University of Pisa

Master's Degree in

Artificial Intelligence and Data Engineering

Internet of Things

SmartWellness

Candidates

Minniti Federico

Del Seppia Matteo

ACADEMIC YEAR 2021/2022

1. INTRODUCTION 3
2. DEPLOYMENT 4
2.1 SWIMMING POOL 4
2.1.1 pH control 4
2.1.2 Chlorine control 5
2.2 GYM 6
2.2.1 Air conditioning 6
2.2.2 Light regulation 7
2.3 STEAM BATH 7
2.3.1 Humidifier 8
2.3.2 Access control 8
2.4 COOJA DEPLOYMENT 9
3. DATA MANAGEMENT AND VISUALIZATION 10
3.1 DATA ENCODING 10
3.2 JAVA COLLECTOR 11
3.3 LOGS 11
3.4 DATABASE 12
3.5 GRAFANA 12

1. INTRODUCTION

The goal of the project is to create a smart area to support wellness activities like swimming, taking a steam bath or working out at the gym. The application that has been developed is capable of:

- automatic remote-control of actuators (e.g., the air conditioning system) based on values collected by sensors;
- detecting when the actuators are being used in manual mode, and behaving accordingly;

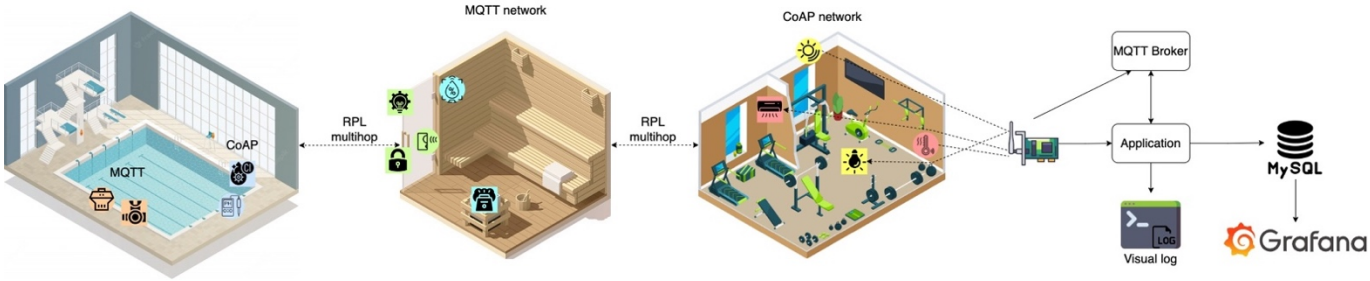
Half of the devices in the network use the CoAP protocol to expose their resources, while the other half use the MQTT protocol. All our devices have been connected to both a sensor and an actuator and can physically display the status of the actuator (e.g., ON or OFF) through their LEDs.

A Java collector has been developed to observe the resources of the devices and control their actuators. The collector has a command line interface capable of making the system's behavior dynamic, as variables that control the actuators can be changed live.

While all six devices were developed and tested to work on real nodes, we had only six Launchpad CC2650 available. Given that one node must act as a border router, only five of the six devices we originally developed were deployed on real Launchpad CC2650 nodes.

(The project code is available here: https://github.com/federicominniti/SmartWellness)

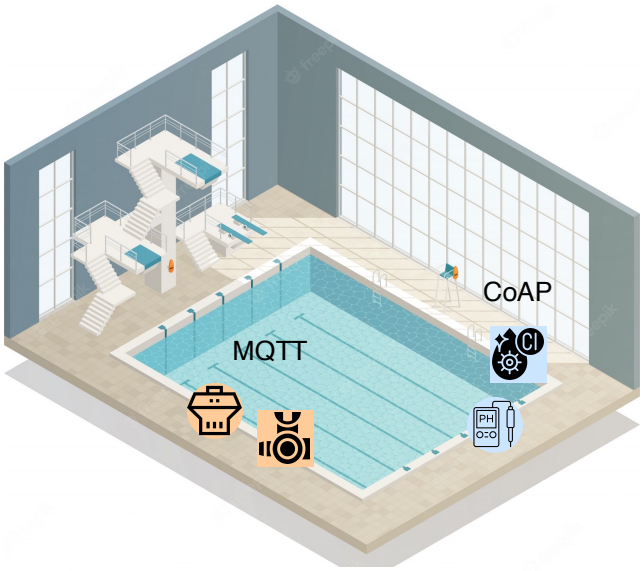
2. DEPLOYMENT



From now on we will consider the deployment on real sensors.

All the devices use their LEDs to communicate the current state of the actuator they're controlling. In addition to this, every device will show a blinking red LED while the connection to the border router or to the collector/broker has not yet been completed.

2.1 SWIMMING POOL



2.1.1 pH control

The pH control device exposes two CoAP resources: a pH sensor and a buffer regulator. The pH is to be kept in the range 7.2-7.8 for the chlorine to be most effective as water sanitizer. Moreover, when the pH

is under 7 the water has acid characteristics and can cause irritation to the skin and the eyes of people. A buffer solution is usually used to increase the pH of swimming pools.

When the pH is under the minimum threshold (under 7.2) the buffer regulator will automatically be activated by the Java collector. The buffer regulator will restore the normal level of the pH inside the swimming pool, and after that it will be switched OFF by the collector again. When the buffer regulator is ON, both the LEDS of the device will for 1 second every 7 seconds. Otherwise, the green LED only will blink for 1 second every 7 seconds.

The left button of the device can be pressed to manually switch ON/OFF the buffer regulator (depending on the current state of the regulator, the operation will always invert the current state of the actuator). The Java collector will not send any commands to the buffer regulator until the button is pressed again, to respect the manual mode.

2.1.2 Chlorine control

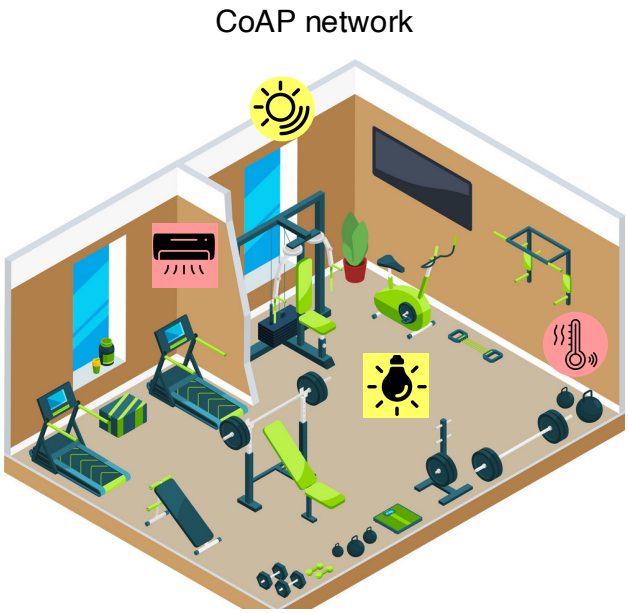
The chlorine control device is linked to a chlorine sensor and a chlorine regulator. The device will notify of the chlorine levels through the MQTT broker, publishing messages on the *ppm* topic. It also receives commands from the Java collector on the *chlorine_regulator* topic.

Chlorine is essential to sanitize a swimming pool. Chlorine level in swimming pools must be between 1ppm and 3ppm. Chlorine sensors on the market have a sensing resolution in the order of ppb. When the chlorine level is too low the Java collector will notify to the device the command to activate the chlorine regulator, which will restore the level of chlorine in the pool. When the level of chlorine has reached at least 2.5ppm, the chlorine regulator will be switched off by the Java collector.

When the chlorine regulator is ON, both the LEDs of the device will blink for 1 second every 7 seconds. Otherwise, the green LED only will blink for 1 second every 7 seconds.

The left button of the device can be pressed to manually switch ON/OFF the chlorine regulator (depending on the current state of the regulator, the operation will always invert the current state of the actuator). The Java collector will not send any commands to change the state of the chlorine regulator until the button is pressed again, to respect the manual mode.

2.2 GYM



2.2.1 Air conditioning

The air conditioning control in the gym is supported by a CoAP device exposing two resources: a temperature sensor and an air conditioning system.

The functioning temperature of the air conditioning system can be changed remotely by the Java collector through a PUT request on the AC resource. By default, the AC temperature is set to 18.

When the temperature in the gym is over a threshold (in the default case, 20 degrees) the AC system will be switched ON by the Java collector. Then again it will be switched OFF when the gym temperature will reach the desired level.

When the AC system is ON, both the LEDs of the device will blink for 1 second every 7 seconds. Otherwise, the green LED only will blink for 1 second every 7 seconds.

The left button of the device can be pressed to manually switch ON/OFF the air conditioning (depending on the current state of the AC, the operation will always invert the current state of the actuator). The Java collector will not send any commands to change the state of the AC system until the button is pressed again, to respect the manual mode.

2.2.2 Light regulation

We assume the gym to be fully capable of being illuminated by the sunlight, if the meteorological conditions make it feasible. To adapt the use of illumination in the gym to the level of sunlight, a device has been connected to a lux sensor and the room lightning system.

By default, when the sensed lux is over 1500, the light will be switched OFF by the Java collector because the sunlight is capable of fully illuminating the gym. When the sensed lux is between 350 and 1500 the lightning system is set to LOW by the Java collector, because the sunlight is not capable of fully illuminating the gym. When the sensed lux is lower than 350, the lightning system is switched to HIGH.

When the light is switched to OFF, the green LED will blink for 1 second every 7 seconds. When the light is switched to LOW, both the green and the red LEDs will blink for 1 second every 7 seconds. When the light is switched to HIGH, the red LED will blink for 1 second every 7 seconds.

The left button of the device can be pressed to manually switch HIGH/OFF the light. If the light is LOW, the device will switch the light OFF, while in any other case the light will be simply inverted. The Java collector will not send any commands to change the state of the gym's lightning until the button is pressed again, to respect the manual mode.

2.3 STEAM BATH

2.3.1 Humidifier

In steam baths the level of humidity must be constant around the 90%-95%. A device associated both with a humidity sensor and a humidifier has the duty to sense the humidity in our steam bath and receive commands from the Java collector to periodically switch ON/OFF the humidifier to maintain the desired level of moisture in the room.

The device uses the MQTT protocol and publishes data about humidity on the *humidity* topic. It also receives commands from the Java collector on the *humidity_control* topic.

By default, when the sensed humidity is over under 87%, the humidifier will be switched to ON by the Java collector to increase the moisture in the air. When the sensed humidity goes over 92%, the humidifier will be switched to OFF by the Java collector, and the moisture level will start to decrease as time passes.

When the humidifier is ON, both the LEDs of the device will blink for 1 second every 7 seconds. Otherwise, the green LED only will blink for 1 second every 7 seconds.

The left button of the device can be pressed to manually switch ON/OFF humidifier (depending on the current state, the operation will always invert the current state of the actuator). The Java collector will not send any commands to change the state of the humidifier until the button is pressed again, to respect the manual mode. When the humidifier is switched to ON, the maximum level of moisture reachable has been set to 99%.

2.3.2 Access control

To regulate the access to the steam bath and avoid overcrowding in the room, a special device has been set at the entrance of the steam bath. The device is connected to a presence sensor, the entrance door and a R/G/Y light above the entrance.

The device uses MQTT and publishes periodically the number of people inside the steam bath on the *number_of_people* topic. It also receives command from the Java collector on the *access_regulator* topic.

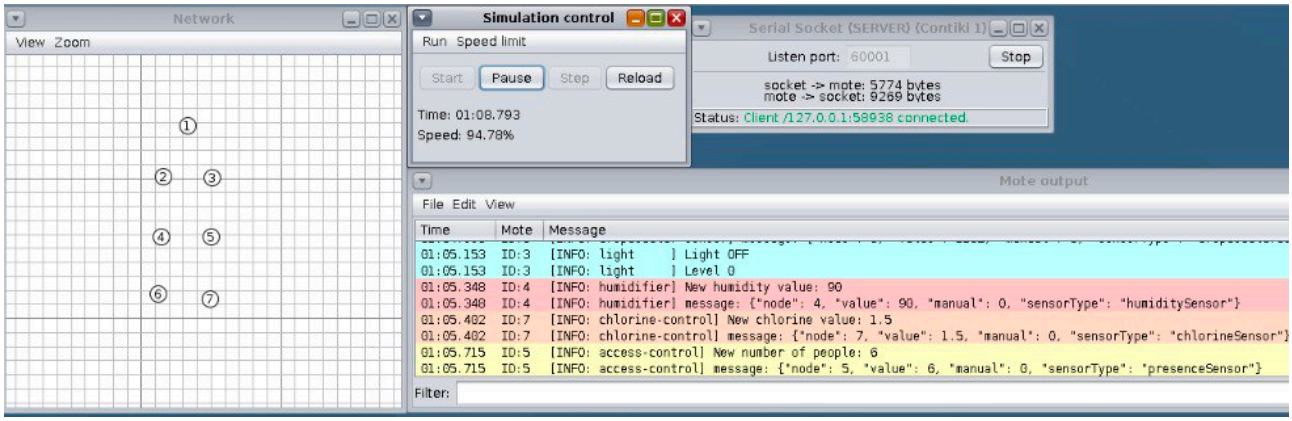
By default, when the number of people in the room is at least 15, the Java collector will switch the door to *CLOSED* and the light outside to *RED*. When the number of people is between 10 and 15, the collector will keep the door *OPEN* but will change the light color to *YELLOW*, to signal possible overcrowding in the near future. When the number of people is lower than 10, Java will keep the door open and set the light to *GREEN*.

When the light is *GREEN*, the green LED of the device will blink for 1 second every 7 seconds. When the light is *RED*, the red LED of the device will blink for 1 second every 7 seconds. When the light is *YELLOW*, both the LEDs of the device will blink for 1 second every 7 seconds.

The left button of the device can be pressed to enter the manual mode, which immediately inverts the current state of the door and sets the light to *GREEN* or *RED* accordingly. If the light was *YELLOW* when the button was pressed, it will stay *YELLOW* until the end of the manual mode. After exiting the manual mode, the Java collector will have again the control of the actuators and set their status according to the value of the presence sensor.

2.4 COOJA DEPLOYMENT

For debugging purposes, the devices were first deployed on Cooja. We tried to keep the topology of the network as real as possible, with the rooms one in front of the other and the border router being in the range of only the first room's two sensors. This forced the devices to form a flat multi-hop WSAN.

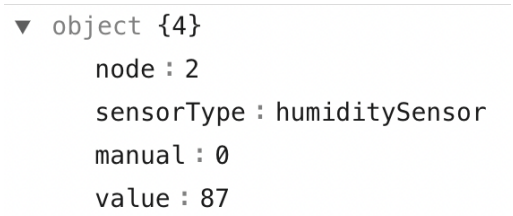


1. Border router
2. Air conditioning (CoAP)
3. Light regulation (CoAP)
4. Humidity regulation (MQTT)
5. Access control (MQTT)
6. pH regulation (CoAP)
7. Chlorine regulation (MQTT)

3. DATA MANAGEMENT AND VISUALIZATION

3.1 DATA ENCODING

While it is true that our devices use CoAP and MQTT to communicate with the collector, this doesn't reveal anything about the data encoding we have used. Given the well-known constraints of our devices in terms of packet size (802.15.4 has a MTU of 127 bytes), we decided to encode our data in the JSON format. Specifically, all our messages have this format:



This allows us to have around 75-80 bytes of payload, hopefully avoiding fragmentation of the IPv6 datagram and less traffic in the network. If we had decided to use an XML encoding, our payload would have looked like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Sample>
    <node>2</node>
    <sensorType>humiditySensor</sensorType>
    <manual>0</manual>
    <value>87</value>
</Sample>
```

with an average length of 150 bytes, which would have resulted in fragmentation for sure, probably doubling the overall network traffic.

3.2 JAVA COLLECTOR

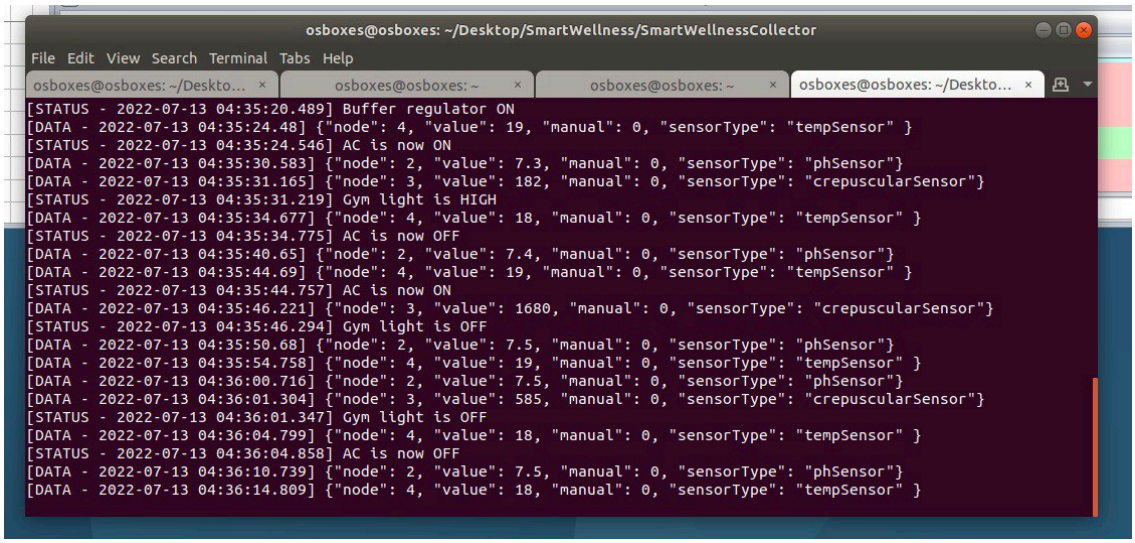
The Java collector, besides managing the data coming from the sensors and the logs, features a CLI which allows the system's manager to quickly get information about the last samples measured by the sensors and to modify the thresholds regulating the de/activation of the actuators. In the case of the air conditioner, the collector also allows the system's manager to set the temperature of the AC system, as specified previously in the description of the node.

```
SmartWellness
Instert a command:

--- GYM ---
get_gym_temperature                          <--- get current temperature in the gym
set_gym_ac_temperature [NUM]                 <--- set AC temperature in the gym
get_gym_ac_temperature                       <--- get AC normal level temperature in the gym
set_gym_ac_upper_bound                       <--- get AC upper bound in the gym
get_gym_lux                                  <--- get gym current lux
set_gym_crepuscular_intermediate_level [NUM] <--- set gym intermediate level of lux for the crepuscular sensor
set_gym_crepuscular_max_level [NUM]          <--- get gym max level of lux for the crepuscular sensor

--- POOL ---
get_pool_ph_level                            <--- get current pool ph level
set_pool_ph_lower_bound [NUM]                <--- set pool lower bound for the ph level
set_pool_ph_normal_level [NUM]               <--- set pool ph normal level
get_pool_chlorine_level                      <--- get current pool chlorine level
set_pool_chlorine_min_ppm [NUM]              <--- set pool minimum value of PPM for the chlorine sensor
set_pool_chlorine_max_ppm [NUM]              <--- set pool max value of PPM for the chlorine sensor

--- STEAM BATH ---
get_steam_bath_people_number                 <--- get current number of people in the steam bath
set_steam_bath_intermediate_number [NUM]     <--- set steam bath intermediate people level
set_steam_bath_max_number [NUM]              <--- set steam bath max number of people
get_steam_bath_humidity_level                <--- get current humidity level in the steam bath
set_steam_bath_humidity_min_percentage [NUM] <--- set steam bath humidity min percentage
set_steam_bath_humidity_max_percentage [NUM] <--- set steam bath humidity max percentage
```

3.3 LOGS

The system's events are showed live in the logs, which can be seen via the `tail -f info.log` command. Each entry in the log has a timestamp, a body (message) and a tag:

- *DATA*: this tag is used to identify data samples received from one of the sensors;
- *STATUS:* this tag is used to identify messages from the collector about the state of one of the actuators;
- *ERROR*: this tag is used to identify messages of error from the collector

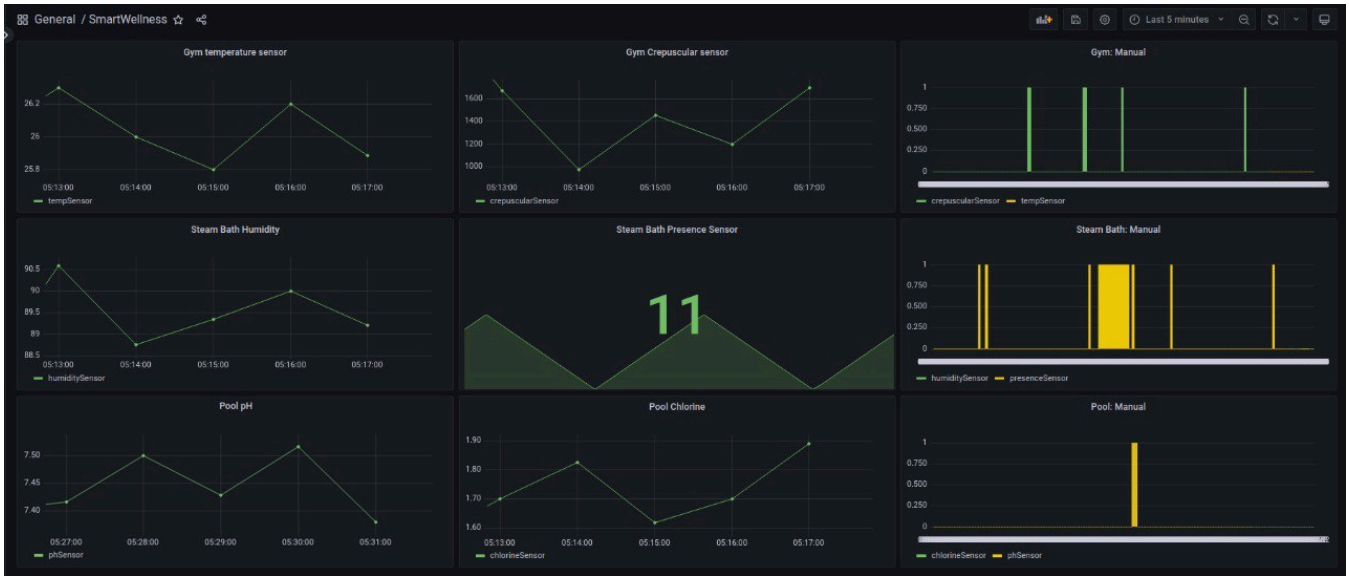


3.4 DATABASE

Each data sample has been saved in a MySQL database. A table for the samples has been created:

```
CREATE TABLE `DataSamples` (
    `sensorType` VARCHAR(30) NOT NULL,
    `timestamp` TIMESTAMP NOT NULL,
    `node` INT NOT NULL,
    `value` FLOAT NOT NULL,
    `manual` INT NOT NULL,
    PRIMARY KEY (`sensorType`,`timestamp`)
);
```

3.5 GRAFANA

Saving the data to the database has allowed us to use Grafana for data visualization. Specifically, a Grafana dashboard has been configured to easily check the values from the sensors.

Each row in the dashboard shows the time series of the average values minute by minute of the two sensors within a certain environment (pool, gym, steam bath). The third plot shows the time series of the *manual* mode for the two devices of a certain environment, without aggregation.