**1. Projektbeschreibung**

Das Ziel des Projekts ist die Entwicklung eines sicheren, skalierbaren und MQTT-kompatiblen Backend-Servers zur Steuerung und Überwachung eines Raumüberwachungssystems. Das System sammelt Sensordaten (z. B. Temperatur, Luftfeuchtigkeit), verarbeitet Vorfälle (Incidents) und bietet eine WebSocket-Schnittstelle für Echtzeitkommunikation.

**2. Technologischer Stack**

| **Komponente** | **Technologie / Paket** |
| --- | --- |
| Programmiersprache | Python 3.x |
| Web-Framework | Flask |
| Authentifizierung | JWT (JSON Web Tokens) |
| WebSocket | Flask-Sock, Flask-SocketIO |
| Datenbank | SQLite |
| Messaging | MQTT mit paho-mqtt |
| Rate-Limiting | Flask-Limiter |
| Sicherheit | Passwort-Hashing mit Werkzeug |
| CORS-Unterstützung | Flask-CORS |
| Datenformatierung | JSON |
| .env-Verwaltung | Python-dotenv |

**3. Hauptfunktionen des Backends**

**3.1 Authentifizierung und Sicherheit**

* JWT-basierte Authentifizierung (/login, @jwt\_required)
* Passwort-Hash-Prüfung (check\_password\_hash)
* IP-basiertes Rate-Limiting mit Sperrfunktion bei zu vielen Fehlversuchen

**3.2 WebSocket-Unterstützung**

* WebSocket-Route (/ws) zur bidirektionalen Kommunikation
* Unterstützung für Live-Datenübertragung über MQTT an Frontend-Clients

**3.3 MQTT-Anbindung**

* Subscribed Topics:
  + RZ/data: Sensordaten (JSON)
  + RZ/config: Konfigurationen
  + RZ/incidents: Vorfälle/Alarme
* Dynamische Nachrichtenübertragung mit zufälligen msgIDs
* Verarbeitung von:
  + Echtzeitdaten (z. B. Temperatur, Türstatus)
  + Konfigurationsbefehlen (setConfig, getConfig)
  + Incidents (Alarme)

**3.4 REST API-Endpunkte (Auswahl)**

| **Route** | **Methode** | **Auth** | **Beschreibung** |
| --- | --- | --- | --- |
| /login | POST | ❌ | Benutzeranmeldung |
| /getConfig | GET | ✅ | Abrufen der Gerätekonfiguration |
| /setConfig | POST | ✅ | Setzen neuer Konfigurationen |
| /addAccessID | POST | ✅ | Generiert eine neue Zugangskarte |
| /data | GET | ✅ | Abruf von Sensorwerten |
| /incidents | GET | ✅ | Abruf von Vorfällen (Alarme etc.) |
| /user/profile | GET | ✅ | Aktuelles Benutzerprofil |
| /health | GET | ❌ | System-Gesundheitsstatus (MQTT Ping) |

**4. Datenbankstruktur (SQLite)**

**users**

| **Spalte** | **Typ** |
| --- | --- |
| id | INTEGER |
| username | TEXT |
| password | TEXT |

**dashboard\_data**

| **Spalte** | **Typ** |
| --- | --- |
| timestamp | TEXT |
| value | TEXT (JSON) |

**incidents**

| **Spalte** | **Typ** |
| --- | --- |
| id | INTEGER |
| timestamp | TEXT |
| type | TEXT |
| value | TEXT (JSON) |

**5. Sicherheitskonzepte**

* **Tokenbasierte Authentifizierung:** JWT mit Ablaufzeit.
* **Rate-Limiting:** Pro IP-Adresse max. 10 Fehlversuche → Sperre für 5 Minuten.
* **CORS-Freischaltung:** Für alle Ursprünge erlaubt.
* **Passwortsicherheit:** Speicherung gehashter Passwörter mit werkzeug.security.

**6. MQTT-Datenfluss (vereinfacht)**

mermaid

KopierenBearbeiten

sequenceDiagram

Client ->> Backend (Flask): /setConfig (POST)

Backend ->> MQTT Broker: send\_message("RZ/config", payload)

Sensoren ->> MQTT Broker: send\_data("RZ/data", json)

MQTT Broker -->> Backend (Handler): on\_message -> Datenbank + WebSocket

**7. Projektstatus und Ausblick**

* ✅ Funktionale REST-API mit Authentifizierung
* ✅ WebSocket für Live-Datenübertragung
* ✅ MQTT vollständig angebunden
* 🟡 Mock-Datengenerator vorhanden (für Debugging)
* 🛠️ **TODO:**
  + Volle WebSocket-Unterstützung für Konfigurationen
  + Asynchrone Verarbeitung von MQTT-Antworten
  + Migration zu PostgreSQL für produktive Umgebung

**8. Beispielhafte API-Calls**

h

KopierenBearbeiten

POST /login

Content-Type: application/json

{

"username": "admin",

"password": "secure123"

}

http

KopierenBearbeiten

GET /data?start\_date=2025-05-01&limit=10

Authorization: Bearer <JWT\_TOKEN>

**9. Umgebungsvariablen (.env Beispiel)**

env

KopierenBearbeiten

JWT\_SECRET\_KEY=supersecretkey

MQTT\_BROKER=10.93.140.165

MQTT\_PORT=1884

MQTT\_USER=grp5

MQTT\_PASS=grp5123!

DB\_PATH=local.db

**10. Fazit**

Dieses Backend-System bietet eine robuste Basis für ein intelligentes Raumüberwachungssystem. Es ist modular aufgebaut, MQTT-fähig, datensicher und auf Erweiterbarkeit ausgelegt – ideal für zukünftige Erweiterungen mit Frontend-Dashboards, mobilen Clients oder erweiterten Alarmsystemen.