IoT board for data visualization in Grafana using MQTT protocol

In the first week I took care of the documentation part for this project, namely about Grafana, the MQTT broker, the ESP8266 board and Node-Red.

The objective for this week was to write a code to be able to collect data from a sensor (temperature and humidity) with an ESP8266 board and send them through the MQTT Broker to Grafana for the data visualization.

Theory:

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for efficient communication between devices, especially in scenarios where bandwidth and power constraints are significant concerns. MQTT is widely used in the Internet of Things (IoT) domain due to its simplicity, low overhead, and ability to work in unreliable and constrained networks.

The ESP8266 is a popular and versatile Wi-Fi microcontroller board designed for Internet of Things (IoT) and embedded systems projects. It is produced by Espressif Systems, a Chinese semiconductor company, and has gained widespread adoption due to its low cost, small form factor, and excellent Wi-Fi capabilities.

Meshlium is a versatile and robust gateway device designed for creating and managing wireless sensor networks in an IoT environment. Meshlium acts as a central hub for connecting and collecting data from multiple Waspmote or other compatible sensor devices deployed throughout an area.

The Code:

```
//the library for the JSON file:
#include <ArduinoJson.h>

//the libraries for connecting to the MQTT server
#include <PubSubClient.h>
#include <ESP8266WiFi.h>

//setting the WI-FI connection
```

```
const char* ssid = "LANCOMBEIA"; // Enter your WiFi name
const char* password = "beialancom"; // Enter WiFi password

//MQTT broker
const char* mqtt_server = "mqtt.beia-telemetrie.ro";
const char* topic = "training/esp8266/StanescuVlad";
const int mqtt_port = 1883;
WiFiClient espClient;  //creates a Wi-Fi client.
PubSubClient client(espClient);
StaticJsonDocument<512> doc;  //store in the stack
```

Link: https://docs.arduino.cc/tutorials/uno-wifi-rev2/uno-wifi-r2-mqtt-device-to-device
Link: https://pubsubclient.knolleary.net/api#PubSubClient1

```
//setting some parameters
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE(50)
char msg[MSG_BUFFER_SIZE];
int value = 0;
String msgStr = "";
char mess[512];
int counter;
```

```
//connecting to a WiFi network
void setup_wifi() {

  delay(10);
  Serial.println();  //Prints data to the serial port as human-readable ASCII text
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA); //station mode: the ESP32 connects to an access point
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

Link: https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/

The code only sets up the WiFi connection of the board and MQTT transmission path to the broker because I didn't have any sensor available to read data at this moment.

Theory:

Waspmote IDE is a development environment for programming and configuring Waspmote devices, which are part of Libelium's Internet of Things (IoT) platform. Libelium is a company that specializes in creating IoT hardware and software solutions.

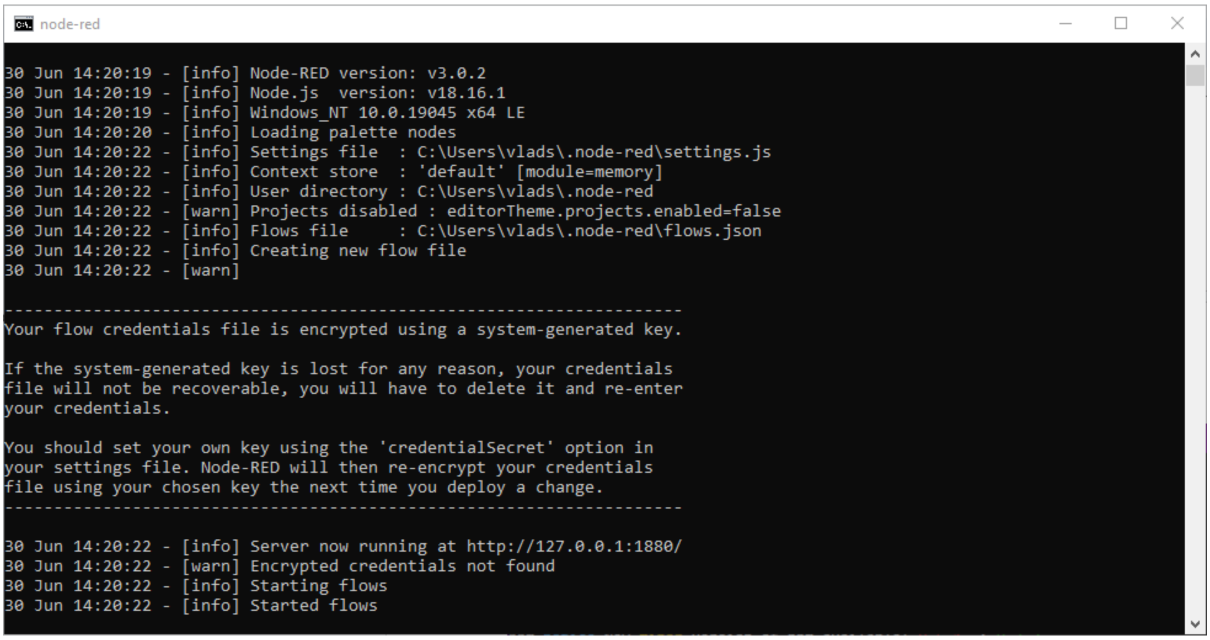
Node-Red:

Theory: Node-RED is an open-source flow-based programming tool designed for visual programming and rapid prototyping of applications. It provides a browser-based, graphical interface that allows users to create, wire, and deploy flows of data between various devices and services. Node-RED is commonly used in the Internet of Things (IoT) space and for connecting and automating different systems and devices.

I downloaded the 18.16.1 LTS version of node.js and used the terminal to download the Node-Red packages to use the interface.

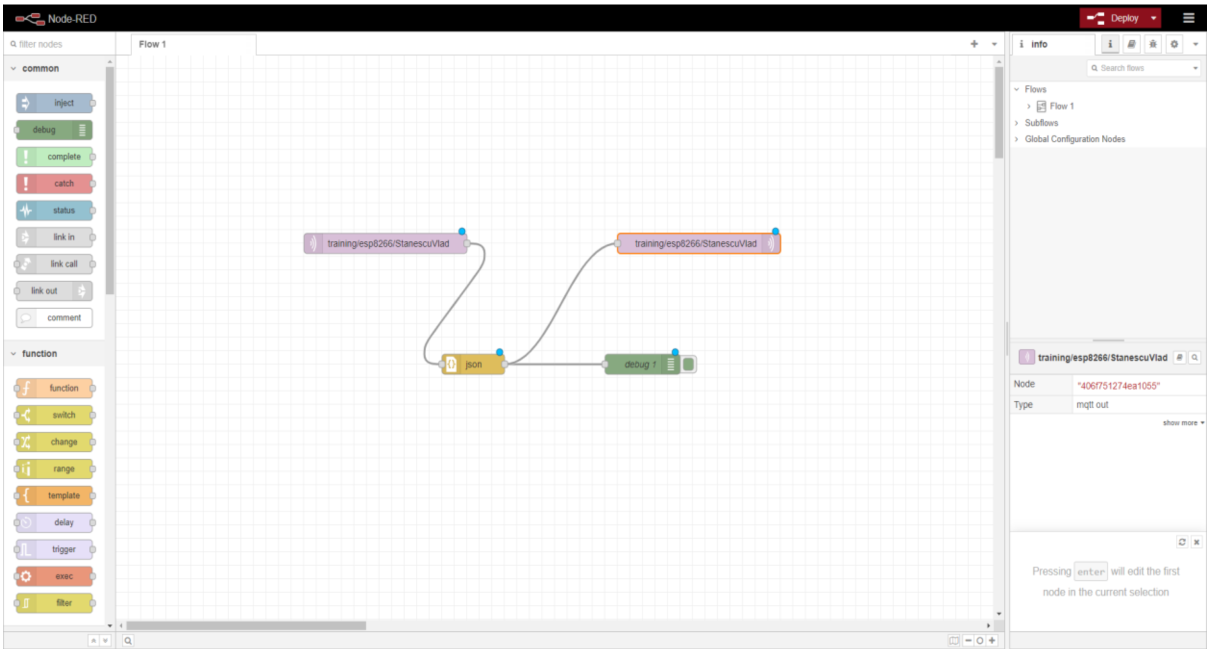
For downloading the packages, I used the command: **npm install -g –unsafe-perm node-red**.

To open the Node-Red terminal, I used the command: **node-red**.

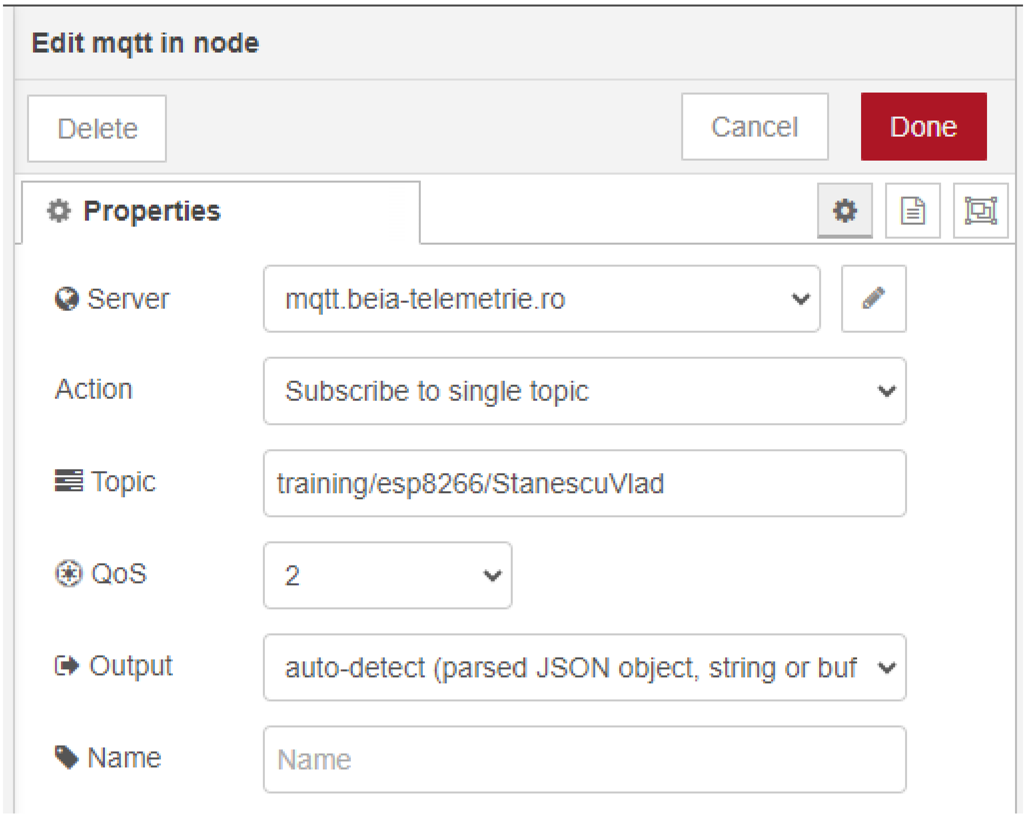


Service is now running on my computer and I can access the Node-Red interface through the link below that points my browser to the localhost. (1880)

http://localhost:1880/#flow/65cf51fda669b22a



I have designed the diagram of the transmission path from the board up to the broker.

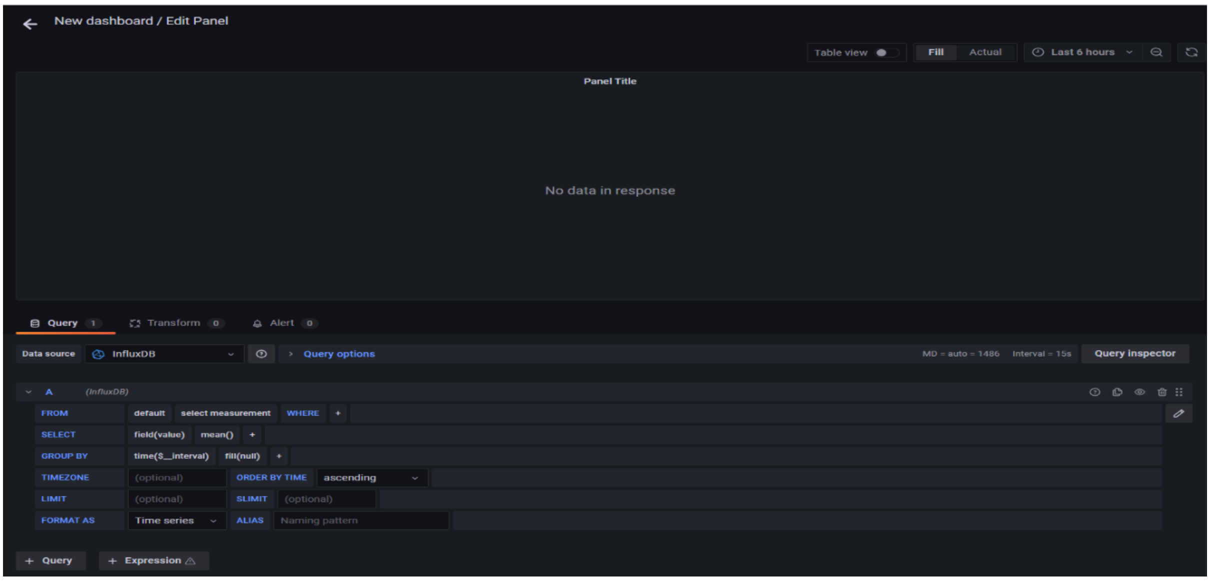


Grafana - Data Visualization

I accessed the following link: https://grafana.beia-telemetrie.ro/
The login account:

user: interviu.practica
password: beiapractica

I created a new dashboard. In the Data Source I chose InfluxDB and I created 1 query.



Theory:

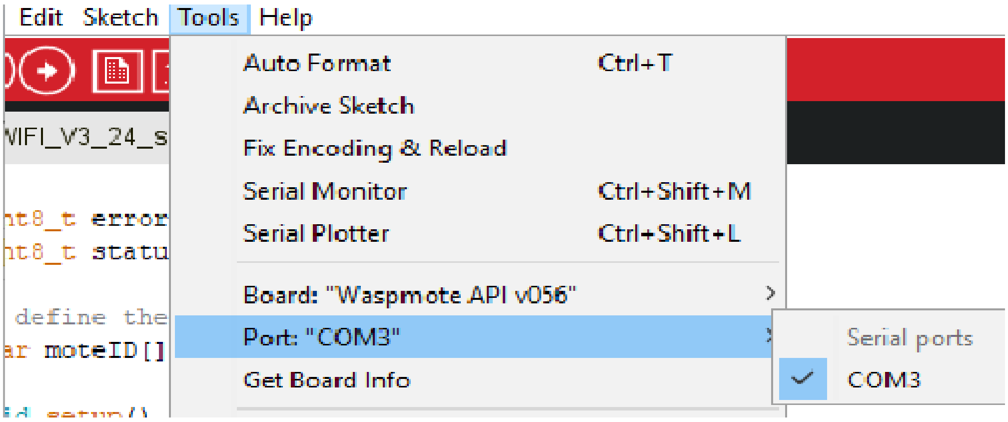
Grafana is an open-source data visualization and monitoring tool. It allows users to create interactive and customizable dashboards to visualize time-series data from various sources, including databases like InfluxDB, Prometheus, Elasticsearch, and others. Grafana supports a wide range of data sources and enables users to create charts, graphs, tables, and alerts to monitor and analyze data in real-time. It is commonly used for monitoring infrastructure, applications, and various systems.

InfluxDB is a time-series database designed to handle and store time-stamped data efficiently. It is an open-source database that provides high-performance storage and retrieval of time-series data. Time-series data consists of data points associated with specific timestamps, such as sensor readings, server metrics, application performance metrics, and more. InfluxDB is optimized for storing and querying such time-series data, making it suitable for applications that require real-time analysis and monitoring.

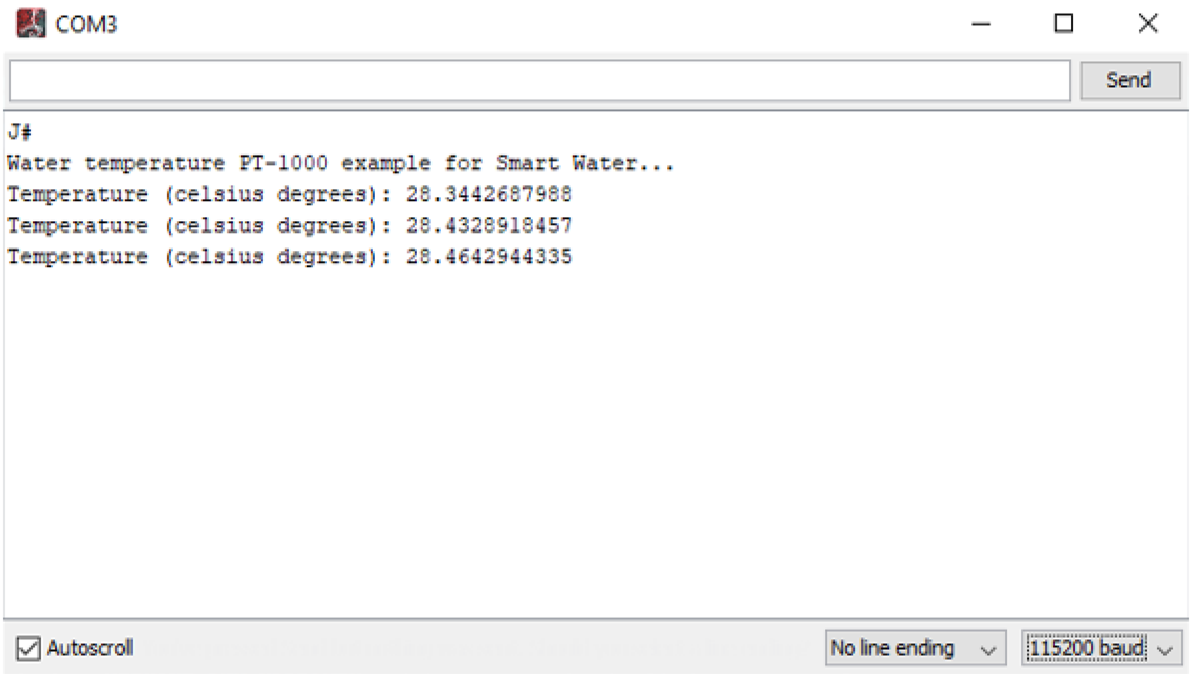
Next Activity summary:

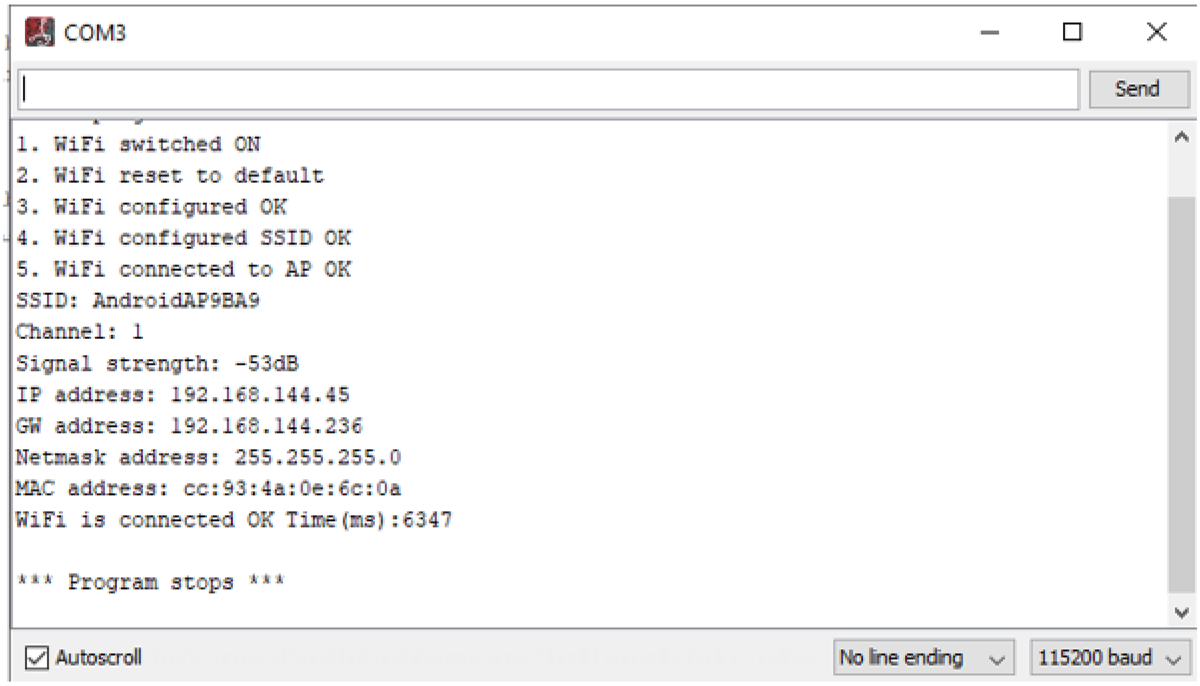
- sent data from a Libelium Smart Water station through WI-FI to Meshlium.
- Meshlium was used to transmit data to BEIA Broker
- data was stored in a database ,,InfluxDB”
- selected and visualized data in Grafana
- set an alert and ran a test

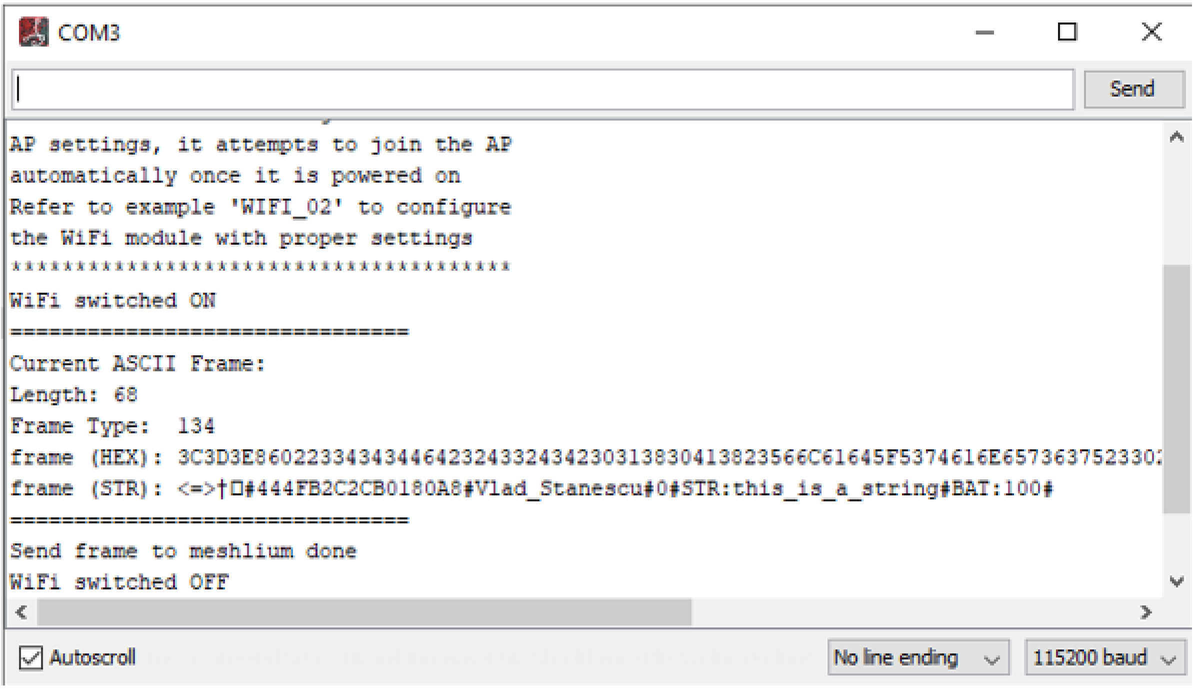
To configure the station and upload the code, a USB cable was needed to connect the station to my personal laptop. For the configuration, I used Waspmote PRO IDE. I selected the port from the “Tools” section and I used the “Examples” section to locate the examples of code which suited my needs: the code to connect the station to WI-FI for the data transmission, the code for temperature sensor reading, the code to get the battery level.



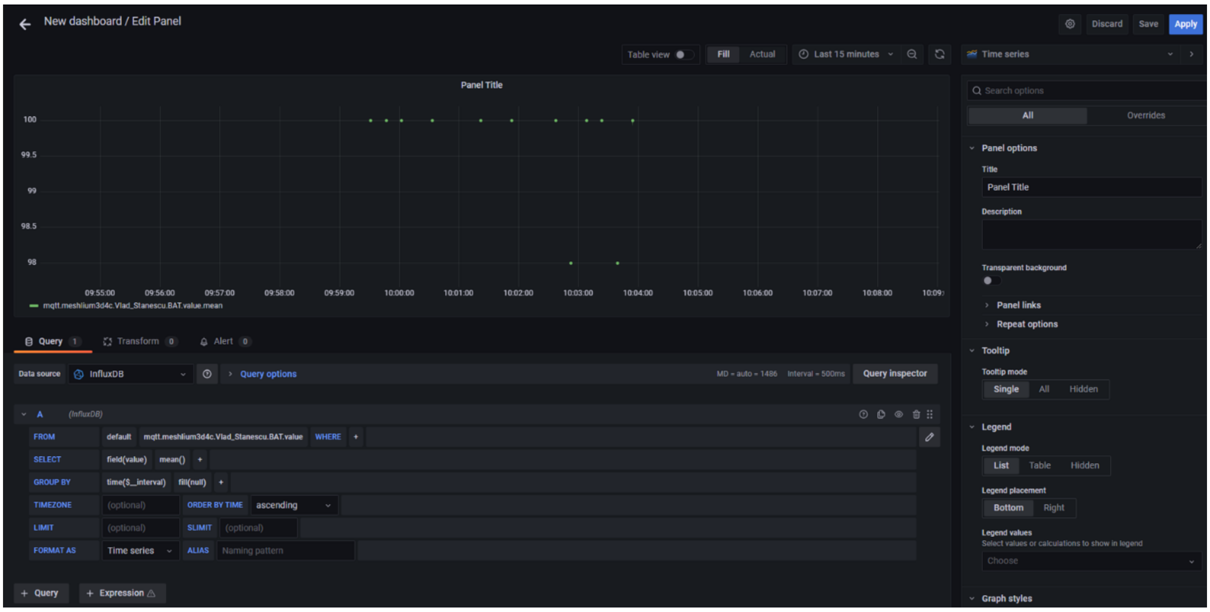
After the verifications of the codes, I uploaded them to the station. The next step was to open the serial monitor to observe the data which the station transmits.



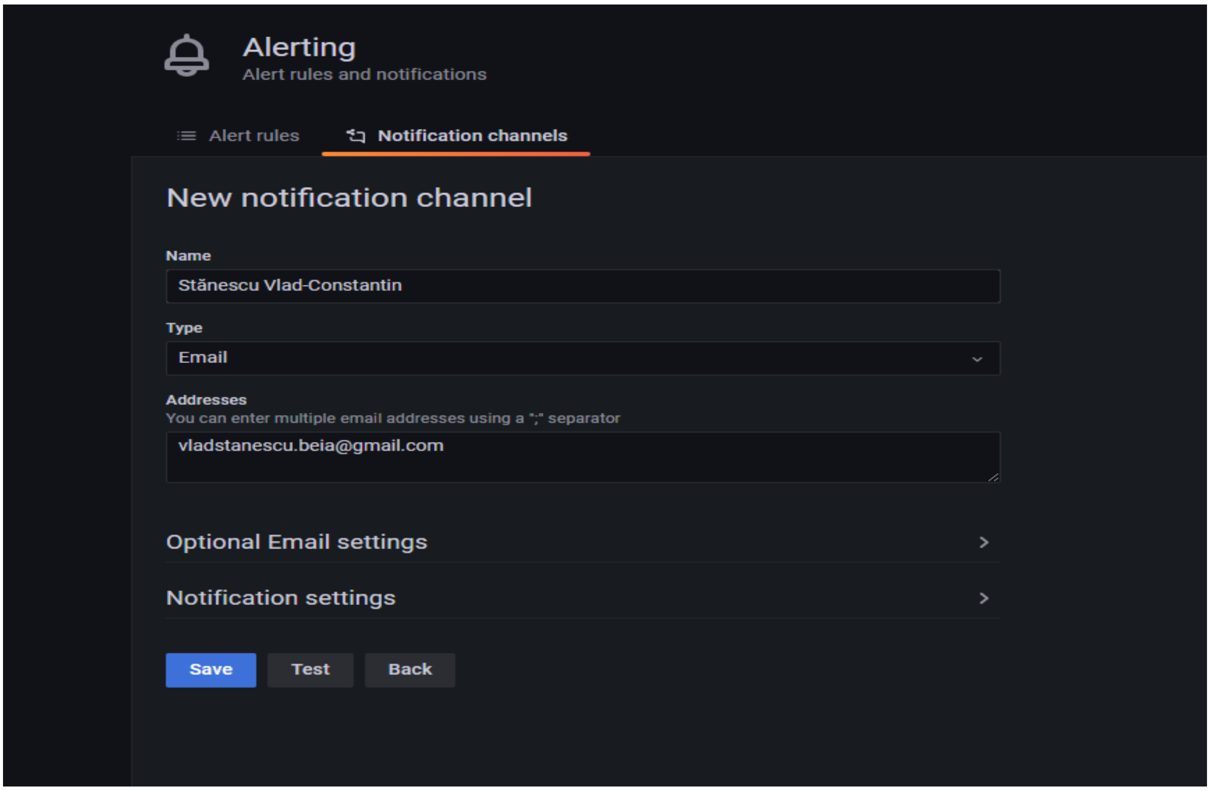




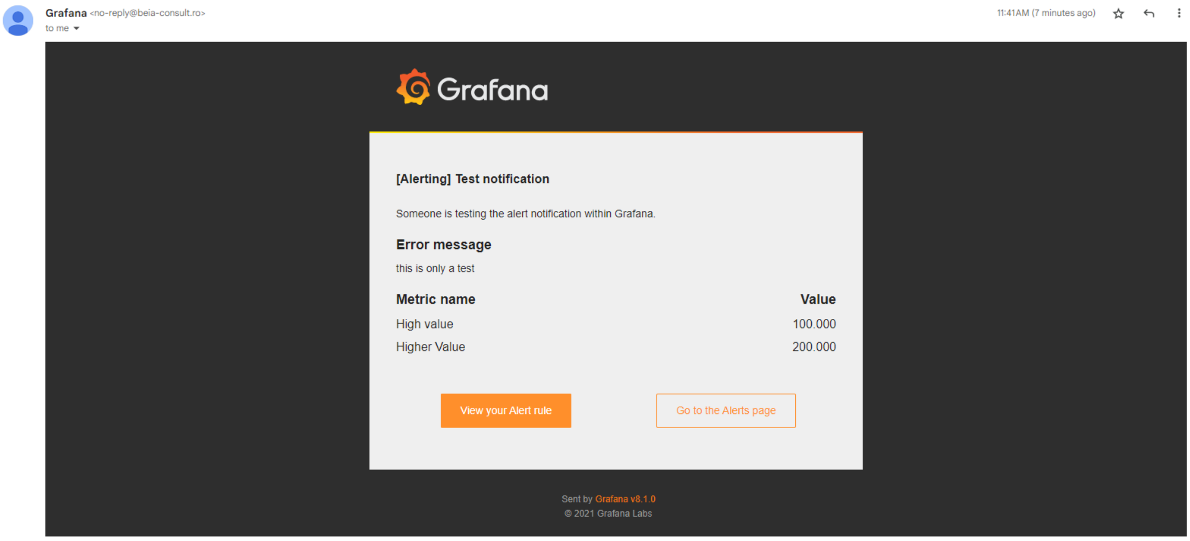
Once sent to the broker, the data is stored in a database which is InfluxDB and in the end the battery level itself can be visualized in Grafana. Now in Grafana all I had to do was to select the data which was sent and represent it on a created panel.

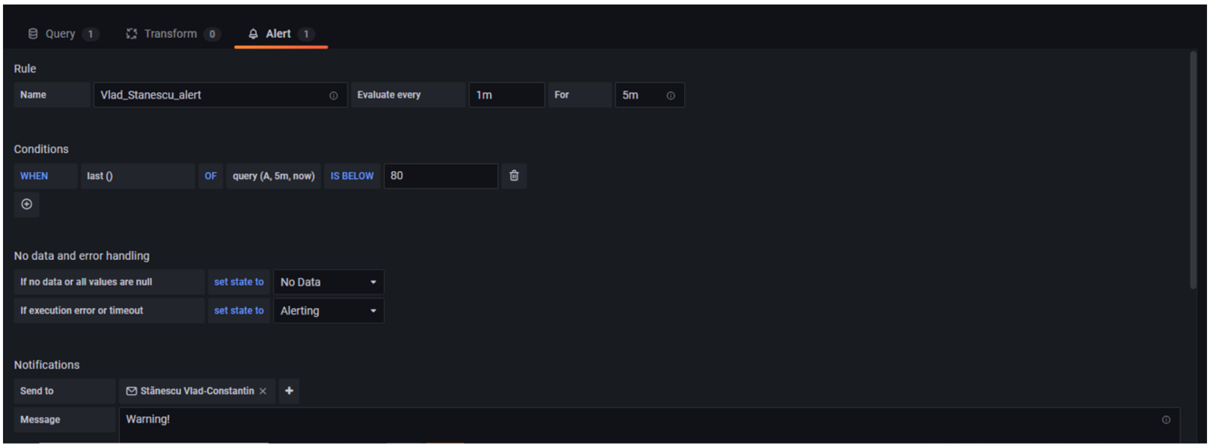


I was tasked with creating an alert in Grafana to send notifications. To personalize my alert, I completed the fields as in the following picture:



To ensure that it works I clicked on the Test button and received a notification on my email:


Grafana <no-reply@beia-consult.ro>
to me
11:41AM (7 minutes ago)
Grafana
[Alerting] Test notification
Someone is testing the alert notification within Grafana.
Error message
this is only a test
Metric name
Value
High value
100.000
Higher Value
200.000
View your Alert rule
Go to the Alerts page
Sent by Grafana v8.1.0
© 2021 Grafana Labs


Query 1
Transform 0
Alert 1
Rule
Name
Vlad_Stanescu_alert
Evaluate every
1m
For
5m
Conditions
WHEN
last ()
OF
query (A, 5m, now)
IS BELOW
80
No data and error handling
If no data or all values are null
set state to
No Data
If execution error or timeout
set state to
Alerting
Notifications
Send to
Stănescu Vlad-Constantin
Message
Warning!

Theory oxidation-reduction potential (ORP) sensor:

The ORP probe is a combination electrode whose output voltage is equivalent to the potential of the solution, so it will share the connection sockets with that sensor. The output of the circuitry to which it is connected is directly read from the analog-to-digital converter of the Smart Water sensor board, being the 2.048 V reference subtracted to obtain the actual oxidation-reduction potential in volts (in this case, since this parameter is directly a voltage it is not necessary to call a conversion function).

The ORP sensor operates based on the principle of redox reactions, where electrons are transferred between different chemical species, resulting in oxidation (loss of electrons) or reduction (gain of electrons). The sensor consists of a sensing electrode and a reference electrode, both immersed in the solution being analyzed.

A positive ORP value indicates that the solution has a greater tendency to undergo oxidation (electron loss), while a negative ORP value suggests a greater tendency for reduction (electron gain). A high positive ORP reading may indicate the presence of strong oxidizing agents, while a low or negative ORP value may indicate the presence of reducing agents.

ORP sensors are valuable tools in water treatment applications, where they help monitor and control disinfection processes by measuring the efficacy of chlorine or other oxidizing agents.

Oxidation-reduction potential sensor

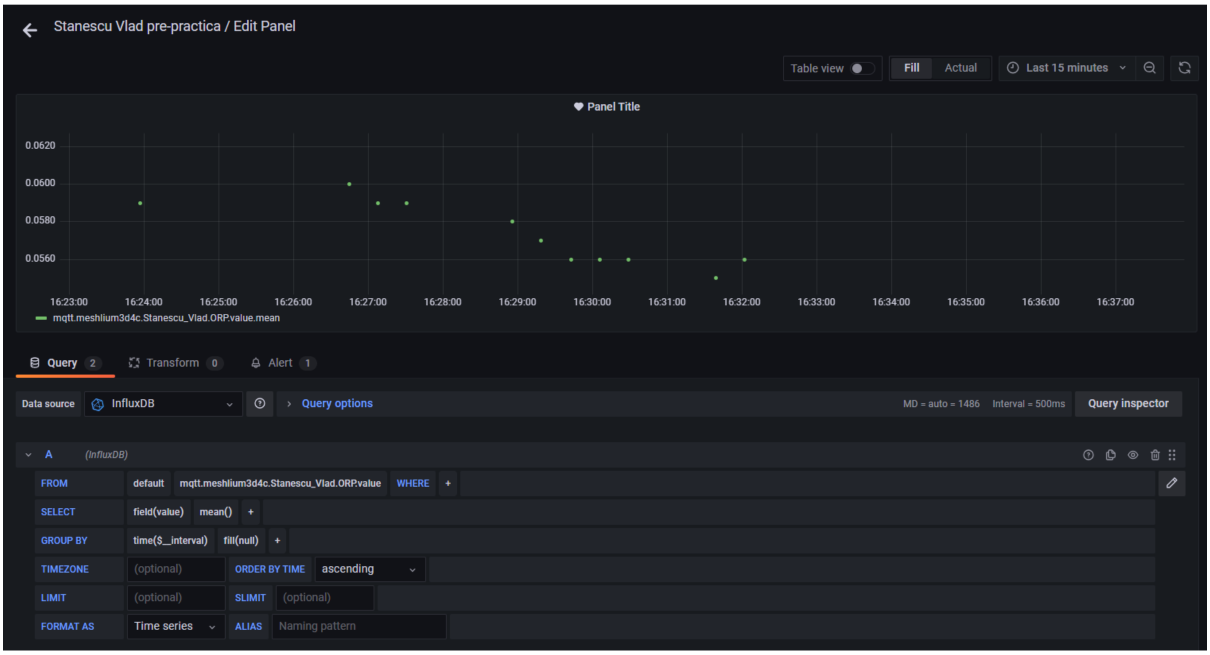
Figure: Image of the oxidation-reduction potential sensor

COM3

```
J#
ORP example for Smart Water...
ORP Estimated: 0.0610058307 volts
ORP Estimated: 0.0638041496 volts
ORP Estimated: 0.0646777153 volts
ORP Estimated: 0.0657649040 volts
ORP Estimated: 0.0664081573 volts
ORP Estimated: 0.0668623447 volts
ORP Estimated: 0.0665240287 volts
ORP Estimated: 0.0669453144 volts
```

COM3

```
J#
Start program
****************************************
It is assumed the module was previously
configured in autoconnect mode and with
the Meshlium AP settings.
Once the module is configured with the
AP settings, it attempts to join the AP
automatically once it is powered on
Refer to example 'WIFI_02' to configure
the WiFi module with proper settings
****************************************
WiFi switched ON
================================
Current ASCII Frame:
Length: 46
Frame Type:  134
frame (HEX): 3C3D3E86012334343446423243324342303138304138235374616E657363755F566C61642330234241543A393123
frame (STR): <=>†□#444FB2C2CB0180A8#Stanescu_Vlad#0#BAT:91#
================================
================================
Current ASCII Frame:
Length: 49
Frame Type:  134
frame (HEX): 3C3D3E86012334343446423243324342303138304138235374616E657363755F566C61642331234F52503A302E30363223
frame (STR): <=>†□#444FB2C2CB0180A8#Stanescu_Vlad#1#ORP:0.062#
================================
Send frame to meshlium done
WiFi switched OFF
```



CODE:

```
// Put your libraries here (#include ...)
#include <WaspWIFI_PRO_V3.h>
#include <WaspFrame.h>
#include <WaspSensorSW.h>

// choose socket (SELECT USER'S SOCKET)
///////////////////////////////////////
uint8_t socket = SOCKET0;
///////////////////////////////////////

// choose HTTP server settings
///////////////////////////////////////
char type[] = "http";
char host[] = "82.78.81.178";
uint16_t port = 80;
///////////////////////////////////////

float value_orp;
float value_orp_calculated;
```

```
// Calibration values
#define CAL_POINT_10 1.985
#define CAL_POINT_7 2.070
#define CAL_POINT_4 2.227
// Temperature at which calibration was carried out
#define CAL_TEMP 23.7
// Offset obtained from sensor calibration
#define CALIBRATION_OFFSET 0.0
// Calibration of the sensor in normal air
#define AIR_CALIBRATION 2.65
// Calibration of the sensor under 0% solution
#define ZERO_CALIBRATION 0.0
// Value 1 used to calibrate the sensor
#define POINT1_COND 10500
// Value 2 used to calibrate the sensor
#define POINT2_COND 40000
// Point 1 of the calibration
#define POINT1_CAL 197.00
// Point 2 of the calibration
#define POINT2_CAL 150.00

ORPClass ORPSensor;
// define the Waspmote ID
char moteID[] = "Stanescu_Vlad";

void setup()
{
  USB.println(F("Start program"));

  USB.println(F("***************************************"));
  USB.println(F("It is assumed the module was previously"));
  USB.println(F("configured in autoconnect mode and with"));
  USB.println(F("the Meshlium AP settings."));
  USB.println(F("Once the module is configured with the"));
  USB.println(F("AP settings, it attempts to join the AP"));
  USB.println(F("automatically once it is powered on"));
  USB.println(F("Refer to example 'WIFI_02' to configure"));
  USB.println(F("the WiFi module with proper settings"));
  USB.println(F("***************************************"));

  // set the Waspmote ID
  frame.setID(moteID);
}

void loop()
{
  /////////////////////////////////////////////////////
  // 1. Switch ON
```

```
////////////////////////////////////////////////
uint8_t error = WIFI_PRO_V3.ON(socket);

if (error == 0)
{
  USB.println(F("WiFi switched ON"));
}
else
{
  USB.println(F("WiFi did not initialize correctly"));
}

// check connectivity
bool status = WIFI_PRO_V3.isConnected();

// check if module is connected
if (status == true)
{
  ///////////////////////////////
  // 3.1. Create a new Frame
  ///////////////////////////////

  // create new frame (only ASCII)
  frame.createFrame(ASCII);

  // add sensor fields

  frame.addSensor(SENSOR_BAT, PWR.getBatteryLevel());

  // print frame
  frame.showFrame();

  Water.ON();
  delay(2000);

  ///////////////////////////////////////////
  // 2. Read sensors
  ///////////////////////////////////////////

  // Reading of the ORP sensor
  value_orp = ORPSensor.readORP();
  // Apply the calibration offset
  value_orp_calculated = value_orp - CALIBRATION_OFFSET;

  ///////////////////////////////////////////
  // 3. Turn off the sensors
  ///////////////////////////////////////////
```

```
    Water.OFF();

    ///////////////////////////////////////////
    // 4. Create ASCII frame
    ///////////////////////////////////////////

    // Create new frame (ASCII)
    frame.createFrame(ASCII);

    // Add ORP value
    frame.addSensor(SENSOR_WATER_ORP, value_orp_calculated);

    // Show the frame
    frame.showFrame();

    // wait 2 seconds
    delay(2000);
    ///////////////////////////////
    // 3.2. Send Frame to Meshlium
    ///////////////////////////////

    // http frame
    error = WIFI_PRO_V3.sendFrameToMeshlium(type, host, port, frame.buffer,
frame.length);

    // check response
    if (error == 0)
    {
      USB.println(F("Send frame to meshlium done"));
    }
    else
    {
      USB.println(F("Error sending frame"));
      if (WIFI_PRO_V3._httpResponseStatus)
      {
        USB.print(F("HTTP response status: "));
        USB.println(WIFI_PRO_V3._httpResponseStatus);
      }
    }
  }
  else
  {
    USB.println(F("2. WiFi is connected ERROR"));
  }
  //////////////////////////////////////////////////
  // 3. Switch OFF
  //////////////////////////////////////////////////
  WIFI_PRO_V3.OFF(socket);
```

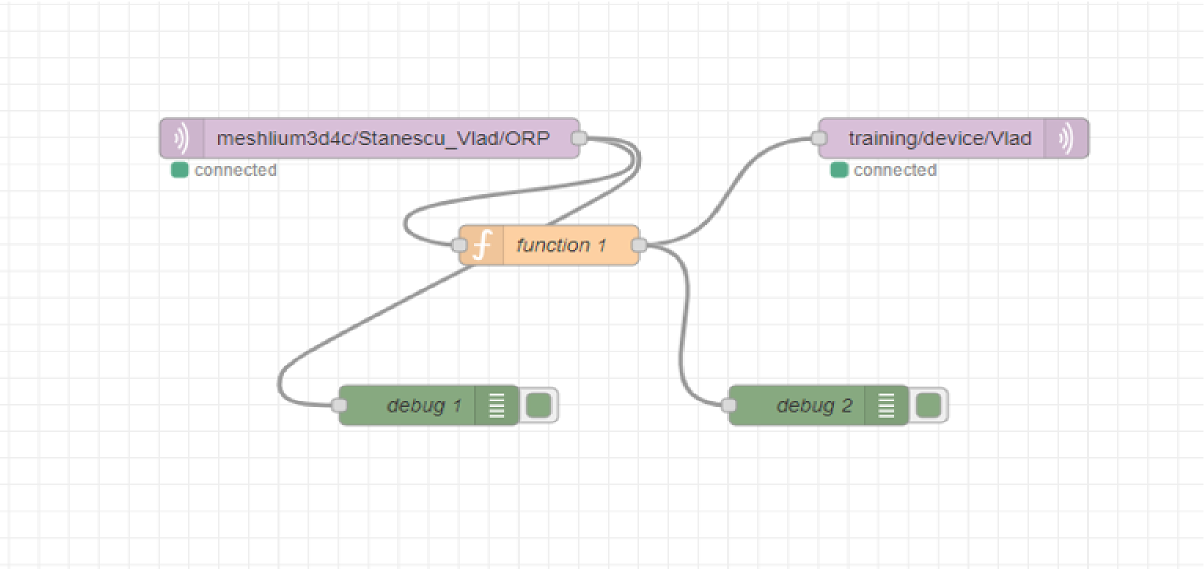
```
  USB.println(F("WiFi switched OFF\n\n"));
  delay(10000);
}
```

I had the task of collecting data (ORP) from the Libellium “Smart Water” station, processing it in Node-Red, and visualizing it in Grafana.

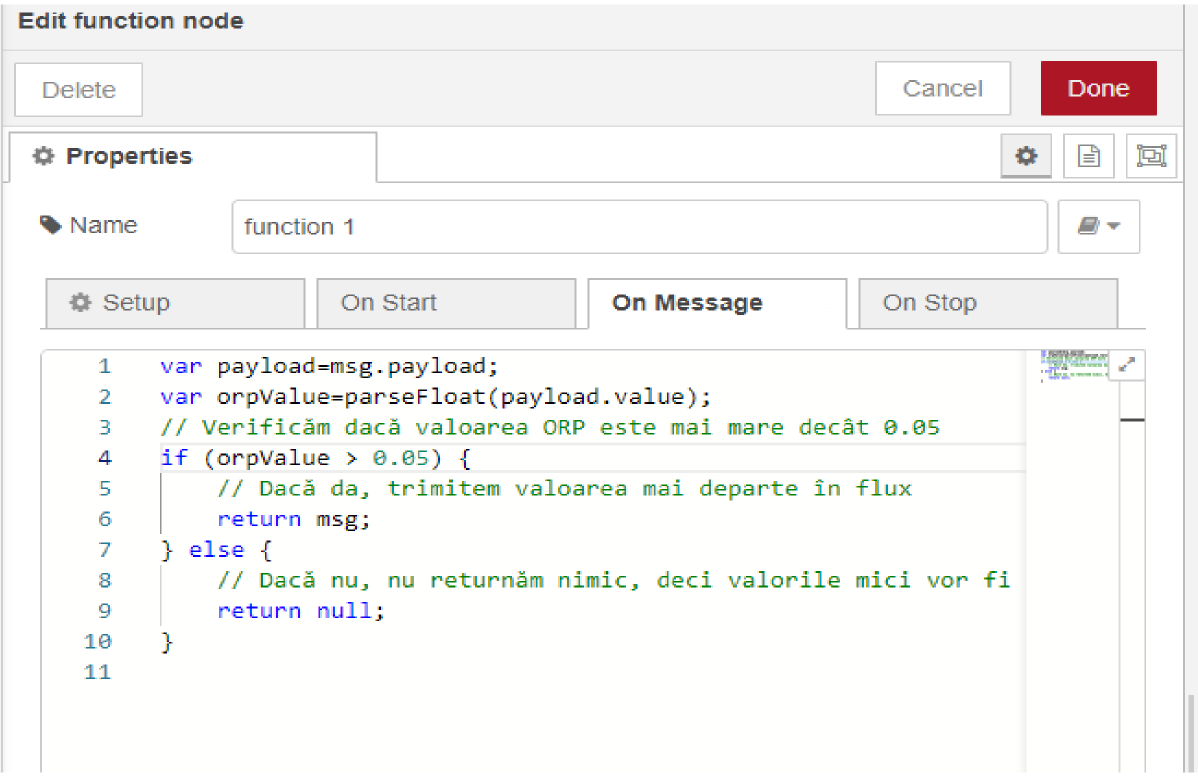
After running the code related to data transmission to the IoT Gateway, a topic was generated from which we could retrieve the ORP values collected by the station.

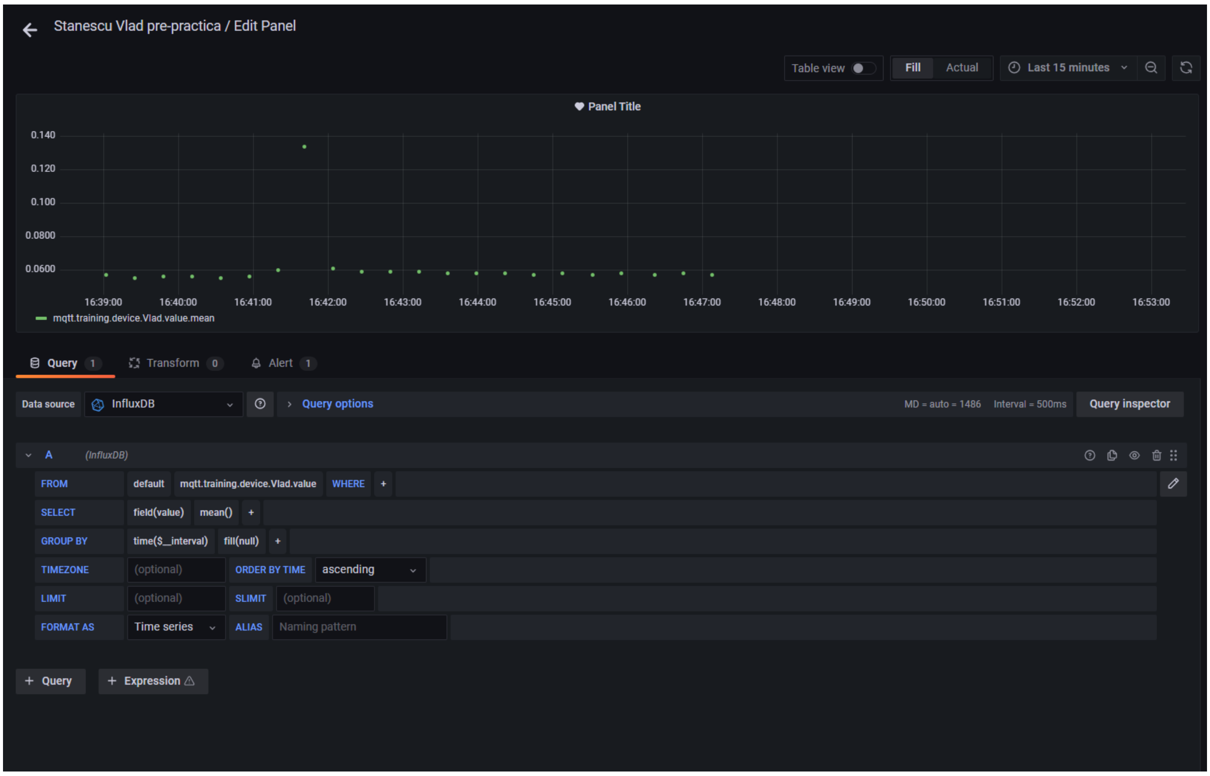
I created two Meshlium nodes: one that retrieves the ORP values directly from the station and another one that retrieves the processed ORP values after passing through one function.

I created a function by which I wanted to collect only the ORP values greater than 0.05 from the data stream, so in the output node, only filtered values were received according to the preferences and finally I visualized those values in Grafana.




debug
all nodes
all
Object
{ id: "2236529", id_wasp: "Stanescu_Vlad", id_secret: "444FB2C2CB0180A8", sensor: "ORP", value: "0.057" … }
7/19/2023, 4:53:33 PM node: debug 2
meshlium3d4c/Stanescu_Vlad/ORP : msg.payload : Object
{ id: "2236529", id_wasp: "Stanescu_Vlad", id_secret: "444FB2C2CB0180A8", sensor: "ORP", value: "0.057" … }
7/19/2023, 4:53:34 PM node: debug 1
meshlium3d4c/Stanescu_Vlad/ORP : msg.payload : Object
{ id: "2236528", id_wasp: "Stanescu_Vlad", id_secret: "444FB2C2CB0180A8", sensor: "ORP", value: "0.057" … }
7/19/2023, 4:53:35 PM node: debug 2
meshlium3d4c/Stanescu_Vlad/ORP : msg.payload : Object
{ id: "2236528", id_wasp: "Stanescu_Vlad", id_secret: "444FB2C2CB0180A8", sensor: "ORP", value: "0.057" … }
7/19/2023, 4:53:36 PM node: debug 1
meshlium3d4c/Stanescu_Vlad/ORP : msg.payload : Object
{ id: "2236527", id_wasp: "Stanescu_Vlad", id_secret: "444FB2C2CB0180A8", sensor: "ORP", value: "0.058" … }





Theory:

When you deploy your Node-RED flow and activate the **"debug"** node, it will print the message content to the debug sidebar in the Node-RED editor. The debug output shows the properties and values of the message object, allowing you to inspect the payload and other message properties.

The function **“parseFloat”** is a JavaScript function used to convert a string into a floating-point number. It is commonly used when working with numeric data in a flow to ensure that the data is represented as a decimal number (floating-point number) rather than a string.

The **"payload"** is a common term used to refer to the main data of a message. A message in Node-RED consists of different properties, and the "payload" property is one of them. It holds the primary information or data that is being passed through the flow.