Smart Light System for home

GitHub Repository

https://github.com/BryantSarabia/SE4IOT

Summary

Introduction

A Smart Light System for the home is designed to enhance both energy efficiency and user experience. This solution leverages a network of sensors, automated controls, and data analytics to dynamically adjust lighting conditions based on real-time environmental factors.

Users can seamlessly control their home lighting through an interface, offering the flexibility to set preferences, monitor energy consumption, and ensure optimal illumination tailored to their needs.

The heart of the system lies in its ability to adapt to changing circumstances. Through the integration of sensors such as light intensity sensors and motion sensors the Smart Light System continuously assesses environmental conditions. When ambient light diminishes or motion is detected, the system autonomously triggers smart lighting devices, including smart bulbs to adjust brightness levels or activate/deactivate as required.

System Description

Smart Home Lighting System is designed to optimize energy usage and improve user experience in residential lighting. By leveraging a combination of sensors, data analytics, and automated control systems, this smart lighting system ensures that lighting conditions within the home are tailored to real-time environmental factors.

Functional requirements

Real time monitoring

The system continuously monitors environmental conditions through a network of sensors, including light sensors and motion sensors.

Automated lighting control

When the ambient light intensity falls below a certain threshold or motion is detected in a room, the system triggers smart lighting devices to adjust brightness levels or turn on/off accordingly.

User interface

The user interface provides real-time status updates, allows users to adjust lighting settings, offers energy consumption information, and influences the rule-based decision-making process.

Data storage

The system stores sensor data for further analysis and system configuration settings, such as light intensity thresholds.

Non-functional requirements

Reliability

The system must operate reliably under varying lighting conditions and user preferences.

Scalability

The system should be able to accommodate additional sensors or smart lighting devices as needed.

System components

Sensors

The sensors are crucial for **monitoring** and gathering real-time data about light conditions. The main sensors are:

- Light intensity sensor
- Motion sensor

Smart lighting devices

Examples of smart lighting devices:

- Smart bulbs

Central Control System

The Central Control System analyzes data from sensors and considers user preferences determine optimal lighting conditions. The system creates a plan based on rule-based decisions, adjusting lighting levels or turning lights on/off as needed.

The central control system is implemented using Node.js. It is used to register the sensors and actuators in the system and to create the dashboards for them in Grafana. It is also used to get and update the user preferences.

It is responsible for evaluating the sensors data by using the user preferences and to send the commands to the actuators It is responsible for evaluating the sensors data by using the user preferences, to send the commands to the actuators and turning off all the lights when the user disables them.

Adaptation goals

Goal	**Description**	**Evaluation metric**
Adjust Lighting	Adjust lighting levels when ambient light intensity falls below or exceeds a certain threshold and motion is detected in the room.	$((Li < Lit) \vee (Li > Lit)) \wedge Md$
Turn off lights	Turn off lights when the room is unoccupied or when lights are disabled.	$Ld \vee \neg Md$

- Li: Light intensity
- Lit: Light intensity threshold
- Md: Motion detected
- Ld: Lights disabled.

Decision function

The Smart Home Lighting System employs a rule-based decision-making approach to determine optimal lighting conditions. The decision logic is governed by predefined rules that consider environmental conditions and user preferences.

Currently, there is an evaluation rule for every type of sensor. The current evaluation rules are the following:

- **Light rule**:
    - If lights are disabled, do not perform any action.
    - If the lights are off, do not perform any action.
    - If no motion is detected in the room, do not perform any action.
    - Otherwise, if the light sensor value is less than the minimum brightness, increase the brightness. If the light sensor value is greater than the maximum brightness, decrease the brightness.

```
export class LightRule extends Rule {
  constructor ({ logger }) {
    super({ type: SENSOR_TYPES.LIGHT })
    this.logger = logger
  }

  evaluate ({ sensorData, userPreferences, room }) {
    let action = null
    const { lightsEnabled } = userPreferences
    if (!lightsEnabled) {
      console.log('Lights are disabled, no need to evaluate light rule.')
      return action
    }
    const { value, type } = sensorData
    const actuatorType = SENSOR_TO_ACTUATOR[type]
    if (!actuatorType) {
      console.log(`No actuator type found for sensor type ${type}`)
      return action
    }
    if (!room.getValue(ROOM_VALUES_KEYS.IS_LIGHT_ON)) {
      console.log(`Lights are OFF on room ${room.name}, no need to evaluate light rule.`)
      return action
    }
    const isMotionDetected = room.getValue(ROOM_VALUES_KEYS.IS_MOTION_DETECTED)
    if (isMotionDetected === undefined || isMotionDetected === null || isMotionDetected === false) {
      console.log(`No motion detected in ${room.name}, no need to evaluate light rule.`)
      return action
    }
    const minimumLightIntensityThreshold = Number(userPreferences.minimumLightIntensityThreshold)
    const maximumLightIntensityThreshold = Number(userPreferences.maximumLightIntensityThreshold)
    if (value < minimumLightIntensityThreshold) {
      action = {
        topic: `actuators/${room.name}/${actuatorType}/increase`
      }
    } else if (value > maximumLightIntensityThreshold) {
      action = {
        topic: `actuators/${room.name}/${actuatorType}/decrease`
      }
    }
    return action
  }       You, 4 weeks ago • feat(Actuators and CCS):
}
```

- **Motion rule**:
    - If lights are disabled, do not perform any action.
    - If there is motion in the room and lights are off, turn on the lights.
    - If there is motion in the room and lights are on, do not perform any action.
    - Otherwise, turn off the lights.

```
export class MotionRule extends Rule {
  constructor ({ logger }) {
    super({ type: SENSOR_TYPES.LIGHT })
    this.logger = logger
  }

  evaluate ({ sensorData, userPreferences, room }) {
    let action = null
    const { lightsEnabled } = userPreferences
    if (!lightsEnabled) {
      console.log('Lights are disabled, no need to evaluate motion rule.')
      return action
    }
    const { value, type } = sensorData
    if (value === null || value === undefined) return
    const isLightOn = room.getValue(ROOM_VALUES_KEYS.IS_LIGHT_ON)
    if (isLightOn === value) {
      console.log(`Lights are already ${isLightOn ? '"ON"' : '"OFF"'} on room ${room.name}`)
      return action
    }
    console.log(`Turning lights ON in room ${room.name}`)
    const actuatorType = SENSOR_TO_ACTUATOR[type]
    if (!actuatorType) {
      console.log(`No actuator type found for sensor type ${type}`)
      return action
    }
    const turn = value ? 'on' : 'off'
    const topic = `actuators/${room.name}/${actuatorType}/${turn}`
    action = {
      topic,
      message: {
        value
      }
    }
    room.updateValue(ROOM_VALUES_KEYS.IS_LIGHT_ON, value)
    room.updateValue(ROOM_VALUES_KEYS.IS_MOTION_DETECTED, value)
    return action
  }
}
```

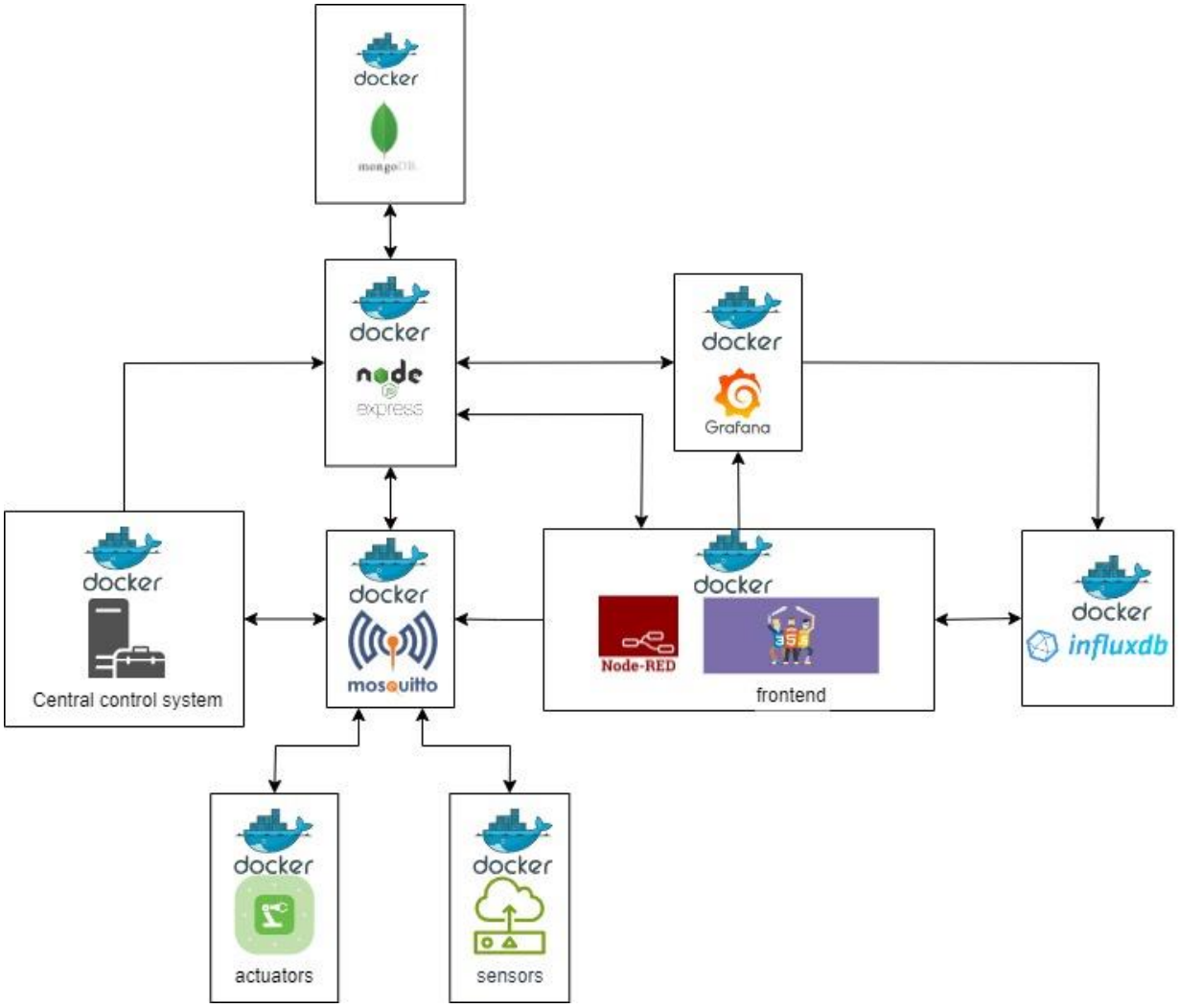
User Interface

The user interface allows users to monitor the system's status, adjust lighting settings, and view energy consumption reports.

Data storage

The data collected by sensors and system configuration settings are stored in a database.

System architecture



Server

The server is implemented using Node.js and Express, it is used to get and update the user preferences and to create a new dashboard on grafana for every sensor and actuator added to the system.

The user preferences are stored in a MongoDB database.

User preferences sent to the backend are valitated using [zod](zod).

Grafana

Grafana is used to create the dashboards for the sensors and actuators. The dashboards are created using the Grafana API.

Influxdb

Influxdb is used to store the data of the sensors and actuators.

Node-RED

Node-RED is used to create the flows of the system. The flows are used to listen for sensor and actuators data and to store it in the Influxdb database.

It is also used to create a dashboard for the user to control the lights and to monitor the sensors and actuators.

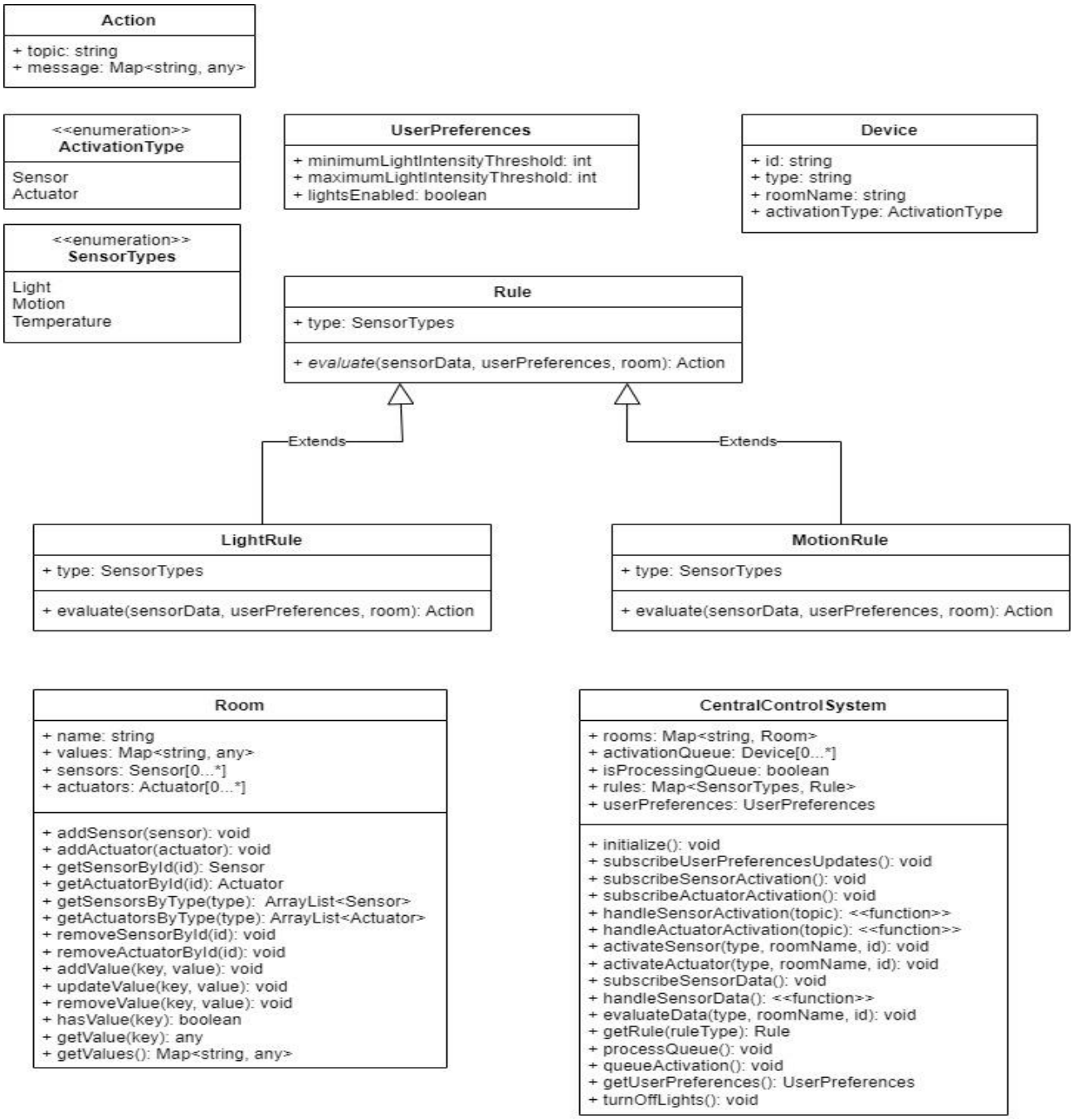
Mosquitto

Mosquitto is used to create the MQTT broker for the system. The sensors and actuators use MQTT to communicate with the server and with Node-RED.

Central Control System

The central control system is implemented using Node.js. It is used to register the sensors and actuators in the system and to create the dashboards for them in Grafana. It is also used to get and update the user preferences.

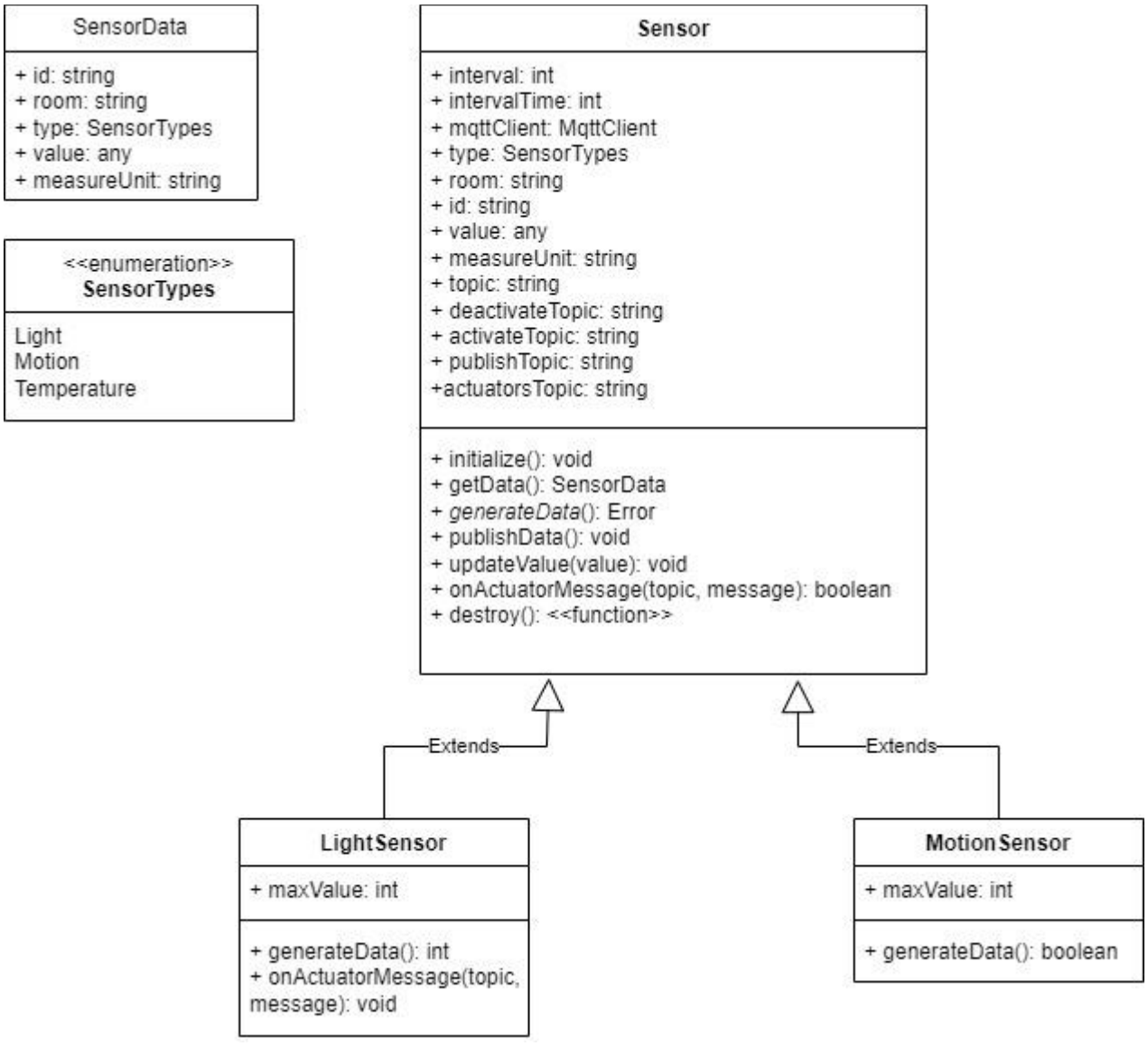
It is responsible for evaluating the sensors data by using the user preferences and to send the commands to the actuators and turning off all the lights when the user disables them.



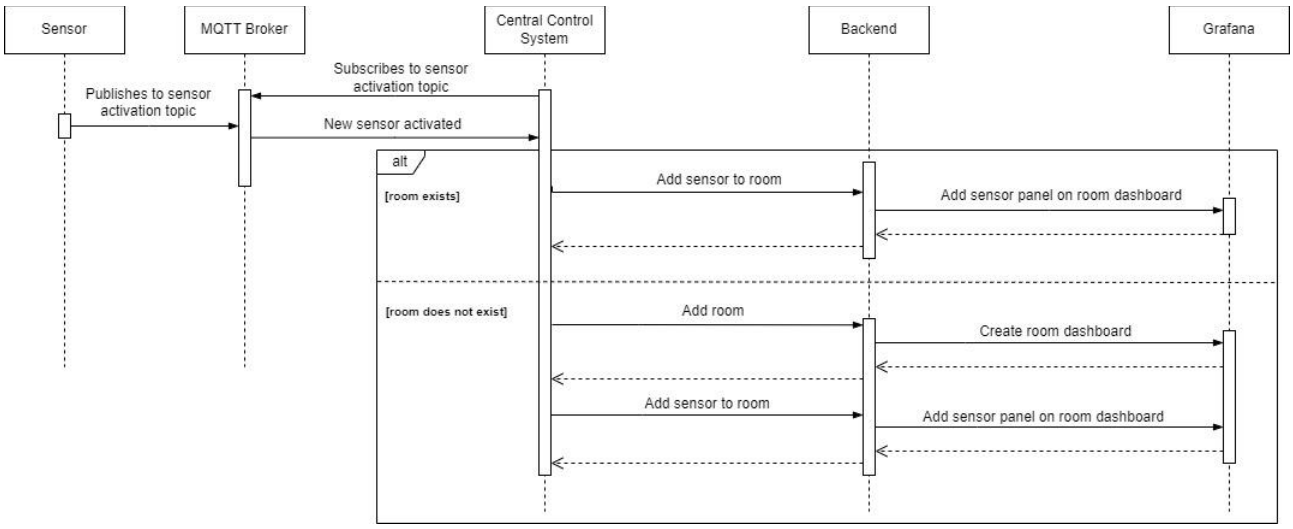
Sensors

The sensors are simulated using Node.js. The measurement unit of the light sensor is **lux,** and the measurement unit of the motion sensor is a **Boolean** value. They are crucial for **monitoring** and gathering real-time data about light conditions. The main sensors are:

- Light intensity sensor: the light sensor is used to measure the brightness of the room. It generates a random number between 0 and 100 every 5 seconds. It receives data from simulated actuators to change its value.
- Motion sensor: the motion sensor is used to detect motion in the room. It generates a random number between 0 and 1 every 5 seconds.



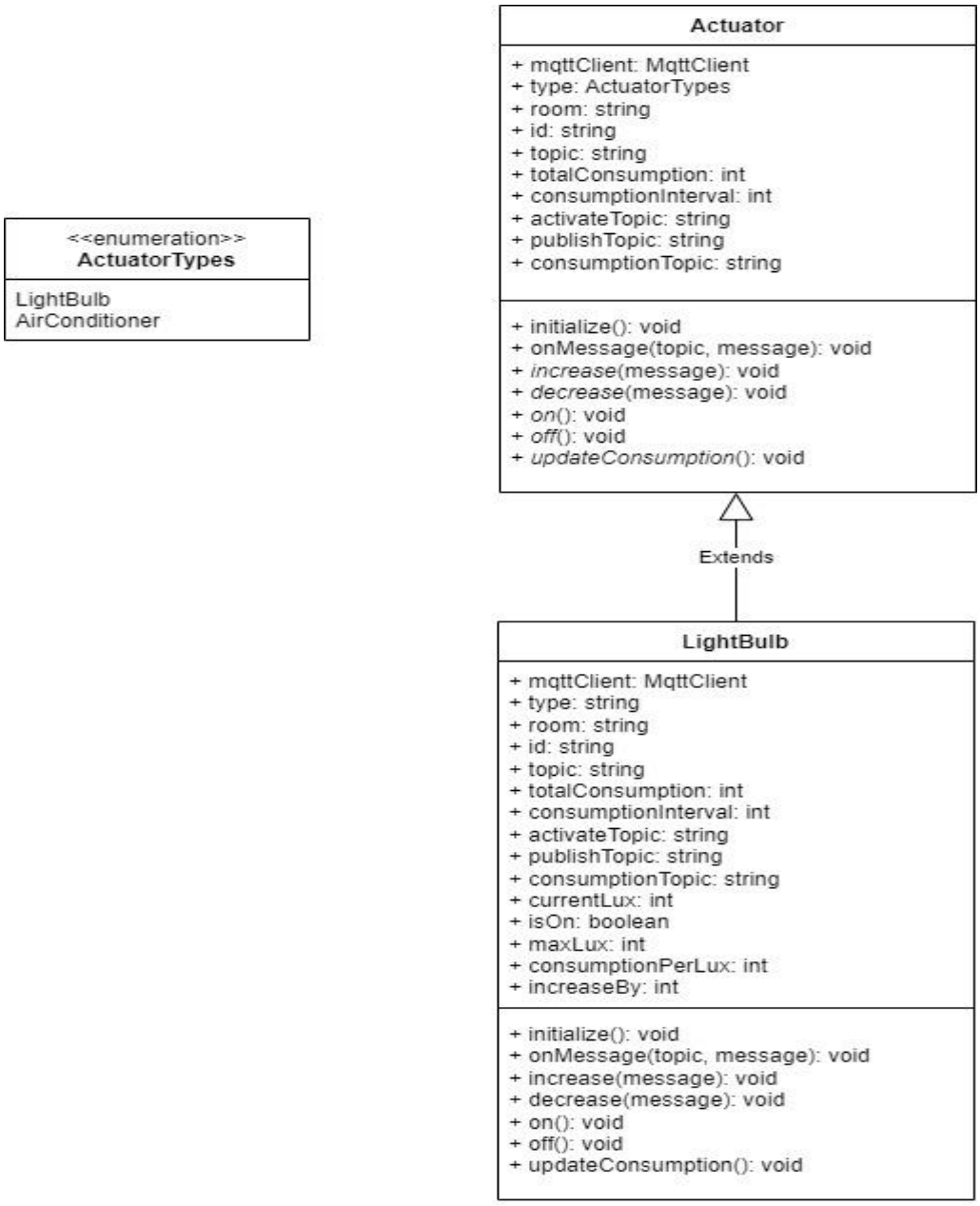
Sensor Activation Sequence diagram



Actuators

The actuators are simulated using Node.js. The main actuator is:

- Smart bulbs: the light bulbs are used to control the brightness of the room. They receive commands from the central control system and have 5 methods.
    - *on*: Turns on the light.
    - *off*: Turns off the light.
    - *increase*: Increases the brightness of the light.
    - *decrease*: Decreases the brightness of the light.
    - *updateConsumption*: Updates the current consumption of the light and publishes the value to the MQTT broker.
        - The current consumption is calculated using the following formula: *currentConsumption* = *currentLux* * *consumptionPerLux*



Actuator activation sequence diagram

Same as sensor sequence diagram.

Technologies

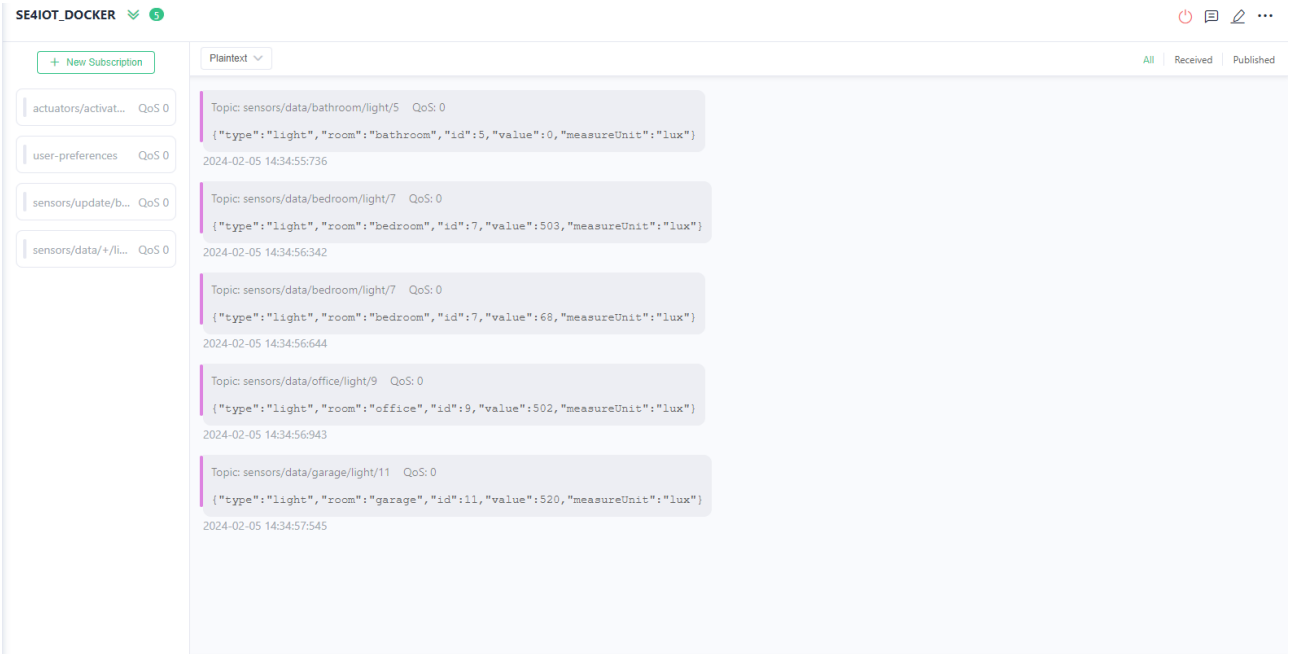
The system has been developed with the following technologies:

- MQTT: as communication protocol.
- Mosquitto: to collect real-time data from sensors and communication between the central control system, devices, and actuators.
- Paho MQTT
- InfluxDB: for storing historical sensor data.
- Grafana: for visualizing system performance and historical data.
- Docker: for containerizing the application.
- Git: for version control.
- Express.js: for the implementation of the backend.
- MongoDB: store system configuration settings.
- NodeRED: for the implementation of the dashboard and storing sensor and actuator data on Influxdb.
- Node.js: for the simulation of the sensors and actuators.

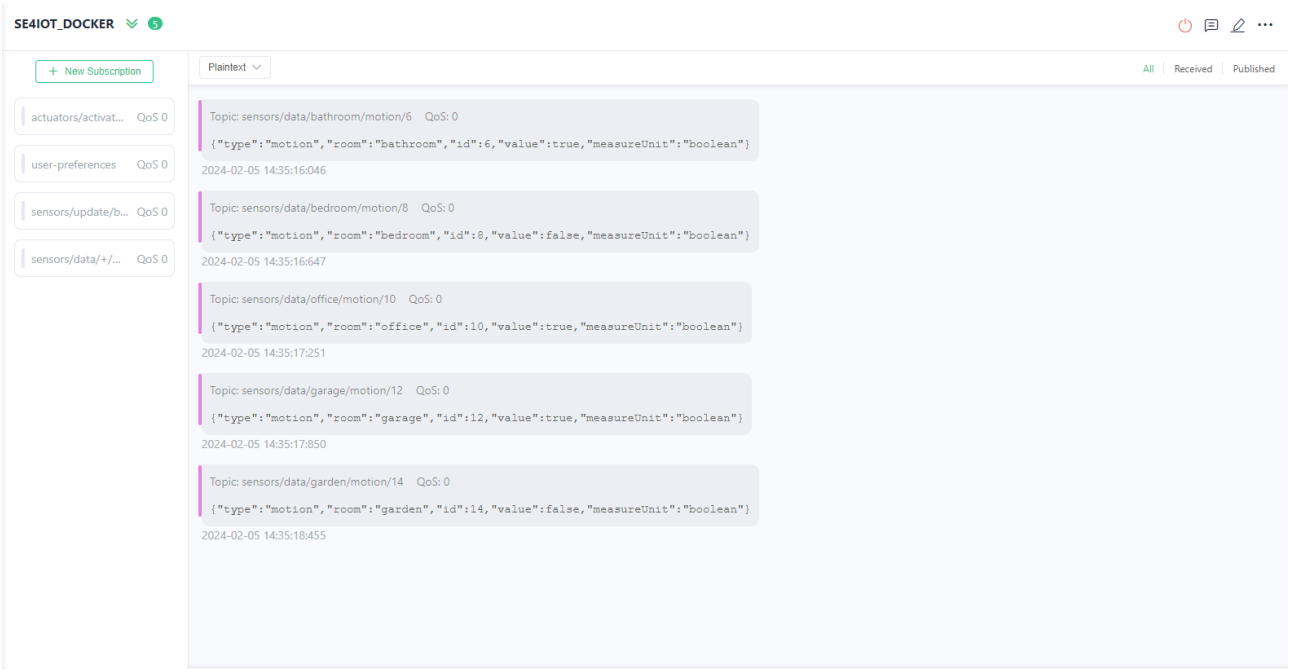
Addressing requirements

Real time monitoring

The figure below depicts the values of light intensity (lux) in the room which have been published by the sensor to the Mosquitto MQTT Broker.

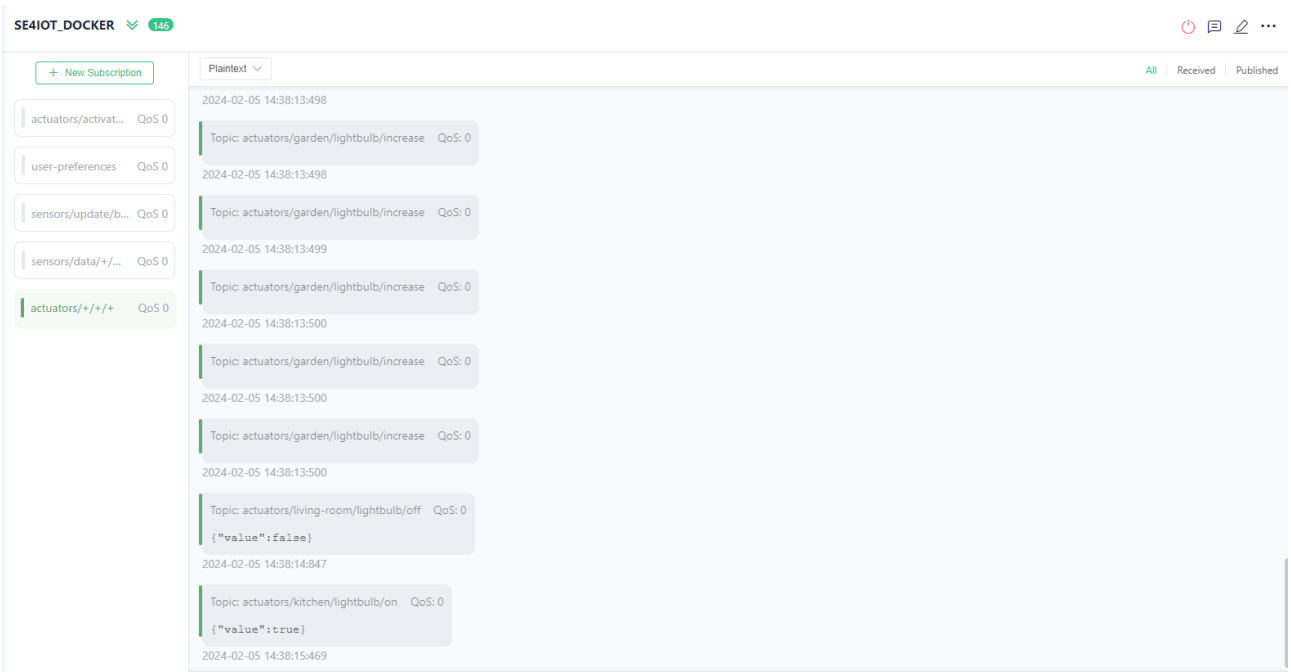


The figure below depicts the values of motion (boolean) in the room which have been published by the sensor to the Mosquitto MQTT Broker.



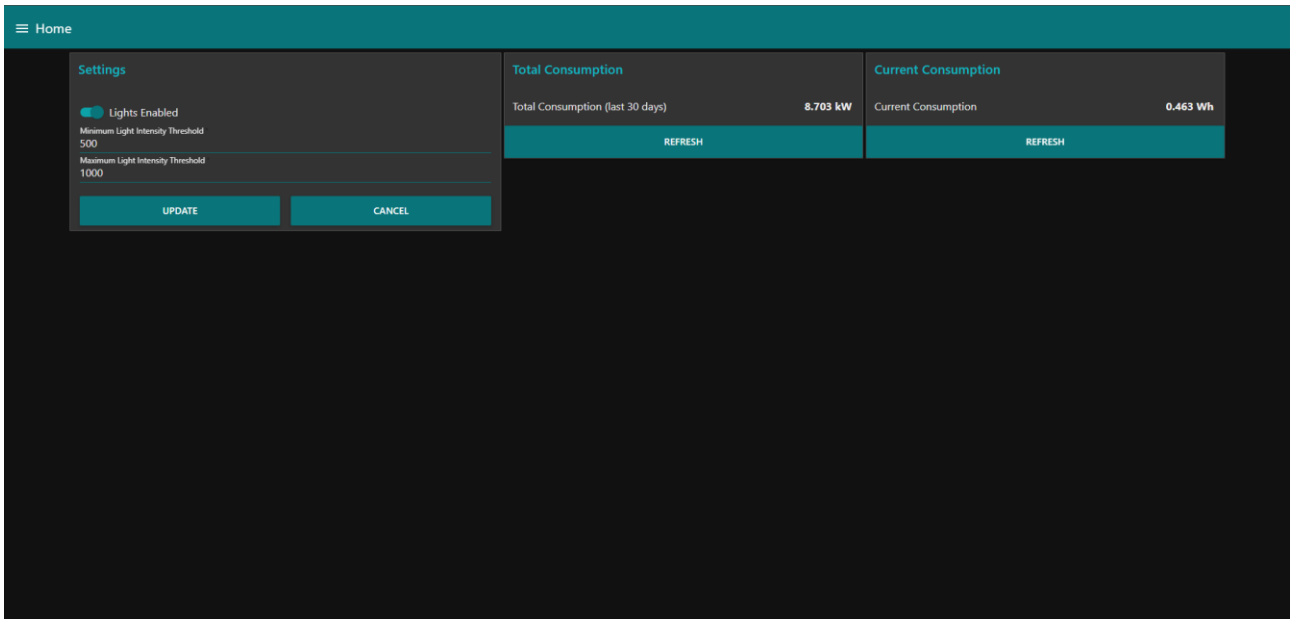
Automated Lighting Control

The figure below depicts the actions taken by the Central Control System after it has evaluated the sensors data.

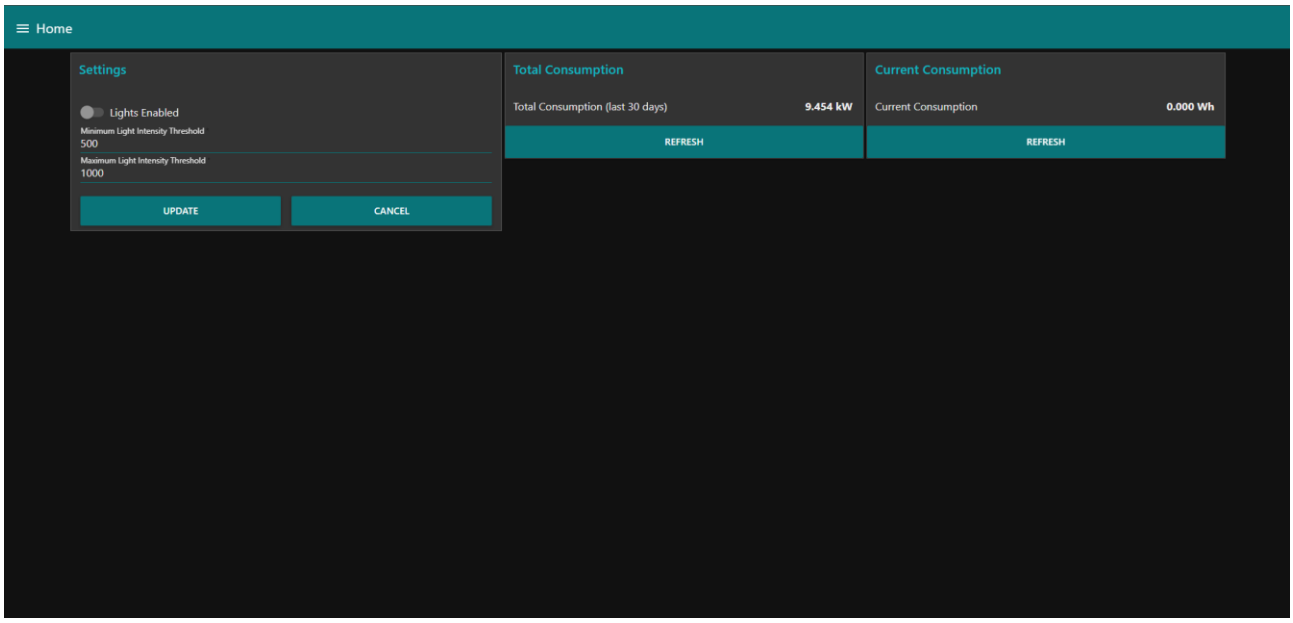


User Interface

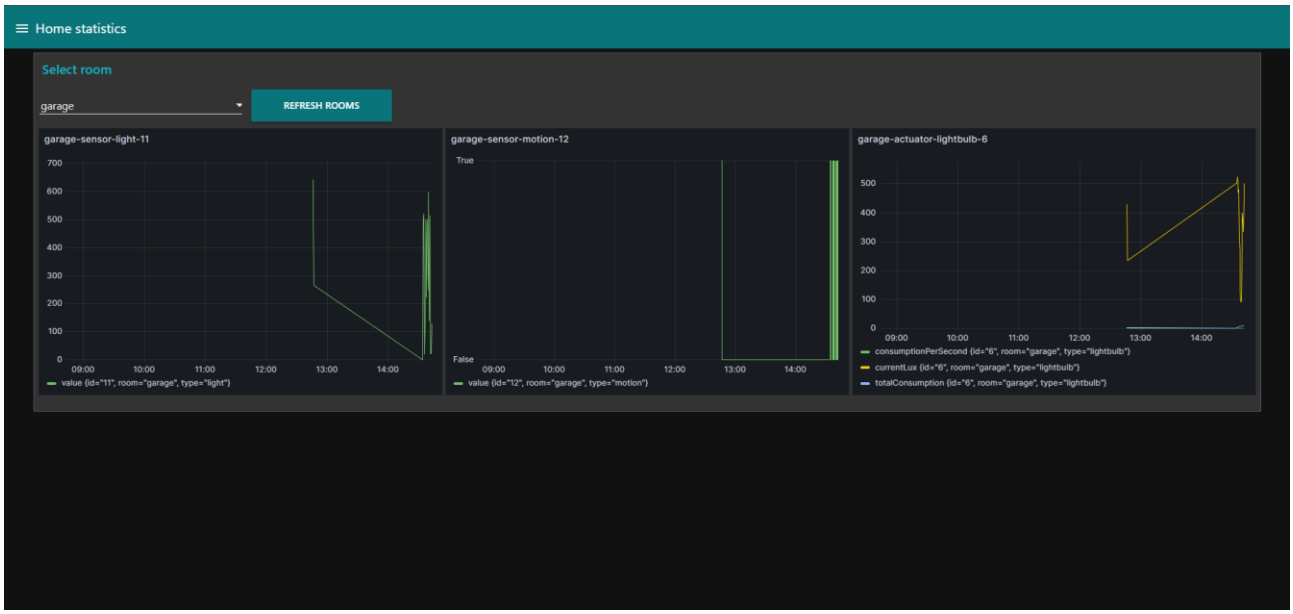
The UI provides real-time status updates, enables users to monitor the system, adjust lighting settings, and access energy consumption information.



When the lights are disabled, we can observe that the current consumption is equal to 0.

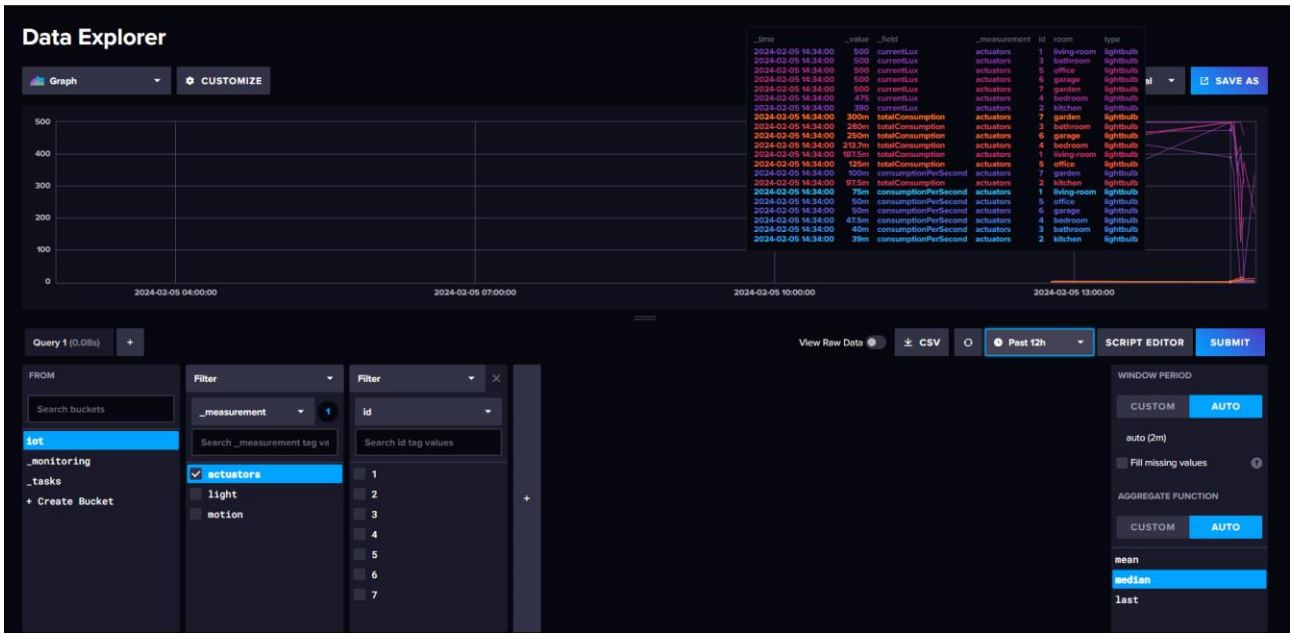


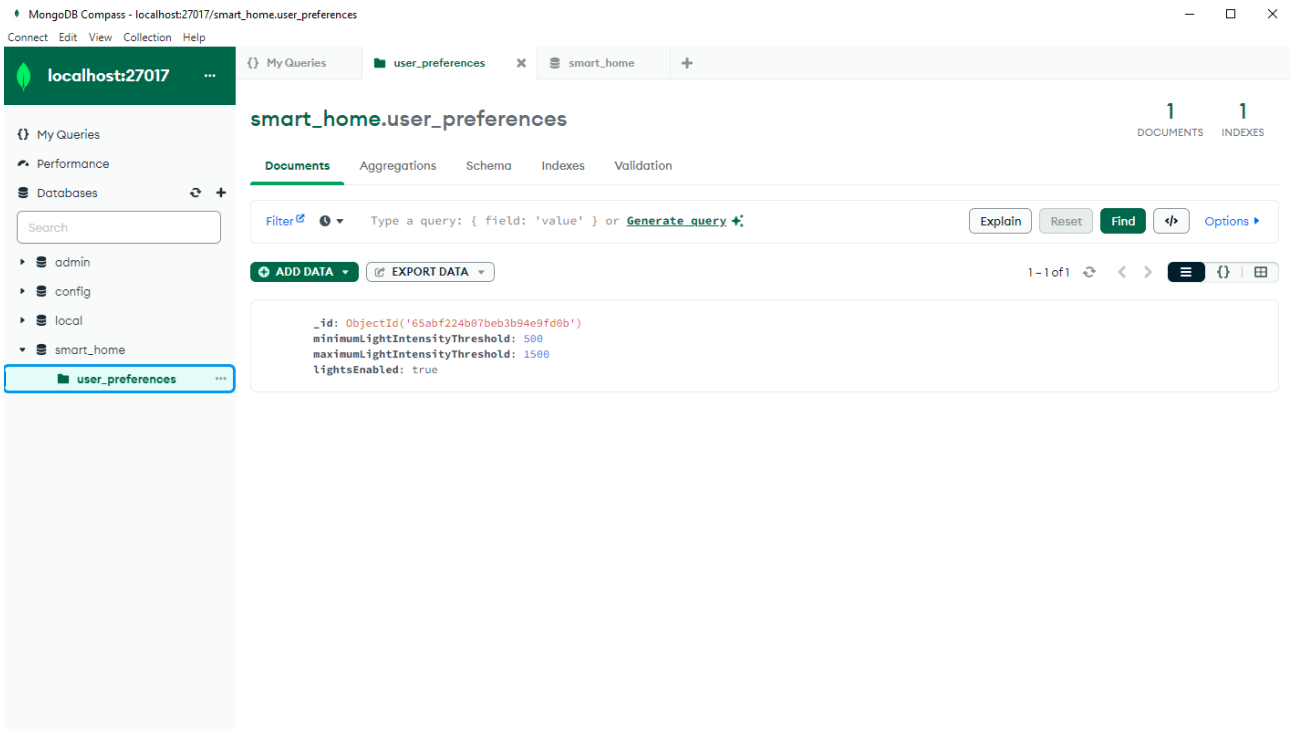
On the "Home Statistics" page, users can visualize sensor and actuator data for a specific room. This information is presented through an iframe, which loads a graphic from Grafana.



Data Storage

The system stores the sensors and actuators data on Influxdb and the user preferences on MongoDB.


Data Explorer
Graph
CUSTOMIZE
SAVE AS
2024-02-05 04:00:00
2024-02-05 07:00:00
2024-02-05 10:00:00
2024-02-05 13:00:00
Query 1 (0.08s)
View Raw Data
CSV
Past 12h
SCRIPT EDITOR
SUBMIT
FROM
Search buckets
iot
_monitoring
_tasks
+ Create Bucket
Filter
_measurement
Search _measurement tag va
actuators
light
motion
Filter
id
Search id tag values
1
2
3
4
5
6
7
WINDOW PERIOD
CUSTOM
AUTO
auto (2m)
Fill missing values
AGGREGATE FUNCTION
CUSTOM
AUTO
mean
median
last


MongoDB Compass - localhost:27017/smart_home.user_preferences
Connect Edit View Collection Help
localhost:27017
My Queries
Performance
Databases
Search
admin
config
local
smart_home
user_preferences
My Queries
user_preferences
smart_home
smart_home.user_preferences
1
DOCUMENTS
1
INDEXES
Documents
Aggregations
Schema
Indexes
Validation
Filter
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
1 - 1 of 1
_id: ObjectId('65abf224b07beb3b94e9fd0b')
minimumLightIntensityThreshold: 500
maximumLightIntensityThreshold: 1500
lightsEnabled: true