UNIVERSITÀ DI PISA

COMPUTER ENGINEERING

INTERNET OF THINGS

A.A. 2022-2023

SMART SKI SLOPE MONITORING

STUDENTS:

FEDERICO CAVEDONI

NICOLO SALTI

CONTENT

1 INTRODUCTIONS

In contrast to the common belief, a ski slope to be fully operational and safe needs considerable attention, both from the skier's safety and from the ski slope itself.

The aim of this project is to develop an automatic system, which is able to collect key data from the environment around the ski slope and promptly respond to them to ensure the practicability of the slope and the safety of the skiers.

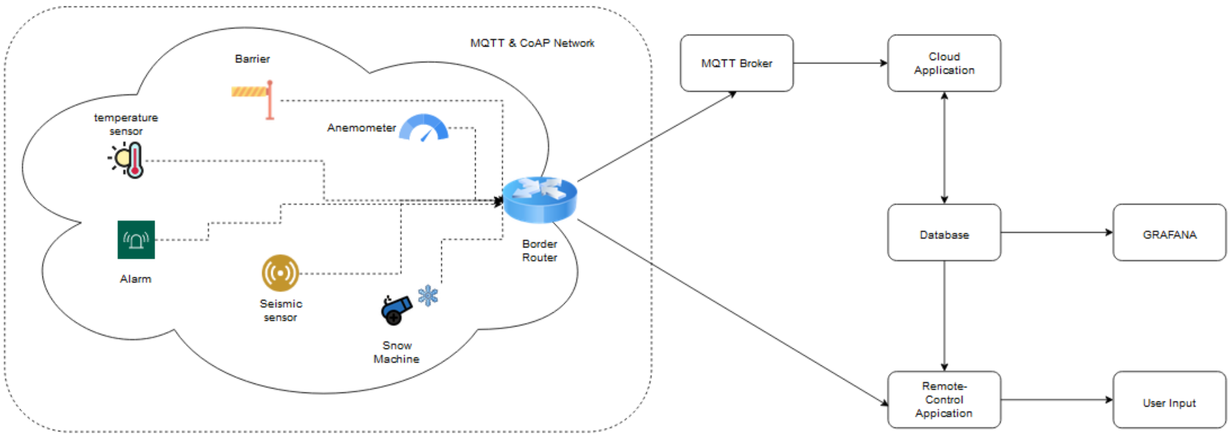
In this case, automation plays a key role, since to guarantee the safety of skiers it is essential that the system reacts to the dangers that a ski slope can present.

Furthermore, in this case the devices will be installed along the entire lane, making manual interaction with them a considerable cost which is eliminated thanks to their automatic and remote management.

The system provides both an automatic mode, in which the network automatically responds to the collected data, and a manual system in which the user manually manages the track.

2 ARCHITECTURE

The system is composed by a network of IoT devices that, depending on their role, implement either MQTT or CoAP protocol. Sensors, which collect data from the external environment, use MQTT to publish them, then, a Cloud Application stores them on the database. Actuators expose their functionalities which are triggered by the control logic implemented by the Remote control application, using the CoAP protocol. A border router is used to allow external access.



In the following chapters we will take a deeper look at the devices located in the network.

2.1 CoAP Devices

In our application we have developed three different types of actuators, which uses CoAP as application protocol. Each of the three actuators implements a resource to expose their functionalities, that can be used by the Remote control application to trigger them, while the application implements a resource to let the actuators to register as CoAP clients.

2.1.1 Snow Machine

The **snow machine** is an actuator which is able to spray the snow on the ski slope. It can be automatically turned on by the Remote control application, depending on the value of the **temperature**, but only if the **barrier** is closed, otherwise there may be skiers on the ski slope. It can be also manually turned on or off, through a button or through the CLI.

2.1.2 Barrier

The **barrier** is an actuator which allows or not skiers to get on the ski lift. It can be automatically open depending on the value of the **wind** speed. It can be also manually closed or opened through a button or through the CLI.

2.1.3 Alarm

The **alarm** is an actuator which informs skiers about the risk of avalanches. It can be automatically turned on depending on the value that are sensed by the **seismic sensor**, the **temperature sensor** and the **anemometer**, in particular when the Remote control application realizes that two out of three sensed values exceed a certain threshold. When the alarm is turned off, the **barrier** is automatically closed.

2.2 MQTT Devices

2.2.1 Temperature Sensor

The **temperature sensor** is able to sense the value of the temperature and send it to the Cloud application, which stores it in the database. The code flashed on the sensor simulates the behaviour of the temperature, which has a mean value of 0 degree and, with a certain period, a value in the interval [-2, +2] is added to the previous sensed value.

2.2.2 Anemometer

The **anemometer** is a particular sensor which is able to sense the speed of the wind. The code flashed on the sensor simulates the behaviour of the wind speed: with a certain probability the value of the speed is increased by a value in the interval [1, 10], and when it reaches a certain threshold, it is decreased by a value in the interval [10, 30].

2.2.3 Seismic Sensor

The **seismic sensor** is able to detect the frequency of seismic vibrations. The code flashed on the sensor simulates the behaviour of the frequency: with a small probability it is increased by a value in the interval [25, 35] and when it exceeds a certain threshold, the frequency is decreased by a value in the interval [20, 30].

For the three sensors, in order to reduce the energy consumption, we have implemented a sort of **adaptive sampling**, which consists of increasing the sampling period of the sensors under certain conditions, with a sampling coefficient which has a maximum value of 4. We have also implemented some commands, through the CLI, that can manipulate the sensed values, with the sole purpose of showing the functionalities of the actuators.

3 Implementation

3.1 Data Encoding

For data encoding we decided to use the **JSON** format. This choice is due to the fact that JSON is a lighter and more compact data encoding language, unlike XML which has a more complex and verbose structure and was not well compatible with constrained devices such as those we are dealing with.

Therefore, every data generated by the sensors is transmitted to the cloud application as a JSON document.

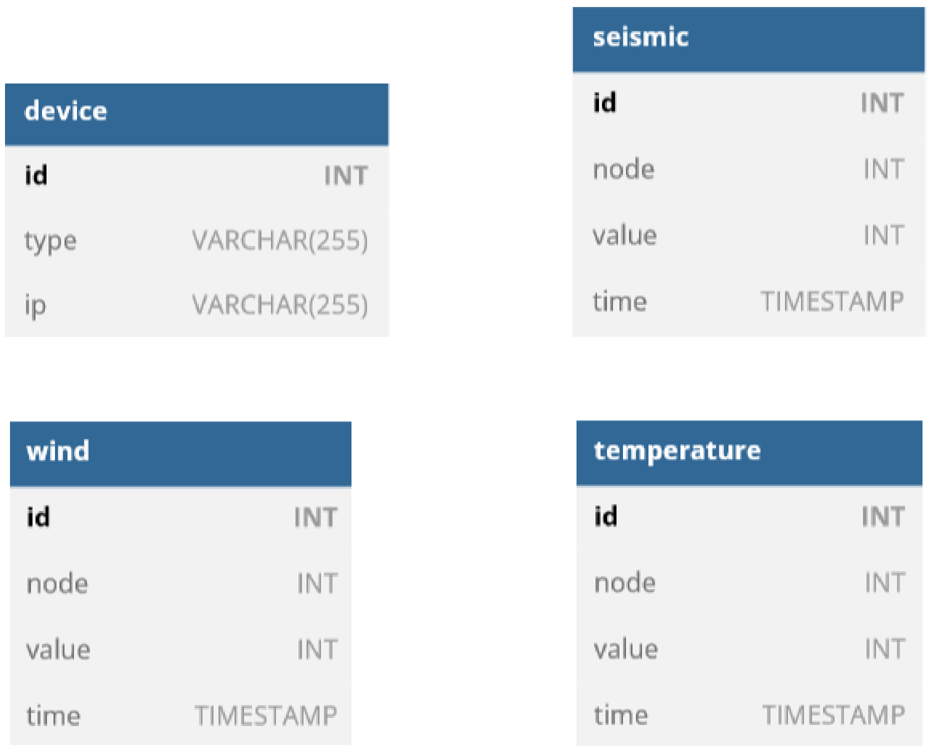
3.2 Database

The data generated by the sensors have been saved in a MySQL database for the correct management of the actuators and the correct use of the GRAFANA tool.

The database used is made up of 4 tables, one for each data collected (Temperature, Wind speed, Frequency of seismic waves) and one to manage the registration of the actuators in the application.

There are no dependencies between the various tables.

Since there can be multiple sensors of each type in the network, the ID of the node that sends the data to the cloud application is also saved.



device	
id	INT
type	VARCHAR(255)
ip	VARCHAR(255)

seismic	
id	INT
node	INT
value	INT
time	TIMESTAMP

wind	
id	INT
node	INT
value	INT
time	TIMESTAMP

temperature	
id	INT
node	INT
value	INT
time	TIMESTAMP

3.3 JAVA Application

We decided to implement our application with the JAVA language, using the JAVA libraries Paho and Californium to manage respectively the MQTT and CoAP protocol.

3.3.1 Cloud Application

The Cloud Application has the role of intermediary between sensors and databases. Through the MQTT protocol it collects data from the sensors and saves them in a MySQL database.

3.3.2 Remote Control Application

The Remote Control Application, when in automatic mode, manages the actuators autonomously according to very specific policies, based on the data collected by the Cloud Application and saved in the database.

The managment policies are the following:

- Average wind speed to close the barrier: >= 65 km/h
- Average temperature to turn ON the Snow Machine: <= -2.5 degrees (And barrier closed)
- To start the alarm two out of three of these conditions must be satisfied:
  - Average wind speed >= 50 km/h
  - Average temperature >= 35 degrees
  - Average seismic frequency >= 30 Hz

Furthermore, it also takes care of providing the user with a series of functions through which it is possible to obtain the last value collected by a sensor or manually manage the actuators, thus deactivating the automatic mode which can in any case be reactivated with the appropriate command.

The functions provided by the Remote Control Application to the user are as follows:

- **!help**: Print a description of the function for each command
- **!getTemperature**: Get the temperature from the temperature sensor
- **!getWind**: Get the wind from the wind sensor
- **!getFrequency**: Get the seismic frequency from the seismic sensor
- **!setTemperature**: Set the temperature of the temperature sensor
- **!setWind**: Set the wind of the wind sensor
- **!setFrequency**: Set the seismic frequency of the seismic sensor
- **!SnowMachineON**: Turn the Snow Machine ON
- **!SnowMachineOFF**: Turn the Snow Machine OFF
- **!BarrierON**: Open the security barrier
- **!BarrierOFF**: Close the security barrier
- **!AlarmON**: Activate the Alarm
- **!AlarmOFF**: Disable the Alarm
- **!SkiLaneON**: Open the ski lane, opening the barrier and turning the snow machine OFF
- **!SkiLaneOFF**: Close the ski lane, closing the barrier and turning the snow machine ON
- **!setAutoAll**: Set the automatic mode for all the actuator
- **!Exit**: Close the program

3.3.3 Main Packages and Classes

To have an efficient organization we have divided the application code into various packages, based on the function they cover within the project. The essential packages are the following:

- **CoAP**
  This package contains the code to manage the communication with the actuators.
  In particular, this package handles the **Registration** of the devices and the **Automatic Management** of the actuators based on the contents of the database, thanks to the various dedicated handlers for each actuator.

- **Database**
  This package contains the functions that communicate with the database, inserting or reading data from it.
  Here we find the dedicated drivers for each type of data that manage the entry of the data collected by the sensor into the database.

- **Model**
  This package contains the classes which represents the data sensed by the sensors and sent to the Cloud Application. Each data has a dedicated class, which has two attributes:
  - Sensor node ID
  - Collected data value.

- **MQTT**
  This package contains the core of the **Cloud Application** which, through the MQTT protocol, takes care of receiving the data sent by the sensors and saving them within the database using the functions contained within the Database package.
  This package also manages the set command performed by the user through the publish mechanism provided by the MQTT Protocol.

- **Service**
  This package contains some utility functions used by the handlers (e.g., the functions to compute the average values of the data retrieved from the MySQL Database).

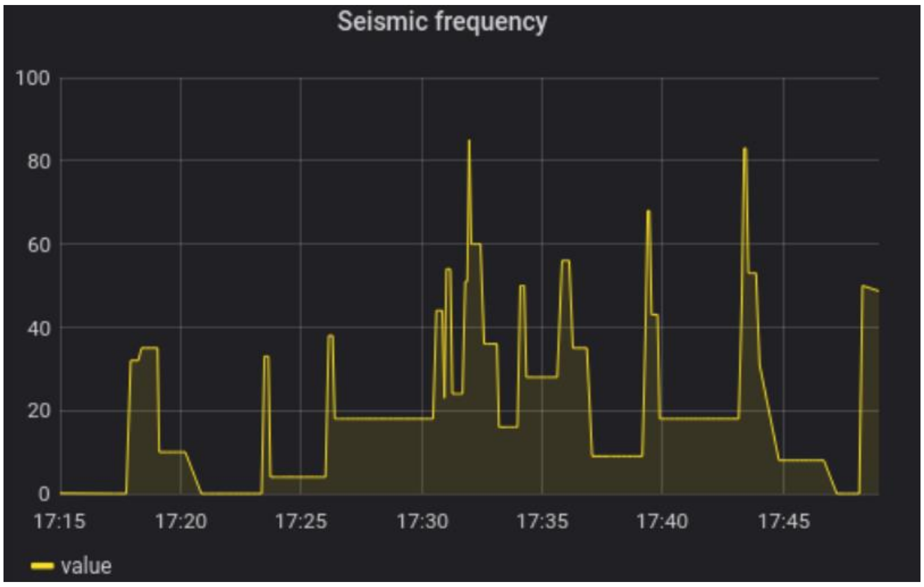
In addition to these packages, we find two fundamental JAVA files for the project, which are:

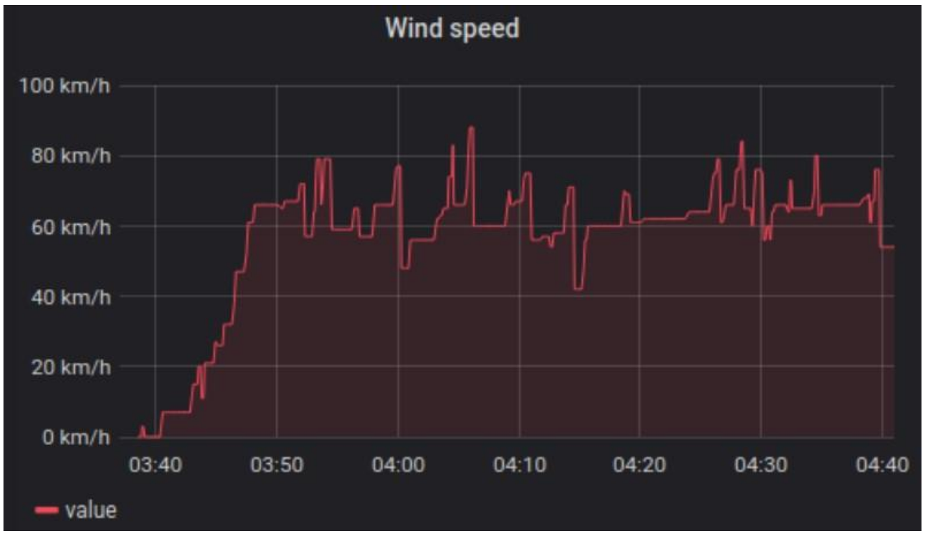
- **SkiTrackMonitoring.java**
  In this file we find the **Remote Control Application**, which contains the various commands that the user can execute and the handling of the automatic/manual management mechanism of the entire system.

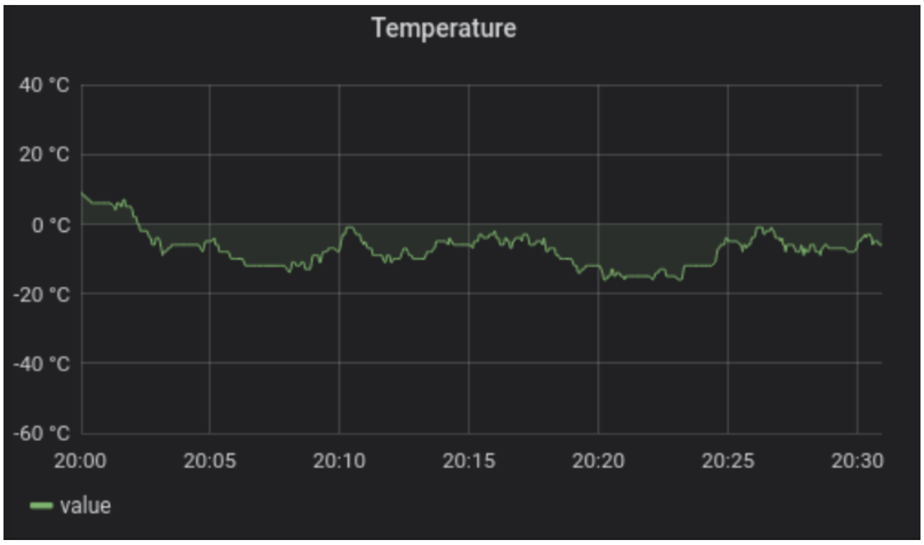
- **Log.java**
  This class is used to create a log containing useful messages for debugging and for monitoring the flow of the application.

3.4 GRAFANA

The GRAFANA tool was used to perform an analysis on the data within the dataset and to check the trends of the monitored data.


Seismic frequency
17:15
17:20
17:25
17:30
17:35
17:40
17:45
value


Wind speed
100 km/h
80 km/h
60 km/h
40 km/h
20 km/h
0 km/h
03:40
03:50
04:00
04:10
04:20
04:30
04:40
value


Temperature
40 °C
20 °C
0 °C
-20 °C
-40 °C
-60 °C
20:00
20:05
20:10
20:15
20:20
20:25
20:30
value