UNIVERSITÀ DEGLI STUDI DELL'AQUILA

LUT University

Software Engineering for IOT

Felix Nahrstedt
Mehdi Karmouche
Alejandro Sebastian Enriquez Mancheno

Sustainable Agriculture System

25. January 2023

Supervision:

Prof. Davide Di Ruscio

Table of Contents

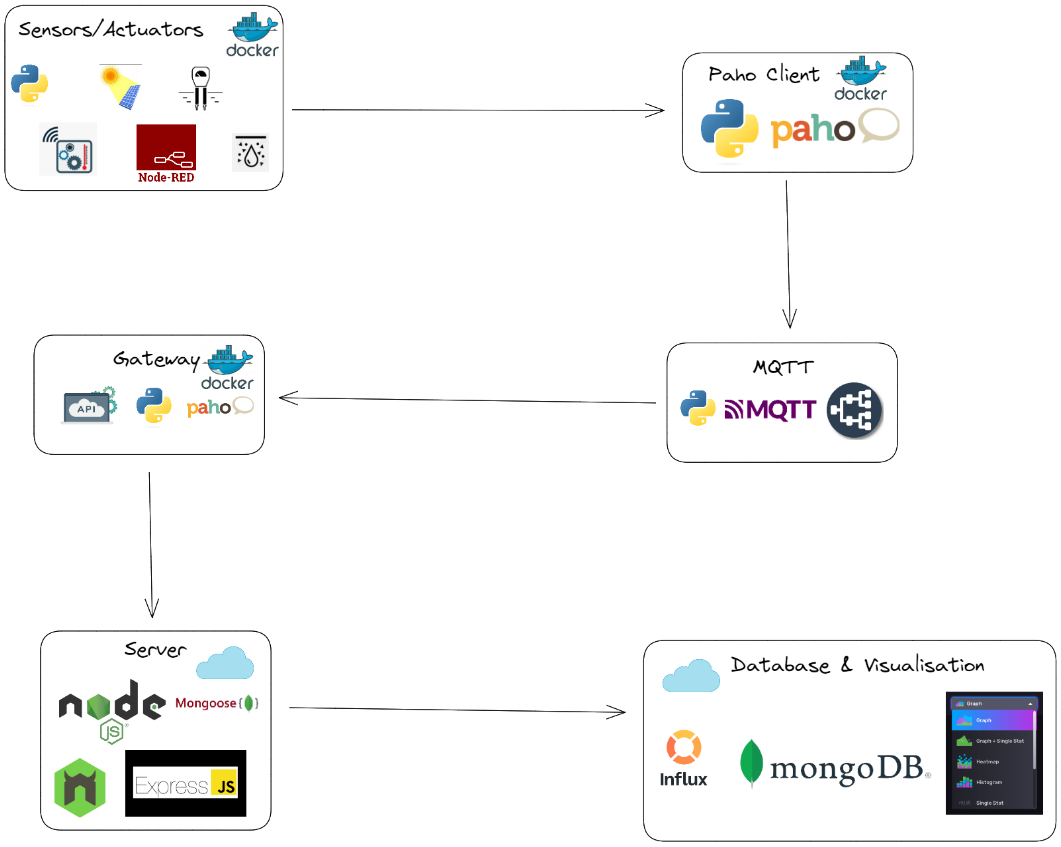
MOTIVATION

Agriculture is one of the most important activities humans perform, having a strong impact in sectors like economic and social as it provides essential materials we need for daily life, such as food and fabric materials. Also, agriculture is a major consumer of natural resources, including land, water, and energy. That is why sustainability in agriculture is key when we talk about climate change, as this can help to mitigate its effects by reducing greenhouse gas emissions and increasing carbon sequestration in soil, also can improve economic aspects by reducing the input costs and times, increasing the final earnings. Considering that the plenty of resources we have nowadays, in the next years are going to be harder to obtain as easily, we as green software engineers need to find a way to be part of the solution to the usage of these resources and create a sustainable solution.

With our *Sustainable Agriculture System*, we propose the usage of self powered sensors to obtain important information about the farm's status. These sensors are powered by solar panels that are activated with the solar energy received from sunrise to sunset. All the processes are done while the panels are receiving solar energy and at night the sensors are deactivated. With this solution, we don't implement batteries that can end up being a potential source of contamination in the mid-long term.

Software Architecture

Technology View



In our Software Architecture we majorly want to focus on having a clean outline of the system while still keeping enough functionality to make it reusable. The Technology food above presents not only the major technologies as an overview but also the connections between them.

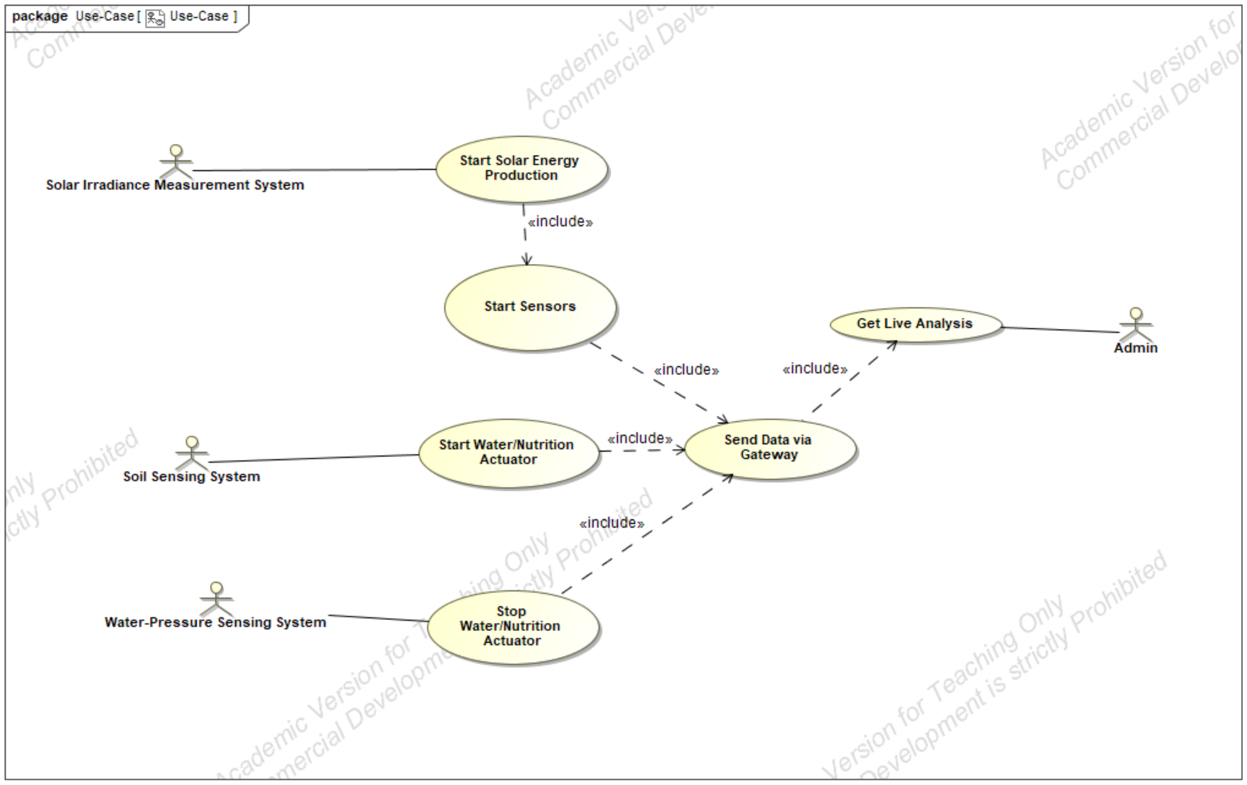
We have a large set of simulated sensors including Solar, Pressure, Temperature, Humidity, NPK, Soil Moisture and PH Sensors together with actuators for Water and Nutrition which are connected with the sensors through MQTT as well.

The output values are being sent over a Paho-Client through MQTT (also
using the MQTT Explorer) to a gateway where they are ordered and sent via API Connection to a deployed Cloud Server Endpoint.

The server uses Node.js as a programming language together with libraries like mongoose or express.js to handle database connections easier.

For database and visualisation purposes, we are using both mongoDB and InfluxDB - while focussing on the usage of Influx because of Influx's possibilities for Visualisations on the admin side as well.

Use Case Diagram

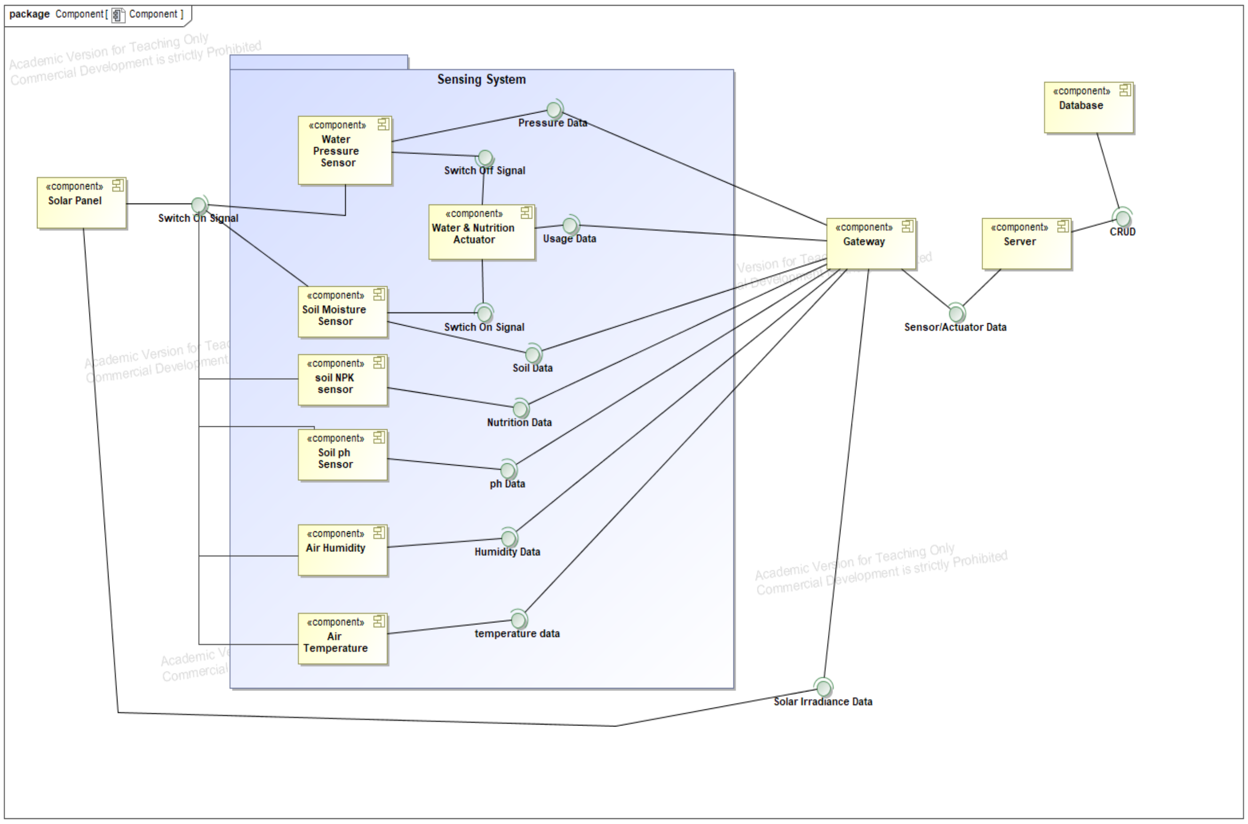

package Use-Case [Use-Case]
Solar Irradiance Measurement System
Start Solar Energy Production
«include»
Start Sensors
«include»
Get Live Analysis
«include»
Admin
Soil Sensing System
Start Water/Nutrition Actuator
«include»
Send Data via Gateway
«include»
Water-Pressure Sensing System
Stop Water/Nutrition Actuator

Generally, we have 3 Actors in our system and 1 external actor (the Admin). The Solar Irradiance Measurement System activates the production by sensing solar irradiance. In other words, when the sun shines the system starts.

This includes starting other sensors as well like pressure, temperature, humidity or ph sensing (and others as well - see system description).

Everytime a sensor is running, it sends data via an api gateway to a server which enables the admin to look at graphs for different sensors (in a way that he can select) by sending the data to the database using CRUD commands.

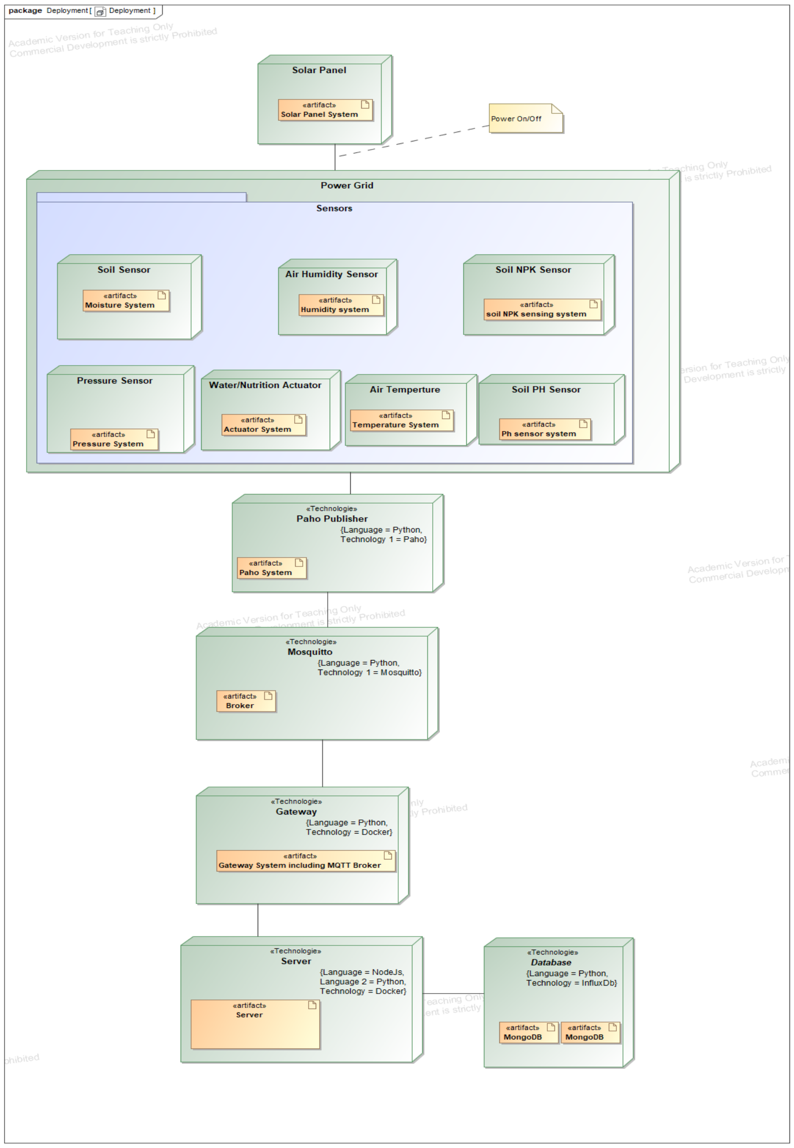
Class Diagram



The class Diagram depicts the described connections between sensors and the server side. The Solar Panel is located outside of the Sensing system, because it works as a power switch to turn the sensors on/off. While the connections inside the sensing system are very self-explanatory, all the data from the sensors are being send to the gateway (through mqtt with Paho clients), where they are prepared to be forwarded to the database. Once the API connection between Gateway and Database is established, the data is send to the Server which stores it in the database to also display the dashboard including graphs.

Deployment Diagram

Finally, In the deployment diagram on the next page, we show in a little bit more detail the deployment nodes and technologies that add to the class diagram. Further Technological details will be discussed in the following sections.


package Deployment[Deployment]
Solar Panel
«artifact» Solar Panel System
Power On/Off
Power Grid
Sensors
Soil Sensor
«artifact» Moisture System
Air Humidity Sensor
«artifact» Humidity system
Soil NPK Sensor
«artifact» soil NPK sensing system
Pressure Sensor
«artifact» Pressure System
Water/Nutrition Actuator
«artifact» Actuator System
Air Temperture
«artifact» Temperature System
Soil PH Sensor
«artifact» Ph sensor system
«Technologie» Paho Publisher
{Language = Python, Technology 1 = Paho}
«artifact» Paho System
«Technologie» Mosquitto
{Language = Python, Technology 1 = Mosquitto}
«artifact» Broker
«Technologie» Gateway
{Language = Python, Technology = Docker}
«artifact» Gateway System including MQTT Broker
«Technologie» Server
{Language = NodeJs, Language 2 = Python, Technology = Docker}
«artifact» Server
«Technologie» Database
{Language = Python, Technology = InfluxDb}
«artifact» MongoDB
«artifact» MongoDB

Data-Sources and Simulation

The Simulations are based on a kaggle dataset that includes weather and solar information for a particular time frame in Spain. We decided to use this database because it includes most of the sensor information that we needed for the simulation part. However, because our project is very dependent on soil values, some code had to be implemented to simulate a crop based scenario.

These include not only both actuators (water and nutrition), that were used to get more fluctuation into the dataset to make it appear more realistic. The first of the self implemented simulated sensors is the N-P-K-Sensor.

These three numbers form what is called the fertilizer's N-P-K ratio — the proportion of three plant nutrients in order: nitrogen (N), phosphorus (P) and potassium (K). The N-P-K numbers reflect each nutrient's percentage by weight. They are strongly based on the last time that nutrients have been added. Hence, the more time has passed since the last “feeding” of the plants - more nutrients need to be added by the actuators because of a lower NPK number.

The ph value of the soil is also based on the nutrients in the soil. Because the npk value is generally in ppm with an optimum between 40 and 60 and a simulated minimum of 0 and maximum of 70, the following formula has been designed for simulating the ph with optimal value of 7 and slightly base or acidic probabilities as well:

$$ph = 0.05 * npk + 5$$

Regarding the soil moisture, it is (in a high level of abstraction) influenced by the temperature because of vaporisation, the air humidity that is sensed and the last time the water sensors have been activated. The humidity has been min-max normalised and randomly fluctuates because of different soil composition (`random.uniform(0.8,0.9)`).

The units for our simulation are as follows:

- Solar irradiance in percent (min-max normalised values out of a large kaggle dataset)
- Pressure - converted from Pascal into Percentages
- Temperature in °C
- Humidity of the air in Percentages
- NPK values in parts per million (ppm)
- Soil Moisture in percent
- Ph values in the ph scale
- Nutrition Actuators in ppm
- Water actuator in percent (just for simulation purposes)

Technologies and Application

MQTT

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe messaging protocol that is designed for use in resource-constrained environments, such as in IoT devices. It is a messaging protocol that is designed to be used on top of the TCP/IP protocol, but it is simpler and more lightweight than other messaging protocols such as HTTP.

Is based on a publish-subscribe model, in which clients (publishers) send messages (publications) to a server (broker) and other clients (subscribers) can receive those messages. Clients can also subscribe to specific topics to receive only the messages that are of interest to them.

MQTT has a number of features that make it well-suited for use in IoT devices, such as:

- Low bandwidth and low power consumption: MQTT uses a small amount of bandwidth and power, making it well-suited for use in resource-constrained devices.
- Low latency: MQTT is designed to have low latency, which is important for real-time applications.
- Quality of Service (QoS) levels: MQTT provides three levels of Quality of Service (QoS) that can be used to ensure that messages are delivered reliably, even in unreliable network conditions.
- MQTT is widely used in IoT, M2M and other embedded systems, it is also used in industrial automation, transportation, and other areas where small and simple messaging protocol is required

Docker

Docker is a platform that allows developers to create, deploy, and run containerized applications. The containers are a lightweight and portable way to package an application and its dependencies, so that it can run consistently across different environments.

Also, it provides a way to create and manage containers, which are lightweight and portable executable packages that include everything needed to run an application: code, runtime, system tools, libraries, and settings. Containers are isolated from each other and from the host system, which makes them more secure and consistent than traditional virtual machines.

Docker has several key features that make it useful for developers, including:

- Portability: Containers created with Docker can run on any machine that supports the Docker runtime, which makes it easy to move applications between different environments.

- Flexibility: Docker allows developers to use different versions of dependencies and services within the same container, which makes it easy to test and deploy multiple versions of an application.
- Isolation: Containers are isolated from each other and from the host system, which makes them more secure and consistent than traditional virtual machines.
- Speed: Containers start up quickly, so applications can be deployed faster than with traditional virtual machines.
- Ease of use: Docker has a simple and consistent command-line interface, which makes it easy for developers to create, manage, and run containers.

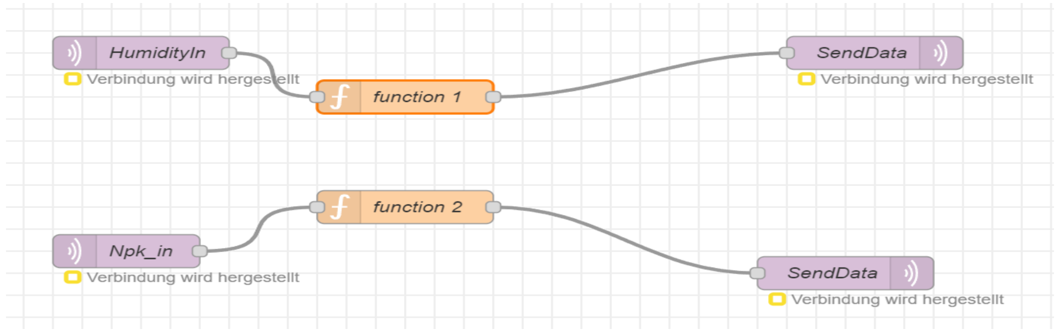
Paho

The Eclipse Paho project is an open-source client implementation of the MQTT (Message Queuing Telemetry Transport) protocol in different programming languages. Provides a simple, lightweight and high-performance MQTT client library that is easy to use and integrate with other systems.

The Paho project provides MQTT client libraries for a variety of programming languages, including Java, C, Python and C++. This code provides a client class which enable applications to connect to an MQTT broker to publish messages, and to subscribe to topics and receive published messages. The Paho libraries are widely used in different systems and applications, and provides a simple and lightweight way to connect to MQTT servers and exchange messages in many technological projects.

Node-Red

We decided to model both of our Actuators in Node-Red. This decision is mainly to split Sensors from Actuators and to use the Node-Red Technology in this project.

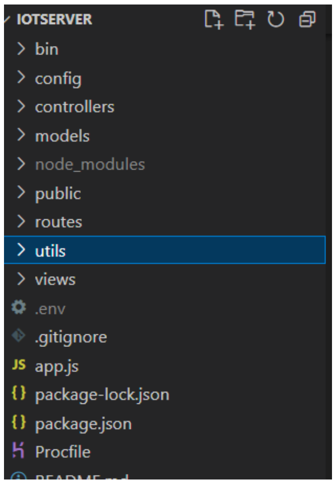

HumidityIn
Verbindung wird hergestellt
function 1
SendData
Verbindung wird hergestellt
function 2
Npk_in
Verbindung wird hergestellt
SendData
Verbindung wird hergestellt

For this, we used the Humidity and NPK Sensors as function inputs that get transformed through 2 functions to be then sent back as the simulated actuators with a timestamp to the MQTT-Broker. In the function, the Humidity/NPK values are checked and - if the value is lower than the different defined thresholds, the actuator is activated.

Node js API

Once the data is sent to the gateway, the latter needs to send it a server application before storing it in cloud databases. For that reason, we developed a server side application using Node.js that is exposed through a REST API. In the server, the schema of data is defined for each data point that we have in our simulation (temperature, moisture). In addition, the server application, once started, connects directly to the cloud databases. Different controllers were developed to implement the API and persist the data correctly while sending back the correct HTTP statuses to the gateway. Furthermore, the server was deployed to Heroku so that the gateway can just hit the right endpoint to persist the desired data. For example: https://iotsegd.herokuapp.com/persist/ph , is used by the gateway to POST the PH data. By deploying the server application, we got rid of a potential overhead of running the application in different machines which improved our collaboration and testing.

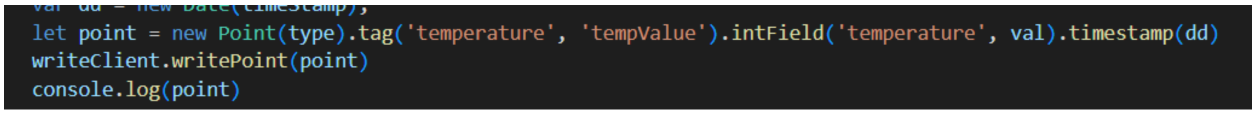
The server had the following file structure:



```
IOTSERVER
> bin
> config
> controllers
> models
> node_modules
> public
> routes
> utils
> views
.env
.gitignore
app.js
package-lock.json
package.json
Procfile
```

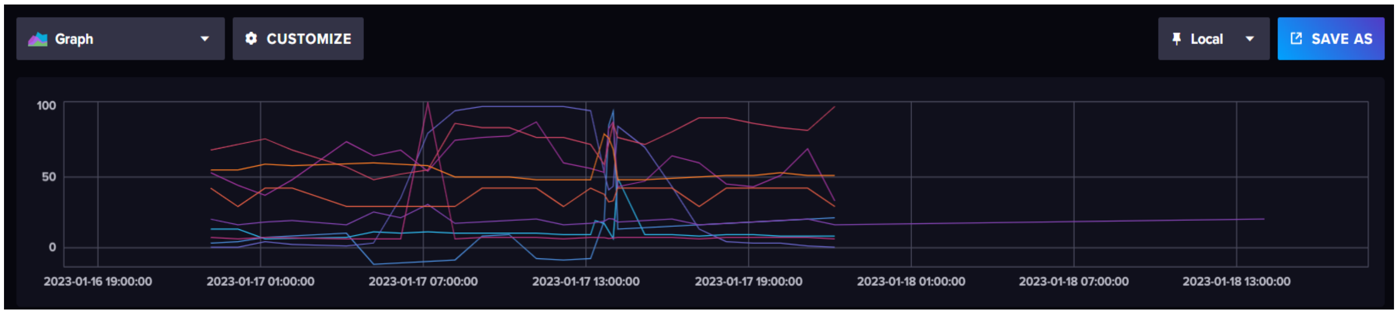
InfluxDB and MongoDB

For influxDB we started by setting up the database on the cloud, choosing the cloud provider, and the region. After that, we created a bucket and got the access tokens and used them in our server. With that, we were able to persist data to the database through our server.

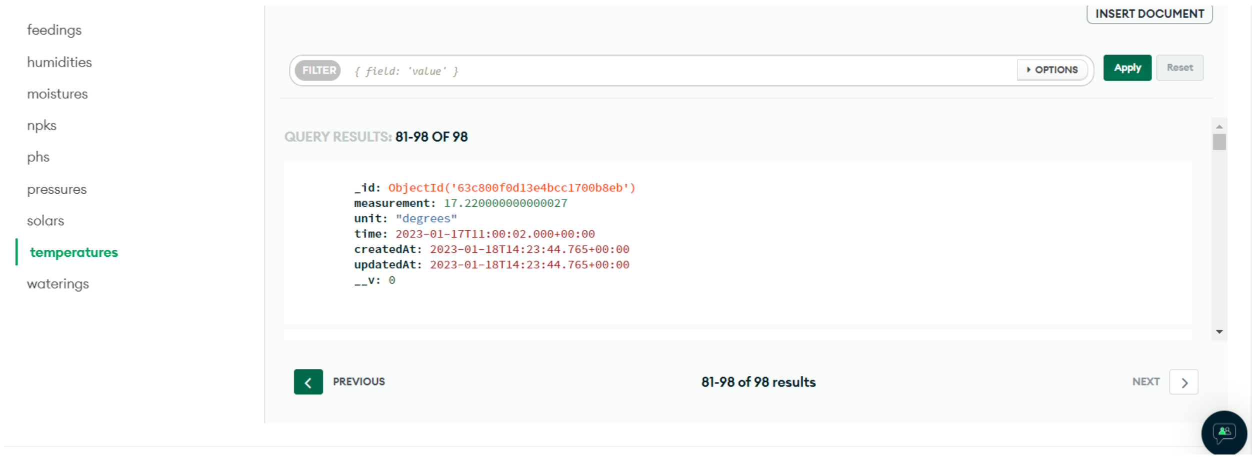


```
var dd = new Date(timestamp);
let point = new Point(type).tag('temperature', 'tempValue').intField('temperature', val).timestamp(dd)
writeClient.writePoint(point)
console.log(point)
```

When the data is stored in the bucket, it is possible to visualise the data using the InfluxDB.



In addition, we had a backup database in which we store all our data in case the influxDB gets compromised. The MongoDB database is deployed to the cloud also. We used Mongoose as an ORM (object data modelling) to manipulate and query our database. The cloud database is deployed in the europe data centre and uses AWS as a cloud provider.

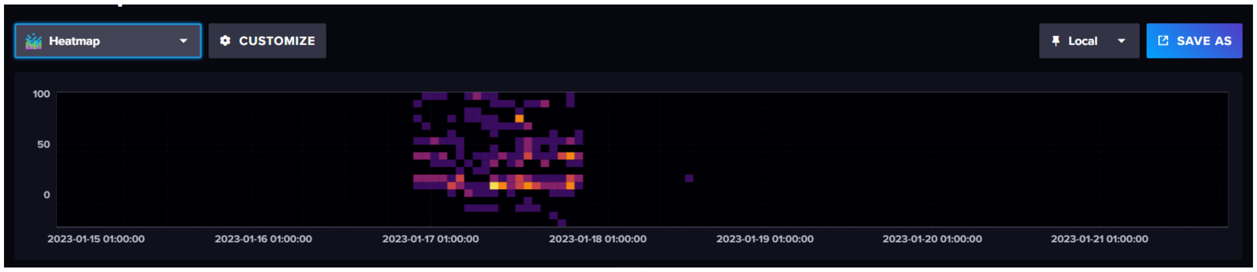


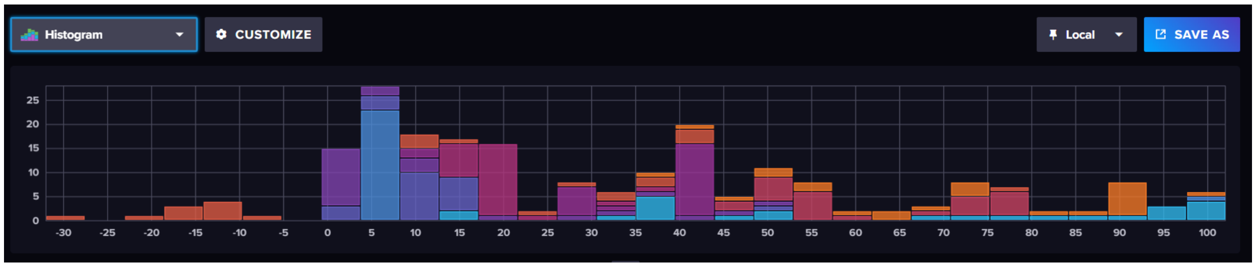
QUERY RESULTS: 81-98 OF 98

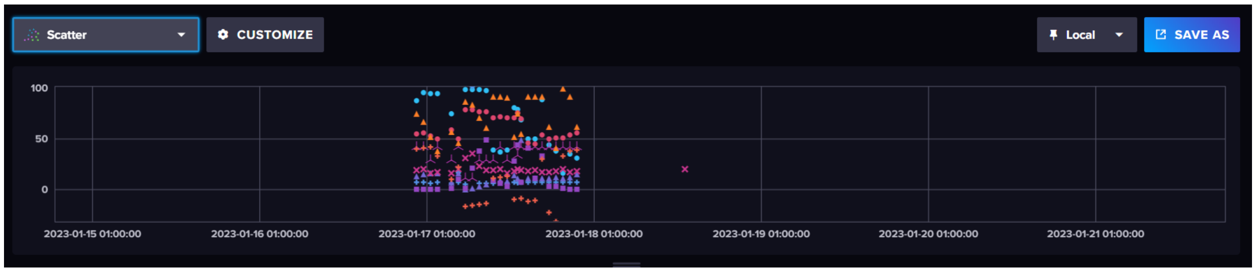
```
_id: ObjectId('63c800f0d13e4bcc1700b8eb')
measurement: 17.220000000000027
unit: "degrees"
time: 2023-01-17T11:00:02.000+00:00
createdAt: 2023-01-18T14:23:44.765+00:00
updatedAt: 2023-01-18T14:23:44.765+00:00
__v: 0
```

81-98 of 98 results

For the visualisation, we made use of the functionalities of InfluxDB to generate graphs, heatmaps, scattered, and histograms etc.







These visualisations bring more insights on data gathered from sensors and enable a more thorough analysis.

Links

Under the following links, the Youtube Description, Docker Hub and the Github pages are available:

GitHub:

https://github.com/AlejoEnriquez2/MQTTAgricultureSystem-master

https://github.com/MehdyKarmouche/iotProjectGate

DockerHub:
https://hub.docker.com/repository/docker/felixnahrstedt/se4gdsmartiotagriculture/general

Youtube:

https://youtu.be/3ITYa1C-KCo