UNIVERSITY OF PISA

Internet of Things - Academic Year 2022/2023

SMARTWINEMAKING

Students:

Emmanuel Piazza

Edoardo Focacci

Table of Contents:

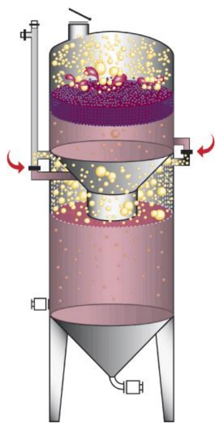
Introduction 3
Architecture 4
  CoAP Network 4
    Bypass 4
    Cooling 4
  MQTT Network 5
    Temperature 5
    Float 5
    Co2 6
Database 7
Data Encoding 8
Collector 8
  CLI 8
Grafana 9

Introduction

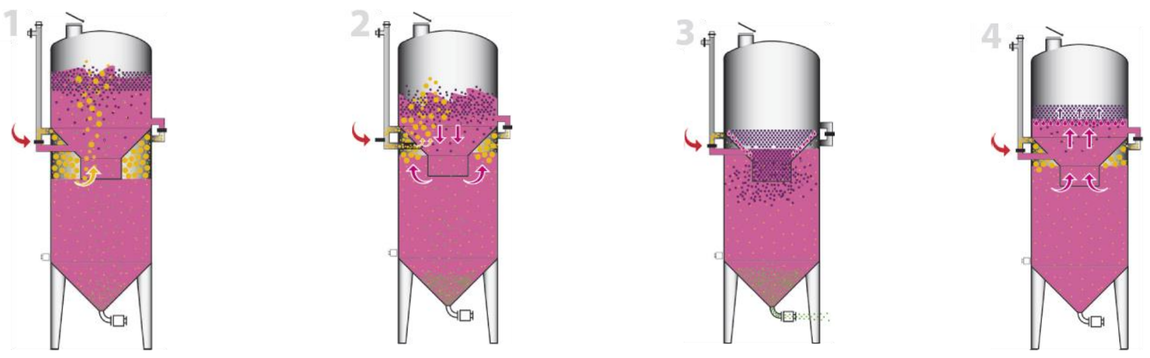
Alcoholic fermentation is the primary stage of wine production where the sugar in the grape juice undergoes transformation into alcohol. When yeast comes into contact with the sugar in the grape juice, it metabolizes the sugar through a series of chemical reactions. The yeast breaks down the sugar molecules into alcohol (ethanol) and carbon dioxide as byproducts. This conversion process is known as fermentation.

In our case the carbon dioxide is released in the air and utilized to mix the must, while in other case this gas can be utilized to make the wine sparkling.

The specific method utilized in this example of "*smart fermente*r" is the one called ***Ganimede***. The method is explained there:



The fermenter is composed by two cavities, linked by an open inverted cone where the liquid can pass through. When the must (in the beginning present only in the cavity below) starts to ferment CO2 and ethanol is produced, CO2 is free to go in the air and is lighter than the air, so some of it goes in the central hole, some of it is instead blocked on the sides of the cone where two valves are closed. The pressure and the adding of other must increase the float level of the must till the must itself goes over the central hole (in the cavity below some CO2 is still trapped on the side, adding pressure to the surrounding liquid). When the level is high the two valves (called bypass) are opened and the must is violently mixed with itself, favoring the deposit of some impurity materials at the bottom of the fermenter where it can be disposed later.



The fermenter obviously must have a method to open the bypass, and some sensors to evaluate the quality of the must and the process of fermentation itself. The application in order to do so has to have the meanings to measure temperature, float level and co2 level in the air, and the capacity to open the bypass and activate a cooling system to maintain a good quality of the fermentation.

Architecture

The system is composed of two different IoT device networks. One network is composed of IoT devices that use MQTT to report data, while the other network uses CoAP as application protocol.

Both MQTT and CoAP network is deployed using real sensors from the IoT testbed. Both networks are connected to a border router that allows them external access.

In this project were used 3 sensors (*Co2, Temperature* and *Float*) and 2 actuators (*Bypass* and *Cooling*).

The collector collects data from both MQTT and CoAP sensors and stores them in a MySQL database. The collected data can then be visualized through a web-based interface developed using *Grafana*.

A simple control logic is executed on the collector to modify the external environment based on the data that has been collected from temperature and float sensors. The collector also exposes a simple command line interface through which the user can retrieve the last measured values or change the ranges within the actuators will turn on or off.

CoAP Network

Bypass

This sensor is used to move up or down the cork linked to the barrel. This cork is used to make the air change inside the barrel and to make progress on the vinification.

By using the data received from the *Float* sensor and by checking the bounds defined by the user, the collector can assess whether it is necessary to move the cork.

The cork will be moved **UP** if the float level of the wine of the barrel surpasses a defined percentage, meanwhile it will turn **DOWN** if the float level returns on a stable percentage, still defined by the user.

To visualize the movement of the cork, a message is printed inside the console to show the action sent to the actuator.

Cooling

This sensor is used to turn on or off the hydraulic cooling system built outside the barrel.

By using the data received from the *Temperature* sensor and by checking the bounds defined by the user, the collector can assess whether it is necessary to activate the cooling system.

The Cooling system will be **ON** if the inner temperature of the barrel surpasses a defined bound, meanwhile it will turn **OFF** if the temperature's registered values return stable and are lower that a defined bound.

To visualize the state of the actuator, a message is printed inside the console to show the action sent to the actuator.

MQTT Network

Temperature

The inner temperature of the barrel is a crucial element to take care of during the vinification process. Because of that, a temperature sensor is installed inside the barrel to check the value's variations during the whole process.

The sensor's data variation is simulated in order to reflect a real wine barrel's behavior. The sensor's value is increased randomly at every loop cycle by 0° or 1°. The variation is also affected by the *Cooling system* status.

This sensor is subscribed to the *cooling* topic. When the coordinator receives a temperature's value that goes higher or lower the defined bounds, it proceeds to send a message for the *Cooling* actuator; this message decides the action taken to maintain the temperature between the defined bounds. When the actuator receives the message (*ON/OFF*) it publishes the action done inside the *cooling* topic in order to let the sensor know its current status.

If the Cooling system is ON, the temperature's variance will always be negative. On the opposite, if the Cooling system is OFF, the temperature's variance will always be positive.

This was implemented for having a better simulation behavior between these two elements.

Float

The float level of the wine inside the barrel is another element to take care of during the vinification process. A float sensor is installed inside the barrel to check the float level variations.

The sensor's data variation is simulated in order to reflect a real wine barrel's behavior. The sensor's value is increased at every loop cycle by a percentage value. The variation is also affected by the *Bypass system* status.

This sensor is subscribed to the *bypass* topic. When the coordinator receives a float value that goes higher or lower the defined bounds, it proceeds to send a message for the *Bypass* actuator; this message decides the action taken to maintain the temperature between the defined bounds. When the actuator receives the message (*UP/DOWN*) it publishes the action done inside the *bypass* topic in order to let the sensor know its current status.

If the cork is UP, the float level will increase by 1%. On the opposite, if the cork is DOWN, the float level variance will decrease by 20%.

The float level can also be manipulated by pressing the button attached to the sensor. When the button is pressed, the float level will go up by 30%.

LEDs are also attached to the sensor. The light will change depending on the current float level: if the level is below 30%, a green light shows up; if the level is between 30% and 60%, a yellow light is turned on; if the level is above 60%, a red light will alert of a critical value reached by the wine.

Co2

This last sensor is used to check the Co2 level in the air near the wine barrel. Because of the continuous release of toxic air from the barrel it is a good use to take care of this element in order to avoid dangerous situations inside the canteen.

The sensor’s data variation is simulated in order to reflect a real wine barrel’s behavior. The sensor’s value is increased at every loop cycle by a percentage value between 1% and 4% and there’s a 50% change of it to be a positive or a negative variation.

The co2 level can also be manipulated by pressing the button attached to the sensor. When the button is pressed, the float level will go up by 30%.

LEDs are also attached to the sensor. The light will change depending on the current co2 level: if the level is below 30%, a green light shows up; if the level is between 30% and 60%, a yellow light is turned on; if the level is above 60%, a red light will alert of a critical value reached by the wine.

Database

Data retrieved from the sensors is stored inside a **MySQL** database called *smart_wine*.

Inside the database different tables are defined:

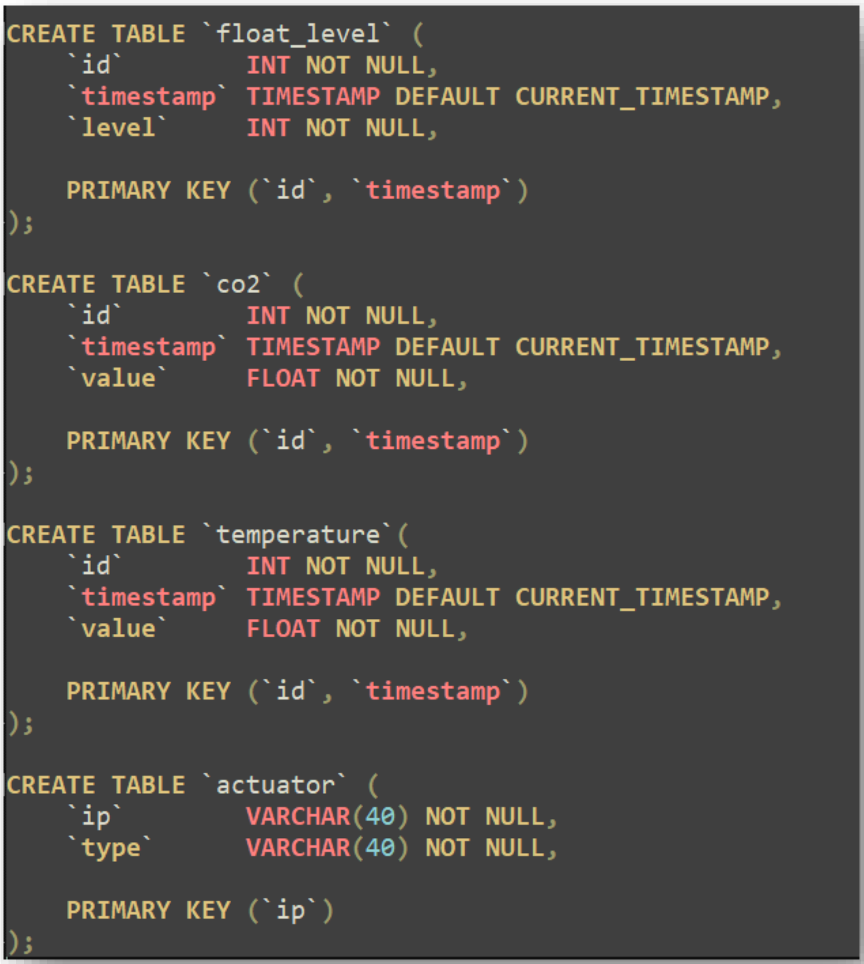
- *temperature*
- *co2*
- *float_level*
- *actuators*

There are no dependencies between the tables.

For each sensor, when a value is stored inside a table, a new entry is created by storing three main values:

- *id* (referring to the sensor that sensed the value)
- *timestamp* (referring to the moment the value was sensed)
- *value* (referring to the actual sensed information; it varies depending on the table)

Inside the *actuator* table are stored the actuators that completed the registration process through the Collector. For each entry the *ip address* and the *type* of actuator are saved inside the table.



```sql
CREATE TABLE `float_level` (
    `id`        INT NOT NULL,
    `timestamp` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    `level`     INT NOT NULL,

    PRIMARY KEY (`id`, `timestamp`)
);

CREATE TABLE `co2` (
    `id`        INT NOT NULL,
    `timestamp` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    `value`     FLOAT NOT NULL,

    PRIMARY KEY (`id`, `timestamp`)
);

CREATE TABLE `temperature`(
    `id`        INT NOT NULL,
    `timestamp` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    `value`     FLOAT NOT NULL,

    PRIMARY KEY (`id`, `timestamp`)
);

CREATE TABLE `actuator` (
    `ip`        VARCHAR(40) NOT NULL,
    `type`      VARCHAR(40) NOT NULL,

    PRIMARY KEY (`ip`)
);
```

Data Encoding

As data encoding, we decided to use the **JSON** format. All the generated data is transmitted to the collector as JSON objects. This choice was taken because the sensors have limited resources, and XML has a too complex structure. JSON is more flexible and less verbose, resulting in less overhead.

Collector

The collector takes care of receiving the data, communicating with the database to save the samples received and decide which actions to take.

It also implements a **CLI** to make it easier for the user to manage. We decided to implement the collector in Java, using the *Paho* and *Californium* libraries.

CLI

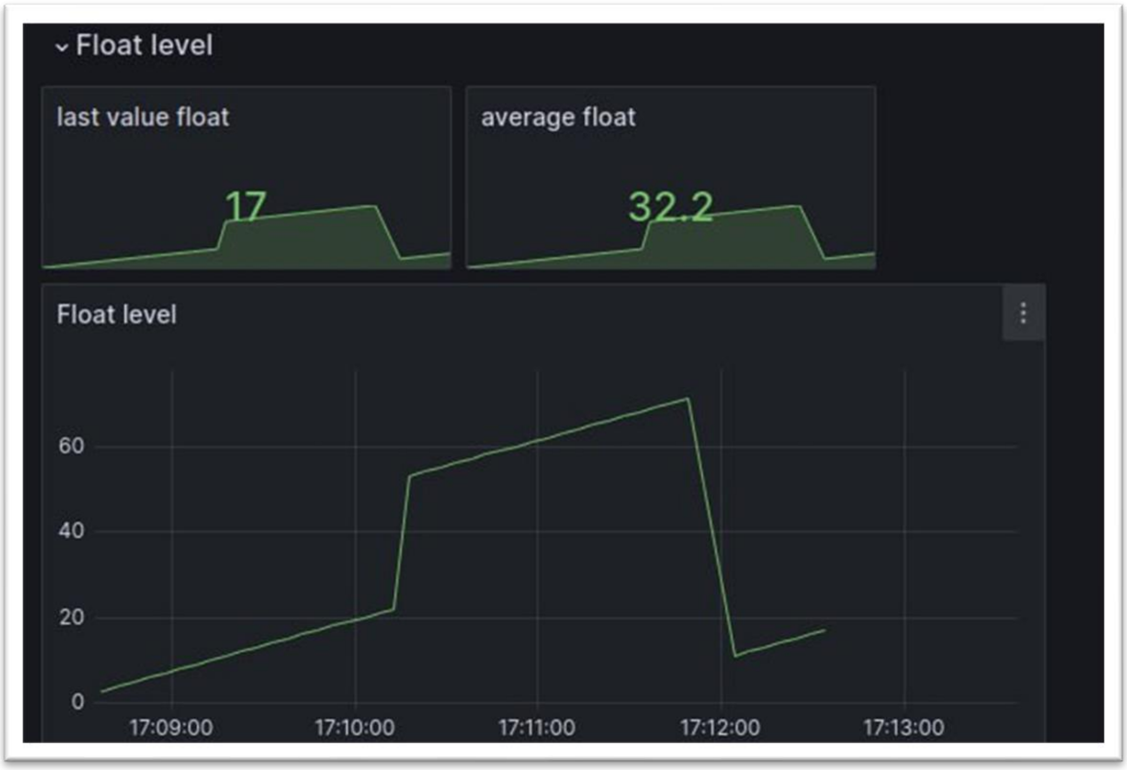
The function available through the CLI are the following:

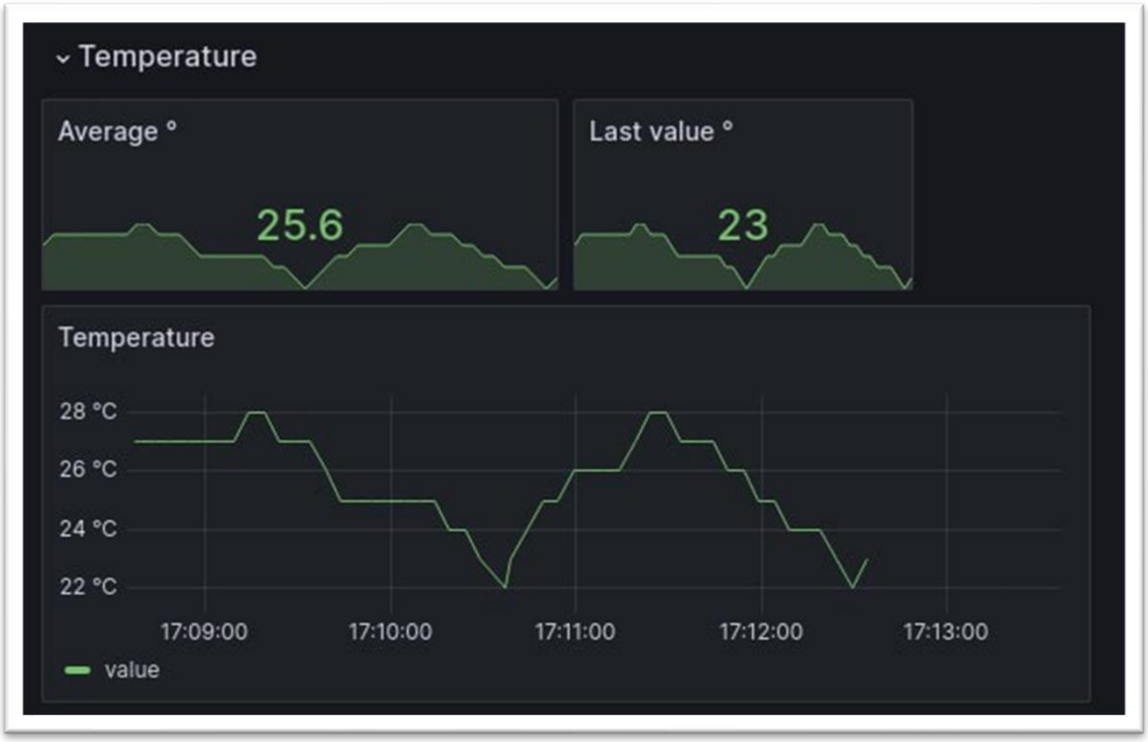
- **set tempLow <integer>**
    - Used for setting the temperature sensor's lower bound. This bound will be checked by the collector before turning OFF the Cooling system.
- **set tempUp <integer>**
    - Used for setting the temperature sensor's upper bound. This bound will be checked by the collector before turning ON the Cooling system.
- **set floatLow <integer>**
    - Used for setting the float sensor's lower bound. This bound will be checked by the collector before turning DOWN the Bypass system.
- **set floatUp <integer>**
    - Used for setting the float sensor's upper bound. This bound will be checked by the collector before turning UP the Bypass system.

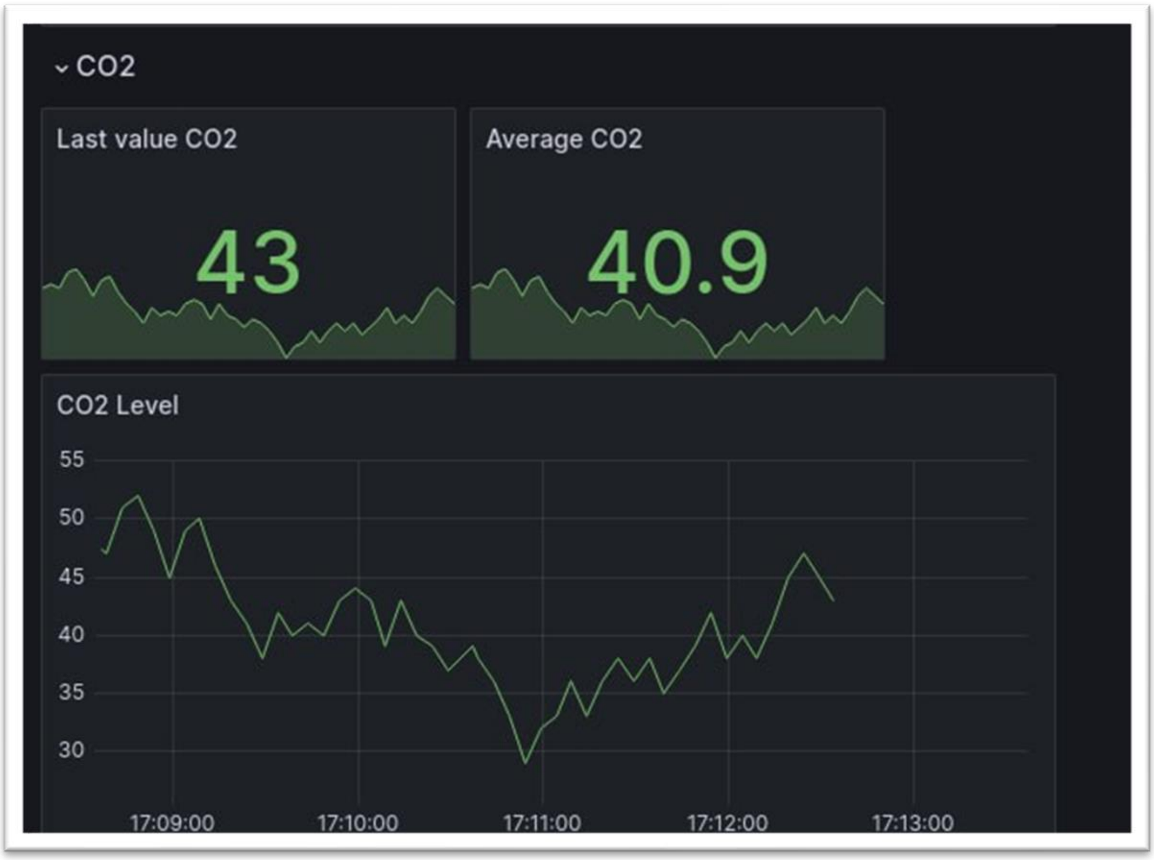
- **get tempBound**
    - Used for checking the temperature sensor's current bounds.
- **get floatBound**
    - Used for checking the float sensor's current bounds.
- **get co2**
    - Used for checking the co2 sensor's last registered value.
- **get temp**
    - Used for checking the temperature sensor's last registered value.
- **get float**
    - Used for checking the float sensor's last registered value.
- **exit**
    - Used for closing the application.

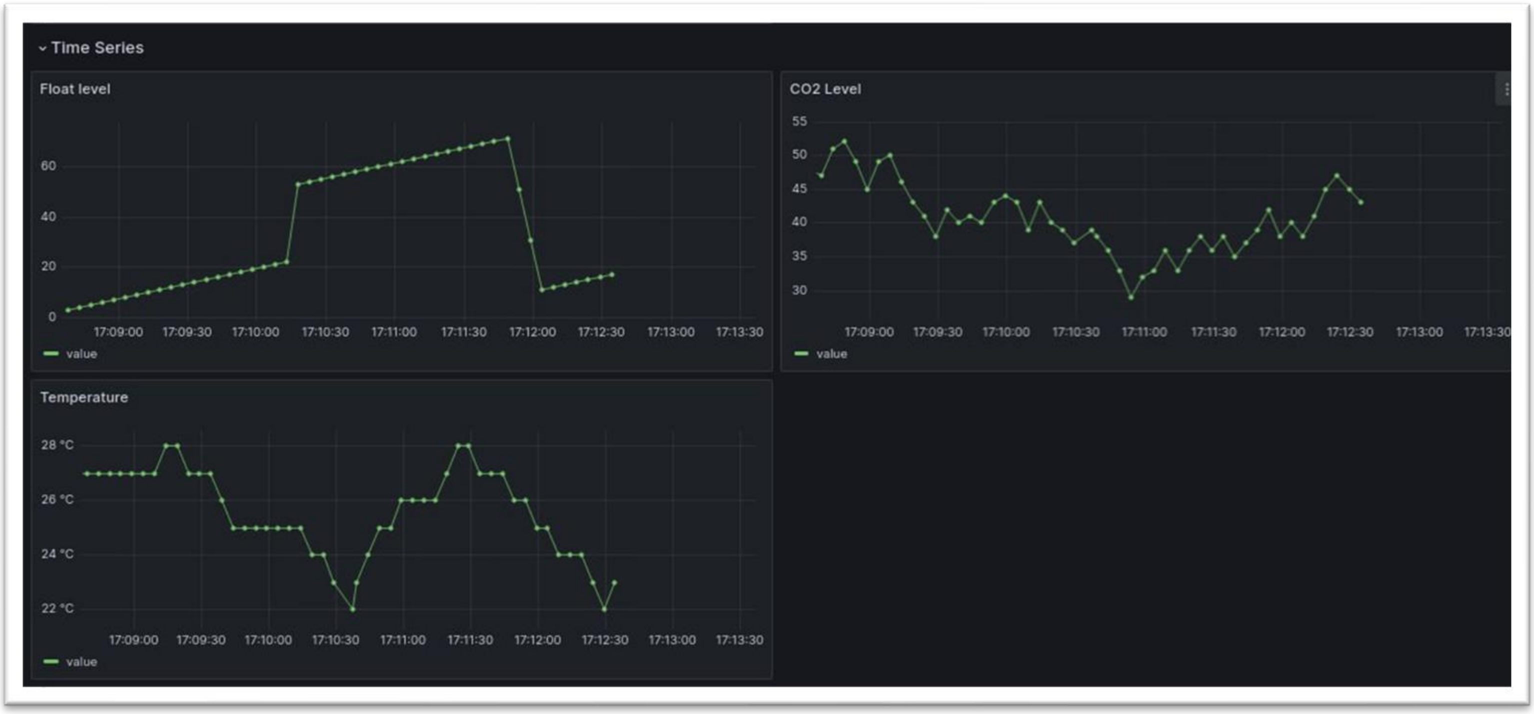
Grafana

Grafana dashboard is used to show analytics and data variation based on the value that are store inside the database. Here's some examples of output data from the sensors:






CO2
Last value CO2
43
Average CO2
40.9
CO2 Level
55
50
45
40
35
30
17:09:00
17:10:00
17:11:00
17:12:00
17:13:00


Time Series
Float level
60
40
20
0
17:09:00
17:09:30
17:10:00
17:10:30
17:11:00
17:11:30
17:12:00
17:12:30
17:13:00
17:13:30
value
CO2 Level
55
50
45
40
35
30
value
Temperature
28 °C
26 °C
24 °C
22 °C
value