Rapport Projet Service Architecture

Yohan Boujon[1], Robin Marin-Muller[1]

[1]INSA Toulouse, National Institute of Applied Sciences, Toulouse, France

Abstract

Dans ce projet, nous avons développé une application Web Proof-of-Concept pour la gestion des salles de l'INSA. Cette application vise à automatiser des actions telles que la fermeture des fenêtres et des portes, l'activation du chauffage ou l'extinction des lumières en fonction des données collectées par des capteurs. L'objectif principal est de permettre une prise de décision intelligente en analysant les informations provenant des capteurs à travers des services logiciels (microservices).

Nous avons conçu l'application en utilisant une architecture basée sur des microservices pour garantir la modularité et l'évolutivité. Les capteurs, comme ceux de température ou de présence, transmettent des données périodiques qui sont analysées pour déclencher des actions spécifiques sur les actionneurs. Par exemple, le chauffage peut s'activer automatiquement si les conditions de température intérieure et extérieure le justifient. Un autre scénario implémenté prévoit l'ouverture des fenêtres si le niveau de particules de dioxyde de carbone est trop élevé.

Architecture

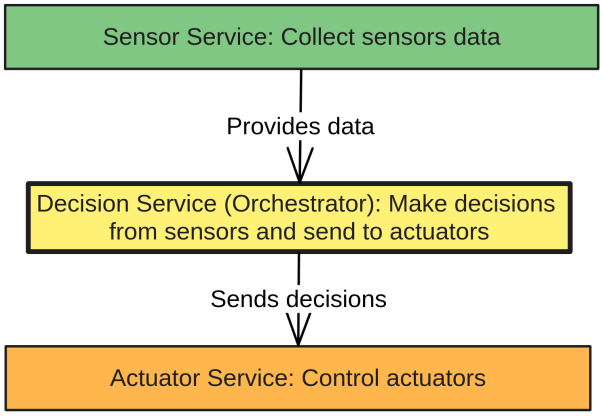

Figure 1: *Microservices Architecture*

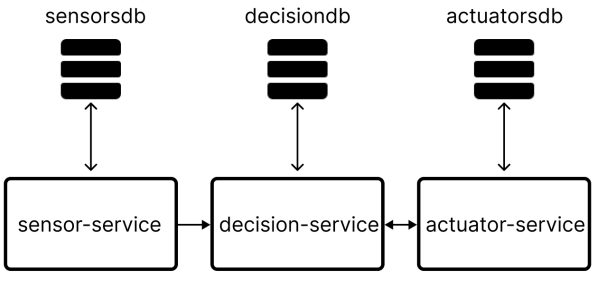

Figure 2: *Database Architecture*

Implementation

Référentiel du projet: [1]

Simulateur en GoLang

Le simulateur permet d'émuler les capteurs et actionneurs hypothétiquement présents dans les pièces de l'INSA. Il expose des API pour récupérer et contrôler les capteurs/actionneurs. De plus, il simule la réponse aux actions effectuées: par exemple si l'on allume les radiateurs, la température de la pièce récupérée par les capteurs de température augmente et inversement. Ce simulateur nous permet de tester notre application dans un environnement analogue à la réalité.

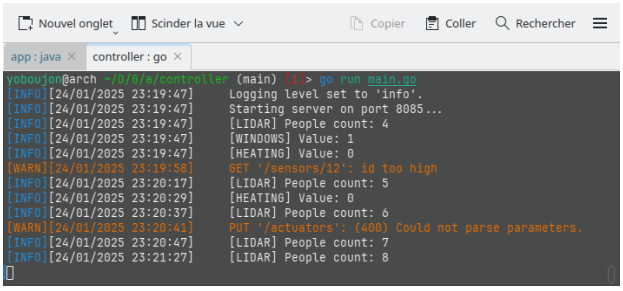

Figure 3: *Console du simulateur*

Via la console il est possible de voir l'évolution de certaines variables internes comme le *lidar* ou le changement effectif d'actionneurs. Les capteurs eux ne sont pas renseignés à moins de mettre le niveau d'enregistrement lancé au début de la simulation. La température ainsi que le niveau de particules dépendent de formules mathématiques qui sont disponible dans le code du programme.

Nous avons choisi *GoLang* pour cette partie car cela nous permet d'apprendre un nouveau langage qui est très focalisé sur l'architecture de service ainsi que les API REST.

Sensor Service

Le microservice "Sensor Service" est responsable de la collecte, de la mise à jour/stockage et de l'expositions (endpoints) des données des capteurs via une API REST.

Récupération des données des capteurs externes Le service fait une requête HTTP GET au simulateur via `http://localhost:8085/sensors/` dans le but de récupérer les données des capteurs, les données récupérées sont ensuite ajoutées ou mises à jour (si les capteurs sont déjà répertoriés) dans la base de donnée locale.

Stockage des données des capteurs Les données des capteurs sont stockées dans une base de données H2 locale, configurée par le fichier `application.properties`. Nous avons trouvé très pratique le fait de pouvoir gérer la base de donnée directement dans le projet Spring Boot, cela nous à fait gagner beaucoup de temps, et dans le même temps nous respectons bien la séparation des bases de données voulu par philosophie microservices.

Les entités `Sensor` représentent les capteurs avec les attributs suivants:

- `name`: Nom du capteur.
- `type`: Type de capteur (Température, Lumière, etc...).
- `value`: Valeur numérique.
- `unit`: Unité.
- `timestamp`: Heure de la mesure.
- `room`: Pièce dans laquelle le capteur est installé.

Exposition des données via une API REST Le microservice expose une API REST à l'endpoint `/sensors` qui permet de partager toutes les données des capteurs stockées dans la base de donnée locale aux autres microservices, l'API est décrite dans la classe `SensorApplication`.

Mise à jour des données des capteurs Quand on fait une requête GET à l'URL `/sensors` le service met à jour les données des capteurs en vérifiant si un capteur avec le même nom et type existe déjà dans la base de données, si oui, il met à jour ses valeurs, dans le cas contraire il ajoute un nouveau capteur.

Résultats Le endpoint /sensors nous renvoi bien une liste de tous les capteurs disponibles.

```
curl -X GET http://localhost:8083/sensors
[{"id":1,"name":"CCS811","type":"CO2","value":
↪ 792.28,"unit":"ppm","timestamp":"2025-01-2
↪ 4T23:07:45.27827824","room":1},{"id":2,"na
↪ me":"LM35","type":"Température
↪ Intérieur","value":14.39,"unit":"°C","time
↪ stamp":"2025-01-24T23:07:45.278932992","ro
↪ om":1}
```

SELECT * FROM SENSOR;

ROOM	value	ID	TIMESTAMP	NAME	TYPE	UNIT
1	592.07	1	2025-01-24 23:10:46.574659	CCS811	CO2	ppm
1	13.52	2	2025-01-24 23:10:46.575418	LM35	Température Intérieur	°C
1	13.15	3	2025-01-24 23:10:46.575977	LM35	Température Extérieur	°C
1	34.5	4	2025-01-24 23:10:46.576498	LM393	Son	dB
1	21.14	5	2025-01-24 23:10:46.577044	DHT22	Humidité	%
1	1.0	6	2025-01-24 23:10:46.577554	Hokuyo UST-10LX	Lidar	pc
2	592.07	7	2025-01-24 23:10:46.578087	CCS811	CO2	ppm
2	13.52	8	2025-01-24 23:10:46.578603	LM35	Température Intérieur	°C
2	13.15	9	2025-01-24 23:10:46.579207	LM35	Température Extérieur	°C
2	34.5	10	2025-01-24 23:10:46.579793	LM393	Son	dB
2	21.14	11	2025-01-24 23:10:46.580408	DHT22	Humidité	%
2	1.0	12	2025-01-24 23:10:46.581017	Hokuyo UST-10LX	Lidar	pc

(12 rows, 2 ms)

Figure 4: *Vue de la Base de Données du Microservice Sensors*

Actuator Service

Récupération des actionneurs disponibles

- Récupération des actionneurs disponibles
- Stockage des actionneurs
- Exposition d'*end-points* pour contrôler les actionneurs 0 ou 1 simple
- Stockage de l'historique des décisions
- Exposition de l'historique des décisions

Résultats Le microservice *actuator* expose bien l'endpoint /actionneurs permettant de récupérer tous les acutators.

```
curl -X GET http://localhost:8042/actuators
[{"id":1,"name":"Windows","type":"AUTOW1","sva
↪ lue":1,"room":1},{"id":2,"name":"Doors","t
↪ ype":"DOOR2032X","svalue":1,"room":1},{"id
↪ ":3,"name":"Heating","type":"HEATING4000",
↪ "svalue":0,"room":1}]
```

Nous pouvons sélectionner un seul actionneur par identifiant en spécifiant ce dernier dans le sous endpoint `/actuator/id`.

```
curl -X GET http://localhost:8042/actuators/3
{"id":3,"name":"Heating","type":"HEATING4000",
↪ "svalue":0,"room":1}
```

Le service prend bien également en compte la mise à jour des états des actionneurs:

```
curl -X PUT http://localhost:8042/actuators/1
↪ -H "Content-Type: application/json" -d
↪ '{"state": 1}'{"id":1,"name":"Windows","ty
↪ pe":"AUTOW1","svalue":1,"room":1}
```

Type: AUTOW1
Name: Windows
Value: 0

Type: AUTOW1
Name: Windows
Value: 1

Figure 5: *Before and After performing the PUT*

SELECT * FROM ACTUATOR;

ROOM	SVALUE	ID	NAME	TYPE
1	0	1	Windows	AUTOW1
1	1	2	Doors	DOOR2032X
1	0	3	Heating	HEATING4000

(3 rows, 1 ms)

SELECT * FROM ACTUATOR;

ROOM	SVALUE	ID	NAME	TYPE
1	1	1	Windows	AUTOW1
1	1	2	Doors	DOOR2032X
1	0	3	Heating	HEATING4000

(3 rows, 1 ms)

Figure 6: *Database View of the PUT Execution*

Decision Service

Le microservice "Decision Service" est responsable de la prise de décision basées sur les données des capteurs et actionneurs. A partir de certains seuils, il va réaliser des actions qu'il va placer dans la base de données avec des libellées précises. Dans notre cas, nous avons défini des actions très précises qui peuvent être décrite comme les suivantes:

SELECT * FROM DECISION;

ID	TIMESTAMP	ACTION	CONDITION
1	2025-01-24 22:27:33.177659	Opening Windows	Temperature too high (> 25°C)
2	2025-01-24 22:27:33.213872	Heating Stopped	Temperature too high (> 25°C)
3	2025-01-24 22:28:40.128434	Starting Heating	Temperature too low (< 20°C)
4	2025-01-24 22:28:40.130619	Closing Windows	Keeping room hot.
5	2025-01-24 22:31:03.328847	Opening Windows	Carbon Dioxide Level too high (> 800 ppm)
6	2025-01-24 22:31:58.527557	Closing Windows	Keeping room hot.

Figure 7: *Actions et Conditions avec le tampon temporel dans la base de donnée de "Decision Service"*

Pour l'instant il n'y a que 3 conditions dont 2 basées sur la température *(trop élevée ou trop basse)* et 1 sur le nombre de ppm *(partie par million)*. Il est bien évidemment possible d'ajouter de nouveaux seuil qui vont réaliser différentes actions.

Récupération des données des capteurs La première étape est de récupérer les données des capteurs en faisant une requête HTTP GET vers l'endpoint exposé par le microservice `Sensor-Service` (`http://localhost:8086/sensors/`).

Prise de décision Vient ensuite l'étape analyse des données de capteurs pour prendre des décisions par chambre (clustering des capteurs selon leur attribut `room`). Par exemple le service peut décider de fermer une fenêtre si une certaine condition est remplie (comme une température trop basse).

Stockage de l'historique des décisions

Exposition des décisions via une API REST Finnalement les décisions prises sont exposées sous `/decisions` dans le but d'exposer l'historique au front-end par exemple.

Front-End

Effectuer des requêtes CURL ou GET à la main n'est n'est pas pratique pour l'utilisateur, nous avons donc mis en place une interface utilisateur (front-end) qui vient automatiser les requêtes et afficher des informations pertinentes à l'utilisateur.

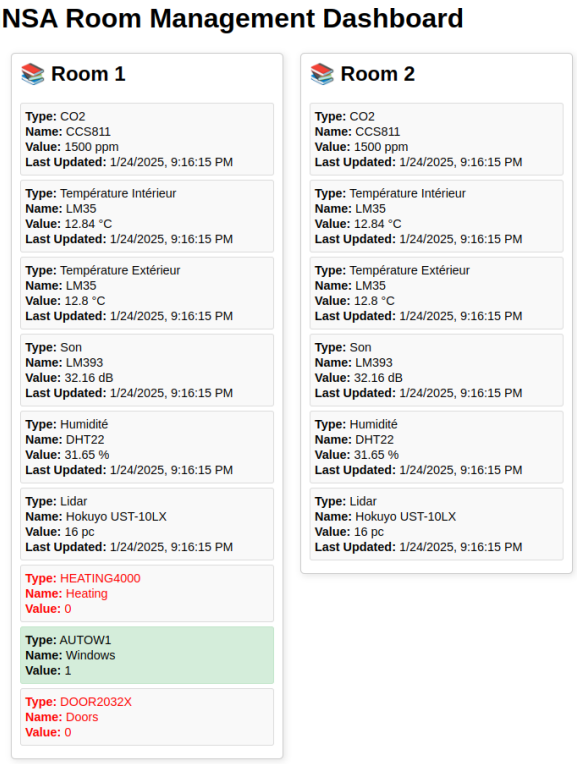

Figure 8: *Front-End Dashboard View*

Nous pouvons voir les différentes pièces avec les capteurs et les métriques. En bas, les actionneurs : rouge pour inactif et vert pour actif. Dans notre cas, le chauffage de la pièce 1 est actif. Le code interroge périodiquement les différentes API pour mettre à jour les données en temps réel.

Conclusion

Ce projet était une occasion de renforcer nos compétences en Java, Maven et SpringBoot et s'inscrit bien dans la continuité des travaux réalisés aux cours des précédents TDs. Durant ceux-ci notre expérience en développement axée service n'étant que très limitée, nous avons pu lors de ce projet prendre du recul sur nos compétences afin de mieux structurer notre solution.

References

[1] Robin Marin-Muller and Yohan Boujon. *Service Architecture Project*. URL: `https://github.com/yoboujon/automatic-management`.