

INSA

**INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE**

REPORT

SERVICE ARCHITECTURE TP

5A ISS - 2024/2025

By BEGHIN Léa
BOUKOUISS Samia
BRUNETTO Marie

Table of Contents

Table of Contents.....	1
Introduction.....	2
I - Architecture of our system.....	2
II - Functional overview.....	3
1. Microservices Design.....	3
2. Communication between services.....	3
3. Functioning of the main loop.....	4
4. Visualization with a front end application.....	4
5. Organisation.....	5
Conclusion.....	5

Our Github Repository:

https://github.com/Pepotrouille/TP_SOA_Beghin_Boukouiss_Brunetto

Introduction

This technical report examines the evolution of the architecture of our application that is designed to monitor sheep, focusing on Microservices architectures with Restful communication. The main goal of this application is to ensure the safety of the sheep by monitoring their location (outside/inside) and activating control mechanisms such as alarms and automated gates to warn the farmer in the event of an escape.

I - Architecture of our system

Before developing the application, its structure and functional requirements had to be defined. The goal was to design a sheep monitoring application integrated with a microservices architecture. After a phase of brainstorming and design, several microservices were defined:

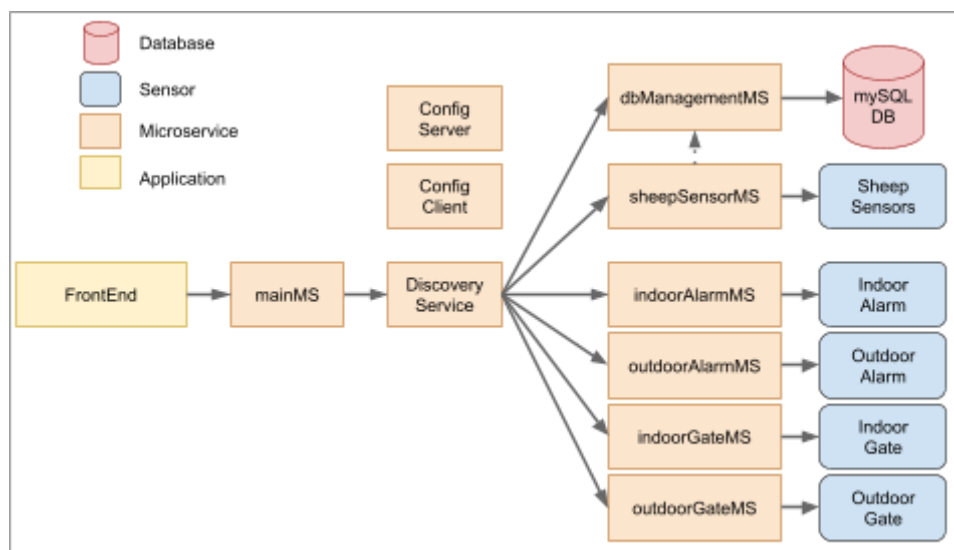


Figure 1 - Diagram of our project

1. **mainMS**: The main microservice that handles global interactions.
2. **Discovery Service**: The service responsible for the service discovery with Eureka.
3. **dbManagementMS**: The microservice responsible for managing data, interacting with a MySQL database.
4. **sheepSensorMS**: The microservice dedicated to managing sensors retrieving the sheep position.
5. **indoorAlarmMS**: The microservice associated with the indoor alarm.
6. **outdoorAlarmMS**: The microservice associated with the outdoor alarm.
7. **indoorGateMS**: The microservice controlling the indoor gate.
8. **outdoorGateMS**: The microservice controlling the outdoor gate.
9. **ConfigServer**: This service creates and externalises the configuration parameters.

10. **ConfigClient**: This service implements the configuration done in the previous service.

If a sheep is detected outside of the zone where it has to be (indoor or outdoor), the application triggers both of the alarms so that the farmer can see that a sheep has burrowed, regardless of whether he is inside or outside. The application automatically closes both gates to prevent any other sheep to follow the one who escaped.

We, therefore, created the managing components and the following resources :

- **Sheep**: Each sheep has an ID, a name, a position in the farm and the ID sensor that tracks its location. This data is collected and processed by the **Sheep Sensor MS** service.
- **Alarms**: The indoor and outdoor alarms are respectively controlled by the **indoorAlarmMS** and **outdoorAlarmMS** services. They are triggered based on the position of the sheep.
- **Gates**: The gates of the indoor and outdoor enclosures are respectively managed via the **indoorGateMS** and **outdoorGateMS** services.

Finally, a front-end interface allows the entire system to be monitored and controlled by the user.

II - Functional overview

1. Microservices Design

Concerning microservices, we decided to have a single service to manage the sensors. Given the possible number of sensors, having a service for each sensor seemed too complicated for our application. We therefore preferred to have a general sensor class managed by a single microservice, which is responsible for communicating with each sensor and separating them via their ID. On the other hand, we implemented a separate microservice for the two alarms and the two doors. Here, unlike with the sensors, we opted for the idea that there might only be one indoor and one outdoor alarm and one indoor and one outdoor enclosure gate to monitor. In addition, the service was not very important because there were only two functions used to retrieve and modify the state of each of the actuators

2. Communication between services

A first question that has been raised when developing the application was “Which services need to communicate together?”.

A first obvious answer was the discovery service, which requires to be aware of most of the services to work as intended. Another service that will need communication with other services is the main loop, which will orchestrate the series of calls for the application. Also, the Sheep Sensor service is in communication with the database service because it needs access to the sheep parameters.

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
DBMANAGEMENTMS	n/a (1)	(1)	UP (1) - insa-20663.insa-toulouse.fr:dbManagementMS:8091
INDOORALARMMS	n/a (1)	(1)	UP (1) - insa-20663.insa-toulouse.fr:indoorAlarmMS:8083
INDOORGATEMS	n/a (1)	(1)	UP (1) - insa-20663.insa-toulouse.fr:indoorGateMS:8082
MAINSERVICEMS	n/a (1)	(1)	UP (1) - insa-20663.insa-toulouse.fr:MainServiceMS:8085
OUTDOORALARMMS	n/a (1)	(1)	UP (1) - insa-20663.insa-toulouse.fr:outdoorAlarmMS:8081
OUTDOORGATEMS	n/a (1)	(1)	UP (1) - insa-20663.insa-toulouse.fr:outdoorGateMS:8084
SHEEPSENSORSMS	n/a (1)	(1)	UP (1) - insa-20663.insa-toulouse.fr:sheepSensorsMS:8090

Figure 2 - Eureka view of all of our services

A final point we had during the development was about the Sheep Sensor service. The sensors aim to take the location of a sheep. We thus asked ourselves if it would be more convenient for our application to have the update of the position done by the sensor itself after each measurement, or by the main application after each call. We concluded that if it should occur after each call, then it would be better and directly integrated, also allowing the update when the service is called outside of the main loop. We therefore update the sheep position directly in the main application.

3. Functioning of the main loop

We will now run through the functioning of the main loop, put in our service mainMS. The loop is controlled by a boolean *service_is_running* that can be stopped through a request to the main service.

The different actions that take place in the loop are the following:

- First, the position of all the sensors/sheep in the database is gathered from the associated service.
- Then, a condition is tested:
 - If there is more than one position, or that the position is not the wanted one, and the actuators are off, then the actuators are turned on.
 - Else, if there is only one position, equal to the wanted one, and the actuators are on, then the actuators are turned off.
- Finally, the loop waits for two seconds to restart the loop.

The time of the loop is currently very short, but in an hypothetical implementation, we would like to avoid sending too many packets, especially as the position of the sheep might not change that often. We could then imagine having the main service to loop over 10 to 30 minutes.

4. Visualization with a front end application

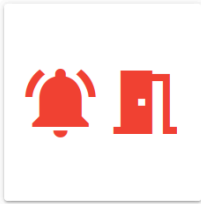
Finally, the last part we wanted to implement to the system was an interface with the user, to be able to use and see from the request more easily.

To proceed, we decided to develop a simple interface using *Python* and the library *niceGUI* for web development. This library provides us components that we could embed flawlessly on our interface. Using a thread to loop over the request, the interface refreshes every second to show the state of the door/alarm and the database. It also allows the user to set the wished position of the herd through a dropdown button.

With the interface, the farmer can see the list of his sheep with their name, ID, actual position and the sensor ID. This list can also serve as the truth in the event of an alarm being triggered, simply by checking whether the sheep's position is inconsistent with where the farmer wants it to be. Besides, when the alarms are triggered and the gates closed, a picture of a bell turns red. Here, we are emphasising the visual side of our application

Sheep Sensors

Name	Position	ID	IdSensor
Fripouille	outdoor	1	0
Nuage	outdoor	2	0
Fluffy	outdoor	3	0
Neige	outdoor	4	4
Larry	outdoor	5	5
Cacao	outdoor	6	0
Shawn	outdoor	7	7
Neige	outdoor	8	4



**Sheep
wanted
position**

INDOOR ▾

REFRESH

Figure 3 - Python Frontend Interface

5. Organisation

To keep track of our progress and share our project to work on each other's session, we used a [shared Github](#) on which all the services are located. We were able to finish the implementation within the time of the labs session, except for the front end python interface, which was made as a bonus.

Conclusion

Through this class, we were able to apply the concept we learnt to build an application through microservices into a more concrete case. In addition to the previous labs, it allowed us to practice the java implementation of said services, and taught us how to make a self-sustained service with a loop. We also learnt how to interface with the REST API that we have created.

