Internet of Things project

“Greenhouse+”

Arduini Luca
Bruni Davide

A.Y. 2022-2023

The code is available on GitHub at:
`https://github.com/DavideBruni/SmartGreenHouse`

Contents

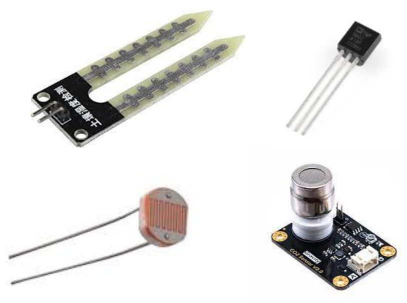
Introduction

Greenhouse+ is an IoT system for the automation of frequent procedures in a smart greenhouse.
In this project, we leverage smart sensors to monitor and optimize four key factors within the greenhouse: light intensity, temperature, soil moisture, and CO_2 levels. Through the use of advanced sensor technology our goal is to enhance the efficiency, productivity, and sustainability of greenhouse cultivation.

In addition to sensor monitoring, our project also incorporates actuator control to modify the environmental conditions within the greenhouse. We utilize three actuators: artificial lighting, windows for ventilation, and sprinkler for a precision irrigation system. These actuators are strategically controlled based on sensor readings to maintain the desired conditions for plant growth.

Our project has a Remote Control Application that manages all the logic to map value provided by the sensor to the action that will be performed eventually by actuators. Moreover the Remote Control Application allows users to have the flexibility to activate the actuators when they deem it necessary.

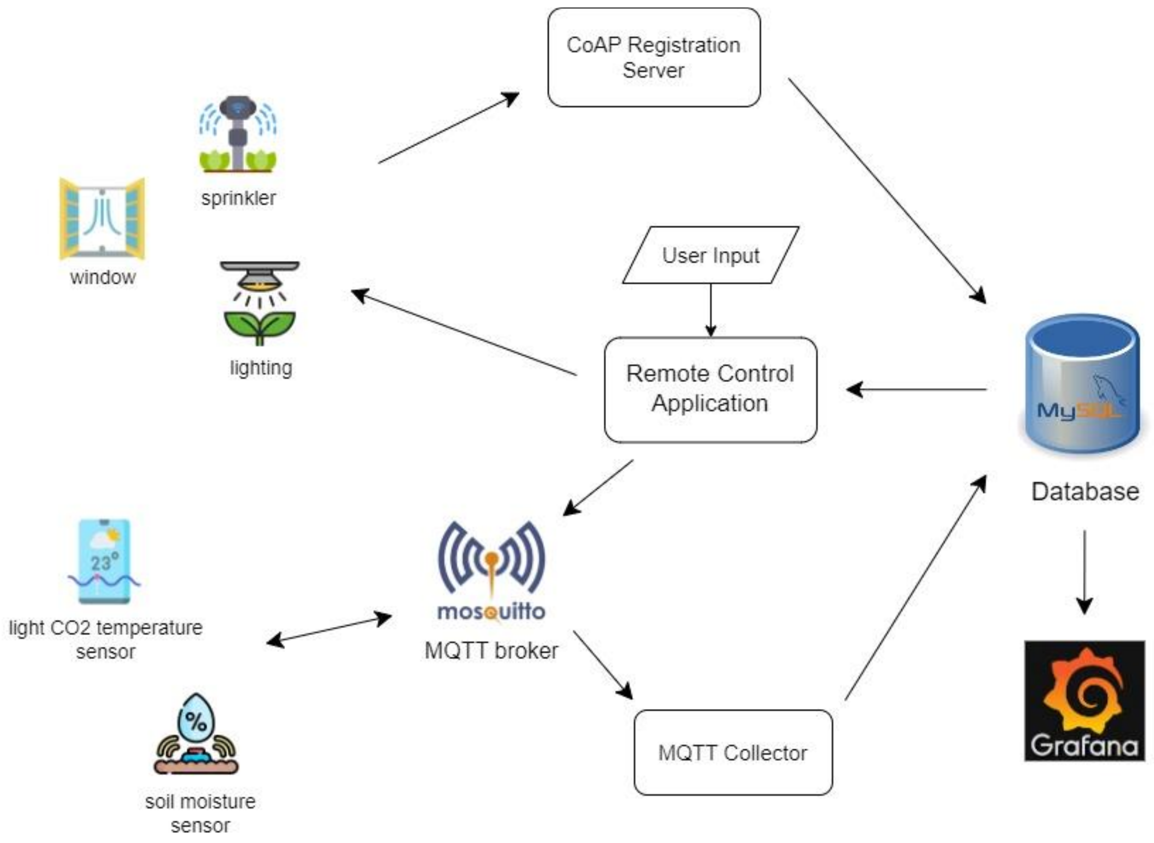
Sensors and actuators are simulated, and we have chosen to simulate the following devices:



- **Sensors**
  - **Soil humidity sensor,** strategically placed near the plant roots to monitor the moisture levels in the soil.
  - **CO_2 sensor,** placed inside the Greenhouse
  - **Temperature sensor,** placed inside the Greenhouse
  - **Photoresistor**, positioned at the top of the plants to monitor the light intensity
- **Actuators**
  - **Sprinkler,** used to have a precision irrigation system
  - **Window,** used to change the CO_2 value and the temperature inside the Greenhouse

Architecture

The system architecture is depicted in the following diagram, intentionally excluding the RPL border-router to enhance clarity.



Essentially the network can be divided into three parts: the MQTT network, the CoAP network and a backend system.

MQTT network

A fundamental part of the MQTT network are the sensors. In our current example scenario, we have considered the presence of only one sensor per type. However, it is important to note that in practical applications, we anticipate having multiple instances of each sensor. Depending on the scale of the agricultural setup, it is common to deploy tens or even hundreds of these sensors throughout the greenhouse area.

Within the MQTT network, there is also the MQTT broker, which is responsible for routing messages between sensors and the MQTT Collector/the Remote control applications by managing topic subscriptions. In this project, we use Mosquitto, an open-source message broker that implements the MQTT protocol.

The MQTT Collector, of which we will discuss shortly, is also part of this network since it reads the data published by the sensors and store them on a database.

CoAP network

The CoAP network consists of actuators that utilize CoAP to receive commands from the Remote Control Application, to expose a resource and to send a registration message upon joining the network. The registration message is useful for implementing a Discovery Procedure: once the registration occurred, the other component of the system can retrieve IPv6 address and resource exposed by the actuator by reading the database.

The CoAP Registration Server is of course an integral component within this network.

Similar to the sensors, for the purpose of this presentation, we have opted to showcase a single instance of each actuator type.

Backend system

The backend system consists of various entities that make the system work.

In the **MQTT Collector** we have a MQTT client and its role is to collect data transmitted by sensors within the MQTT network. To achieve this, the application subscribes to all the topics where the sensors publish their messages. When a new publication is received, the Collector interprets it and insert the new data into the database.

The **CoAP Registration Server** is responsible for registering nodes within the CoAP network in the database. In fact, the actuators are programmed to send a message to this server as soon as they join the RPL DODAG created by the border router. After that, the server will proceed to register their presence in the database through a specific query.

The **Remote Control Application** is the brain of our system and performs multiple functions.
Periodically, it checks if new data has been published by the sensors in the database and interprets them accordingly. If these values exceed user-defined threshold values, it proceeds to send commands (using CoAP) to the actuators to bring these values back to normal.
Moreover, this component is designed to allow user interaction. Users can modify system parameters through specific commands, and these updated parameters are transmitted to the sensors using MQTT.
Additionally, the Remote Control Application enables users to directly send commands to the actuators.
It's important to highlight that, since the application knows the status of the actuators, it avoids sending useless commands, for example "open the window" if the window is already open, in order to reduce the traffic on the network.

We utilize a MySQL **database** to store the data captured by the sensors, which are then transmitted to the database by the MQTT Collector. Additionally, the database maintains a record for each actuator, ensuring an up-to-date repository of currently accessible actuators for interaction with the Remote Application.

In the end we have **Grafana**, an open-source data visualization and monitoring platform that allows users to visualize and analyze data collected by the sensors. We will talk more about this component later.

Data encoding

In resource-constrained IoT scenarios, the efficiency of the chosen protocol is crucial. Since we haven't constraints that force us to use XML (there aren't other devices already deployed to interact with using XML), we opted to use JSON, which offers a simpler and more lightweight syntax compared to XML. As a result, JSON generally yields smaller data payloads since it utilizes concise syntax without the inclusion of verbose tags and attributes found in XML. This characteristic leads to reduced bandwidth usage and reduce the resource used by the sensor for computing the packet.

Deployments

Sensors

Our sensors publish sensed value on topic: "sensor/*x*", where *x* is the kind of value sensed.
We have the following topics:

- ***sensor/humidity***
- ***sensor/co2_light_temp***

About the "sensor/co2_light_temp" topic: for the reasons mentioned below, we opted to consolidate three distinct sensed values into a single topic instead of assigning a separate topic for each value. Since these values are transmitted by the same dongle and, unless they fall outside the specified range, they are published together once every hour, we prefer sending a slightly longer message as opposed to three separate ones, aiming to minimize energy consumption.

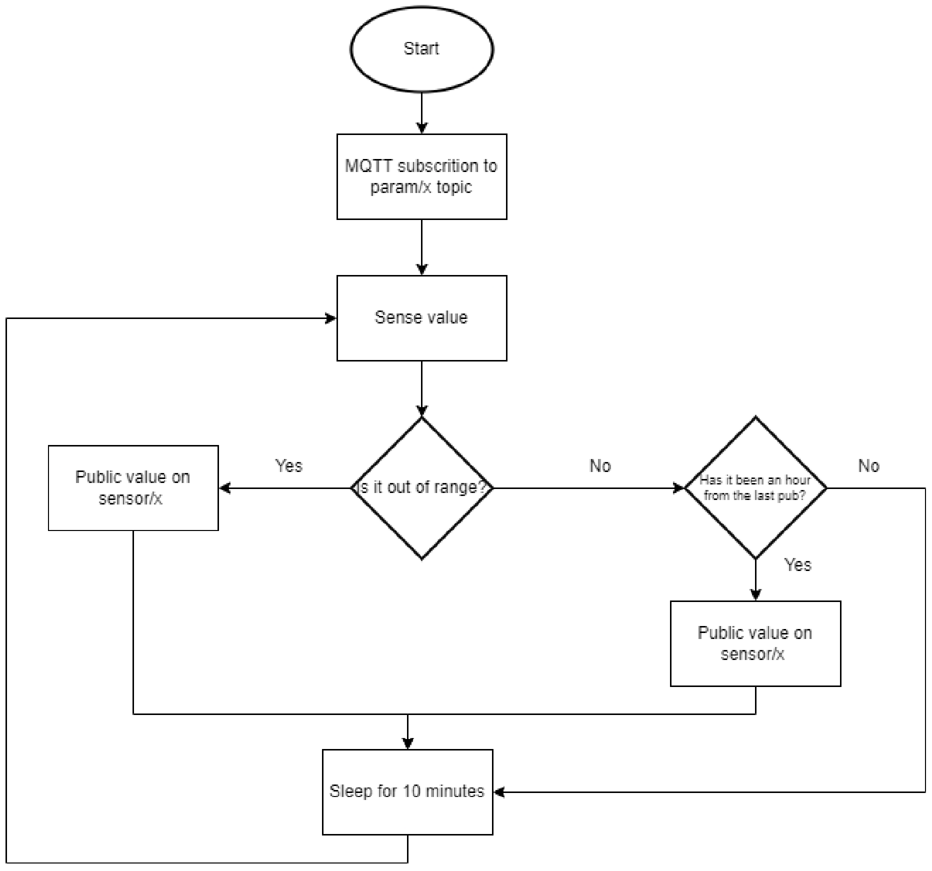
But how can we say that a value is out of range in order to public the value in case of "emergency"? I.e. how can we determine if a value is above the *max_value* or below the *min_value*? The sensors have a predefined max/min threshold. However, this threshold can be modified from CLI of the Remote Control Application. For this reason we need other topics to whom sensors must subscribe:

- **param/humidity**
- **param/co2**
- **param/temp**
- **param/light**

In the param/x topic the min and max value are published except for:

- ***CO_2***: we supposed that, in a greenhouse, we don't have problem of high CO_2 levels
- ***Light***: there's another parameter called "*is_night*". If the value of this parameter is equal to 1, lights cannot be turned on, whatever it takes and moreover, light sensor will not publish anything if the light value is under the min threshold value. This parameter is introduced in order to simulate the natural light cycle that plants must respect. This parameter can be set by the user through the CLI on the remote control application.

During their execution, sensors perform the following actions:



We need to make a few consideration:

- As we can see from the flowchart above, sensors sample values every 10 minutes, but if these values are not out of range, they are not published. This approach is aimed at saving energy: the quantity measured by the sensors, doesn't change in a fast way, so we can reduce the traffic.

- as we can see from the flowchart, the sensors public values only under two different conditions:
    - value is out of range, and this is necessary in order to activate actuators and change the actual greenhouse conditions; or
    - it's been an hour from the last not "out-of-range" publication. This is due in order to have a general monitoring of the Greenhouse: all the measured quantities doesn't change rapidly, so we can avoid to send frequent messages and save some energy.
- about the *humidity value*: if it's out of range, the sensor goes to an "ALERT STATE", it implies that the sleeping time change from 10 minutes to 2 minutes. Why? Because, in order to increase the field humidity, the sprinkler actuator is on, and we want to avoid that the field becomes too wet or, worse, flooding the Greenhouse! (In our deployment, in order to be able to test our environments, 1 hour has been modified with 45 seconds, 10 minutes with 15 seconds and 2 minutes with 7 seconds).

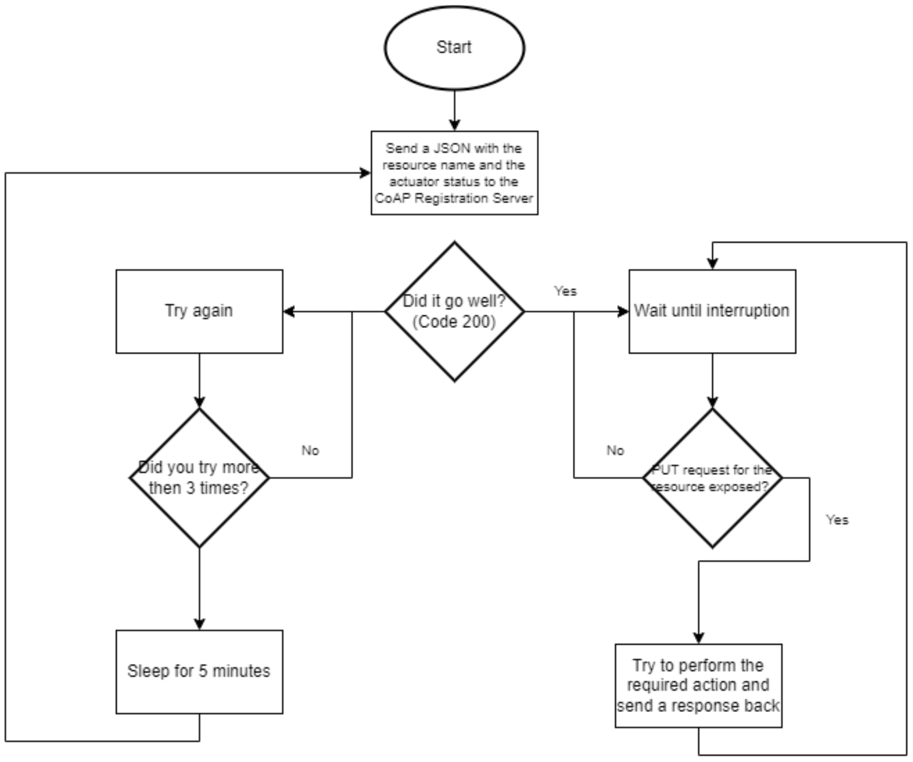
In order to simulate value out of range, we use the button placed on our Nordic Semiconductor nrf52840 dongle:

- *Humidity sensor node*: if pressed, the button start simulating value under the min_value
- *Co2_light_temp node*:
    - *1st pressure*: only temperature value generated are out of range
    - *2nd pressure*: only light value generated are out of range
    - 3rd pressure: only CO_2 value generated are out of range
    - 4th pressure: all values generated are in range

Actuators

In order to be reachable from the remote control application (which at the start doesn't know the IPv6 address of the actuators), they must registrate themselves with a CoAP Registration Server. Each actuator sends a JSON containing the name of the resource exposed and the state of the actuators itself. The CoAP registration server then will store these information and the IP address of the actuator in a MySQL database.

During their execution, actuators perform the following actions:


Start
Send a JSON with the resource name and the actuator status to the CoAP Registration Server
Try again
Did it go well? (Code 200)
Yes
Wait until interruption
No
Did you try more then 3 times?
No
PUT request for the resource exposed?
Yes
Sleep for 5 minutes
Try to perform the required action and send a response back

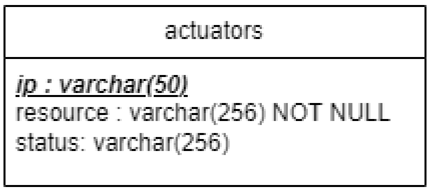
What actuators can do?

- ***Sprinkler actuator***: used for increasing the field humidity. The action accepted are “on”/”off” in order to turn on/off the sprinkler (this situation is simulated using the LEDS: green if the sprinkler is turned on, red otherwise).
- ***Light actuator***: used for changing the light intensity or for changing the temperature if the window cannot be opened because of CO_2 level constrained (i.e. the CO_2 level is low and the window cannot be closed, so in order to increase the temperature, we increase the light intensity). The action accepted are “up”/ “down” in order to increase or decrease the light intensity and “off” in order to turning off the lights. This situation is simulated using the LEDS and 3 state: yellow for low intensity, green for high intensity and turned off if the light must be turned off).
  If the parameter “is_night” is set and an “up” command arrives, the actuator can’t turn on the lights.
- ***Window actuator***: used for changing the CO_2 values or for changing the temperature. The action accepted are “open”/ “close” in order to open/close the window (this situation is simulated using the LEDs: green if the window is open and red if it is closed).

Actuators send back a response to the Remote control application that can contain the following CODEs:

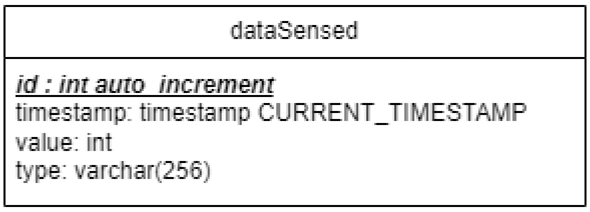
- *CHANGED_204,* if the required action was performed correctly
- *BAD_OPTION_402,* if the request contain an unknown action
- *BAD_REQUEST_400, if the request does not contain any action*

Database schema



actuators

ip : varchar(50)
resource : varchar(256) NOT NULL
status: varchar(256)



dataSensed

id : int auto_increment
timestamp: timestamp CURRENT_TIMESTAMP
value: int
type: varchar(256)

The ***actuators*** table is populated by the Registration Server to store actuators information, which will be retrieved by the Remote Control Application to communicate with them.
The ***dataSensed*** table instead, is populated by the MQTT client on the MQTT Collector in order to store data published by the sensors. These values will be retrieved:

- by Grafana, to show the status of the greenhouse through several plots
- by the Remote Control Application to implement the application logic and decide whether to activate the actuators.

Remote Control Application CLI

The remote Control Application provides a CLI useful to interact with the system:

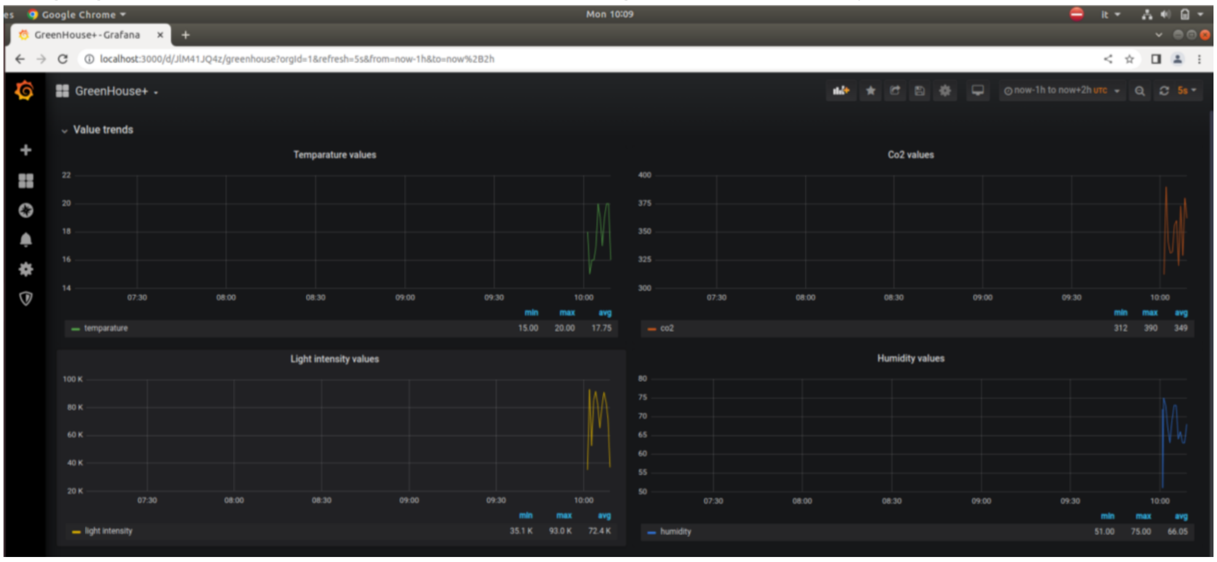
- ***\action:*** open the menu to send a command to an actuator
  - ***\window_open:*** open the window
  - ***\window_close:*** close the window
  - ***\sprinkler_active:*** active the sprinkler
  - ***\sprinkler_deactivate:*** deactivate the sprinkler
  - ***\light_up:*** increase the light brightness
  - ***\light_down:*** decrease the light brightness
- ***\change_params:*** open the menu to change sensors parameters
  - ***\min_light_parameter***: change the minimum value of the brightness

  - ***\max_light_parameter:*** change the maximum value of the brightness
  - ***\light_hours:*** change the interval in which there will be light on
  - ***\min_co2_parameter:*** change the minimum value of the CO_2 inside the greenhouse
  - ***\min_temp_parameter:*** change the minimum value of the temperature inside the greenhouse
  - ***\max_temp_parameter:*** change the maximum value of the temperature inside the greenhouse
  - ***\min_humidity_parameter:*** change the minimum value of the field humidity
  - ***\max_humidity_parameter:*** change the maximum value of the field humidity
- ***\set_day_mode:*** the light sensor will report anomaly if needed
- ***\set_night_mode:*** light will be turned off immediately and the light sensor will not report anomaly
- ***\show_actuators_status:*** show the current status of the available actuators

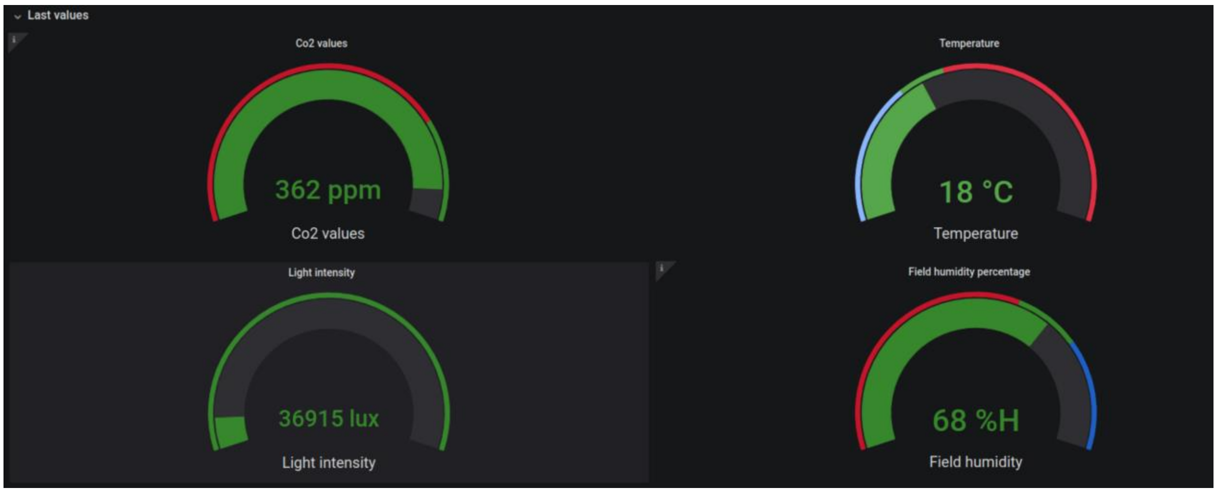
Grafana

Our graphical interface has two section:

- *Value trends*: for each quantity measured, it shows the line plot of the last 4 hours values. It highlight also the min value, the max value and avg value in that time period.


GreenHouse+ - Grafana
localhost:3000/d/JIM41JQ4z/greenhouse?orgId=1&refresh=5s&from=now-1h&to=now%2B2h
GreenHouse+
Value trends
Temparature values
Co2 values
Light intensity values
Humidity values
temparature min 15.00 max 20.00 avg 17.75
co2 min 312 max 390 avg 349
light intensity min 35.1 K max 93.0 K avg 72.4 K
humidity min 51.00 max 75.00 avg 66.05

- *Last value*: for each quantity measured, it shows the last value sensed, highlighting if it's out of range or not.



Use case

Let's explore some practical examples of what can occur within our smart greenhouse.
When the system is operative, all the sensors will be sampling data, and all the available actuators ready to be activated will be visible from outside the network (and they are also present in the actuators table of the database).

Irrigation

As is well known, plants continuously absorb water from the soil for nourishment, causing the soil moisture to decrease. When the dedicated sensor samples a moisture level is lower equal than the user-set minimum threshold, it immediately records the detected value in the database without any constraints on data timing.
The Remote Control Application, which constantly monitors the database, notices the new publication and compares the new value with the threshold value, determining the need to activate the irrigation system. An activation command is then sent to the sprinkler which, by watering the soil, will slowly raise the humidity value of the soil.

The objective is to irrigate the soil until it reaches a second threshold value. To achieve this, the humidity sensor, after sending the initial alert signal, will increase its sampling frequency beyond the standard rate to prevent overwatering the plant. All the values recorded during this "alert phase" are transmitted to the database without undergoing the usual time check for outgoing messages.

When a sampling exceeds the user-defined maximum humidity value, the sensor will send a final alert message before resuming sampling at the pre-established frequency. Upon detecting this new value, the Remote Control Application will send a command to deactivate the sprinkler.

Environmental conditions

The data monitored by the second type of sensors initiates procedures that are similar to those introduced for humidity levels, with some obvious differences:

- In the case of a low CO_2 value, the greenhouse window will be opened to ventilate the area and restore normal CO_2 levels.
- If the temperature drops below a dangerous threshold, the window will be immediately closed if it is open or the light intensity is increased if the window cannot be closed due to CO_2 value constraints.
- If there is a shortage of light due to cloudy skies, the artificial lighting system will be turned on. This system has two levels of brightness and will be adjusted based on the current amount of light.
  In both cases the light should be turned on, there's a check if the day_mode is set or not.

There is another significant difference. Unlike the soil moisture, which varied rapidly due to artificial irrigation, these other quantities do not change at the same speed. Therefore, in these cases, it was decided not to increase the sampling frequency, but the corresponding actuator will be turned off when an out-of-range value is detected.