

Seminararbeit-microservices-ws2022
von 4934001

Theorieteil + Erklärungen zu den Praxisteilen
Ausführliche Dokumentation und
Installation/Integration-Anweisungen

Arbeitsaufwand ca. 30 Stunden

Sehr ausführliche Dokumentation die ein sehr gute Note anstrebt.

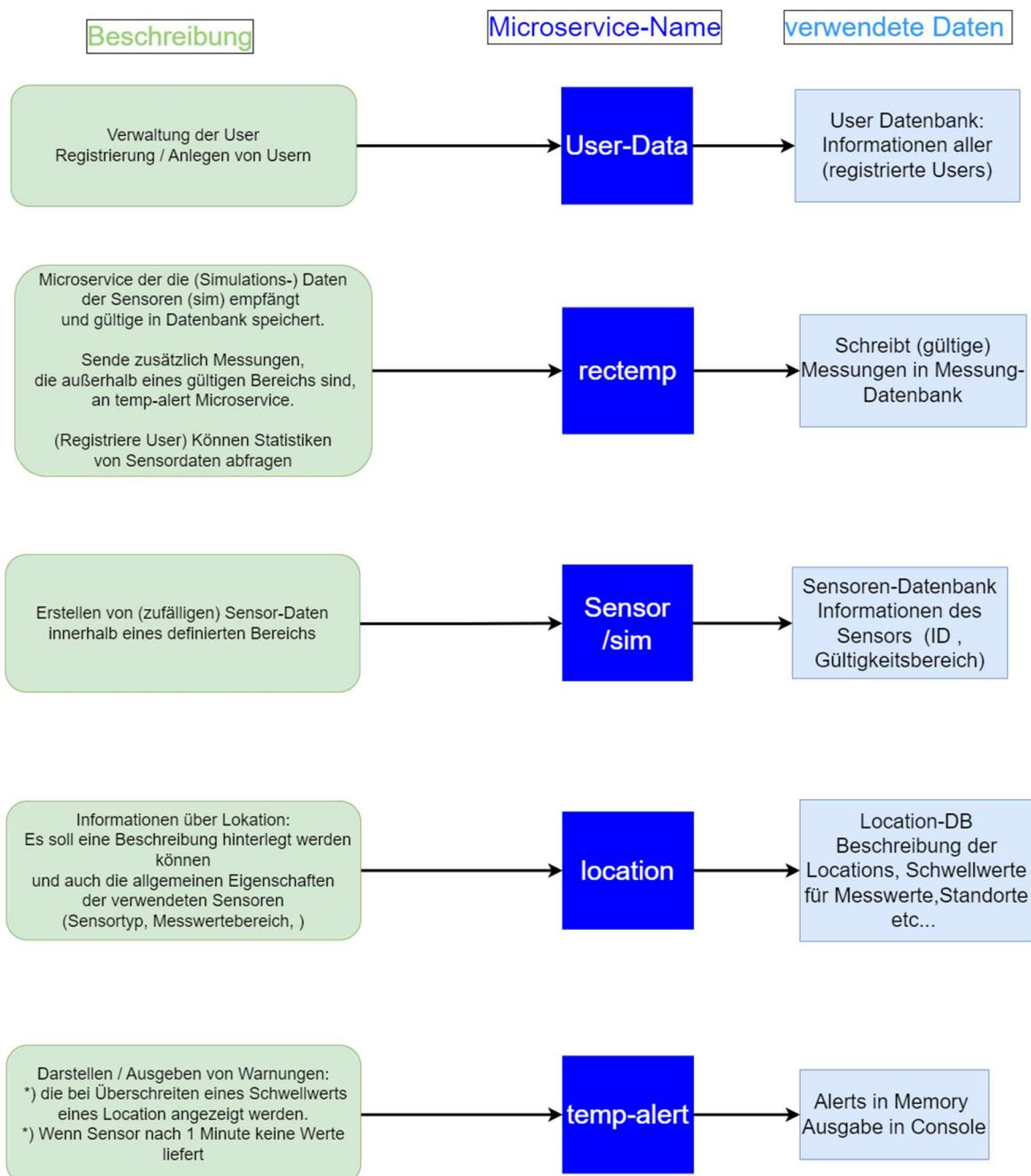
Inhalt

Theoretischer Teil: Aufgabe 1 Domain Driven Design.....	3
Theoretischer Teil: Aufgabe 2 Datenmodell	4
Theoretischer Teil: Aufgabe 3 Eventliste.....	5
Theoretischer Teil: Aufgabe 4 Deployment Modell der Microservices.....	6
Praktischer Teil: Beschreibung des MVPs	7
Sim:.....	8
MongoDB.....	8
MQTT	9
Rectemp	10
Config.js	11
Wertebereich / Gültigkeitsbereich der Sensoren:	11
MQTT-CLIENTID:.....	12
Schreiben in MongoDB:.....	12
Sende Warnung bei Überschreiten von Schwellwert:	13
Zusätzliche Features:.....	14
Healthz-Schnittstelle:	16
Temp-alert.....	16
Schaubild MQTT Messages:.....	17
Praktischer Teil: Kubernetes.....	17
Netzwerk	17
Konfigurierbare Verbindungsparameter für den Zugriff auf MongoDB:	17
Resilienz gegen Ausfall	18
Quota Limit für rectemp.....	18
Integrationstest / (Installations-Anleitung).....	19
Lokaler Bau der Container mit docker-compose.....	19
Deployment in Kubernetes.....	22

Theoretischer Teil: Aufgabe 1

Domain Driven Design

Domain Driven Design



Theoretischer Teil: Aufgabe 2

Datenmodel

rectemp	
Name	Type
Sensorid	Integer (zeigt auf Sensor/SIM)
Messung	double (zeigt auf Sensor/SIM)
Datum	DateTime (zeigt auf Sensor/SIM)
Messungstyp	Char (zeigt auf Sensor/SIM)
Sensorid	Integer (zeigt auf Sensor/SIM)
Locationid	String (zeigt auf location)
gpslatitude	String (zeigt auf location)
gpslongitude	String (zeigt auf location)
Beschreibung	String (zeigt auf location)
Schwellwert_unten	Array/Map (zeigt auf location)
Schwellwert_oben	Array/Map (zeigt auf location)

Sensor/SIM	
Name	Type
Sensorid	Integer
Messung	double
Datum	DateTime
Messungstyp	Char
Gueltig_ab	Integer
Gueltig_bis	Integer
Locationid	Integer (zeigt auf location)

location	
Name	Type
Locationid	Integer
gpslatitude	String
gpslongitude	String
Beschreibung	String
Schwellwert_unten*	Array/Map definiert unteren Schwellwert pro Messungstyp
Schwellwert_oben*	Array/Map definiert oberen Schwellwert pro Messungstyp

temp-alert	
Name	Type
Sensorid	Integer (zeigt auf Sensor/SIM)
Messung	double (zeigt auf Sensor/SIM)
Datum	DateTime (zeigt auf Sensor/SIM)
Messungstyp	Char (zeigt auf Sensor/SIM)
Locationid	String (zeigt auf rectemp)
gpslatitude	String (zeigt auf rectemp)
gpslongitude	String (zeigt auf rectemp)
Beschreibung	String (zeigt auf rectemp)

User	
Name	Type
Username	String
Email	String (E-Mail)
Password	String (Hashed)

Anmerkung*:
 Variablen bzw. Werte für **Schwellwert_unten** und **Schwellwert_oben** werden pro **Location** gesetzt, da angenommen wird, dass für jeden Standort unterschiedliche Schwellwerte (pro Sensortyp) definiert werden können sollen. Bspw. Soll in einem Raum mehr Personen erlaubt sein als in einem anderem.

Theoretischer Teil: Aufgabe 3

Eventliste

Entity type	Event type	Event Topic	Event Data	Comments
User	User_created	User/created	Username,Email, Passwort,	
User	User_updated	User/updated	Username,Email, Passwort,	
User	User_deleted	User/deleted	Username	
location	Location_created	Location/created	Locationid,gpslatitude,gpslongitude,Beschreibung,Schwellwert_unten,Schwellwert_oben	Eine neue Location wurde erstellt
location	Location_updated	Location/updated	Locationid,gpslatitude,gpslongitude,Beschreibung,Schwellwert_unten,Schwellwert_oben	Eine Location wurde geändert
location	Location_deleted	Location-deleted	Locationid	Eine Location wurde gelöscht
Sensor/SIM	sende_gültige_messung	rec/messungen	JSON:{messungsdaten}	Gültige Messung wird gesendet
Sensor/SIM	sende_ungültige_messung	Alert/ungültig	JSON:{messungsdaten}	Ungültige Messung außerhalb des Gültigkeitsbereichs des Sensors wird an Alert- ungültig gesendet
rectemp	sende_messung_warnung	Alert/Schwellwert		Messung außerhalb des Schwellwerts wird als Warnung gesendet

Anmerkung:

temp-alert abonniert die Topics: Alert/ungültig und Alert/Schwellwert

dadurch kann unterschieden werden, ob eine Messung bereits bei der Erstellung am Sensor ungültig ist, oder die (gültige Messung) im nicht akzeptieren Bereich liegt (durch rectemp gesendet)

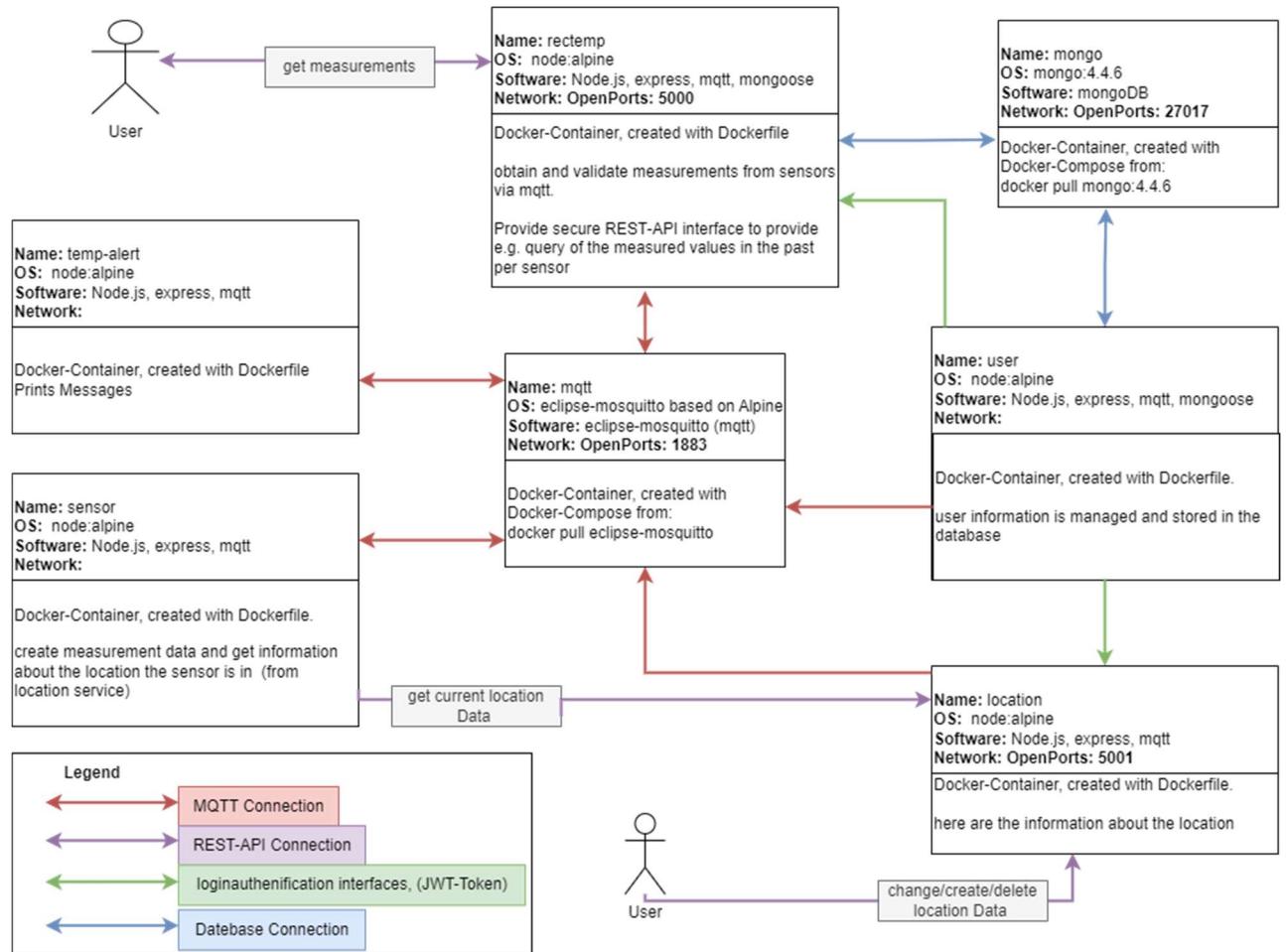
Sensor/SIM abonniert die Topics: Location/created und Location/updated und Location/delete, sodass beim Senden der Messungen die aktuellen Daten/Informationen der Locations verwendet werden (sind auch lokal zwischengespeichert)

rectemp abonniert das Topic: : rec/messungen. Hier kommen die gültigen Messungen des Sensors/SIM an

Theoretischer Teil: Aufgabe 4

Deployment Modell der Microservices

Deployment Model für theoretischen Teil wie er so in Docker/Kubernetes Deployt werden kann:

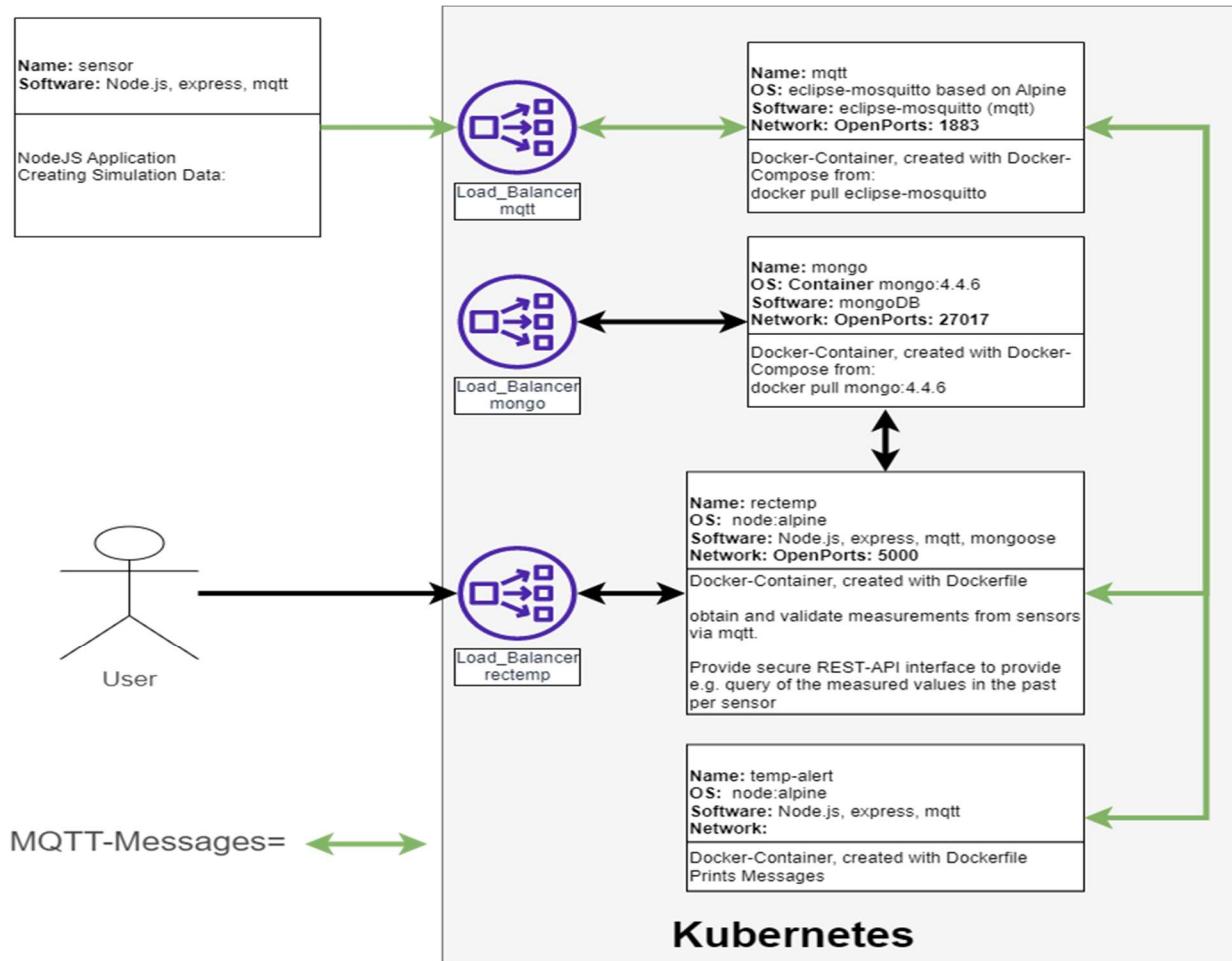


Praktischer Teil: Beschreibung des MVPs

Der MVP bildet eine abgespeckte Funktionalität unseres Messdatenerfassungssystems ab.

Anders als beispielsweise im Theorieteil gibt es in diesem MVP erstmal kein eigenen Microservice zur Userverwaltung. Auch ist die Verwaltung der Locations durch einen eigenen Microservice so auch noch nicht abgebildet:

Das Deployment-Model des MVPs kann mit folgendem Schaubild gezeigt werden:



Der MVP-Besteht aus folgenden Komponenten:

Sim:

Das Sim-Programm ist ein NodeJS Programm, das Messungen eines Sensors von einem angegebenen Standort simuliert.

Dabei müssen beim Start des Programms bestimmte Parameter definiert werden:

- arg1 = timeinterval in dem die Daten gesendet werden in sec
- arg1 = unique id of the sensor station - 4 stellig
- arg2 = Anzahl der Daten in Zyklen - 2stellig
- arg3 = Sensortyp - T=temperature, X=co2, P=people in a room, H=luftfeuch
- arg4 = start value
- arg5 = end value

npm start 10 1234 5 T 20 25

→ alle 10 Sekunden, werden vom Standort 1234, 5-mal, Temperaturdaten, zwischen 20 und 25 generiert

Der generierte Messwert wird im Bereich der letzten zwei Argumente zufällig pro Iteration generiert

Diese Messungen werden über MQTT jeweils an das Topic .

mqtopic="4934001/messungen" gesendet.

Dabei ist wichtig die korrekte URL des MQTT-Host bestimmen. Diese ist in Zeile 12 des NodeJS Scripts zusetzen:
(Bei Verwendung von Docker-Desktop mit aktiviertem Kubernetes ist diese angezeigte URL so korrekt und muss nicht geändert werden

```

9 //#####
10 //Setze mqtturl (auf LB von Kubernetes)
11 //var mqttHOSTurl = "192.168.49.2:31501"
12 var mqttHOSTurl = "localhost:1883"
```

Aufgabe-5/sim/index-skeleton.js

```

3
4 if ((args.length) == 6) {
5   timeinterval = args[0];
6   locid = args[1];
7   simanzahl = args[2];
8   sensortype = args[3];
9   min = parseInt(args[4]);
10  max = parseInt(args[5]);
11  MessdatensensorID = args[6];
12  mqtopic="4934001/messungen"
13 } else {
```

Aufgabe-5/sim/index-skeleton.js

MongoDB

Der Zugriff der MongoDB ist mit einem Usernamen und Passwort zu schützen. Dies ist durch das Setzen von Environment Variablen in der docker-compose-yaml Aufgabe 7 einzustellen.

```

33   - "27017:27017"
34   environment:
35     - MONGO_INITDB_ROOT_USERNAME=mongoadmin
36     - MONGO_INITDB_ROOT_PASSWORD=secret
37     - MONGO_INITDB_DATABASE=Temperatur
38   #command: [--auth]
39   volumes:
40     -
```

Aufgabe-6-7\docker-compose.yaml"

Dasselbe gilt für die Konfiguration von MongoDB innerhalb von Kubernetes. (Mehr dazu im Abschnitt: *Konfigurierbare Verbindungsparameter für den Zugriff auf MongoDB*: auf Seite 17)

Damit die Daten (innerhalb von Kubernetes) persistiert werden (also bei Neustart des Containers nicht gelöscht werden), ist die Datenbank in einem Volumen auszulagern, das über ein PersistentVolumeClaim durch Kubernetes bereitgestellt wird.

```

57   volumeMounts:
58     - name: "mongo-data-dir"
59       mountPath: "/data/db"
60   volumes:
61     - name: "mongo-data-dir"
62       persistentVolumeClaim:
63         claimName: "mongo-data"
```

Aufgabe-8\mongodb\mongodb-deployment.yaml

Diese Anforderungen, dass die Daten beim Neustart der Container von MongoDB nicht gelöscht werden, wurde entsprechend auch in Aufgabe 7 konfiguriert. Hier wird bei der Erstellung mittels der docker-compose.yaml Datei, dem Service mongo ein separates Volume erstellt in diesem die MongoDB Datenbank gespeichert ist und bei Neustart des Containers beibehalten wird.

Verbindungs-URL für MongoDB:

mongodb://mongoadmin:secret@localhost:27017/Temperatur?authSource=admin

MQTT

Als Message Broker soll innerhalb von Kubernetes Mosquitto-mqtt laufen.

Für die passende Konfiguration von Mosquitto mqtt wird im Kubernetes eine Configmap verwendet:

In dieser ist unter anderem definiert, dass mqtt auf Port 1883 lauschen soll und dass anonyme Anfragen erlaubt sind.

```
1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: mqtt-configmap
5   namespace: default
6 data:
7   mosquitto.conf: |
8     allow_anonymous true
9     listener 1883
10    persistence true
11    persistence_location /mosquitto/data/
12    log_dest file /mosquitto/log/mosquitto.log
```

Aufgabe-8\mqtt\mqtt-configmap.yaml

Auch die Persistenz der Daten wird in dieser Configmap eingestellt. Dazu ist, wie bei MongoDB auch, wieder ein persistentes Volumen innerhalb von Kubernetes notwendig. Dieses wird auch wieder mit einem PersistentVolumeClaim dem Pod zugewiesen, sodass die Daten/vorgehaltenen MQTT-Nachrichten bei einem Neustart des Pods nicht verloren gehen.

Auch für Aufgabe 7 die Bereitstellung über Docker-Compose ist die Configdatei entsprechend eingebunden worden. Dabei muss diese Konfigurationsdatei physisch auf dem Hostsystem liegen. Diese wird über die Docker Volume Mapping Funktion entsprechend dem Container bereitgestellt

```
ABGABE > Aufgabe-6-7 > mqtt > mosquitto > config > mosquito.conf
1  allow_anonymous true
2  listener 1883
3  persistence true
4  persistence_location /mosquitto/data/
5  log_dest file /mosquitto/log/mosquitto.log
```

Aufgabe-6-7\mqtt\mosquitto\config\mosquitto.conf

```
39   volumes:
40     - mongodb_volume:/data/db
41   volumes:
42     mongodb_volume:
```

Aufgabe-6-7\docker-compose.yaml

```
15   spec:
16     containers:
17       - name: mqtt
18         image: eclipse-mosquitto
19         ports:
20           - containerPort: 1883
21         volumeMounts:
22           - name: config-file
23             mountPath: /mosquitto/config/mosquitto.conf
24             subPath: mosquitto.conf
25           - name: mqtt-data
26             mountPath: /mosquitto/
27         volumes:
28           - name: config-file
29           configMap:
30             name: mqtt-configmap
31           - name: mqtt-data
32             persistentVolumeClaim:
33               claimName: "mqtt-data"
```

Aufgabe-8\mqtt\mqtt-deployment.yaml



```
18   mqtt:
19     image: eclipse-mosquitto
20     container_name: mqtt
21     restart: always
22     volumes:
23       - ./mqtt/mosquitto/config:/mosquitto/config
24       - ./mqtt/mosquitto/data:/mosquitto/data
25       - ./mqtt/mosquitto/log:/mosquitto/log
26     ports:
27       - "1883:1883"
```

Aufgabe-6-7\docker-compose.yaml

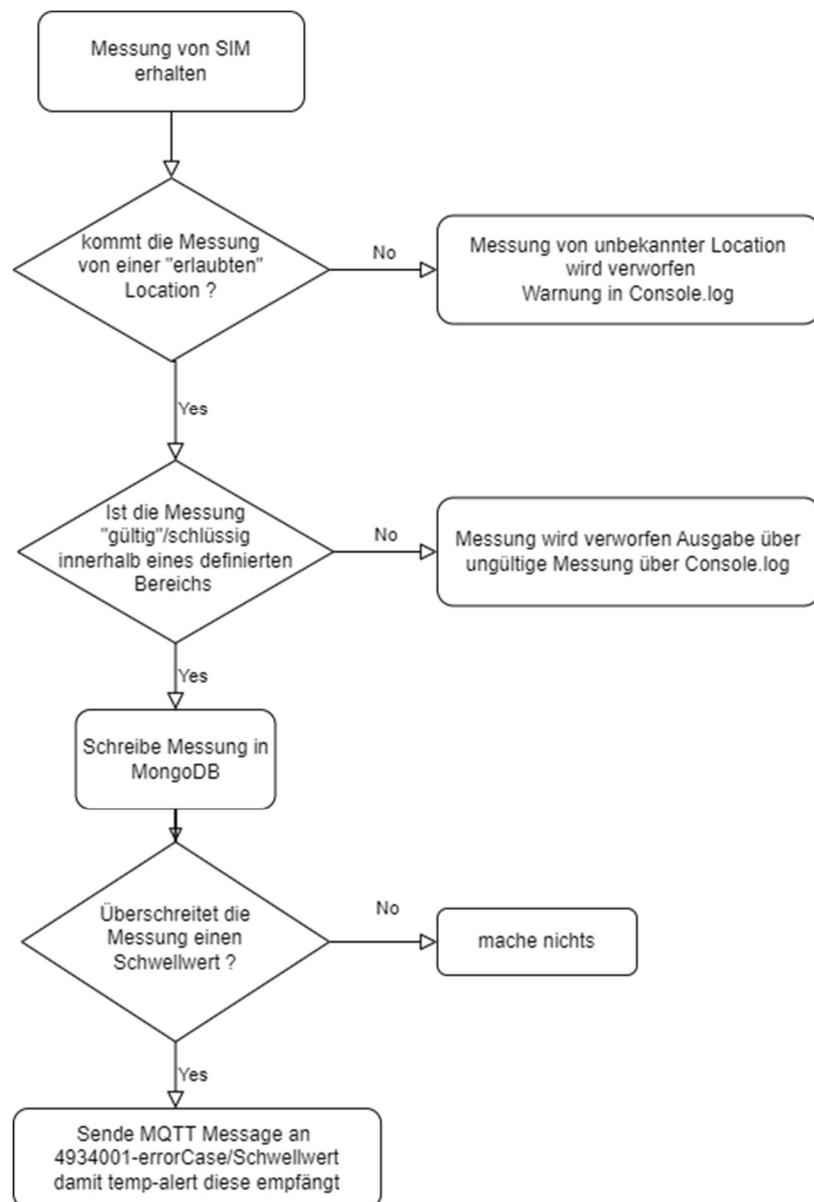
Rectemp

Ein NodeJS Programm, das die MQTT-Messages des SIM-Programms entgegennimmt und diese Überprüft:

- Ob eine Messung von einer erlaubten Location kommt
- Ob die Werte schlüssig sind (liegen die Messungen innerhalb eines Gültigkeitsbereichs eines Sensors)
- und ob, die Messungen einen bestimmten Schwellwert überschreiten.

Zusätzlich werden Messungen in eine MongoDB geschrieben und beim Überschreiten eines Schwellwerts einer Messung wird zusätzlich eine mqtt Nachricht an den temp-alert Service geschickt:

Der genaue Ablauf ist mit folgendem **Ablaufdiagramm** beschrieben:



Ablaufdiagramm erhält einer MQTT-Message in rectemp

Config.js

Bei dem rectemp Node-JS Programm gibt es eine extra Java-Script Datei in der Konfigurationen, unter anderem für die Verbindungs-Parameter der MongoDB, den MQTT-Service gespeichert sind (Config.js) . Auch sind hier global die erlaubten Locations in einem Array gespeichert und die Schwellwerte -innerhalb diesen eine Messung als normal gilt - pro Sensortyp definiert:

**Wichtig zum Testen: Es sind nur Messungen aus den Locations (1234,3456,4567 und 5678) erlaubt
(Zeile 6, Config.JS)**

```

2  ######
3  //      Config
4  // Nur definierte Sensoren werden akzeptiert
5  // hier werden die locid der Sensoren definiert, von denen Nachrichten empfangen und gespeichert werden
6  global.loc_configs = [1234, 3456, 4567, 5678]
7
8  //Schwellwerte:
9  //((Temperatur: Innerräume dürfen nur zwischen 16 und 19 Grad beheizt werden)
10 //var global.T_unten = 16;
11 global.T_unten = 16;
12 //module.export = {global.T_unten}
13 global.T_oben = 19;
14 //CO2: Innerräume dürfen nicht mehr als 2000 ppm Co2 beinhalten
15 global.X_unten = 2000;
16
17 //Personen: in Innerräume dürfen nicht mehr 10 Personen gleichzeitig sein
18 // Zusätzlich soll, eine Warnung ausgegeben werden wenn der raum leer ist
19 global.P_unten = 1;
20 global.P_oben = 10;
21
22
23 //Luftfeuchtigkeit: in Innerräume dürfen nicht mehr als 60% relativer Luftfeuchtigkeit haben
24 // Sollten aber mindestens 30 % haben
25 global.H_unten = 30;
26 global.H_oben = 60;
27
28 //Feinstaub: Im Raum darf nicht mehr als 5 µg/m³ Feinstaub vorhanden sein
29 //Die Weltgesundheitsorganisation WHO hat einen Richtwert für PM2,5 von 5 µg/m³
30 global.p_unten = 5;
31
32
33 // Config Parameter für MongoDB und MQTT und Zugangsdaten für Admins für die API Calls
34 global.env = {
35   MQTT_HOST: "mqtt",
36   MQTT_PORT: "1883",
37   DB_HOST: "mongo",
38   DB_PORT: "27017",
39   DB_DATABASE: "Temperatur",
40   DB_IS_SECURED: "true",
41   DB_USERNAME: "mongoadmin",
42   DB_PASSWORD: "secret",
43   USERID: "admin",
44   PASSWORD: "admin",
45 }

```

Aufgabe-6-7\rectemp\configs.js"

Wertebereich / Gültigkeitsbereich der Sensoren:

Die Wertebereiche, innerhalb diesem eine erhaltene Messung als gültig/schlüssig gilt, wurde aufgrund der Einfachheit des Programms hard-coded:

```

89  function validation_test(topic, message) {
90    // Überprüft ob Sensoren Normale/gültige Werte liefern.
91    // Nur Werte im Gültigkeitsbereich werden akzeptiert und weiter verarbeitet
92
93    var messung = JSON.parse(message);
94    sensorTyp = messung.sensorType
95    value = messung.value;
96    gueltig = 0;
97    switch (sensorTyp) {
98      case 'T':
99        //Temperatur Werte nur gültig wenn sie zwiswchen -10 und 50 Grad sind
100       if (value >= -10 && value < 50)
101         gueltig = 1;
102       break;
103     case 'X':
104       //CO2 Werte nur gültig wenn sie zwiswchen 1000 und 2500 ppm sind
105       if (value >= 1000 && value < 2500)
106         gueltig = 1;
107       break;
108     case 'P':
109       //Personenzahl nur gültig wenn zwischen 0 und 25
110       if (value >= 0 && value < 25)
111         gueltig = 1;
112       break;
113     case 'H':
114       //Luftfeuchtigkeit nur gültig wenn zwischen 0 und 100
115       if (value >= 0 && value < 100)
116         gueltig = 1;
117       break;
118     case 'p':
119       //Feinstaub nur gültig wenn zwischen 0 und 25
120       if (value >= 0 && value < 25)
121         gueltig = 1;
122       break;
123     case 'L':
124       //Lichtintensität nur gültig wenn zwischen 0 und 100
125       if (value >= 0 && value < 100)
126         gueltig = 1;
127       break;
128   }
129
130   return gueltig;
131 }

```

Aufgabe-6-7\rectemp\mqtt.js

MQTT-CLIENTID:

Beim Start der Node-JS Applikation wird zunächst die Verbindung zum MQTT-Broker initiiert und das passende Topic wird abonniert:

```
unique_id = crypto.randomBytes(16).toString('hex')

var Topic = '4934001/#'; //subscribe to all topics from postapp
var client = mqtt.connect("mqtt://" + global.env.MQTT_HOST + ":" + global.env.MQTT_PORT, { clientId: "rectemp-" + unique_id });
```

Aufgabe-6-7\rectemp\mqtt.js

Dabei ist sicherzustellen, dass bei jedem Start der Anwendung eine (neue) einzigartige ID als Client ID mitgegeben wird. Dies ist notwendig, da in Kubernetes diese Anwendung mehrfach laufen soll und daher ist eine unterschiedliche Client ID notwendig, da sonst der MQTT-Broker die Verbindungen verweigern wird.

Diese wird mit der Funktion `crypto.randomBytes(16).toString('hex')` sichergestellt die einen zufällige 16-stellige zahl generiert die die Client-id mitgebenden wird.

(dieses Vorgehen wird auch genauso beim Simulationsprogram SIM gehandhabt, da auch dies theoretisch mehrfach zur Ausführen kommen muss)

Schreiben in MongoDB:

Empfängt der rectemp Service eine „gültige“ Messung wird diese in die MongoDB Datenbank geschrieben. Dazu muss zunächst das Daten-Schema für MongoDB definiert werden:

```
1 const mongoose = require("mongoose");
2 const Schema = mongoose.Schema;
3
4 let Sensor_Schema = new Schema(
5   {
6     unique_sensor_id: { type: String },
7     timestamp: { type: String },
8     locid: { type: Number },
9     gpslatitude: { type: String },
10    gpslongitude: { type: String },
11    sensortype: { type: String },
12    value: { type: String }
13  }
14);
15
16 module.exports = mongoose.model("sensor", Sensor_Schema);
```

Aufgabe-6-7\rectemp\model\sensor_werte.js

Anschließend wird die MongoDB URL entsprechend den Parametern der config.js Datei definiert:

Es wird die mongoosurl mit Authentifizierung genommen

`mongodb://mongoadmin:secret@mongo:27017/Temperatur?authSource=admin`

```
4 // Hier wird entschieden mit welchen Paramtern auf die MongoDB verbunden wird
5 // Für diesen MVP wurde Mongo DB OHNE zugriffsschutz als anonyme Zugangsdaten konfiguriert
6
7 if (global.env.DB_HOST && global.env.DB_PORT) {
8   // Überprüfe ob Zugriffsparameter vorhanden sind
9   if (global.env.DB_IS_SECURED === 'True') {
10     // Falls Mongo DB mit Username und Passwort authentifiziert werden muss verwenden diesen connection String:
11     var mongoosurl = "mongodb://" + global.env.DB_USERNAME + ":" + global.env.DB_PASSWORD + "@" + global.env.DB_HOST + ":" + global.env.DB_PORT;
12   } else {
13     // mongo DB Parameter aus Config aber keine Authentifizierung mittels Username oder Password
14     var mongoosurl = "mongodb://" + global.env.DB_HOST + ":" + global.env.DB_PORT + "/" + global.env.DB_DATABASE;
15   }
16 }
17
18 }
19 else {
20   //Datenbank-Config nicht vorhanden nutzte Vorgaben aus dem Anfordungsdokument
21   var mongoosurl = "mongodb://localhost:27017/Temperatur";
22 }
23 //console.log(mongoosurl);
24 mongoose.connect(mongoosurl, { useNewUrlParser: true, useUnifiedTopology: true });
25 console.log(mongoosurl)
```

Aufgabe-6-7\rectemp\db.js

Dann kann auch die Messung in die richtige DB geschrieben werden:

```
module.exports = {
  insert_mongodb: function (message) {
    const sonor_daten = require('../model/sensor_werte.js');
    var messung = JSON.parse(message);
    //console.log(messung)

    const insert_sensor_messung = {
      unique_sensor_id: messung.unique_sensor_id,
      timestamp: messung.timestamp,
      locid: messung.locid,
      gpslatitude: messung.gpslatitude,
      gpslongitude: messung.gpslongitude,
      sensortype: messung.sensortype,
      value: messung.value
    };

    //const duplicate = await User.findOne({userid: newUser.userid}).exec();
    sonor_daten.insertMany(insert_sensor_messung);
    console.log("Schreibe in MongoDB ");
  }
}
```

Aufgabe-6-7\rectemp\db.js

Sende Warnung bei Überschreiten von Schwellwert:

Eine Messung wird auf Überschreiten von Schwellwerten überprüft.
(Dabei kommen die Schwellwerte aus der Config.js Datei).

Geschieht dies, wird die *sende_schwellwert_mqtt_message* Funktion ausgeführt, die wie der Name bereits vermuten lässt, eine mqtt Message an das topic *4934001-errorCase/Schwellwert* sendet.
Dieses Topic wird vom temp-alert Service überwacht.

```
if (valide_messung = validation_test(topic, message)) {
  // Nur Messungen im Gültigkeitsbereich werden weiter untersucht
  db.insert_mongodb(message); //schreibe in DB
  switch (valide_messung.sensortyp) {
    // Check den Schwellenwert der GÜLTIGEN Sensorenprüfung ab.
    case 'T': {
      //Überprüfe Schwellwert für T=temperature
      if (valide_messung.value < global.T_unten || valide_messung.value > global.T_oben)
        sende_schwellwert_mqtt_message(valide_messung.sensortyp, valide_messung.value, valide_messung.messung)
      break;
    }
    case 'X': {
      //Überprüfe Schwellwert für X=co2
      if (valide_messung.value > global.X_oben)
        sende_schwellwert_mqtt_message(valide_messung.sensortyp, valide_messung.value, valide_messung.messung)
      break;
    }
    case 'P': {
      //Überprüfe Schwellwert für P=people in a room
      if (valide_messung.value < global.P_unten || valide_messung.value > global.P_oben)
        sende_schwellwert_mqtt_message(valide_messung.sensortyp, valide_messung.value, valide_messung.messung)
      break;
    }
    case 'H': {
      //Überprüfe Schwellwert für H=luftfeuchtigkeit
      if (valide_messung.value < global.H_unten || valide_messung.value > global.H_oben)
        sende_schwellwert_mqtt_message(valide_messung.sensortyp, valide_messung.value, valide_messung.messung)
      break;
    }
    case 'p': {
      //Überprüfe Schwellwert für p=Feinstaub (Feinstaub = PM2,5)
      if (valide_messung.value > global.p_oben)
        sende_schwellwert_mqtt_message(valide_messung.sensortyp, valide_messung.value, valide_messung.messung)
      break;
    }
  }
}
```

Aufgabe-6-7\rectemp\mqtt.js

```
Topic=4934001/messungen Message={"unique_sensor_id":"0e573477bcf8a91c6d52f4e701c91605","timestamp":"2022-12-13T21:08:55.963Z","loc
Schreibe in MongoDB
>  Schwellwert für H erreicht --> 75 <--      Publishe an 4934001-errorCase/Schwellwert

Topic=4934001/messungen Message={"unique_sensor_id":"2cb769572292613afea8d557599e3b6e","timestamp":"2022-12-13T21:08:56.719Z","loc
Schreibe in MongoDB
>  Schwellwert für T erreicht --> 46 <--      Publishe an 4934001-errorCase/Schwellwert

Topic=4934001/messungen Message={"unique_sensor_id":"988549d1830b60c4f0abb7b729e0bfd9","timestamp":"2022-12-13T21:08:58.154Z","loc
Schreibe in MongoDB
>  Schwellwert für p erreicht --> 22 <--      Publishe an 4934001-errorCase/Schwellwert
```

Ausgabe in rectemp bei Überschreiten von Schwellwert

Zusätzliche Features:

(Rest-)API-Schnittstelle:

Es wurde ein (Rest-)API-Schnittstelle implementiert die **anonym**:

- Alle Messungen in der Datenbank ausgeben kann:

Dazu muss ein GET-Request an rectemp Service über localhost:5000/api/ gesendet werden:

```

GET      localhost:5000/api/
Params   Authorization • Headers (10) Body Pre-request Script Tests
Body    Cookies (1) Headers (7) Test Results
Pretty Raw Preview Visualize JSON
54     "value": "3",
55     "_v": 0
56   },
57   {
58     "_id": "6398d9e5ee8703cf75180d77",
59     "unique_sensor_id": "09a4diff950a854b87718fed4c8ef21",
60     "timestamp": "2022-12-13T20:00:37.358Z",
61     "locid": 1234,
62     "gpslatitude": "48.60000",
63     "gpslongitude": "8.90000",
64     "sensorstype": "T",
65     "value": "22",
66     "_v": 0
67   },
68   {
69     "_id": "6398d9e63aff613241c93f6",
70     "unique_sensor_id": "cd4256f31a30a77de77f2e7b8da6a446",
71     "timestamp": "2022-12-13T20:00:42.360Z",
72     "locid": 5678,
73     "gpslatitude": "48.60000",
74     "gpslongitude": "8.90000",
75     "sensorstype": "p",
76     "value": "5",
77     "_v": 0
78   },
79   {
80     "_id": "6398d9eae8703cf75180d79",
81     "unique_sensor_id": "cd4256f31a30a77de77f2e7b8da6a446",
82     "timestamp": "2022-12-13T20:00:42.360Z",
83     "locid": 5678,
84     "gpslatitude": "48.60000",
85     "gpslongitude": "8.90000",
86     "sensorstype": "p",

```

```

11 const router: Router
12 router.get("/", async(req, res) => {
13   console.log("find all docs in mongodb");
14   alldocs = await sensor_werte.find({}).exec();
15   return res.status(200).send(alldocs);
16 });
17

```

Aufgabe-6-7|rectemp|Routes|api.js"

- Alle Messungen zu einer bestimmten Location in der Datenbank ausgeben kann:

Dazu muss ein GET-Request an rectemp Service über localhost:5000/api/getLocid/**1234** gesendet werden: (wobei **1234** eine gewünschte location ist)

```

GET      localhost:5000/api/getLocid/1234
Params   Authorization • Headers (12) Body Pre-request Script Tests Settings
Body    Cookies (1) Headers (8) Test Results
Pretty Raw Preview Visualize JSON
4     "unique_sensor_id": "09a4diff950a854b87718fed4c8ef21b",
5     "timestamp": "2022-12-13T20:00:37.358Z",
6     "locid": 1234,
7     "gpslatitude": "48.60000",
8     "gpslongitude": "8.90000",
9     "sensorstype": "T",
10    "value": "22",
11    "_v": 0
12  },
13  {
14    "_id": "6398d9e563aff613241c93f4",
15    "unique_sensor_id": "09a4diff950a854b87718fed4c8ef21b",
16    "timestamp": "2022-12-13T20:00:37.358Z",
17    "locid": 1234,
18    "gpslatitude": "48.60000",
19    "gpslongitude": "8.90000",
20    "sensorstype": "T",
21    "value": "22",
22    "_v": 0
23  },
24  {
25    "_id": "6398d9e6ee8703cf75180d77",
26    "unique_sensor_id": "09a4diff950a854b87718fed4c8ef21b",
27    "timestamp": "2022-12-13T20:00:37.358Z",
28    "locid": 1234,
29    "gpslatitude": "48.60000",
30    "gpslongitude": "8.90000",
31    "sensorstype": "T",
32    "value": "22",
33    "_v": 0
34  },
35  {
36    "_id": "6398d9e8f76c978a844bb29",
37    "unique_sensor_id": "09a4diff950a854b87718fed4c8ef21b",
38    "timestamp": "2022-12-13T20:00:42.360Z",
39    "locid": 1234,
40    "gpslatitude": "48.60000",

```

```

18 Complexity is 5 Everything is cool!
19 router.get ('/getLocid/:locid', async(req, res) => {
20   locid=(req.params.locid);
21   Complexity is 4 Everything is cool!
22   sensor_werte.find({locid: locid}, function(err,docs) {
23     if (err) {
24       console.log(err);
25       return res.sendStatus(400);
26     }
27   });

```

Aufgabe-6-7|rectemp|Routes|api.js"

- Alle Messungen zu einem bestimmten Sensors in der Datenbank ausgeben kann:
Dazu muss ein GET-Request an rectemp Service über localhost:5000/api/getSensor/**T** gesendet werden: (wobei **T** eine gewünschter Sensor ist)

```

localhost:5000/api/getSensor/T

GET /api/getSensor/T
Params Authorization Headers (7) Body Pre-request Script Tests Settings
Body Cookies (1) Headers (7) Test Results
Pretty Raw Preview Visualize JSON ▾

1 [
2   {
3     "_id": "6398d9e58f76c978a844bb26",
4     "unique_sensor_id": "09a4d1ff950a854b87718fed4c8ef21b",
5     "timestamp": "2022-12-13T20:00:37.358Z",
6     "locid": 1234,
7     "gpslatitude": "48.60000",
8     "gpslongitude": "8.90000",
9     "sensorType": "T",
10    "value": "22",
11    "_v": 0
12  },
13  {
14    "_id": "6398d9e58f76c978a844bb26",
15    "unique_sensor_id": "09a4d1ff950a854b87718fed4c8ef21b",
16    "timestamp": "2022-12-13T20:00:37.358Z",
17    "locid": 1234,
18    "gpslatitude": "48.60000",
19    "gpslongitude": "8.90000",
20    "sensorType": "T",
21    "value": "22",
22    "_v": 0
23  },
24  {
25    "_id": "6398d9e58f76c978a844bb26",
26    "unique_sensor_id": "09a4d1ff950a854b87718fed4c8ef21b",
27    "timestamp": "2022-12-13T20:00:37.358Z",
28    "locid": 1234,
29    "gpslatitude": "48.60000",
30    "gpslongitude": "8.90000",
31    "sensorType": "T",
32    "value": "22",
33    "_v": 0
34  },
35  {
36    "_id": "6398d9e58f76c978a844bb26",
37    "unique_sensor_id": "09a4d1ff950a854b87718fed4c8ef21b",

```

Aufgabe-6-7|rectemp\Routes\api.js

- Des Weiteren wurde ein POST-Schnittstelle implementiert die **authentifizierten** Personen es möglich macht die Schwellwerte (aus der Config.js) zu ändern. Dadurch ist das Anpassen der Schwellwerten zur Laufzeit des Services möglich und das Auslösen der Warnungen bei unter/überschreiten eines Schwellwerts kann beeinflusst werden.

Dazu muss ein POST an localhost:5000/api/change-parameter ausgeführt werden und im Body müssen verpflichtend *userid* und *password* (richtig) gesetzt werden. Zusätzlich können die Variablen aus der Config.js im Body angegeben werden, auf welchen Wert diese geändert werden sollen.

KEY	VALUE
<input checked="" type="checkbox"/> userid	admin
<input checked="" type="checkbox"/> password	admin
<input checked="" type="checkbox"/> T_unten	0
<input checked="" type="checkbox"/> T_oben	10
<input checked="" type="checkbox"/> H_oben	50
<input checked="" type="checkbox"/> H_unten	40
<input checked="" type="checkbox"/> P_oben	10
<input checked="" type="checkbox"/> P_unten	0
<input checked="" type="checkbox"/> X_oben	2500
<input checked="" type="checkbox"/> p_oben	3

```

router.post('/change-parameter', async (req, res) => {
  var rueckgabe = '';
  const login = (req.body.userid === authentification.userid && req.body.password === authentification.password);

  if (login) {
    if (isFinite((req.body.T_unten))) {
      global.T_unten = (req.body.T_unten);
      console.log(global.T_unten);
      rueckgabe += "\n T_unten wurde geändert auf: " + global.T_unten;
    }

    if (isFinite(req.body.T_oben)) {
      global.T_oben = req.body.T_oben;
      console.log(global.T_oben);
      rueckgabe += "\n T_oben wurde geändert auf: " + global.T_oben;
    }

    if (isFinite(req.body.X_oben)) {
      global.X_oben = req.body.X_oben;
      console.log(global.X_oben);
      rueckgabe += "\n X_oben wurde geändert auf: " + global.X_oben;
    }

    if (isFinite(req.body.P_unten)) {
      global.P_unten = req.body.P_unten;
      console.log(global.global.P_unten);
      rueckgabe += "\n P_unten wurde geändert auf: " + global.P_unten;
    }

    if (isFinite(req.body.P_oben)) {
      global.P_oben = req.body.P_oben;
      console.log(global.P_oben);
      rueckgabe += "\n P_oben wurde geändert auf: " + global.P_oben;
    }

    if (isFinite(req.body.H_unten)) {
      global.H_unten = req.body.H_unten;
      console.log(global.H_unten);
      rueckgabe += "\n H_unten wurde geändert auf: " + global.H_unten;
    }

    if (isFinite(req.body.H_oben)) {
      global.H_oben = req.body.H_oben;
      console.log(global.H_oben);
      rueckgabe += "\n H_oben wurde geändert auf: " + global.H_oben;
    }

    if (isFinite(req.body.p_oben)) {
      global.p_oben = req.body.p_oben;
      console.log(global.p_oben);
      rueckgabe += "\n p_oben wurde geändert auf: " + global.p_oben;
    }
  } else {
    console.log('no such userid/password');
    return res.status(404).send("no such userid/password");
  }
  //req.session.user = users;
  if (rueckgabe) return res.status(200).send(rueckgabe);
  else { return res.status(404).send("Fehler! Keine Daten geändert") };
}

```

Aufgabe-6-7|rectemp\Routes\api.js

Healthz-Schnittstelle:

Für die Liveness-Probe für Kubernetes, also die Probe, ob der Service noch läuft und reagiert ist eine einfach Healthcheck Schnittstelle mittels eines GET-Requests implementiert worden, der bei Aufruf von localhost:5000/healthz Status 200 und „OK“ zurück gibt.

The screenshot shows a Postman interface with a GET request to 'localhost:5000/healthz'. The 'Authorization' tab is selected, showing 'Inherit auth from parent'. The response status is 200 OK, and the body contains '1 OK'.

Temp-alert

Einfaches Node-Js Programm, das das Topic „4934001-errorCase“ abonniert, in welches rectemp die Messungen, die außerhalb des Schwellwerts liegen sendet.

Auch hier muss, wie bei dem SIM-Programm ein MQTT-Host definiert werden.

Da dieser Service mit dem MQTT-Service allerdings innerhalb von Kubernetes läuft muss dieser Host (mqtt:1883) so bestehen bleiben. Dieser verweist durch einen Loadbalancer innerhalb von Kubernetes auf den mosquitto MQTT-Broker

```
4 #####  
5 //Setzte mqtturl (auf LB von Kubernetes)  
6 //var mqttHOSTurl = "192.168.49.2:31501"  
7 var mqttHOSTurl = "mqtt:1883"  
8 #####  
9 var Topic = '4934001-errorCase/#'; //subscribe to all topics from postapp  
10 var client = mqtt.connect('mqtt://'+mqttHOSTurl, { clientId: "tempalert-random1234" });  
11
```

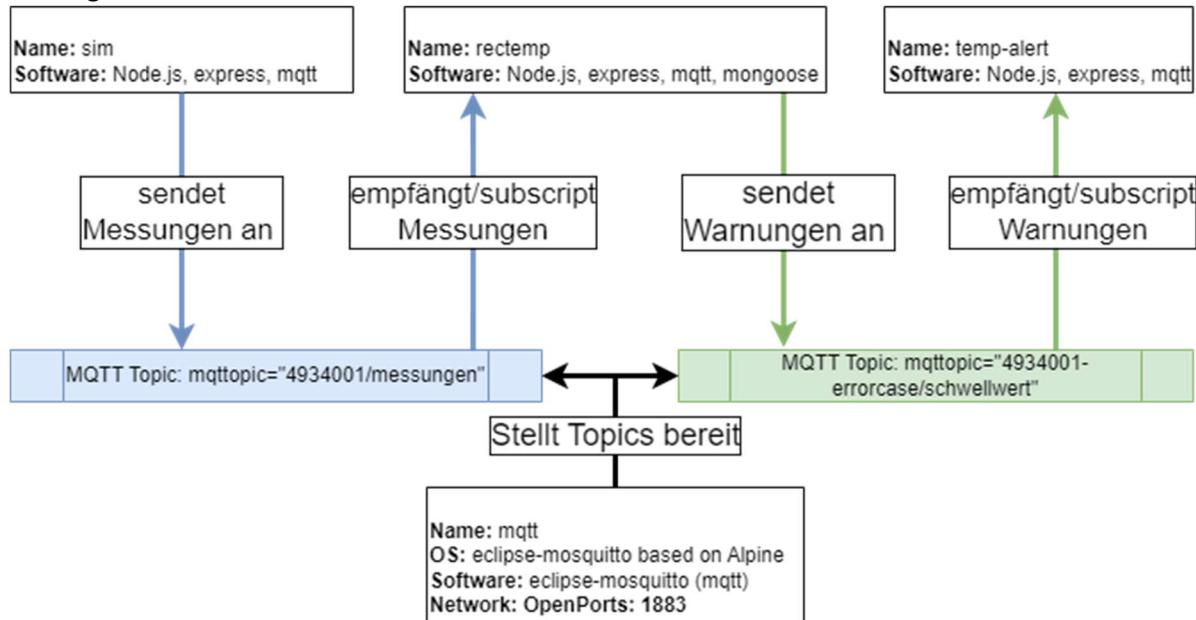
Aufgabe-6-7\temp-alert\index-skeleton.js

Ausgabe in Temp-alert:

```
Achtung Schwellwert überschritten!  
Wert: 26 bei Sensor 0e573477bcf8a91c6d52f4e701c91605 am Standort 1234:  
Topic=4934001-errorCase/Schwellwert  
Message={"unique_sensor_id":"0e573477bcf8a91c6d52f4e701c91605","timestamp":"2022-12-13T21:10:31.113Z","locid":"1234","gpslatitude"  
  
Achtung Schwellwert überschritten!  
Wert: 26 bei Sensor 0e573477bcf8a91c6d52f4e701c91605 am Standort 1234:  
Topic=4934001-errorCase/Schwellwert  
Message={"unique_sensor_id":"0e573477bcf8a91c6d52f4e701c91605","timestamp":"2022-12-13T21:10:31.113Z","locid":"1234","gpslatitude"
```

Schaubild MQTT Messages:

Folgendes Schaubild hilft zum Verständnis wie die einzelnen Services des MVPs sich gegenseitig mqtt Messages senden:



Praktischer Teil: Kubernetes

Netzwerk:

Wie aus dem Schaubild auf Seite 7 zusehen sind die Services (rectemp mongo,mqtt und temp-alert) alle in Kubernetes (im selben Namespace), damit diese sich untereinander und auch von außen erreichbar sind, sind die Services jeweils (alle außer temp-alert) mit einem internen Kubernetes Loadbalancer verbunden.

Diese Loadbalancer ermöglichen es, dass innerhalb von Kubernetes anhand eines Namens des Loadbalancers auf die Pods zugegriffen werden kann. Auch die **Lastverteilung für den Service rectemp, der mit 3 Pods aufgesetzt** wird, kann damit gewährleistet werden.

Konfigurierbare Verbindungsparameter für den Zugriff auf MongoDB:

Im MongoDB Deployment von Kubernetes wird auf den Username und das Passwort (was in einem erstellten Secret steht) referenziert. Damit MongoDB innerhalb von Kubernetes mit Username und Passwort initialisiert wird, sind diese Values über die sogenannten Kubernetes-Secrets bei Erstellung des Deployment mitzugeben.

```
Aufgabe-8 > mongodb > mongodb-secrets.yaml
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    creationTimestamp: null
5    name: mongo-creds
6  data:
7    username: bw9uZ29hZG1pbpg==
8    password: c2VjcmV0
```

Aufgabe-8\mongodb\mongodb-secrets.yaml

```
46
47
48
49
50
51
52
53
54
55
56
57
58
env:
- name: MONGO_INITDB_DATABASE
  value: Temperatur
- name: MONGO_INITDB_ROOT_USERNAME
  valueFrom:
    secretKeyRef:
      name: mongo-creds
      key: username
- name: MONGO_INITDB_ROOT_PASSWORD
  valueFrom:
    secretKeyRef:
      name: mongo-creds
      key: password
```

Aufgabe-8\mongoDB\mongodb-deployment.yaml

Resilienz gegen Ausfall

Um die Resilienz gegen Ausfall zu sichern, wurde der rectemp bereits mit einer Health-check ausgestattet. (siehe Seite 17)

Mit der Kubernetes Funktion *livenessProbe* wird in einem definierten Abstand an GET-Request auf eine URL innerhalb des Pods durchgeführt. Schlägt dies nach (standartmäßig) 3 Versuchen fehl (kein http Status Code 200 oder 301) , wird der Pod neugestartet.

```
livenessProbe:  
  httpGet:  
    path: /healthz  
    port: 5000  
    initialDelaySeconds: 15  
    periodSeconds: 15
```

Aufgabe-8\rectemp\rectemp-deployment.yaml

```
Tolerations:  
  - key: node.kubernetes.io/not-ready  
    operator: Exists  
    value: "true"  
  - key: node.kubernetes.io/unreachable  
    operator: Exists  
    value: "true"  
  
Events:  
  Type Reason Age From Message  
  ---- ---- -- -- --  
  Normal Scheduled 5s default-scheduler Successfully assigned default/rectemp-69bb5f5bfc-6n5ht to docker-desktop  
  Normal Pulled 5s kubelet Container image "tomisboy/pub:rectemp" already present on machine  
  Normal Created 5s kubelet Created container rectemp  
  Normal Started 4s kubelet Started container rectemp  
  Warning Unhealthy 1s (x3 over 3s) kubelet Liveness probe failed: Get "http://10.1.0.94:5000/healthz": dial tcp 10.1.0.94:5000: connect: connection refused  
  Normal Killing 1s kubelet Container rectemp failed liveness probe, will be restarted
```

livenessProbe schlägt fehl, da bspw. NodeJs abgestürzt ist → Pods wird neugestartet

Quota Limit für rectemp

Um ein Quotalimit festzulegen ist folgendes zu konfigurieren

```
resources:  
  limits:  
    memory: "128Mi"  
    cpu: "100m"
```

Aufgabe-8\rectemp\rectemp-deployment.yaml

Integrationstest / (Installations-Anleitung)

Voraussetzung für ein reibungslosen Test / Deployment ist ein Windows Hostsystem mit Docker-Desktop und installiertem Kubernetes

Bevor der MVP in Kubernetes deployt werden kann, müssen die Docker-Container zuerst gebaut werden. Dazu ist es hilfreich zuerst in das Verzeichnis „Aufgabe-6-7“ zu wechsel und anschließend über die docker-compose.yaml file die Container zu bauen:

Lokaler Bau der Container mit docker-compose

```
PS C:\Users\Thomas\Aufgabe-6-7> docker-compose-v1.exe build
```

Anschließend stehen die Images zur Verfügung:

□	NAME	TAG	STATUS	CREATED	SIZE	ACTIONS
□	tomisboy/pub 7b96a2daa14b	tempalert	In use	about 1 hour ago	192.68 MB	▶ ⋮ 🗑
□	tomisboy/pub 4713f1697287	rectemp	In use	about 2 hours ago	772.63 MB	▶ ⋮ 🗑

Dann können die Docker-Container testweise lokal gestartet werden. Dazu muss zuerst das Netzwerk *mqtt* erstellt werden:

```
PS C:\Users\Thomas\Aufgabe-6-7> docker network create mqtt
```

Dann können die Container mit docker-compose gestartet werden.

```
PS C:\Users\Thomas\Aufgabe-6-7> docker-compose-v1.exe up
```

```
temp-alert  | [nodemon] to restart at any time, enter `rs`  
temp-alert  | [nodemon] watching path(s): *.*  
temp-alert  | [nodemon] watching extensions: js,mjs,json  
temp-alert  | [nodemon] starting `node index-skeleton.js`  
temp-alert  | Connecting MQTT  
temp-alert  | Subscribed to 4934001-errorCase/#  
rectemp    | mongodb://mongo:27017/Temperatur  
rectemp    | MQTT_HOST= mqtt://mqtt:1883  
rectemp    | (node:30) [MONGOOSE] DeprecationWarning: Mongoose: the `strictQuery` option  
u want to prepare for this change. Or use `mongoose.set('strictQuery', true);` to suppress  
rectemp    | (Use `node --trace-deprecation ...` to show where the warning was created)  
rectemp    | Server started  
rectemp    | Connecting MQTT  
rectemp    | Subscribed to 4934001/#  
mongo      | MongoDB init process complete; ready for start up.  
mongo      |  
mongo      | {"t":{"$date":"2022-12-13T22:24:18.820+00:00"}, "s": "I", "c": "CONTROL", "i": "ledProtocols", "v": "none"}  
mongo      | {"t":{"$date":"2022-12-13T22:24:18.823+00:00"}, "s": "W", "c": "ASIO", "i": "f+", "v": "2022-12-13T22:24:18.823+00:00"}  
mongo      | {"t":{"$date": "2022-12-13T22:24:18.823+00:00"}, "s": "T", "c": "NETWORK", "i": "f+"}
```

Die Container sollten nun lokal in Docker laufen.

Startet man das Sim- Programm auf einer weiteren Shell sieht man auch schon diesen MQTT-Messages senden und von rectemp empfangen

PS C:\Users\Thomas\Aufgabe-5> npm start 5 1234 500 T 16 88

```

mqttmsg = {
    unique_sensor_id: '24776d0293cec8765d0e8da22dc883d',
    timestamp: '2022-12-13T23:06:16.659Z',
    locid: '1234',
    gpslatitude: '48.60000',
    gpslongitude: '8.90000',
    sensortype: 'T',
    value: 19
}
mqtttopic = 4934001/messungen
mqtt msg {"unique_sensor_id": "24776d0293cec8765d0e8da22dc883d", "timestamp": "2022-12-13T23:06:16.659Z", "locid": "1234", "gpslatitude": "48.60000", "gpslongitude": "8.90000", "sensortype": "T", "value": 19}
mqttmsg = {
    unique_sensor_id: '24776d0293cec8765d0e8da22dc883d',
    timestamp: '2022-12-13T23:06:21.660Z',
    locid: '1234',
    gpslatitude: '48.60000',
    gpslongitude: '8.90000',
    sensortype: 'T',
    value: 21
}
mqtttopic = 4934001/messungen
mqtt msg {"unique_sensor_id": "24776d0293cec8765d0e8da22dc883d", "timestamp": "2022-12-13T23:06:21.660Z", "locid": "1234", "gpslatitude": "48.60000", "gpslongitude": "8.90000", "sensortype": "T", "value": 21}
mqttmsg = {
    unique_sensor_id: '5d08690db41b11cce65f1ab141f04710',
    timestamp: '2022-12-13T23:10:16.567Z',
    locid: '1234',
    gpslatitude: '48.60000',
    gpslongitude: '8.90000',
    sensortype: 'T',
    value: 89
}
mqtttopic = 4934001/messungen
mqtt msg {"unique_sensor_id": "5d08690db41b11cce65f1ab141f04710", "timestamp": "2022-12-13T23:10:16.567Z", "locid": "1234", "gpslatitude": "48.60000", "gpslongitude": "8.90000", "sensortype": "T", "value": 89}

```

Simulationsprogramm sendet 3 Messwerte der Temperaturen. Einmal 19 Grad 21 Grad und 89 Grad

Die drei Messungen kommen an rectemp an:

- Messung 19 Grad ist in Ordnung und wird in die MongoDB gespeichert
- Messung 21 Grad ist nicht in Ordnung, da der Schwellwert überschritten (Temperatur darf nur zwischen 16 und 19 Grad sein) wird dennoch in MongoDB geschrieben
- Messung 89 Grad ist offensichtlich ein Fehler und liegt außerhalb des Gültigkeitsbereichs des Sensors, daher wird die Messung verworfen

```

rectemp | Topic=4934001/messungen Message={"unique_sensor_id": "24776d0293cec8765d0e8da22dc883d", "timestamp": "2022-12-13T23:06:16.659Z", "locid": "1234", "gpslatitude": "48.60000", "gpslongitude": "8.90000", "sensortype": "T", "value": 19}
rectemp | Schreibe in MongoDB
rectemp | Topic=4934001/messungen Message={"unique_sensor_id": "24776d0293cec8765d0e8da22dc883d", "timestamp": "2022-12-13T23:06:21.660Z", "locid": "1234", "gpslatitude": "48.60000", "gpslongitude": "8.90000", "sensortype": "T", "value": 21}
rectemp | Schreibe in MongoDB
rectemp | > Schwellwert für T erreicht --> 21 <-- Publish an 4934001-errorCase/Schwellwert
rectemp |
temp-alert | Achtung Schwellwert überschritten!
temp-alert | Wert: 21 bei Sensor 24776d0293cec8765d0e8da22dc883d am Standort 1234:
temp-alert | Topic=4934001-errorCase/Schwellwert
temp-alert | Message={"unique_sensor_id": "24776d0293cec8765d0e8da22dc883d", "timestamp": "2022-12-13T23:06:21.660Z", "locid": "1234", "gpslatitude": "48.60000", "gpslongitude": "8.90000", "sensortype": "T", "value": 21}
temp-alert |
rectemp | >>ACHTUNG<< Fehlerhafte Messung, außerhalb des Gültigkeitsbereich, wird nicht übernommen: Topic=4934001/messungenMessage={"unique_sensor_id": "5d08690db41b11cce65f1ab141f04710", "timestamp": "2022-12-13T23:10:16.567Z", "locid": "1234", "gpslatitude": "48.60000", "gpslongitude": "8.90000", "sensortype": "T", "value": 89}
rectemp |

```

Ausgabe der docker-container temp-alert und rectemp

Diese Daten sind dann auch in der MongoDB zu finden:

Connection: String:

`mongodb://mongoadmin:secret@localhost:27017/Temperatur?authSource=admin`

The screenshot shows the MongoDB Compass interface connected to the 'Temperatur' database. The 'sensors' collection is selected. Two documents are listed:

```
_id: ObjectId('63990ba02c92a199508a61abd')
unique_sensor_id: "813c30f6a3be4aa7c8ee6e3a09f364aa"
timestamp: "2022-12-13T23:30:10.618Z"
locid: 1234
gpslatitude: "48.60000"
gpslongitude: "8.90000"
sensorType: "T"
value: "21"
__v: 0

_id: ObjectId('63990ba02c92a199508a61abd')
unique_sensor_id: "813c30f6a3be4aa7c8ee6e3a09f364aa"
timestamp: "2022-12-13T23:30:10.618Z"
locid: 1234
gpslatitude: "48.60000"
gpslongitude: "8.90000"
sensorType: "T"
value: "19"
__v: 0
```

Und auch die Abfrage über die REST-Schnittstelle: `http://localhost:5000/api/getSensor/T` liefert die Ergebnisse:

The screenshot shows a POSTMAN request to `http://localhost:5000/api/getSensor/T`. The response body contains the following JSON data:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
{
  "_id": "63990ba02c92a199508a61abd",
  "unique_sensor_id": "813c30f6a3be4aa7c8ee6e3a09f364aa",
  "timestamp": "2022-12-13T23:30:10.618Z",
  "locid": 1234,
  "gpslatitude": "48.60000",
  "gpslongitude": "8.90000",
  "sensorType": "T",
  "value": "21",
  "__v": 0
},
{
  "_id": "639912143caf9683768dcf7",
  "unique_sensor_id": "f71afdf77349bd75a07f280004f77b9fb",
  "timestamp": "2022-12-14T00:00:20.740Z",
  "locid": 1234,
  "gpslatitude": "48.60000",
  "gpslongitude": "8.90000",
  "sensorType": "T",
  "value": "19",
  "__v": 0
}
```

Somit ist sichergestellt, dass alles funktioniert und es kann mit dem Kubernetes Deployment fortgefahren werden. Dazu müssen zunächst Container gestoppt und gelöscht werden. Mit **CTRL + C** werden die Container gestoppt. Anschließend müssen sie gelöscht werden.

```
PS C:\Users\Thomas\Aufgabe-6-7> docker-compose-v1.exe down
PS C:\Users\Thomas\Aufgabe-6-7> docker-compose-v1.exe rm
```

Deployment in Kubernetes

Damit die Anwendungen in Kubernetes laufen, müssen die Deployment YAML-Files aus den Ordner Aufgabe-8 gestartet werden:

```
PS C:\Users\Thomas\Aufgabe-8> kubectl.exe apply -f .\mongodb\
PS C:\Users\Thomas\Aufgabe-8> kubectl.exe apply -f .\mqtt\
PS C:\Users\Thomas\Aufgabe-8> kubectl.exe apply -f .\rectemp\
PS C:\Users\Thomas\Aufgabe-8> kubectl.exe apply -f .\temp-alert\
```

Pods:

NAME↑	PF	READY	PODS (DEFAUTL/10)		NODE
			RESTARTS	STATUS	
mongo-d66f974d7-n6h5n	●	1/1	0	Running	10.1.0.164
mqtt-5455fbf565-8vb7q	●	1/1	0	Running	10.1.0.152
rectemp-65588c9c-fhfb1	●	1/1	0	Running	10.1.0.167
rectemp-65588c9c-hzvd6	●	1/1	0	Running	10.1.0.165
rectemp-65588c9c-xvz68	●	1/1	0	Running	10.1.0.166
temp-alert-845784449b-hrcxl	●	1/1	0	Running	10.1.0.153

Services:

NAME↑	TYPE	CLUSTER-IP	SERVICES (DEFAUTL/14)		AGE
			EXTERNAL-IP	PORTS	
kubernetes	ClusterIP	10.96.0.1		https:443<-->0	30h
mongo	LoadBalancer	10.103.146.150	localhost	mongo:27017<-->30087	19s
mqtt	LoadBalancer	10.106.57.84	localhost	mqtt:1883<-->32295	24m
rectemp	LoadBalancer	10.107.160.85	localhost	rectemp:5000<-->32181	9m30s

Beispiele:

Anschließend kann das Sim-Programm (mehrfach) gestartet werden und weiter wieder Messungen zu generieren. Diese landen dann im rectemp Service innerhalb von Kubernetes

Erinnerung: Es sind nur Messungen von den Locations 1234,3456,4567 und 5678 erlaubt.

```
PS C:\Users\Thomas\Aufgabe-5> npm start 5 1234 500 T 16 88
PS C:\Users\Thomas\Aufgabe-5> npm start 5 3456 500 p 2 6
PS C:\Users\Thomas\Aufgabe-5> npm start 5 1234 500 H 20 70
PS C:\Users\Thomas\Aufgabe-5> npm start 5 4567 500 P 0 30
PS C:\Users\Thomas\Aufgabe-5> npm start 5 1234 500 X 1997 2502
```

rectemp Ausgabe:

```
>>> ACHTUNG<< Fehlerhafte Messung, außerhalb des Gültigkeitsbereich, wird nicht übernommen: Topic=4934001/messungenMessage={"unique_sensor_id":"8e0f7bcc2d2a195a2b814d102dd7959b","timestamp": "2022-12-14T01:35:28.643Z","locid": "1234","gpslatitude": "48.60000","gpslongitude": "10.96000"}>>>
Topic=4934001/messungen Message={"unique_sensor_id":"8e0f7bcc2d2a195a2b814d102dd7959b","timestamp": "2022-12-14T01:35:28.643Z","locid": "1234","gpslatitude": "48.60000","gpslongitude": "10.96000"}>>> Schreibe in MongoDB
> Schwellwert für T erreicht --> 22 <-- Publish an 4934001-errorCase/Schwellwert
Topic=4934001/messungen Message={"unique_sensor_id":"c9023a2318150d96d75e6269c92848b6","timestamp": "2022-12-14T01:35:28.962Z","locid": "3456","gpslatitude": "48.60000","gpslongitude": "10.96000"}>>> Schreibe in MongoDB
> Schwellwert für H erreicht --> 17 <-- Publish an 4934001-errorCase/Schwellwert
Topic=4934001/messungen Message={"unique_sensor_id":"8e0f7bcc2d2a195a2b814d102dd7959b","timestamp": "2022-12-14T01:35:30.586Z","locid": "5678","gpslatitude": "48.60000","gpslongitude": "10.96000"}>>> Schreibe in MongoDB
> Schwellwert für P erreicht --> 19 <-- Publish an 4934001-errorCase/Schwellwert
```

temp-alert Ausgabe:

```
Achtung Schwellwert überschritten!
Wert: 75 bei Sensor c9023a2318150d96d75e6269c92848b6 am Standort 3456:
Topic=4934001-errorCase/Schwellwert
Message={"unique_sensor_id":"c9023a2318150d96d75e6269c92848b6","timestamp": "2022-12-14T01:36:19.022Z","locid": "3456","gpslatitude": "48.60000","gpslongitude": "8.90000","sensortype": "H"}>>>
Achtung Schwellwert überschritten!
Wert: 14 bei Sensor 8e0f7bcc2d2a195a2b814d102dd7959b am Standort 5678:
Topic=4934001-errorCase/Schwellwert
Message={"unique_sensor_id":"8e0f7bcc2d2a195a2b814d102dd7959b","timestamp": "2022-12-14T01:36:20.656Z","locid": "5678","gpslatitude": "48.60000","gpslongitude": "8.90000","sensortype": "P"}>>>
```

monogdb:

Temperatur.sensors

207 1 DOCUMENTS INDEXES

Documents Aggregations Schema Explain Plan Indexes Validation

Filter Type a query: { field: 'value' } Reset Find More Options

ADD DATA EXPORT COLLECTION

_id: ObjectId('630992865226494291bdcc2e9')
unique_sensor_id: "8e0f7bcc2d2a195a2b814d102dd7959b"
timestamp: "2022-12-14T01:35:28.643Z"
gpslatitude: "48.60000"
gpslongitude: "10.96000"
sensortype: "T"
value: 88
__v: 0

_id: ObjectId('630992865226494291bdcc2e9')
unique_sensor_id: "8e0f7bcc2d2a195a2b814d102dd7959b"
timestamp: "2022-12-14T01:35:28.643Z"
gpslatitude: "48.60000"
gpslongitude: "10.96000"
sensortype: "T"
value: 88
__v: 0