INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

TP Service Architecture

Web Application Connected INSA Room

Cédric Chanfreau

Timothé Bigot

Promotion 58 – 2024 / 2025

20/12/2024

Introduction 2
I - Contexte et objectif du projet 2
II - Conception de l'application basée sur micro-services et choix technologiques 3
1. Structure en micro-services 3
2. Définition des scénarios d'utilisation 4
3. Implémentation et appel des micro-services 4
4. Implémentation de l'interface web pour suivi et tests des différents cas d'utilisations 8
5. CI / CD 10
III - Conclusion 11

Introduction

Ce compte rendu et les séances de TP auxquelles il se rapporte font suite au précédent projet réalisé au cours des séances de TD, concernant l'application de mise en relation de volontaires et de personnes sollicitant une aide. Le but de ce TP est de développer une nouvelle application en micro-services, en y ajoutant de nouvelles fonctionnalités vues dans les tutoriels mais non appliquées encore sur un projet concret. Nous avons fait le choix du projet de salle de classe intelligente, au travers de divers capteurs et actionneurs et de son implémentation technique, que nous allons détailler dans les prochaines parties.

I - Contexte et objectif du projet

Nous avons donc déjà pu mettre en place et comprendre divers types d'architectures en TD. Parmi lesquels, SOAP, une architecture qui combine plusieurs protocoles standardisés, afin de communiquer au travers de messages SOAP. La description des communications se base sur XML, ainsi que le fichier WSDL qui détaille l'accès et les interactions entre le client, le serveur et les services. Bien qu'offrant un bon niveau de sécurité et de fiabilité, c'est une architecture assez rigide, et limitée dans le cadre d'applications complexes et où la bande passante est réduite. C'est pourquoi nous avons ensuite implémenté le projet avec une approche REST.

REST est une architecture plus conçue pour l'interaction de services web, également basée sur HTTP, mais diffère par rapport à SOAP avec une vision orientée ressource, avec des requêtes HTTP via des mots clés dédiés. Cela permet d'avoir un système plus simple, léger et de meilleures performances mais moins de fonctionnalités et de sécurité. Nous avons pu aborder cette architecture en TD au travers de l'application Jersey et de la plateforme Postman pour le test des fonctions développées.

Enfin nous avions aussi développé notre projet de TD avec une forme hybride des deux précédentes, à savoir sous forme de micro-services en découpant l'application en plusieurs micro-services et plusieurs avantages qui en résultent que nous allons approfondir par la suite.

Le but de ces TP est la mise en place d'une nouvelle architecture de micro-services, en y apportant une plus grande variété d'options que lors des travaux dirigés.

Notre choix pour le thème de ce projet s'est porté sur la gestion de salles de cours à l'INSA et sur le développement de l'application y attenant. Le but est que cette application puisse obtenir des relevés de données ascendants depuis la salle de cours, et agir sur du matériel en conséquence, et ce, de manière automatisée. Il y a donc une partie relevée, et analyse de

données d'un ou plusieurs capteurs, et ensuite d'actions à prendre au travers d'actionneurs. Par exemple, ouvrir une fenêtre si la température est trop importante. En complément de cela, une interface web pour un suivi de l'historique des données et actions. Nous allons maintenant exposer notre démarche de conception pour ce projet.

II - Conception de l'application basée sur micro-services et choix technologiques

Tout d'abord, dans une démarche de simplification, nous nous sommes concentrés sur la gestion d'une salle seulement, et si jamais nous en avons le temps, d'étendre à plusieurs salles. Ensuite, concernant les capteurs et actionneurs, ils seront simulés et des valeurs entrées directement sur l'interface web, afin de ne pas rajouter de problématique supplémentaire avec des communications de formats différents.

Pour tout le reste du projet, on se focalise sur un développement basé sur une architecture de micro-services que nous allons maintenant détailler.

1. Structure en micro-services

En se basant sur des micro-services, on fait le choix d'une approche distribuée et décentralisée offre de nombreux avantages parmi lesquels une garantie d'évolutivité sans trop de contraintes lorsque le développement a intégré cet aspect, mais aussi une indépendance des micro-services qui permettent de ne pas compromettre la totalité du système dans le cas d'un problème sur l'un d'eux. Également, cette approche permet d'avoir une abstraction du protocole du capteur (même si virtualisé dans notre cas) vis-à-vis de l'interaction avec le monde extérieur, à savoir les autres micro-services.

C'est dans cette logique aussi que nous avons décidé de choisir de définir un micro-service par capteur, et non un micro-service par famille de capteurs, qui bien que cela aurait réduit le nombre de services de notre architecture nous aurait apporté une évolution moins bonne (arrêt du micro-service ou le dupliquer pour ajouter un nouveau capteur). Ce choix reste acceptable dans la mesure où notre architecture comporte finalement peu de capteurs.
Dans notre projet, nous avons au total deux capteurs, de présence et de température, desquels nous allons agir sur des actionneurs en fonction de scénarios bien définis. En particulier, nous avons fait le choix des actionneurs suivants : activation d'une alarme, d'une lumière, et ouverture d'une porte ou d'une fenêtre.

2. Définition des scénarios d'utilisation

Nous avons défini nos trois scénarios différents, faisant intervenir les divers capteurs et actionneurs :

- Scénario 1 : Ouvrir les fenêtres si la température extérieure est inférieure à la température intérieure et entre 18 et 27°.
- Scénario 2 : Fermer les portes, fenêtres et lumières en dehors des heures de travail si personne n'est présent.
- Scénario 3 : Activer l'alarme si une présence détectée après 22h00.

3. Implémentation et appel des micro-services

Dans Maven, nous avons établi une liste des plusieurs micro-services, qui permettent de réaliser l'application que nous avons définie plutôt suivant les différents cas d'utilisations :

Figure 1 Micro - service du projet

Pour chacun des micro-services, à l'exception de orchestratorAction, le micro-service est composé de quatre fichiers java :

- Action : Permet de lire et d'appliquer l'action à réaliser
- Controller : Gère les requêtes HTTP entrantes et appelle les services appropriés pour traiter ces requêtes.
- Management Application : Point d'entrée principal de l'application Spring Boot.
- Service : Contient la logique métier pour traiter les actions demandées.
- Client (pour les services temperatureManagement, doorManagement et windowManagement) : Permet de faire des appels HTTP vers d'autres micro-services.

Figure 2: Fichiers java pour le microservice Alarm

Le micro-service Orchestrateur se compose de plusieurs éléments, pour agir sur les actionneurs (fenêtre, porte, alarme et lumière) et les capteurs (température extérieur, intérieur, et la présence au travers du capteur de lumière)

- ActuatorState : Permet de récupérer et centraliser l'état actuel des actionneurs en testant leurs valeurs d'états au travers de booléens

- SensorData: Permet de récupérer les valeurs des capteurs ainsi que la donnée temporelle associé

- AppConfig : Configure et fournit un bean RestTemplate pour effectuer des appels HTTP vers d'autres micro-services.

- SensorHistory : Représente une entité JPA pour stocker l'historique des états des capteurs et actionneurs dans la base de données.

- OrchetratorActionApplication : Point d'entrée principal de l'application Spring Boot pour le micro-service orchestrateur.

- OrchestratorController : Gère les requêtes HTTP entrantes pour obtenir l'état actuel et l'historique des actionneurs, ainsi que pour gérer les dispositifs en fonction des données des capteurs.

- OrchestratorService : Permet de faire appel aux diverses micro-services qui agissent sur les actionneurs, en fonction des valeurs reçu par les capteurs, et d'agir en conséquence sur les actionneurs, au travers de ActuatorState, tout en mémorisant Le temps de toutes les actions, et en retournant ces informations à l'interface web

Concernant l'orchestrateur, son rôle est le suivant :
L'orchestrateur est un composant central de notre architecture de micro-services. Il joue le rôle de coordinateur sur les différents actionneurs (fenêtres, portes, lumières, alarme) en fonction des données reçues des capteurs (température intérieure, température extérieure, présence). Les principales responsabilités des actionneurs sont les suivantes :

1) **Réception des données des capteurs** : L'orchestrateur reçoit les données des capteurs via des requêtes HTTP POST. Ces données incluent la température intérieure, la température extérieure, la présence détectée et l'heure actuelle.

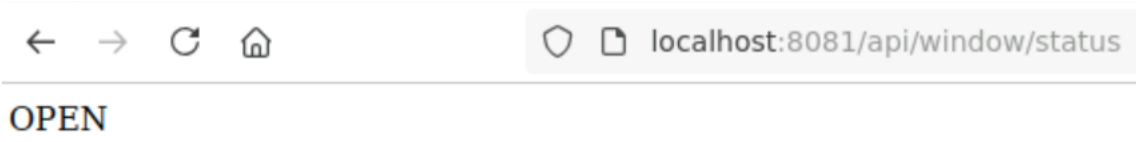
2) **Analyse des données et prise de décision** : En fonction des données reçues, l'orchestrateur analyse et prend des décisions pour agir sur les actionneurs. Par exemple :
   - Ouvrir les fenêtres si la température extérieure est inférieure à la température intérieure et se situe entre 18 et 27°C.
   - Fermer les portes, fenêtres et lumières en dehors des heures de travail si aucune présence n'est détectée.
   - Activer l'alarme si une présence est détectée après 22h.

3) **Appel aux micro-services des actionneurs** : L'orchestrateur envoie des requêtes HTTP aux micro-services des actionneurs pour exécuter les actions décidées.

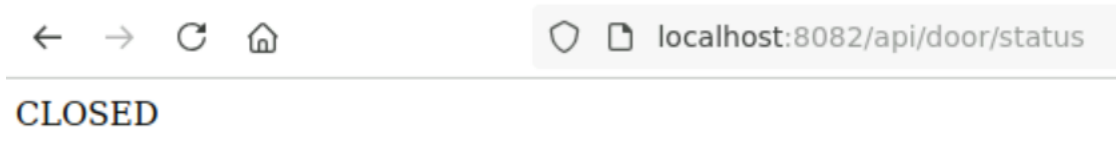
4) **Mise à jour de l'état des actionneurs** : L'orchestrateur maintient un état actuel des actionneurs en utilisant la classe ActuatorState. Cet état est mis à jour après chaque action effectuée.

5) **Enregistrement de l'historique des actions** : Chaque action effectuée par l'orchestrateur est enregistrée pour garder une trace de l'historique des actions. La classe SensorHistory est utilisée pour stocker ces informations.

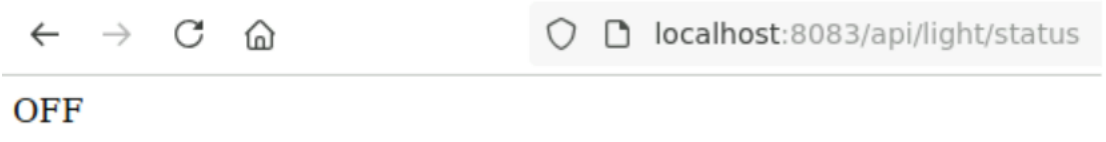
Nous pouvons faire appel à ces services en local, suivant les ports que nous avons définis et obtenir l'état du capteur/actionneur.



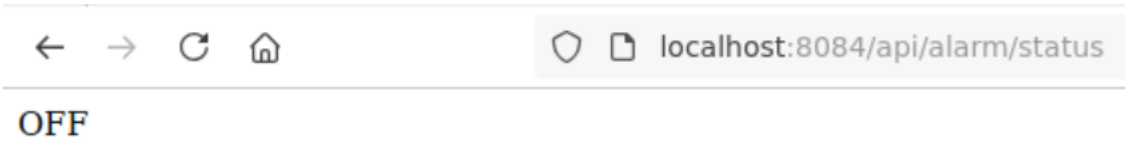
Appel du microservice windowManagement



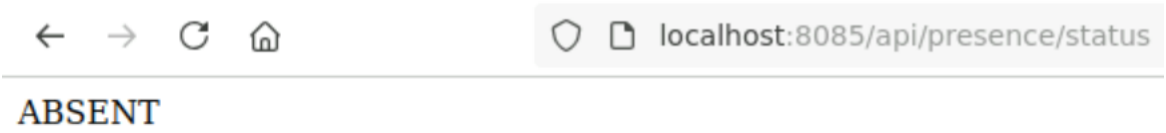
Appel du microservice doorManagement



Appel du microservice lightManagement



Appel du microservice alarmManagemen

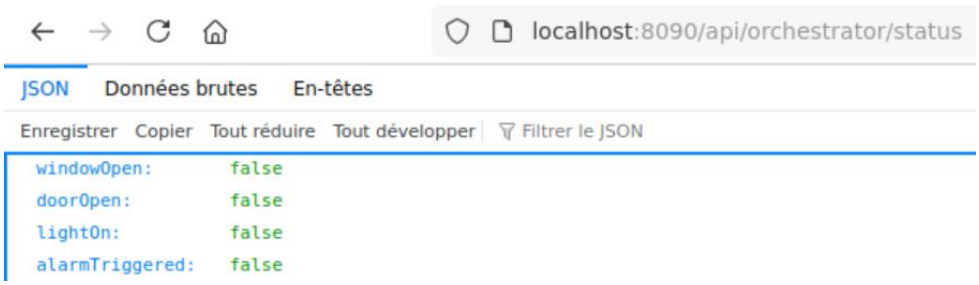


Appel du microservice presenceManagement

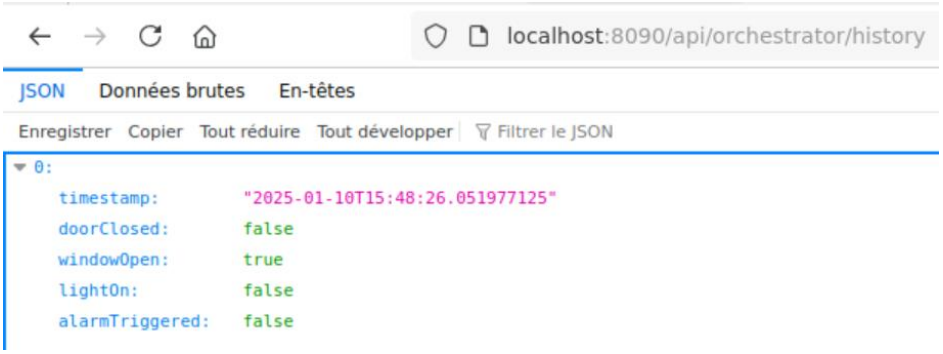
Le tableau en ci-bas récapitule les micro-services avec les ports auquels ils sont associés ainsi que les capteurs correspondants.

Micro-services	Type de composant associé	Port de communication
temperatureManagement	Capteur	8080
windowManagement	Actionneur	8081
doorManagement	Actionneur	8082
LightManagement	Actionneur	8083
alarmManagement	Actionneur	8084
presenceManagement	Capteur	8085
orchestratorAction	Actionneur	8090

Nous pouvons avoir un retour du statut et de l'historique de l'orchestrateur, avec les valeurs booléennes des micro-services que nous avons. Plus tard dans le projet, nous avons utilisé une base de données pour avoir un retour des actions qui ont été effectué, de manière plus accessible pour un usager utilisant notre application.



Status de l'orchestrateur

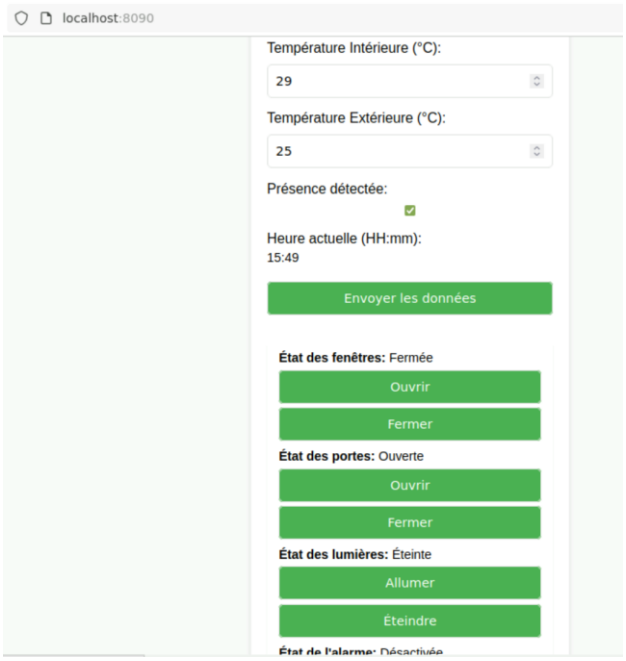


Historique de l'orchestrateur

4. Implémentation de l'interface web pour suivi et tests des différents cas d'utilisations

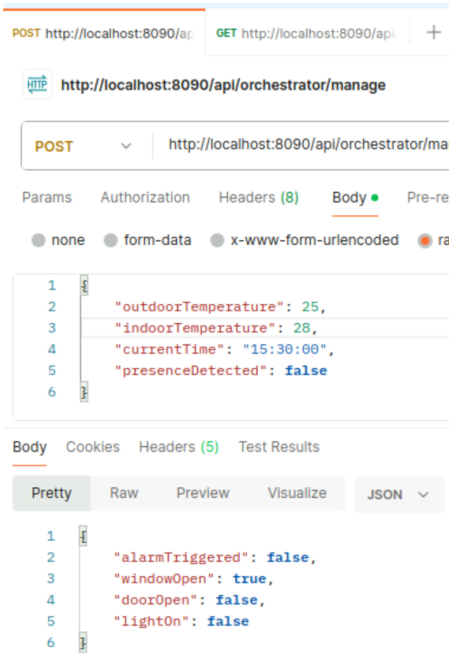
L'interface graphique permet d'entrer manuellement la température intérieur, extérieur ainsi que de préciser l'heure et s'il y a une présence au sein de la classe.

En plus de cela, une section dédiée aux capteurs, avec la possibilité d'agir sur les actionneurs, indépendamment des cas d'utilisation, afin de vérifier le fonctionnement du système, ce qui est plutôt optionnel en fin de compte.

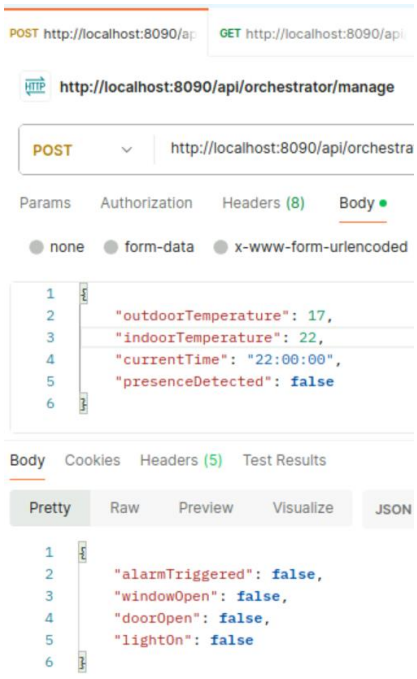


Interface graphique utilisateur

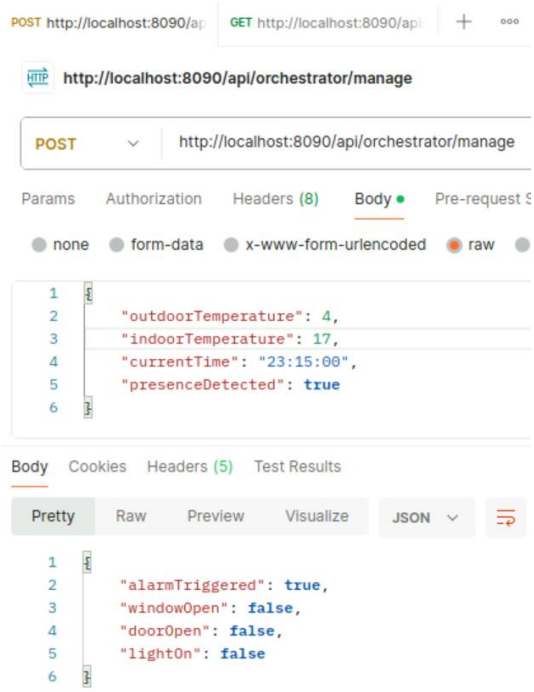
Nous avons ensuite pu vérifier que l'architecture étais fonctionnelle, avec les différents scénarios que nous avions définis :



Cas 1 : Ouverture de la fenêtre pour une température extérieure inférieure à la température intérieure et entre 18 et 27°



Cas 2 : Fermeture des portes, fenêtres et lumières en dehors des heures de travail si personne n'est présent



Cas 3 : Activation de l'alarme si une présence est détectée après 22h00

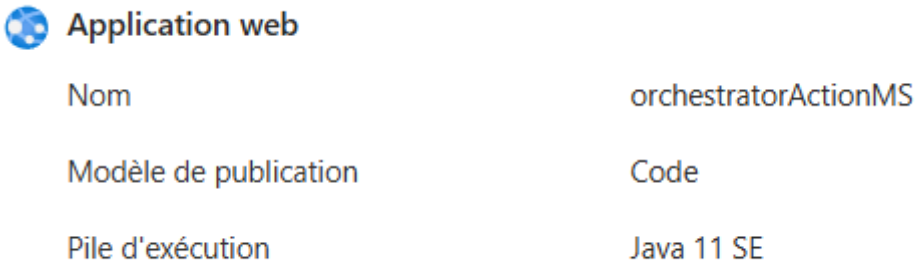
5. CI / CD

Un des objectifs de ce TP est de pouvoir implémenter certains des concepts qui relèvent du développement logiciel en lien avec l'administration des infrastructures, du DevOps en soi. Cela inclut notamment de l'intégration continue, et du déploiement continu.

Le but pour nous est, pour un de nos micro-services, crée un workflow qui va automatiser les processus : lorsqu'une modification est apportée au code sur GitHub, des actions comme l'exécution de tests ou le déploiement peuvent être lancées automatiquement. Pour cela, nous avons utilisé Microsoft Azure comme plateforme de déploiement pour héberger les applications et GitHub Action.

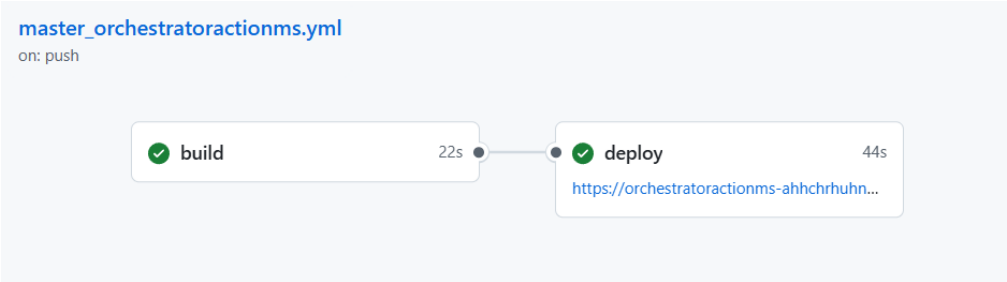
Au-delà de l'automatisation de notre micro-service, toute cette démarche est pertinente à plus grande échelle, pour déployer des applications de manière fluide et continue, garantissant des mises à jour régulières sans interruption.

Une fois le groupe de ressource MS-DeploymentTP et l'application web orchestratorActionMS définie dans Azure, nous avons effectué les modifications dans le fichier yml pour avoir les bonnes dépendances avec notre micro-service orchestratorAction, susceptible d'être modifié par la suite.



Application web sur Microsoft Azure

Après la compilation du workflow, le service est maintenant déployé et à chaque fois que orchestratorAction sera modifié, il sera testé et déployé indépendamment, assurant ainsi une intégration continue de nouvelles fonctionnalités ou corrections sans perturber le système global.



Déploiement fonctionnel sur github

III - Conclusion

En rétrospective, le développement de nos microservices a été un parcours enrichissant. De la conception initiale à la mise en œuvre avec Spring Boot et Spring Cloud, chaque étape a offert des opportunités d'apprentissage. La mise en place du Config Server a permis une gestion centralisée des configurations, essentielle pour une architecture de microservices bien gérée.

Ce projet a été une immersion complète dans l'écosystème des microservices, transformant chaque défi en opportunité de croissance. Nous avons développé une architecture robuste et flexible pour notre application de gestion des actionneurs et capteurs. Chaque micro-service fonctionne de manière indépendante, assurant modularité et scalabilité. L'orchestrateur coordonne les actions des actionneurs en fonction des données des capteurs, garantissant une gestion intelligente et automatisée. Grâce à l'intégration continue, chaque modification du service orchestratorAction est testée et déployée automatiquement, assurant une mise à jour fluide et sans interruption. Notre vision est de continuer à améliorer cette architecture pour intégrer de nouvelles fonctionnalités et optimiser la gestion des ressources et la performance globale du système.

Lien vers le dépôt GitHub:

https://github.com/CedricChnfr/service-architecture-insa/tree/master/TP