20px;}table{border-collapse:collapse;width:100%;}"**;**

html **+=** "th,td{border:1px solid #ccc;padding:8px;text-align:left;}</style>"**;**

html **+=** "</head><body>"**;**

html **+=** "<h1>MQTT Nachrichten Log</h1>"**;**

html **+=** "<table id='messageLogTable'><thead><tr><th>Topic</th><th>Payload</th><th>Zuletzt empfangen</th></tr></thead><tbody></tbody></table>"**;**

html **+=** "<h2>Retained Messages</h2><table id='retainedMessagesTable'><thead><tr><th>Topic</th><th>Payload</th></tr></thead><tbody></tbody></table>"**;**

html **+=** "<h2>Connected MQTT Clients</h2><table id='connectedClientsTable'><thead><tr><th>Client ID</th><th>Last Seen (ms)</th><th>Subscribed Topics</th></tr></thead><tbody></tbody></table>"**;**

html **+=** "<p>Verbunden mit: <span id='wifiSsid'></span><br>IP-Adresse: <span id='wifiIp'></span></p>"**;**

html **+=** "<script>"**;**

html **+=** "function fetchData() {"**;**

html **+=** " var xhr = new XMLHttpRequest();"**;**

html **+=** " xhr.onreadystatechange = function() {"**;**

html **+=** " if (xhr.readyState == 4 && xhr.status == 200) {"**;**

html **+=** " var data = JSON.parse(xhr.responseText);"**;**

html **+=** " var messageLogTableBody = document.getElementById('messageLogTable').getElementsByTagName('tbody')[0];"**;**

html **+=** " messageLogTableBody.innerHTML = '';"**;**

html **+=** " data.messageLog.forEach(function(item) {"**;**

html **+=** " var row = messageLogTableBody.insertRow();"**;**

html **+=** " row.insertCell(0).innerText = item.topic;"**;**

html **+=** " row.insertCell(1).innerText = item.payload;"**;**

html **+=** " row.insertCell(2).innerText = new Date(item.timestamp).toLocaleTimeString();"**;**

html **+=** " });"**;**

html **+=** " var retainedMessagesTableBody = document.getElementById('retainedMessagesTable').getElementsByTagName('tbody')[0];"**;**

html **+=** " retainedMessagesTableBody.innerHTML = '';"**;**

html **+=** " for (var topic in data.retainedMessages) {"**;**

html **+=** " if (data.retainedMessages.hasOwnProperty(topic)) {"**;**

html **+=** " var row = retainedMessagesTableBody.insertRow();"**;**

html **+=** " row.insertCell(0).innerText = topic;"**;**

html **+=** " row.insertCell(1).innerText = data.retainedMessages[topic];"**;**

html **+=** " }"**;**

html **+=** " }"**;**

html **+=** " var connectedClientsTableBody = document.getElementById('connectedClientsTable').getElementsByTagName('tbody')[0];"**;**

html **+=** " connectedClientsTableBody.innerHTML = '';"**;**

html **+=** " data.connectedClients.forEach(function(item) {"**;**

html **+=** " var row = connectedClientsTableBody.insertRow();"**;**

html **+=** " row.insertCell(0).innerText = item.id;"**;**

html **+=** " row.insertCell(1).innerText = item.lastSeen + ' ms ago';"**;**

html **+=** " var topicsCell = row.insertCell(2);"**;**

html **+=** " item.subscribedTopics.forEach(function(topic) {"**;**

html **+=** " topicsCell.innerHTML += topic + '<br>';"**;**

html **+=** " });"**;**

html **+=** " });"**;**

html **+=** " document.getElementById('wifiSsid').innerText = data.wifi\_ssid;"**;**

html **+=** " document.getElementById('wifiIp').innerText = data.wifi\_ip;"**;**

html **+=** " }"**;**

html **+=** " };"**;**

html **+=** " xhr.open('GET', '/mqtt\_data', true);"**;**

html **+=** " xhr.send();"**;**

html **+=** "}"**;**

html **+=** "setInterval(fetchData, 2000);"**;**

html **+=** "window.onload = fetchData;"**;**

html **+=** "</script>"**;**

html **+=** "</body></html>"**;**

web**.**send**(**200**,** "text/html"**,** html**);**

**}**

void setupWebInterface**()** **{**

web**.**on**(**"/"**,** handleRoot**);**

web**.**on**(**"/mqtt\_data"**,** handleMqttData**);**

web**.**begin**();**

Serial**.**println**(**"HTTP server gestartet"**);**

**}**

void setup**()** **{**

Serial**.**begin**(**115200**);**

WiFi**.**mode**(**WIFI\_STA**);**

WiFi**.**begin**(**ssid**,** password**);**

Serial**.**print**(**"Verbinde mit WLAN"**);**

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

delay**(**500**);**

Serial**.**print**(**"."**);**

**}**

Serial**.**println**();**

Serial**.**println**(**"Verbunden mit WLAN!"**);**

Serial**.**print**(**"IP-Adresse: "**);**

Serial**.**println**(**WiFi**.**localIP**());**

Serial**.**println**(**"Scanne nach WLANs in der Nähe..."**);**

int n **=** WiFi**.**scanNetworks**();**

**if** **(**n **==** 0**)** **{**

Serial**.**println**(**"Keine Netzwerke gefunden."**);**

**}** **else** **{**

**for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **{**

Serial**.**print**(**i **+** 1**);**

Serial**.**print**(**": "**);**

Serial**.**print**(**WiFi**.**SSID**(**i**));**

Serial**.**print**(**" (RSSI: "**);**

Serial**.**print**(**WiFi**.**RSSI**(**i**));**

Serial**.**println**(**" dBm)"**);**

**}**

**}**

mqttServer**.**begin**();**

Serial**.**printf**(**"MQTT Broker läuft auf Port %d\n"**,** mqttPort**);**

setupWebInterface**();**

**}**

// Funktion zum Senden von MQTT Publish Paketen an Client

void sendPublish**(**WiFiClient **&**client**,** const String **&**topic**,** const String **&**payload**,** uint8\_t qos **=** 0**,** bool retain **=** **false,** uint16\_t packetId **=** 0**)** **{**

**if** **(!**client**.**connected**())** **{**

Serial**.**println**(**"Error: Tried to send PUBLISH to disconnected client."**);**

**return;**

**}**

int len **=** topic**.**length**();**

int payloadLen **=** payload**.**length**();**

uint8\_t fixedHeader **=** 0x30**;**

fixedHeader **|=** **(**qos **<<** 1**);**

**if** **(**retain**)** fixedHeader **|=** 0x01**;**

int remLen **=** 2 **+** len **+** payloadLen**;**

**if** **(**qos **>** 0**)** **{**

remLen **+=** 2**;**

**}**

uint8\_t encodedRemLen**[**4**];**

int i **=** 0**;**

**do** **{**

uint8\_t digit **=** remLen **%** 128**;**

remLen **/=** 128**;**

**if** **(**remLen **>** 0**)** **{**

digit **|=** 0x80**;**

**}**

encodedRemLen**[**i**++]** **=** digit**;**

**}** **while** **(**remLen **>** 0**);**

client**.**write**(**fixedHeader**);**

**for** **(**int j **=** 0**;** j **<** i**;** **++**j**)** **{**

client**.**write**(**encodedRemLen**[**j**]);**

**}**

client**.**write**((**len **>>** 8**)** **&** 0xFF**);**

client**.**write**(**len **&** 0xFF**);**

client**.**print**(**topic**);**

**if** **(**qos **>** 0**)** **{**

client**.**write**((**packetId **>>** 8**)** **&** 0xFF**);**

client**.**write**(**packetId **&** 0xFF**);**

**}**

client**.**print**(**payload**);**

**}**

// Funktion zum Verarbeiten einer eingehenden PUBLISH-Nachricht

void handlePublish**(**MQTTClientState**&** clientState**,** byte header**,** byte remainingLength**)** **{**

WiFiClient**&** client **=** clientState**.**client**;**

uint8\_t qos **=** **(**header **>>** 1**)** **&** 0x03**;**

bool retain **=** **(**header **&** 0x01**);**

**if** **(!**waitForBytes**(**client**,** remainingLength**))** **{**

Serial**.**println**(**"Timeout bei PUBLISH Payload"**);**

**return;**

**}**

byte topicLenMsb **=** client**.**read**();**

byte topicLenLsb **=** client**.**read**();**

int topicLen **=** **(**topicLenMsb **<<** 8**)** **|** topicLenLsb**;**

char topicChar**[**topicLen **+** 1**];**

**for** **(**int i **=** 0**;** i **<** topicLen**;** i**++)** **{**

topicChar**[**i**]** **=** client**.**read**();**

**}**

topicChar**[**topicLen**]** **=** 0**;**

String topic **=** String**(**topicChar**);**

uint16\_t packetId **=** 0**;**

**if** **(**qos **>** 0**)** **{**

byte packetIdMsb **=** client**.**read**();**

byte packetIdLsb **=** client**.**read**();**

packetId **=** **(**packetIdMsb **<<** 8**)** **|** packetIdLsb**;**

**}**

int payloadLen **=** remainingLength **-** **(**2 **+** topicLen**);**

**if** **(**qos **>** 0**)** payloadLen **-=** 2**;**

String payload **=** ""**;**

**if** **(**payloadLen **>** 0**)** **{**

char payloadChar**[**payloadLen **+** 1**];**

**for** **(**int i **=** 0**;** i **<** payloadLen**;** i**++)** **{**

payloadChar**[**i**]** **=** client**.**read**();**

**}**

payloadChar**[**payloadLen**]** **=** 0**;**

payload **=** String**(**payloadChar**);**

**}**

Serial**.**printf**(**"PUBLISH empfangen: Topic='%s', Payload='%s', QoS=%d, Retain=%d\n"**,** topic**.**c\_str**(),** payload**.**c\_str**(),** qos**,** retain**);**

Message msg **=** **{**topic**,** payload**,** millis**()};**

messageLog**.**push\_back**(**msg**);**

**if** **(**messageLog**.**size**()** **>** 50**)** **{**

messageLog**.**erase**(**messageLog**.**begin**());**

**}**

**if** **(**retain**)** **{**

**if** **(**payload**.**length**()** **==** 0**)** **{**

retainedMessages**.**erase**(**topic**);**

Serial**.**printf**(**"Retained message for topic '%s' cleared.\n"**,** topic**.**c\_str**());**

**}** **else** **{**

retainedMessages**[**topic**]** **=** payload**;**

Serial**.**printf**(**"Retained message for topic '%s' updated.\n"**,** topic**.**c\_str**());**

**}**

**}**

**for** **(**auto **&**sub **:** activeSubscriptions**)** **{**

**if** **(**sub**.**clientState**->**client**.**connected**()** **&&** topicMatches**(**topic**,** sub**.**topicFilter**))** **{**

Serial**.**printf**(**"Weiterleiten an Subscriber '%s' für Topic: %s\n"**,** sub**.**clientState**->**clientId**.**c\_str**(),** sub**.**topicFilter**.**c\_str**());**

sendPublish**(**sub**.**clientState**->**client**,** topic**,** payload**,** 0**,** retain**);**

**}**

**}**

**if** **(**qos **==** 1**)** **{**

byte puback**[]** **=** **{**0x40**,** 0x02**,** **(**byte**)(**packetId **>>** 8**),** **(**byte**)(**packetId **&** 0xFF**)};**

client**.**write**(**puback**,** **sizeof(**puback**));**

Serial**.**printf**(**"PUBACK gesendet für Packet ID %d\n"**,** packetId**);**

**}**

**}**

void handleSubscribe**(**MQTTClientState**&** clientState**,** byte header**,** byte remainingLength**)** **{**

WiFiClient**&** client **=** clientState**.**client**;**

**if** **(!**waitForBytes**(**client**,** remainingLength**))** **{**

Serial**.**println**(**"Timeout bei SUBSCRIBE Payload"**);**

**return;**

**}**

byte packetIdMsb **=** client**.**read**();**

byte packetIdLsb **=** client**.**read**();**

int packetId **=** **(**packetIdMsb **<<** 8**)** **|** packetIdLsb**;**

byte topicLenMsb **=** client**.**read**();**

byte topicLenLsb **=** client**.**read**();**

int topicLen **=** **(**topicLenMsb **<<** 8**)** **|** topicLenLsb**;**

char topicChar**[**topicLen **+** 1**];**

**for** **(**int i **=** 0**;** i **<** topicLen**;** i**++)** **{**

topicChar**[**i**]** **=** client**.**read**();**

**}**

topicChar**[**topicLen**]** **=** 0**;**

String topic **=** String**(**topicChar**);**

byte qos **=** client**.**read**();**

Serial**.**printf**(**"Client '%s' SUBSCRIBEd to Topic: %s with QoS %d\n"**,** clientState**.**clientId**.**c\_str**(),** topic**.**c\_str**(),** qos**);**

byte suback**[]** **=** **{**0x90**,** 0x03**,** packetIdMsb**,** packetIdLsb**,** qos**};**

client**.**write**(**suback**,** **sizeof(**suback**));**

Serial**.**println**(**"SUBACK gesendet"**);**

ActiveSubscription newSub**;**

newSub**.**clientState **=** **&**clientState**;**

newSub**.**topicFilter **=** topic**;**

activeSubscriptions**.**push\_back**(**newSub**);**

clientState**.**subscribedTopics**.**push\_back**(**topic**);**

**for** **(**auto const**&** **[**retainedTopic**,** retainedPayload**]** **:** retainedMessages**)** **{**

**if** **(**topicMatches**(**retainedTopic**,** topic**))** **{**

Serial**.**printf**(**"Sende Retained Message für Topic '%s' an neuen Subscriber '%s'.\n"**,** retainedTopic**.**c\_str**(),** clientState**.**clientId**.**c\_str**());**

sendPublish**(**client**,** retainedTopic**,** retainedPayload**,** 0**,** **true);**

**}**

**}**

**}**

void handlePingReq**(**MQTTClientState**&** clientState**)** **{**

WiFiClient**&** client **=** clientState**.**client**;**

byte pingresp**[]** **=** **{**0xD0**,** 0x00**};**

client**.**write**(**pingresp**,** **sizeof(**pingresp**));**

Serial**.**printf**(**"PINGRESP gesendet an Client '%s'\n"**,** clientState**.**clientId**.**c\_str**());**

clientState**.**lastSeen **=** millis**();**

**}**

void processMQTTClient**(**MQTTClientState**&** clientState**)** **{**

WiFiClient**&** client **=** clientState**.**client**;**

**if** **(!**client**.**connected**())** **{**

**return;**

**}**

clientState**.**lastSeen **=** millis**();**

**if** **(!**clientState**.**connectedACKSent**)** **{**

**if** **(!**waitForBytes**(**client**,** 2**))** **{**

**return;**

**}**

byte header **=** client**.**read**();**

byte remainingLength **=** client**.**read**();**

**if** **(**header **!=** 0x10**)** **{**

Serial**.**printf**(**"Client '%s' sent non-CONNECT first: 0x%X. Disconnecting.\n"**,** clientState**.**clientId**.**c\_str**(),** header**);**

client**.**stop**();**

**return;**

**}**

**if** **(!**waitForBytes**(**client**,** remainingLength**))** **{**

Serial**.**printf**(**"Timeout processing CONNECT payload for client '%s'. Disconnecting.\n"**,** clientState**.**clientId**.**c\_str**());**

client**.**stop**();**

**return;**

**}**

**for** **(**int i **=** 0**;** i **<** 8**;** **++**i**)** client**.**read**();**

byte clientIdLenHi **=** client**.**read**();**

byte clientIdLenLo **=** client**.**read**();**

int clientIdLen **=** **(**clientIdLenHi **<<** 8**)** **|** clientIdLenLo**;**

char clientId**[**clientIdLen **+** 1**];**

**for** **(**int i **=** 0**;** i **<** clientIdLen**;** i**++)** **{**

clientId**[**i**]** **=** client**.**read**();**

**}**

clientId**[**clientIdLen**]** **=** 0**;**

clientState**.**clientId **=** String**(**clientId**);**

// --- NEUE DEBUG-AUSGABE START ---

**if** **(**clientState**.**clientId**.**length**()** **==** 0**)** **{**

Serial**.**printf**(**"DEBUG: Client verbunden OHNE ID (Länge 0). Remote IP: %s\n"**,** client**.**remoteIP**().**toString**().**c\_str**());**

// Optional: Generieren Sie hier eine ID, wenn gewünscht

// clientState.clientId = "AutoGen\_" + String(client.remoteIP()[3]) + "\_" + String(millis() % 1000);

**}** **else** **{**

Serial**.**printf**(**"DEBUG: Client '%s' verbunden MIT ID. Remote IP: %s\n"**,** clientState**.**clientId**.**c\_str**(),** client**.**remoteIP**().**toString**().**c\_str**());**

**}**

// --- NEUE DEBUG-AUSGABE ENDE ---

Serial**.**printf**(**"Client '%s' CONNECTED. Sending CONNACK.\n"**,** clientState**.**clientId**.**c\_str**());**

byte connack**[]** **=** **{**0x20**,** 0x02**,** 0x00**,** 0x00**};**

client**.**write**(**connack**,** **sizeof(**connack**));**

clientState**.**connectedACKSent **=** **true;**

**}**

**if** **(**client**.**available**()** **>=** 2**)** **{**

byte h **=** client**.**read**();**

byte len **=** client**.**read**();**

int packetType **=** h **>>** 4**;**

**switch** **(**packetType**)** **{**

**case** 3**:** // PUBLISH

handlePublish**(**clientState**,** h**,** len**);**

**break;**

**case** 8**:** // SUBSCRIBE

handleSubscribe**(**clientState**,** h**,** len**);**

**break;**

**case** 12**:** // PINGREQ

handlePingReq**(**clientState**);**

**break;**

**case** 14**:** // DISCONNECT

Serial**.**printf**(**"Client '%s' DISCONNECT requested. Closing connection.\n"**,** clientState**.**clientId**.**c\_str**());**

client**.**stop**();**

**break;**

**default:**

Serial**.**printf**(**"Client '%s': Unknown packet type 0x%X, length %d. Disconnecting.\n"**,** clientState**.**clientId**.**c\_str**(),** packetType**,** len**);**

**for** **(**int i **=** 0**;** i **<** len **&&** client**.**available**();** **++**i**)** client**.**read**();**

client**.**stop**();**

**break;**

**}**

**}**

**}**

void loop**()** **{**

web**.**handleClient**();**

WiFiClient newClient **=** mqttServer**.**available**();**

**if** **(**newClient**)** **{**

Serial**.**println**(**"Neue TCP-Verbindung erkannt."**);**

MQTTClientState newState**;**

newState**.**client **=** newClient**;**

newState**.**lastSeen **=** millis**();**

connectedMQTTClients**.**push\_back**(**newState**);**

Serial**.**printf**(**"Anzahl verbundener MQTT-Clients: %d\n"**,** connectedMQTTClients**.**size**());**

**}**

**for** **(**int i **=** connectedMQTTClients**.**size**()** **-** 1**;** i **>=** 0**;** **--**i**)** **{**

**if** **(**connectedMQTTClients**[**i**].**client**.**connected**())** **{**

processMQTTClient**(**connectedMQTTClients**[**i**]);**

**}** **else** **{**

Serial**.**printf**(**"Client '%s' (%s) hat die Verbindung getrennt. Entferne ihn.\n"**,**

connectedMQTTClients**[**i**].**clientId**.**c\_str**(),**

connectedMQTTClients**[**i**].**client**.**remoteIP**().**toString**().**c\_str**());**

activeSubscriptions**.**erase**(**std**::**remove\_if**(**activeSubscriptions**.**begin**(),** activeSubscriptions**.**end**(),**

**[&](**const ActiveSubscription**&** sub**)** **{**

**return** sub**.**clientState **==** **&**connectedMQTTClients**[**i**];**

**}),**

activeSubscriptions**.**end**());**

connectedMQTTClients**.**erase**(**connectedMQTTClients**.**begin**()** **+** i**);**

**}**

**}**

**}**

Gerne, hier ist eine kurze Erklärung der Änderungen, die am ESP32 MQTT Broker Programm vorgenommen wurden, um QoS 1 & 2 sowie Authentifizierung und Autorisierung zu integrieren:

**Erweiterungen des ESP32 MQTT Brokers: QoS 1 & 2 und Authentifizierung/Autorisierung**

Das aktualisierte Programm erweitert die Funktionalität des ESP32 MQTT Brokers erheblich, um eine zuverlässigere Nachrichtenübermittlung und verbesserte Sicherheit zu gewährleisten.

**1. Volle QoS-Implementierung (Quality of Service 1 & 2):**

* **Zustandsverwaltung:** Neue Strukturen (OutgoingMessage und IncomingQoS2State) wurden zur MQTTClientState-Struktur hinzugefügt. Diese speichern den Status und die Paket-IDs von Nachrichten, die sich im QoS-Handshake befinden, sowohl für Nachrichten, die der Broker an Clients sendet, als auch für solche, die er von Clients empfängt.
* **Erweiterter sendPublish:** Diese Funktion generiert nun eindeutige Paket-IDs für QoS 1 und 2 Nachrichten und speichert sie in einer Warteschlange, um auf Bestätigungen von den Clients zu warten.
* **Mehrstufige Handshake-Handler:**
  + handlePublish wurde angepasst: Bei eingehenden QoS 1 Nachrichten sendet der Broker ein PUBACK. Bei QoS 2 Nachrichten sendet er ein PUBREC und speichert den Zustand, leitet die Nachricht aber noch nicht weiter.
  + Neue Funktionen (handlePubAck, handlePubRec, handlePubRel, handlePubComp) wurden hinzugefügt. Diese verarbeiten die jeweiligen Antwortpakete des Clients und steuern den mehrstufigen Handshake für QoS 1 (PUBLISH -> PUBACK) und QoS 2 (PUBLISH -> PUBREC -> PUBREL -> PUBCOMP).
* **Retransmission-Logik:** In der processMQTTClient-Schleife wurde eine Logik implementiert, die regelmäßig die Warteschlangen für ausstehende QoS-Nachrichten überprüft. Werden Bestätigungen nicht innerhalb eines definierten Timeouts empfangen, werden die entsprechenden Pakete erneut gesendet, um die Zustellung zu gewährleisten.
* **Subscriber-QoS:** Beim Senden von Nachrichten an Subscriber berücksichtigt der Broker nun den vom Subscriber im SUBSCRIBE-Paket gewünschten QoS-Level.

**2. Authentifizierung und Autorisierung:**

* **Benutzer- und ACL-Definitionen:** Es wurden neue Datenstrukturen (UserCredentials, TopicACL) und Maps (users, userACLs) eingeführt. Diese ermöglichen es, Benutzer mit Benutzernamen und Passwörtern zu definieren sowie Zugriffsrechte (Lesen, Schreiben, beides) für spezifische Topics festzulegen.
* **setupUsersAndACLs:** Eine neue Funktion initialisiert diese Benutzer und deren Zugriffsrechte beim Start des Brokers.
* **Authentifizierung im CONNECT-Paket:**
  + Der Broker liest nun den Benutzernamen und das Passwort aus dem eingehenden CONNECT-Paket des Clients.
  + Diese Anmeldedaten werden gegen die intern definierten Benutzer überprüft.
  + Bei erfolgreicher Authentifizierung wird der Benutzername im MQTTClientState des Clients gespeichert. Bei Fehlschlag sendet der Broker einen "Not Authorized"-CONNACK-Code und trennt die Verbindung.
* **Autorisierungsprüfung (checkPermission):** Eine neue Hilfsfunktion checkPermission wurde erstellt, die anhand des authentifizierten Benutzernamens und der definierten ACLs überprüft, ob ein Client die Berechtigung hat, auf ein bestimmtes Topic zuzugreifen (Lesen oder Schreiben).
* **Autorisierung bei PUBLISH:** Bevor eine eingehende PUBLISH-Nachricht verarbeitet und an Subscriber weitergeleitet wird, prüft der Broker, ob der authentifizierte Client die Schreibberechtigung für das betreffende Topic besitzt. Ist dies nicht der Fall, wird die Nachricht ignoriert (obwohl der QoS-Handshake für den sendenden Client abgeschlossen werden kann).
* **Autorisierung bei SUBSCRIBE:** Bevor ein Abonnement hinzugefügt wird, prüft der Broker, ob der authentifizierte Client die Leseberechtigung für das angefragte Topic besitzt. Bei fehlender Berechtigung wird das Abonnement abgelehnt und ein entsprechender SUBACK-Fehlercode an den Client gesendet.

Diese Ergänzungen machen den ESP32 MQTT Broker deutlich robuster und sicherer, indem sie die Einhaltung des MQTT-Protokolls verbessern und eine grundlegende Zugriffskontrolle ermöglichen.

Ja, das Webinterface bietet noch viel Potenzial für interessante Erweiterungen, die die Benutzerfreundlichkeit und die Interaktion mit dem Broker verbessern würden.

Hier sind einige Ideen, was man noch hinzufügen könnte:

* **Benutzer- und ACL-Verwaltung (Admin-Interface):**
  + Eine Seite, auf der Administratoren **Benutzer hinzufügen, bearbeiten oder löschen** können.
  + Möglichkeit, die **ACLs (Zugriffsrechte)** für jeden Benutzer direkt im Webinterface zu verwalten (welche Topics gelesen/geschrieben werden dürfen).
  + Dies würde die aktuelle statische Definition im Code dynamisch machen.
* **Nachrichten senden (Publisher-Funktion):**
  + Ein **Formular**, über das man direkt aus dem Webinterface MQTT-Nachrichten an ein bestimmtes Topic senden kann.
  + Eingabefelder für Topic, Payload, QoS-Level und Retain-Flag. Ideal zum Testen oder manuellen Steuern von Geräten.
* **Abonnieren und Live-Ansicht (Subscriber-Funktion):**
  + Möglichkeit, sich im Webinterface für bestimmte Topics zu **abonnieren** und die eingehenden Nachrichten in Echtzeit anzuzeigen (ähnlich einem MQTT-Client-Tool).
  + Dies könnte eine separate Live-Log-Ansicht sein, die nur die abonnierten Nachrichten anzeigt.
* **Detaillierte Client-Informationen:**
  + Beim Klick auf einen verbundenen Client in der Tabelle könnten **detailliertere Informationen** angezeigt werden:
    - Verbindungszeitpunkt
    - Keep-Alive-Intervall
    - Anzahl der gesendeten/empfangenen Nachrichten
    - Der authentifizierte Benutzername (bereits im Code vorhanden)
    - Status der ausstehenden QoS-Nachrichten (wenn man diese im Webinterface sichtbar machen möchte).
* **Broker-Status und Statistiken:**
  + **Echtzeit-Diagramme** für die Anzahl der verbundenen Clients, die Nachrichtenrate (Nachrichten pro Sekunde).
  + Anzeige der Speichernutzung (RAM) des ESP32.
  + Uptime des Brokers.
  + Fehler-Log (z.B. fehlgeschlagene Authentifizierungsversuche, ungültige Pakete).
* **Konfigurationsverwaltung:**
  + Möglichkeit, grundlegende Broker-Einstellungen (z.B. WLAN-Zugangsdaten, MQTT-Port) über das Webinterface zu ändern und zu speichern (würde Persistenz erfordern, z.B. LittleFS).
  + Neustart des Brokers über das Webinterface.
* **Themenübersicht:**
  + Eine Liste aller aktiven Topics (die jemals veröffentlicht wurden oder abonniert sind).
  + Möglichkeit, nach Topics zu filtern oder zu suchen.
* **Responsives Design und UI/UX-Verbesserungen:**
  + Anpassung des Layouts für mobile Geräte (Smartphones, Tablets).
  + Verbesserung der visuellen Darstellung (CSS-Frameworks, Icons).
  + Interaktive Elemente (z.B. Sortierfunktionen für Tabellen, Suchfelder).

Diese Erweiterungen würden das Webinterface von einem reinen "Monitor" zu einem mächtigen Werkzeug für die Verwaltung und Interaktion mit deinem Smart Home MQTT-System machen.