Master's degree in Artificial Intelligence and Data Engineering

Internet of Things project documentation

SMART SUIT

Domenico D'Orsi, Matteo Razzai

A.Y. 2022/2023

GitHub repository: https://github.com/dominicofthebears/SmartSuit

Sommario

1. Introduction

SmartSuit is an IoT system having the objective of increasing the safety of workers in hazardous environments, with possible leakage of gases, radiation, or electromagnetic fields with abnormal values.

For example, considering the gas radon, it is a naturally occurring radioactive gas that cannot be perceived by our senses, this gas, in fact, is odorless and colorless. Through a radioactive decay mechanism, radon is transformed, originating other substances, which once inhaled by breathing, are deposited in the lungs, where they emit radiation that damages lung tissue.

Thinking that it is only one of the possible substances that can be found in some work environments, the needed came up to have some solution to detect and protect oneself from these kinds of harmful substances.

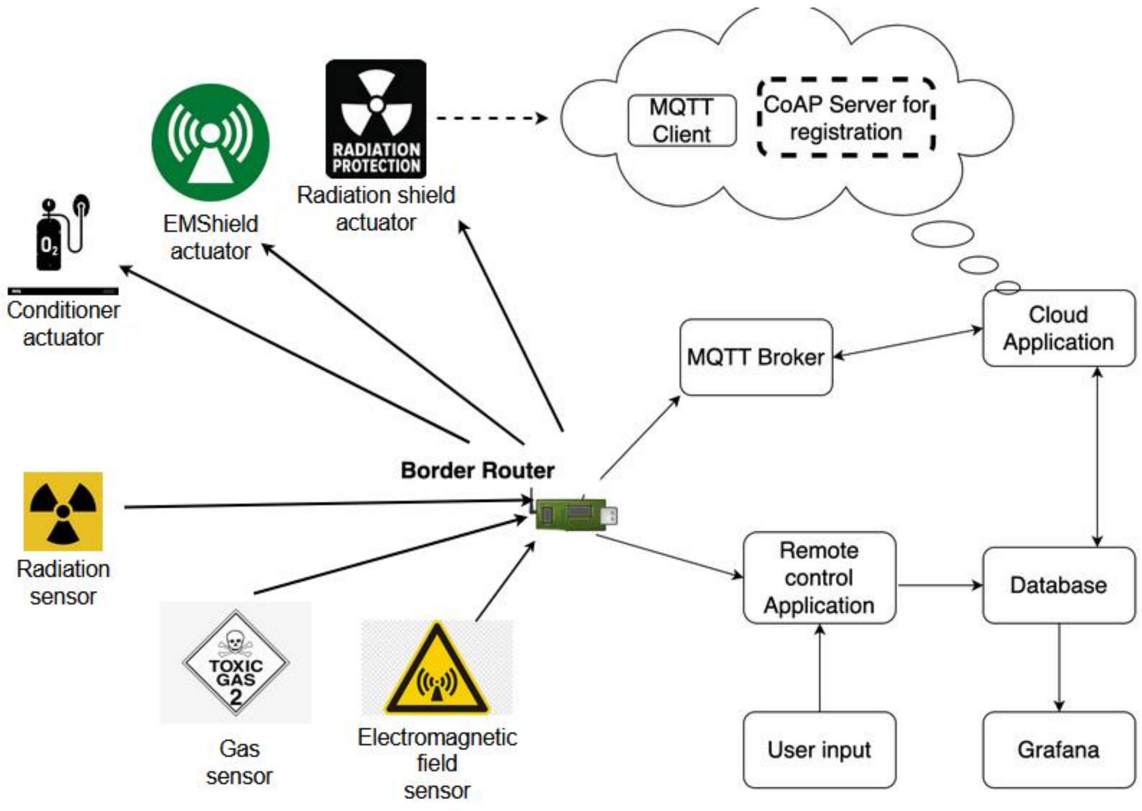
The goal of this system is precisely to provide a system that can determine external toxic agents and then can protect the worker from them.

To obtain that we thought of a smart suit equipped with sensors that can detect such anomalies and actuators that can act to protect the worker. The sensors are used to detect values of gas, radiation, and electromagnetic field in the environment and the actuators are used to manage the defense mechanisms, which are respectively: a conditioner, a shielding from ionizing radiation, and a shielding from electromagnetic fields.

The application provides an automatic mechanism to protect the worker, and in addition to this it also provides a mechanism to allow the user to activate/deactivate defense mechanisms remotely. It provides also the possibility to change the thresholds for which values become critical.

2. Architecture

The overall architecture of the application is defined below.



For what regards the IoT devices, we have two kinds of them: sensors and actuators.

The sensor network is made up of three different types of devices, which are a Radiation sensor, a Gas sensor and an Electromagnetic field sensor. Each of them is part of the data-producing MQTT network, since their role is to retrieve external values (sintetically generated in this case) and then to publish them on the MQTT topics, where then they will be processed by the application.

The actuator network instead receives CoAP calls to exposed resources in order to perform some operation in response to given sensed values. We have one actuator for each type of sensor.

On the other side we have two different components to process the retrieved data. The first one is the Cloud application which subscribes to the MQTT topics, in order to obtain the sensed values and save them in the database, while the other main component is the Remote control application: this part is in charge to retrieve data from the database and process it, performing some CoAP calls to the actuators as a response to dangerous values sensed. Moreover, it also processes the User requestes submitted through Command Line Interface.

2.1 Sensing/Actuating network

2.1.1 Sensors MQTT network

The main components of the MQTT network are the sensors. In our application there are 3 types of sensors:

- **sensor-gas**: The gas sensor is responsible for detecting the level of gas in the environment. The code flashed on the sensor simulates this behavior by generating gas values within a plausible range, which we set between 25 PPM and 40 PPM, with values dangerous to the technician above 35 PPM.
- **sensor-electromagnetic:** The electromagnetic sensor is responsible for detecting the level of electromagnetic field in the environment. The code flashed on the sensor simulates this behavior by generating electromagnetic field values within a plausible range, which we set between 30 V/m and 45 V/m, with values dangerous to the technician above 40 V/m.
- **sensor-radiation:** The radiation sensor is responsible for detecting the level of radiation in the environment. The code flashed on the sensor simulates this behavior by generating radiation values within a plausible range, which we set between 90 millisievert and 105 millisievert, with values dangerous to the technician above 100 millisievert.

To simulate the most real behavior possible, we decided to make the values increase by two units every 7 seconds, and then passed the maximum limit make it decrease in turn by 2 units every 7 seconds until it reaches the minimum limit imposed in the range of possible values and starts rising again.

Each type of sensor was deployed on a single dongle, it might be useful to increase the number of sensors for each type of item to be sensed, since the danger is high it is necessary to be sure to detect all kinds of abnormal values, and with more sensors we can better handle the failure of a single sensor. Regarding the area over which the sensors are distributed, the number of sensors used in this project could be sufficient to carp the outlier value near the worker.

The three sensors publish respectively on the topic “gas”, “electromagnetic” and “radiation” every 7 seconds.

2.1.2 Actuators CoAP network

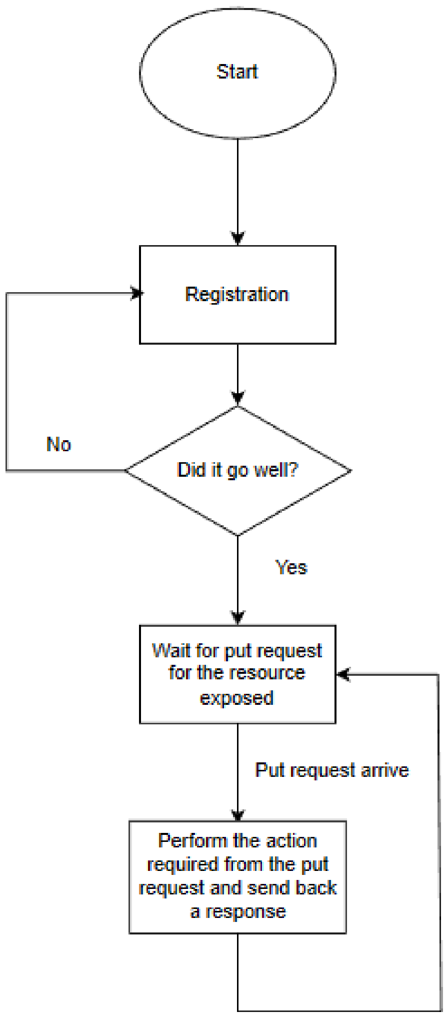
The actuator as the first action must register itself with a CoAP Registration server so that it can be reached by the remote control application, which at first is not aware of its IPv6 address. To register each actuator sends a JSON containing a key-value pair describing the actuator type (Example: {"type" : " gas"}).

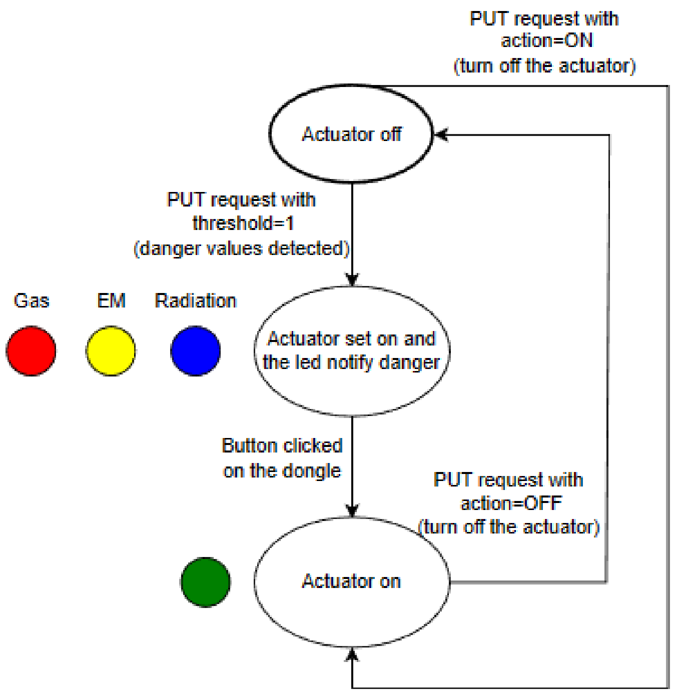
There are three different actuators:

- **actuator conditioner:** The actuator is used to control the air conditioner, which is used to clean the air after detecting a gas value in the air higher than normal. The led to notify the danger is set to red.
- **actuator electromagnetic shielding:** The electromagnetic shielding actuator is used to control the shielding to protect the worker from anomalous electromagnetic field. The led to notify the danger is set to yellow.
- **actuator radiation shielding:** This actuator controls another type of shielding, that which is used when abnormal values of radiation are detected in the environment. The led to notify the danger is the blue one.

When an abnormal value is sensed, the actuator is turned on and the danger is represented by the specific led for the type of value sensed, the technician wearing the suit and seeing the danger led on a sensor, understands the type of danger and has to press the button on that specific actuator to notify that he/she has received the danger message and is okay despite the dangerous situation. After the button is clicked, the green LED will remain on, notifying the actuator that it is on. If no led is on, it means the situation is normal and the actuator is off.

The workflow of the actuators is visible on the left and the behavior, of the actuator and the LEDs of the dongle on which it is running, after receiving a put request is visible on the right:


Start
Registration
Did it go well?
No
Yes
Wait for put request for the resource exposed
Put request arrive
Perform the action required from the put request and send back a response


PUT request with action=ON (turn off the actuator)
Actuator off
PUT request with threshold=1 (danger values detected)
Gas
EM
Radiation
Actuator set on and the led notify danger
Button clicked on the dongle
PUT request with action=OFF (turn off the actuator)
Actuator on

2.2 Application side

The application side is in charge to store, process and visualize all the retrieved data and act in consequence if danger is detected by making actuators performing some specific actions. In the following paragraphs each specific part of this side is defined. Each of these classes is implemented as a singleton and extends Runnable, in order to be used along with a ScheduledExecutor in the entry point on the application.

2.2.1 Remote control application

The application core part is the remote control application, which is made up of three different components to manage the workflow:

1. **Periodic data retrieval:** this first part stores as first the thresholds defined by default and the potentially changed by the used for each type of sensed informations. The actual running part, instead, retrieves the most recent data for each type in order to check whether we have some danger I.E. some values which goes over the threshold. In this first case, we set another variable to indicate that danger is present, then we advertise the actuator to turn on (in case it is not already in this state). On the other hand, if there's no danger and the value sensed is below the 30% of the threshold, we automatically turn off the actuator to save energy. This operation will be perfomed iteratively each 7 seconds, which is synchronized with the data production from the sensors and also a small enough value required by the criticity of the application.

2. **Coap client**: here we store the information about the presence at the moment of danger for the three types of sensors, but the main task of this class is to send commands to the actuators. The first check is used to avoid turning off the actuators in case of danger, otherwise we send to the specific actuator an action to perform, encoded as "ON" or "OFF" by using a CoAP PUT request.

3. **Command line interface**: at last, we use this class in order to process the user inputs. The action that can be performed are mainly 4: print the help list, select new thresholds for the sensed values, change the status of the actuators (only in case the specified one is different from the state the actuator is already in) and check their status. Those actions are performed by simply inserting the corresponding number specified by the help command, in the way specified below

   1 - Help list

   2 - Select a new value for the radiation threshold

   3 - Select a new value for the gas threshold

   4 - Select a new value for the electromagnetic threshold

   5 - Change the status of the radiation shield

   6 - Change the status of the oxygen flow system

   7 - Change the status of the electromagnetic shield

   8 - Check the actuators status

2.2.2 Cloud application

The other main component of the application side is the Cloud Application. This component takes care of two different communications held with the sensors and the actuators.

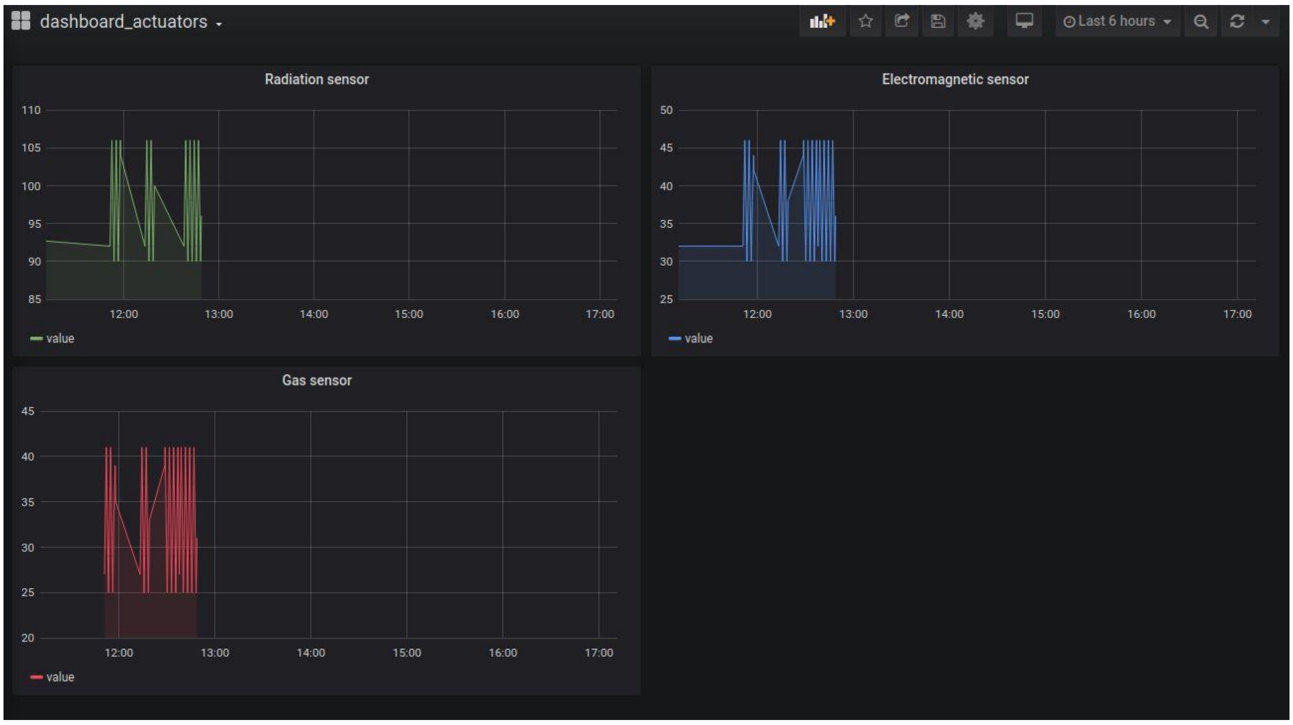
1. **MQTT collector**: this, along with the sensors and the broker, is the other end of the MQTT network. In particular, it acts as the subscriber of the three queues used by the sensors to send data over (topics are "radiation", "gas" and "electromagnetic").
   When new data is received, the "*messageArrived*" method is invoked, the retrieved payload is processed, and the obtained data is saved. In case of connection lost, instead, an error message is printed.

2. **CoAP registration**: the CoAP registration component is a Server that exposes a resource (RegistrationResource) which defines only a method to handle a POST request, since we are creating a new resource by this call. This resource listens for actuators calls wanting to register themselves to the application by sending a payload containing only the type of the actuator, since the IP address is retrieved by the request itself. The actuator will then be inserted in the database in the "OFF" status. It will remain active for the entire time the application is running.

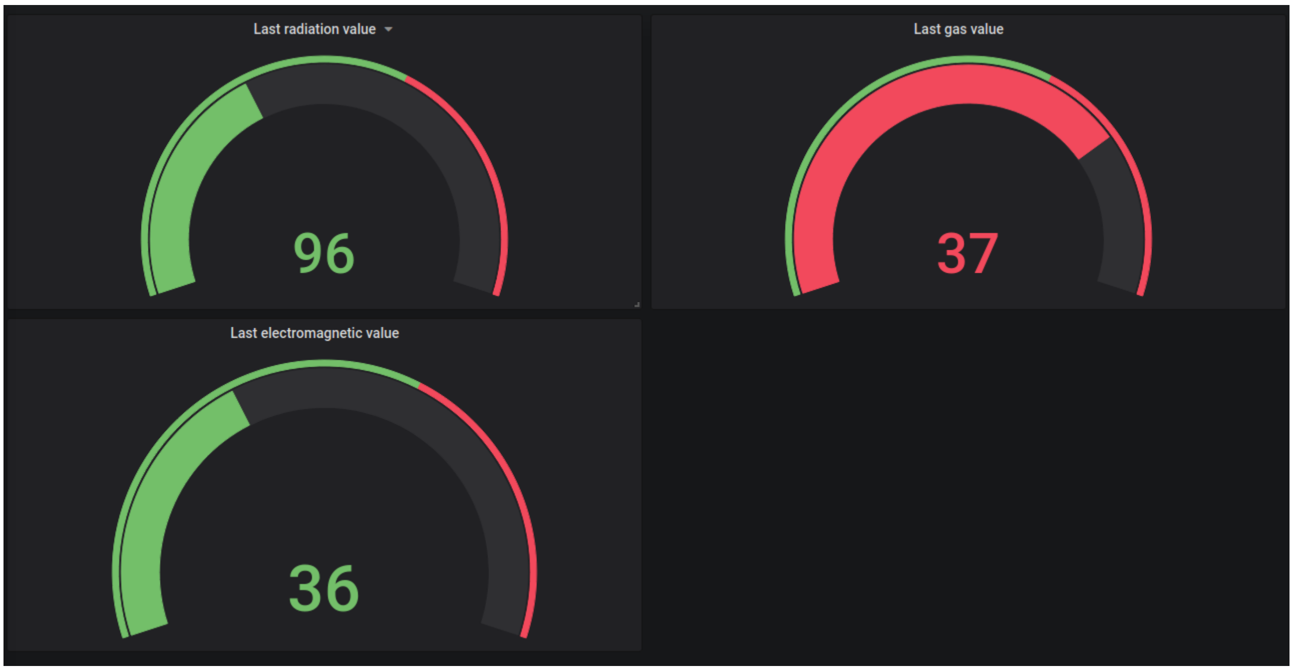
2.2.3 Grafana

In order to monitor the ongoing of the sensed data, we attached to the homonimous database table a Grafana instance. This allowed us and a potential user to monitor the external environment behaviour and the way it changes. In particular, we created two main groups of dashboards

- **Values fluctuation** to see how the values change over time



- **Most recent value** to check whether we are in a danger situation for one of the three sensors



This dashboard is reachable by looking up to the http://localhost:3000/ address on a web browser

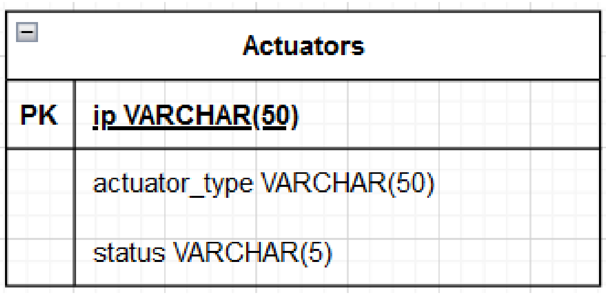
3. Data encoding

When it comes to choosing a data encoding language, we had mainly two options: XML and JSON. We evaluated the possible pros and cons, and got to the decision that JSON would be more suitable for our application for several reasons, such as less verbosity and simpler messages which allowes us to use less bandwidth when exchanging data from one side to the other.

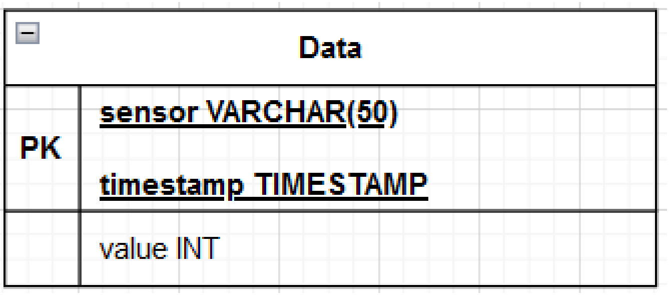
This also made the payload management more simple, since there are plenty of libraries that allow to build up and parse JSON messages with minimal effort.

4. Database

The database component is used to store two different pieces of information in two specific tables inside of an SQL Database, in order to be retrieved later. Those two aspects are the data retrieved by the sensors and the actuators registered to the application, both retrieved from the Cloud Application and stored by using the **DatabaseAccess** class, which acts as a DAO. The two tables are composed in the following way:



	Actuators
PK	ip VARCHAR(50)
	actuator_type VARCHAR(50)
	status VARCHAR(5)



	Data
PK	sensor VARCHAR(50) timestamp TIMESTAMP
	value INT

For what regards the actuators, we identify them by using the IP address, since it will be for sure different for each of them thus uniquely identifying them, then we have the type of the actuator and the status, which can be “ON” or “OFF”. The data, instead, is identified by the sensor who provided ita long with the timestamp of the data insertion, since this as well will identify each information, then we have the sensed data in INT format.

The class that manages insertions and retrievals from the tables is, as previously said, DatabaseAccess: it provides a DBMS connection towards our SQL instance, and 5 different methods/queries

- **resetActuators**: used to reset the actuators tables before the start of the CoAP server. Query: DELETE from actuators

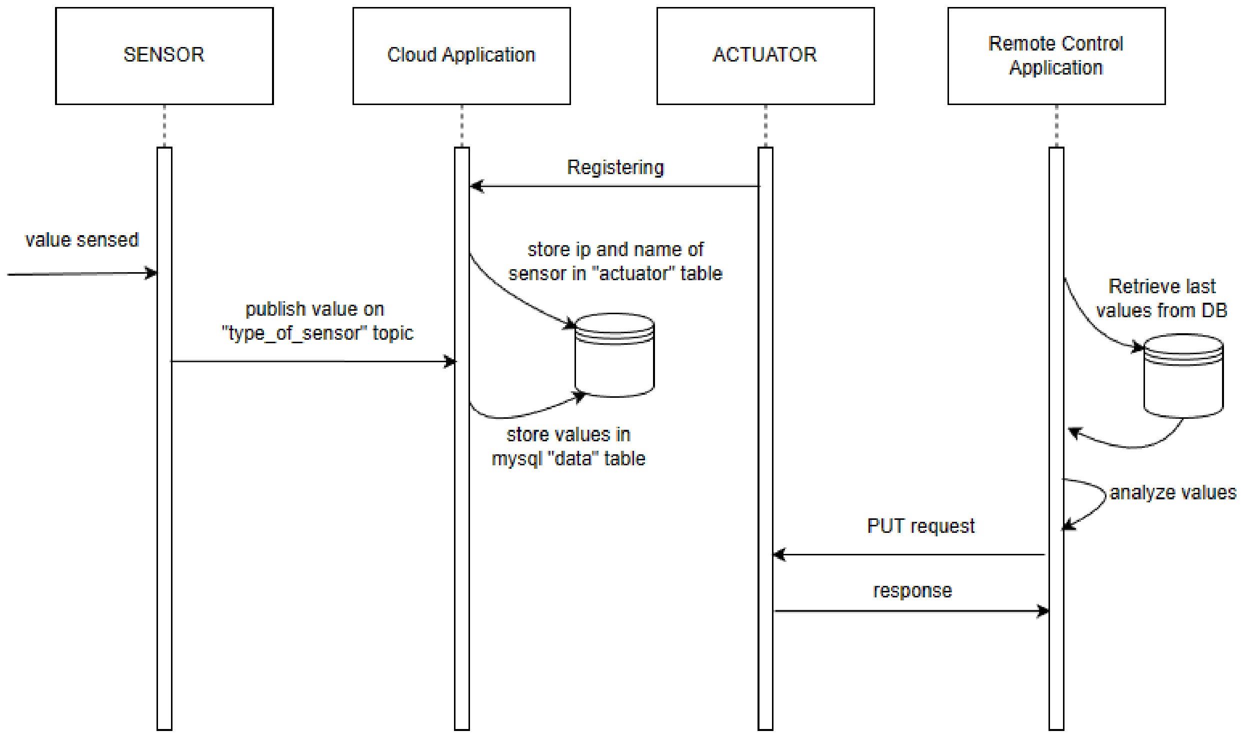
- **updateActuators**: used to insert the registered actuators and to change their status when they get turned on/off
  Query: REPLACE INTO actuators (ip, actuator_type, status) VALUES(?,?,?)
- **retrieveActuator**: used to retrieve the status and the ip address of a given actuator
  Query: SELECT ip, status FROM actuators WHERE actuator_type = ?

- **insertData**: used to insert the data retrieved from the MQTT queues, sensed by the IoT devices
  Query: INSERT INTO data (value, sensor) VALUES(?,?)

- **retrieveData**: used to retrieve the most recent value for each distinct type of sensor
  Query: SELECT sensor, value FROM data WHERE (sensor, timestamp) IN (SELECT sensor, MAX(timestamp) FROM data GROUP BY sensor)

5. Workflow

The execution flow is briefly described in the following diagram:


SENSOR
Cloud Application
ACTUATOR
Remote Control Application
Registering
value sensed
store ip and name of sensor in "actuator" table
publish value on "type_of_sensor" topic
Retrieve last values from DB
store values in mysql "data" table
analyze values
PUT request
response

6. Use cases

Our application has been thought to be used on working suits for people who operate in dangerous fields and environment (that’s where the name comes from), where both the sensors and the actuators are placed in specific part of the working suit where they can both properly work and also not be exposed to the possibility of getting destroyed/compromised. These use cases can go from military battlefield to firefighters suits (with proper sensors adaptation).

Following, we show some possible specific usages of our system in a more detailed way.

Dangerous nuclear plant exploration

The first possibility is to mount the sensors on the suit for operators of nuclear plants, which are commonly known as environments where both ionic radiations and electromagnetic ones may be present, thus eliminating the necessity to use more complicated and more cumbersome instruments, while our application may both advertise them of the present danger and at the same time monitor how the environmental values change over the area. Data then can be also shared with authorities and perform even more drastic operation in case of danger.

Mines exploration

When it comes to explore several types of mines, we know that various dangerous situations may occur, such as the presence of dangerous gasses or the presence of radioactive materials (such as uranium).

In those context, our application may simplify the work of an operator, helping him to determine which parts of the mine may be dangerous and wheter the entire mine is operable or not, needing some operations in order to make it exploitable.