INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

SERVICE ORIENTED ARCHITECTURE

Rapport Projet “Salle autonome”

Module **Middleware and Service**

SOUMIS PAR

Yacine BENCHEHIDA 5ISS-A2
Hugo LE BELGUET 5ISS-A2
Sami BEYAH 5ISS-A2

SOUS LA DIRECTION DE

Prof. **Nawel GHERMOUCHE**

Année scolaire : 2021-2022
Janvier, 2021

Lien Git : https://github.com/yacine180496/SOA_Project

Sommaire

Introduction

Dans le cadre du cours « d'architecture de services » à l'INSA Toulouse, nous avons dû développer une application Web pour la gestion des salles de l'INSA.

En utilisant un service web RESTful, le système doit être capable de contrôler les éléments des salles à travers une interface web et de prendre des décisions basées sur les données récupérées des capteurs.

Chapitre 1 – Gestion de projet

Pour le développement de cette application, nous avons utilisé la méthode Agile Scrum et réalisé plusieurs sprints. Nous avons utilisé Jira pour son interface utilisateur intuitive mais aussi car c'est une solution non payante. En parallèle nous avons également travailler avec GitHub pour pouvoir centraliser nos différentes parties de codes. De plus nous nous sommes également aidés de Jenkins pour l'intégration continue de notre projet.

I) Notre utilisation de Jira

Nous avons décidé d'opérer avec des sprints d'une semaine chacun, soit forme de trois sprints distincts. Ils abordent tous un aspect différent du projet global, cette façon de travailler nous a beaucoup aidé dans la fluidité du travail individuel car nous savions tous quoi faire et quand le faire, ce qui évite de perdre du temps en discussion inutile.

De plus dans chaque story au sein des sprint nous pouvons mettre des tickets enfants pour rentrer encore plus dans les détails des tâches à effectuer. Le fait d'avoir un statut concernant l'avancement, non seulement des story d'un sprint mais également des tickets enfants de ces story, permet de savoir rapidement d'un coup d'œil où en est chacun et s'il faut venir aider sur quelque chose pour pouvoir faire avancer le projet.

On peut également associer clairement chaque tache à une personne en particulier il n'y a donc aucune erreurs ou perte de temps à ce sujet, par exemple si deux personnes travaillent sur la même chose à cause d'un manque de communication.

Nous avons également exploité le système d'importance au sein des sprints pour savoir quelles tâches étaient à effectuer en priorité par rapport à d'autres qui pourraient être moins vitales au fonctionnement du projet, ce qui permet de concentrer son travail efficacement dans la bonne direction.

II) Nos sprints

Nous allons maintenant rentrer dans les détails de nos trois différents sprints pour voir ce que nous avons mis en place.

i) Sprint 1 : Mise en place de l'architecture

Dans ce premier sprint, notre objectif est de concevoir nos différents scénarios dans un premier temps, de les développer et les regrouper sous la coupelle d'un superviseur dans un second temps puis finalement de simuler les données comme si nous avions un réseau de capteur.

Pour organiser ces différentes étapes voici comme nous nous y sommes pris.

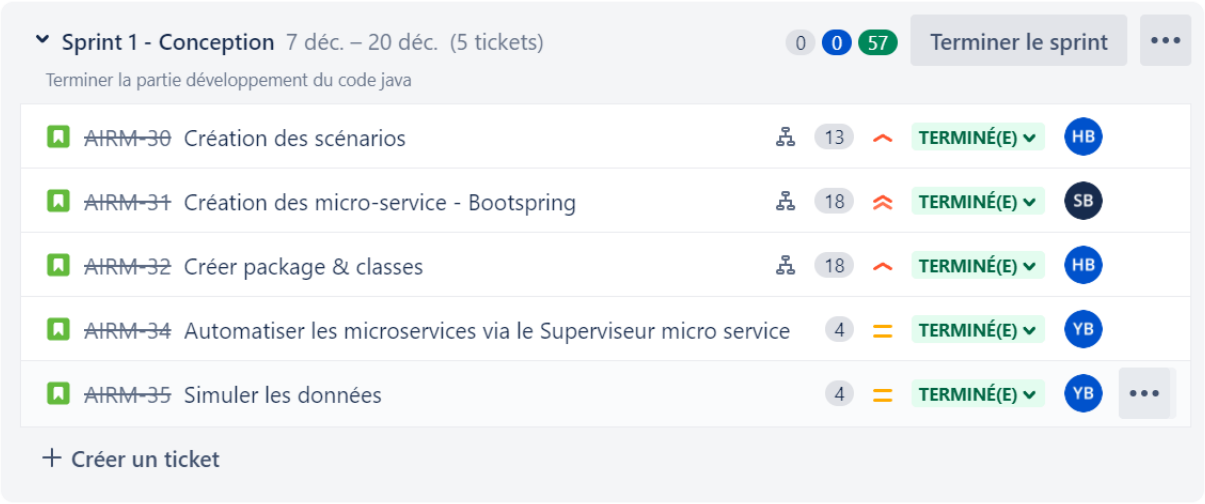


Figure 1 : Sprint 1

Nous avons voulu fragmenter le plus possible les différentes étapes de ce sprint car il s'agit de la plus grande partie de notre projet, celle qui nous a pris le plus de temps également, ainsi dans un souci de précision avoir beaucoup d'étapes nous a permis de ne pas nous disperser et de ne rien oublier.

On constate également un arbre sur les trois premières story, cela signifie qu'il y a des tickets enfants.

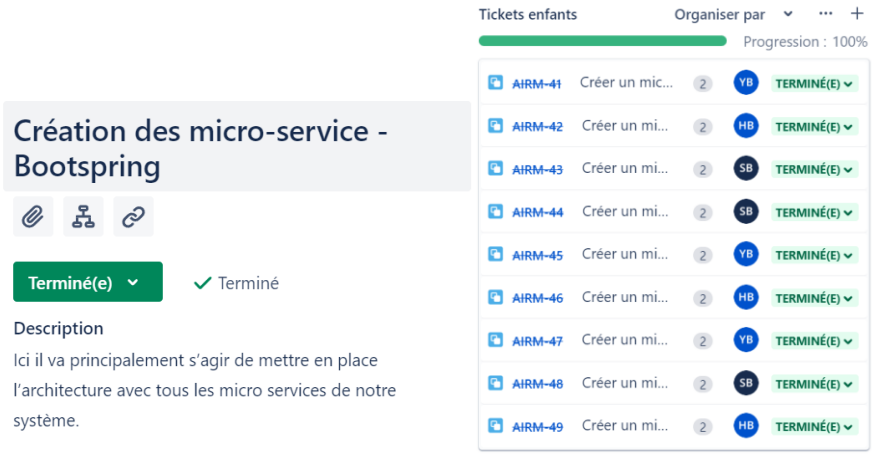


Figure 2 : Création des micro-services – Bootspring

Ainsi on a un objectif général contenant plein de sous objectif plus précis ce qui permet de suivre à bas niveau l'avancement du projet.

Le but à la fin de ce sprint était d'avoir une infrastructure de service fonctionnelle, gérée par un superviseur et visible par des requêtes visible sur une interface web simple, cet objectif a été atteint dans les temps grâce à l'organisation apporté par Jira.

ii) Sprint 2 : phase de tests

Suite à la première partie, nous avons maintenant fini le développement de nos différents scénarios, il est donc temps de les mettre à l'épreuve pour voir si tout opère comme nous l'avons prévu.

C'est l'objectif de ce sprint qui prend la forme suivante.

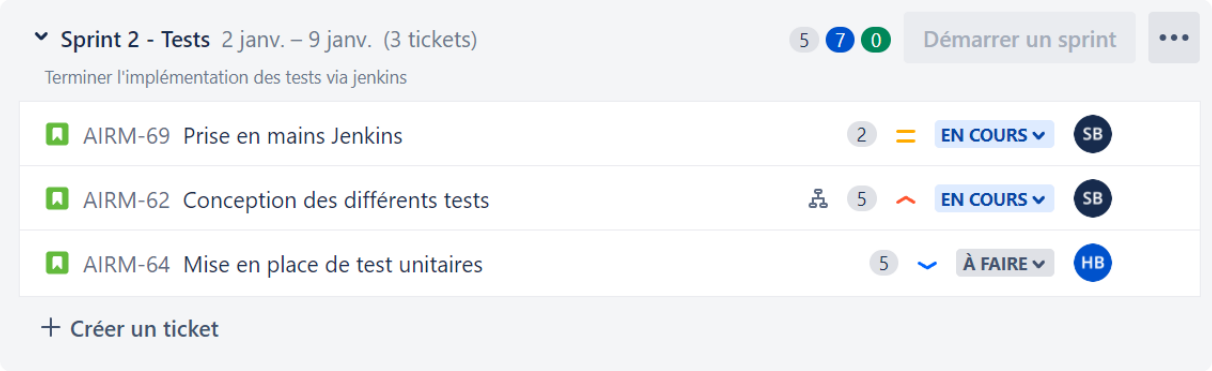


Figure 3 : Sprint 2

Nous avons également utilisé ce sprint pour passer du temps sur Jenkins que nous avons découvert avec ce projet, dans ce sens il nous a paru important de prendre du temps pour apprendre à s'en servir, nous l'avons principalement utilisé pour sa fonction de versioning.

Néanmoins la majeure partie du temps a été passée à mettre en place différents cas d'utilisation pour tester l'efficacité de nos scénarios. Une fois cela fait, il est prévu de mettre en place des tests unitaire visant à s'assurer que le code est fiable et résilient. Cependant cela s'affiche plutôt dans une optique d'amélioration future du projet et nous ne serons pas en mesure de la faire avant la date de rendu, c'est pourquoi cette partie reste au statut « à faire ».

iii) Sprint 3 : Mise en place d'une interface utilisateur

Ce dernier sprint rejoint la fin du précédent et ne sera pas mis en place dans le cadre de ce prochain, il s'agit plus d'une piste d'amélioration, nous savons ce qu'il reste à faire pour rendre le projet encore plus abouti, et nous l'avons exprimé dans ce sprint.

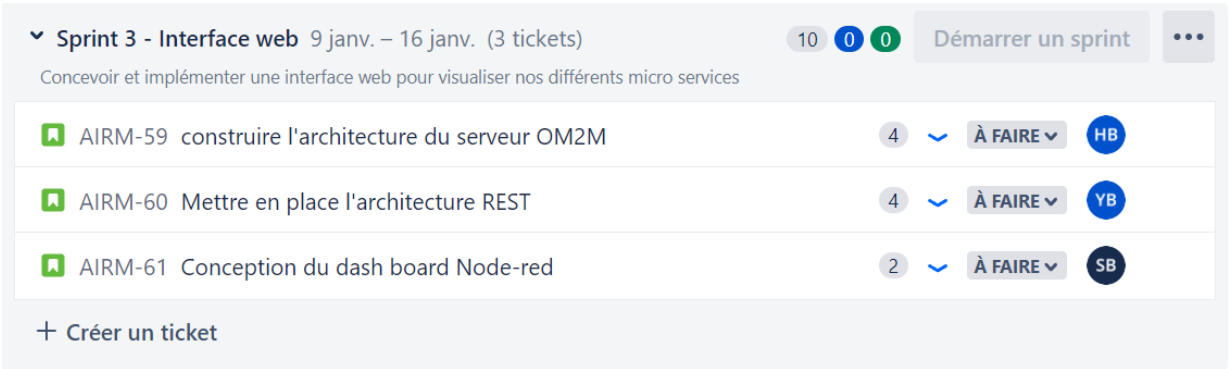


Figure 4 : Sprint 3

Pour commencer nous devons mettre en place l'infrastructure OM2M, nous aurions choisi ça car nous avons déjà travaillé avec, en accord avec nos différents scénarios et les capteurs nécessaires.

Ensuite il faudra mettre en place l'architecture REST afin de récupérer les données des capteurs et de les faire interagir avec notre infrastructure de service déjà établie.

Finalement, pour le confort des utilisateurs, nous devrons créer un *dashboard* clair disposant toutes les informations obtenues, ceci grâce à Node-Red.

II) Nos logiciels pour le « versionning » de notre code

Pour le développement de notre code, nous avons principalement utilisé deux outils logiciels : Git et Jenkins.

Nous avons utilisé git pour la gestion des versions (versionning). Nous sommes habitués à utiliser cet outils dans nos projets informatiques car il permet d'avoir un historique des modifications en local.

Nous avons fait le choix de centralisé notre code sur la plateforme Github. Ce dernier complète Git par des capacités de collaborations pour la mise en place d'un projet. Il est particulièrement utile dans les projets collaboratifs comme le nôtre.

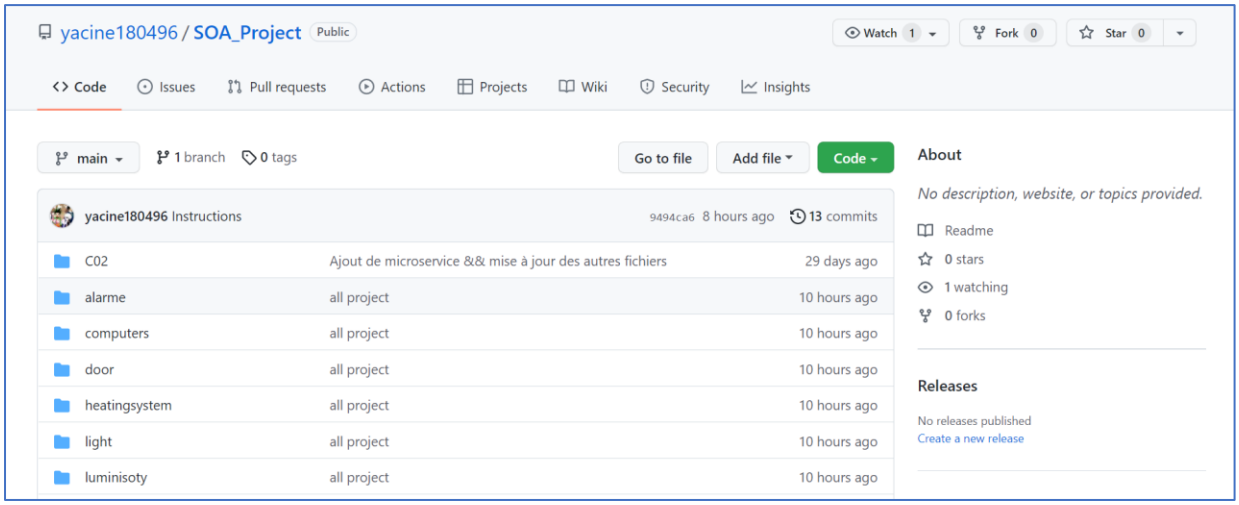


Figure 5 : dépôt Github

En parallèle, nous avons utilisé Jenkins pour l'intégration continue de notre projet. Jenkins est un outils open source développé en Java. A chaque modification de code d'une application dans le gestionnaire de configuration, Jenkins se charge automatiquement de la recompiler, et de la tester. Contrairement, à Git, nous n'avons jamais utilisé cet outils. La prise en main de cet outils a pris du temps.

Nous avons lié notre outils Jenkins à notre dépôt Github. Pour chaque microservice, nous avons créer un Job sur Jenkins. Ce dernier vérifie le code et le compile. Si une erreur dû à un changement dans le code surgit, nous sommes immédiatement notifiés.

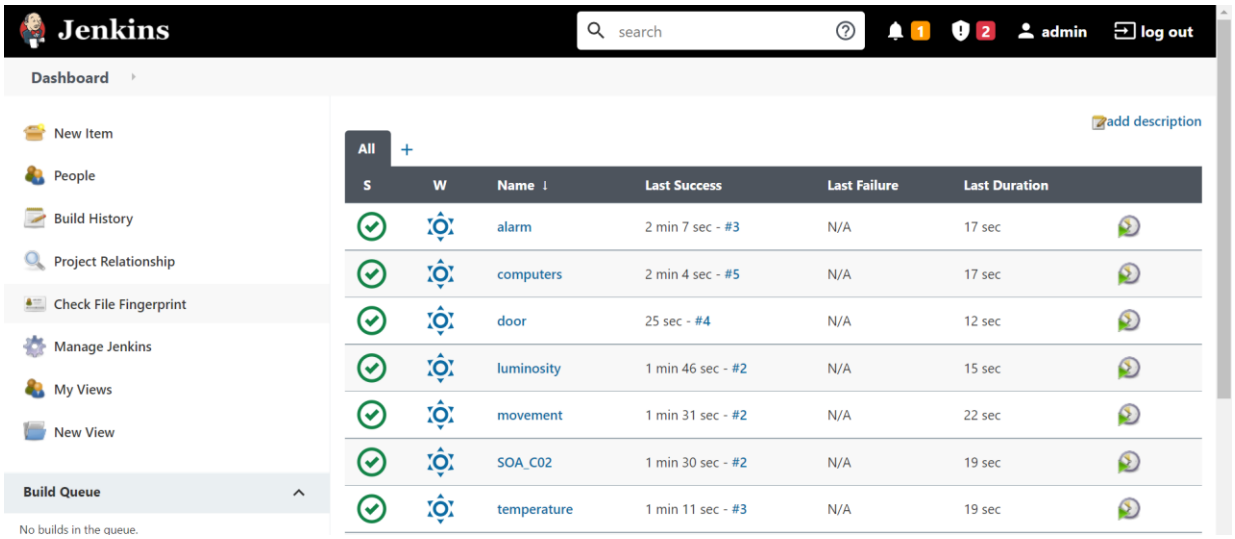


Figure 6 : Jenkins

Vous pouvez remarquer ci-dessus que les builds ont été réalisés avec succès pour nos différents microservices. Donc notre projet ne contient aucune erreur et le projet a été compilé avec succès.

Chapitre 2 – Architecture de notre solution

Comme spécifié, notre architecture devrait être basée sur une *API Rest* pour récupérer les données des capteurs et donner des com- commandes aux actionneurs.

Selon les demandes de nos utilisateurs, nous avons implémenté 11 services web avec l'un de ces services qui supervisent les autres et prend les décisions en fonction des données qui récupèrent.

Pour la phase de test, nous avons fait le choix de **simuler** notre architecture OM2M, du fait de l'utilisation de plusieurs capteurs et actionneurs. On va donc « *randomiser* » les données des capteurs afin d'avoir une première architecture fonctionnelle.

I) Conception de l'architecture OM2M

Pour modéliser notre système, nous avons conçu une architecture utilisant *OM2M*. Le nœud d'infrastructure (*IN*) représente le bureau de travail dans lequel se trouve un nœud intermédiaire (*MN*) représentant la pièce où nous déployons notre système intelligent.

Nous avons 11 appareils représentant chacun un micro-service qui a un rôle différent selon le cas d'usage : capteurs ou actionneurs. Voici un diagramme de notre architecture *OM2M* montrant le choix d'implémentation de celle-ci :

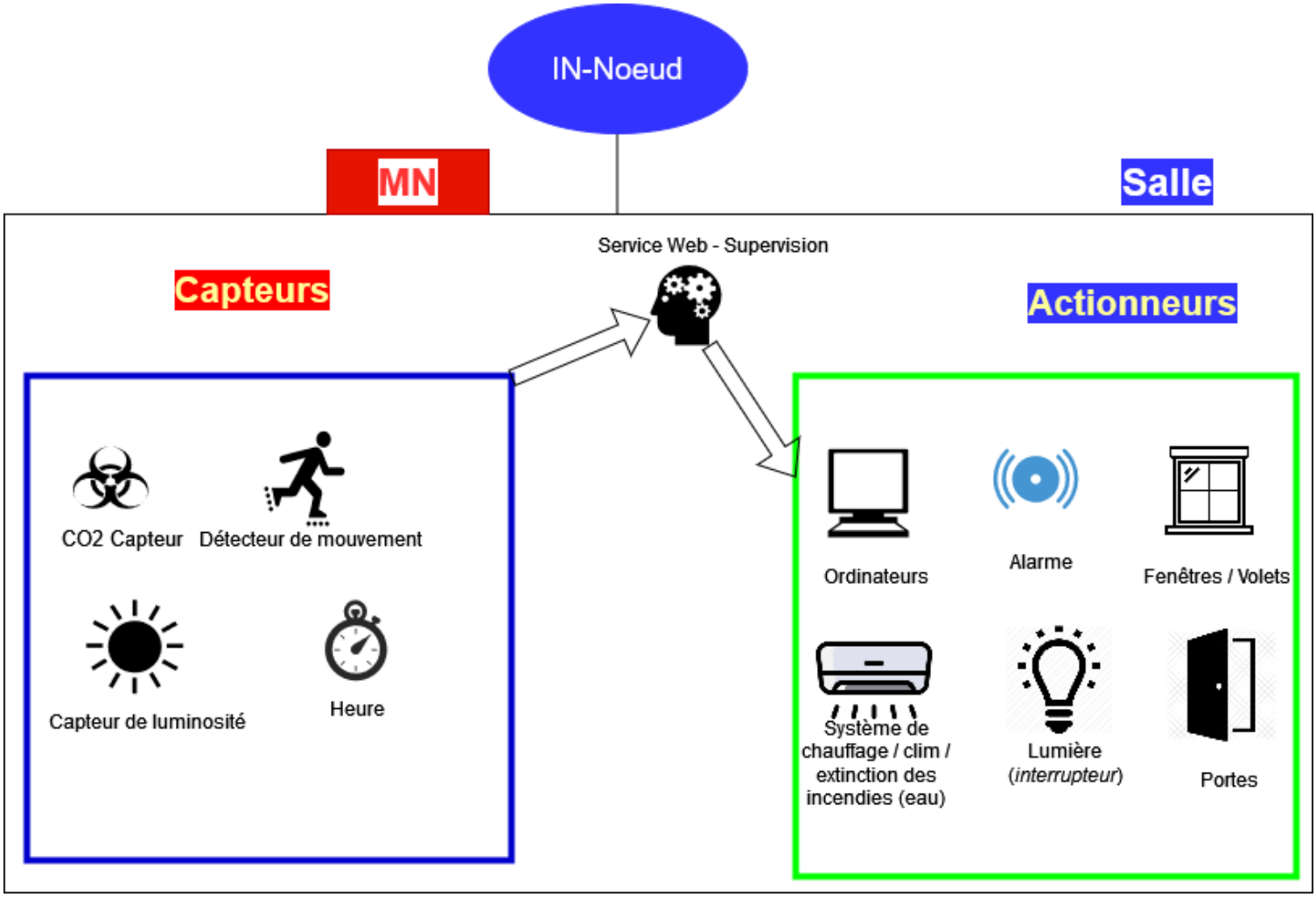


Figure 1 : Diagramme montrant notre architecture OM2M

En effet, l'ensemble des capteurs et actionneurs sont représentés dans OM2M par des entités d'application (AEs).

C'est une vue générale de notre architecture qui permet de comprendre l'implémentation qui suit.

II) Mise en œuvre des services

Selon les demandes de nos utilisateurs et d'après la figure précédente, nous avons implémenté :

- **4 services** jouant le rôle de capteurs
- **6 services** jouant le rôle d'actionneurs

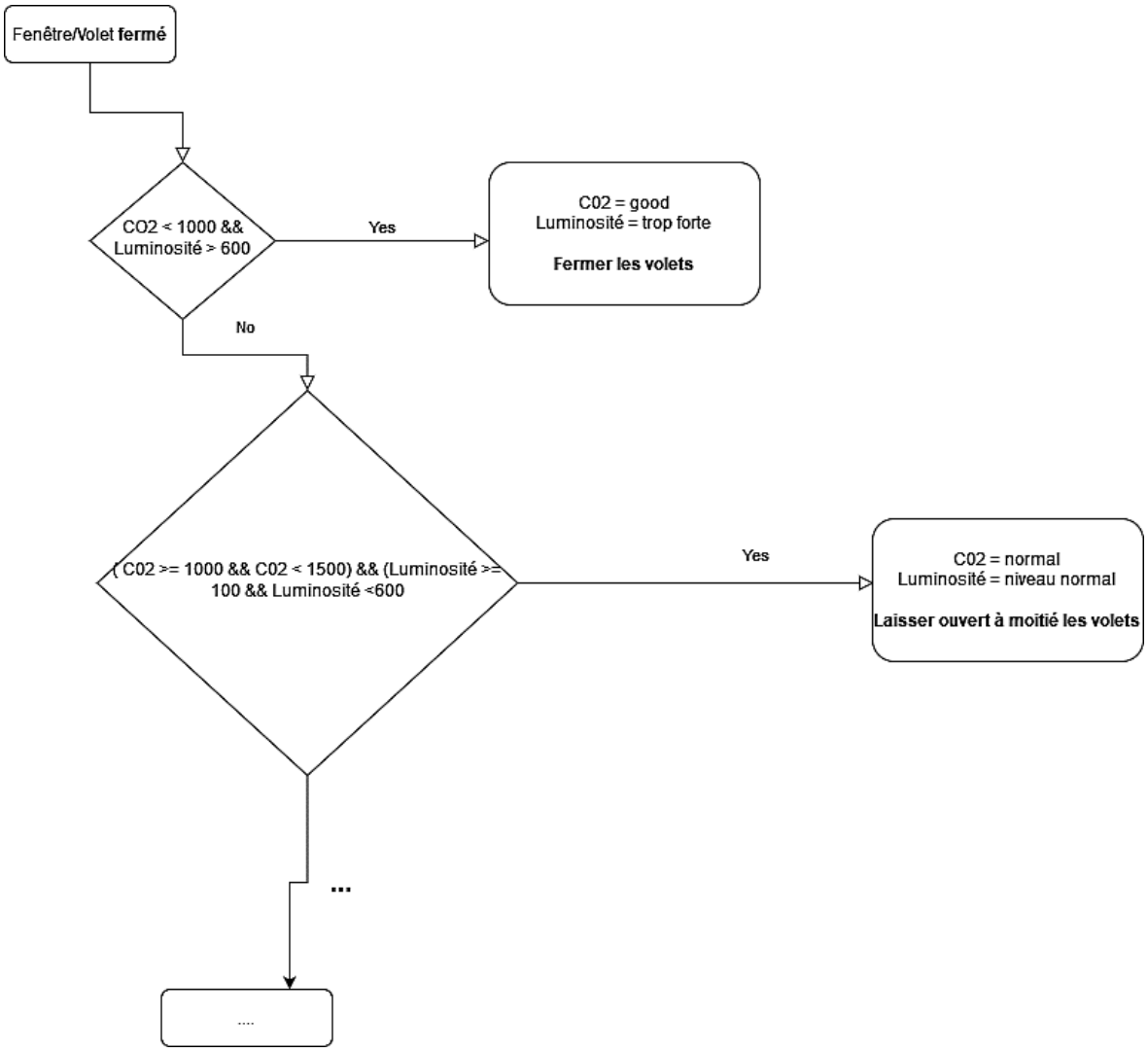
L'objectif de cette partie est de présenter les différents scénarios que nous avons implémenté pour contrôler les éléments d'une salle.

i) Scénario 1 – Luminosité ,Fenêtre et C02

Ce scénario se représente de la façon suivante : en fonction du taux de luminosité et du taux du C02 dans la pièce, le superviseur va décider d'ouvrir ou pas la fenêtre afin de purifier l'air.

En effet, **le taux de C02** sera toujours prioritaire sur le taux de luminosité. Même si le taux de luminosité est normal, **si le taux de C02 dépasse le seuil autorisé** : le superviseur ouvrera automatiquement la fenêtre pour purifier automatiquement l'air.

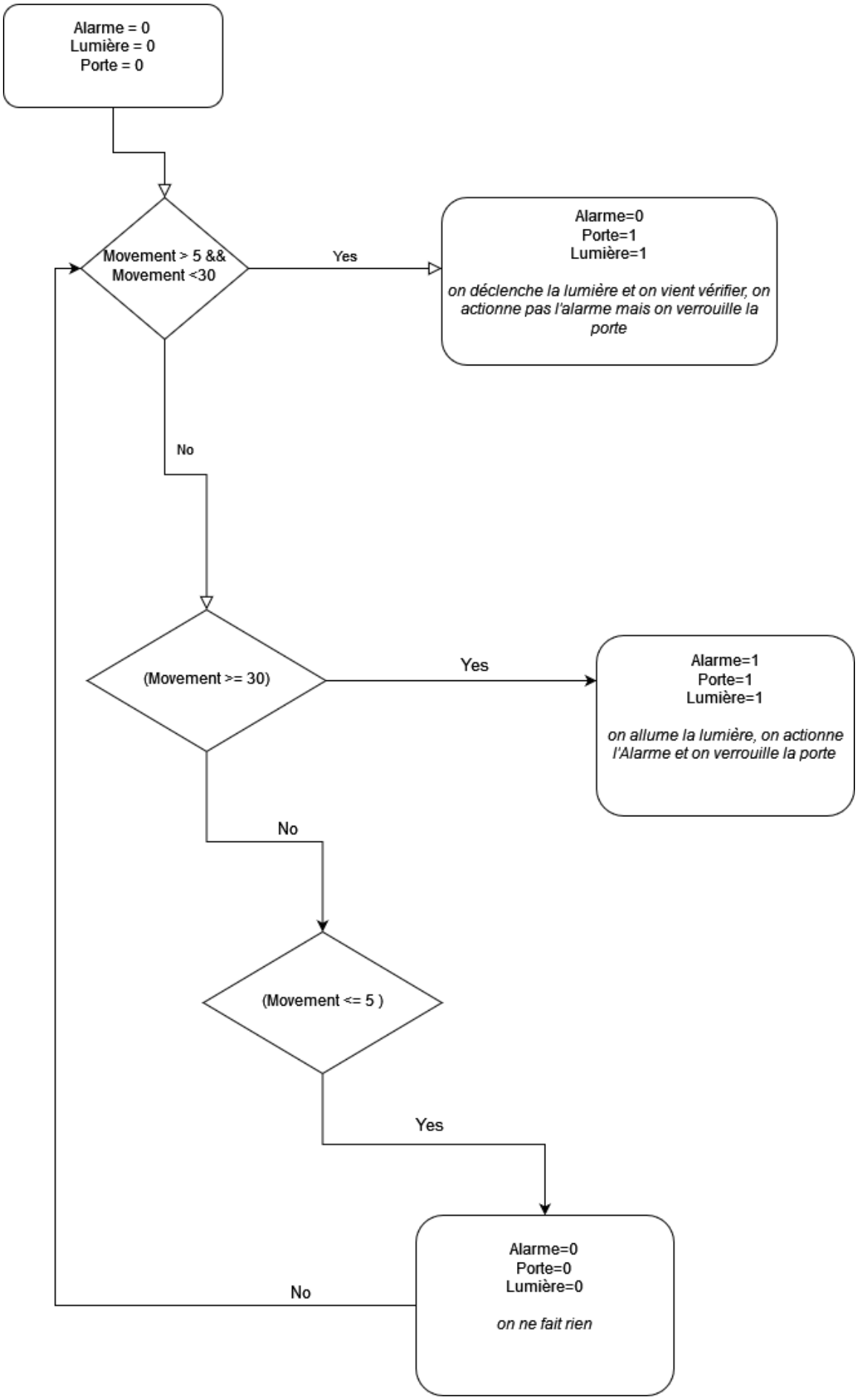
Voici un extrait de l'algorigramme illustrant la structure du scénario :



ii) Scénario 2 - Sécuriser la pièce contre les intrusions pendant la nuit

Un système de sécurité est mis en route pour empêcher les intrusions dans la pièce. En fonction de la valeur du faisceau de mouvement, le superviseur peut décider de déclencher la lumière, d'actionner l'alarme et/ou de verrouiller la porte de la salle.

Ci-dessous, l'algorithme que nous avons implémenté :

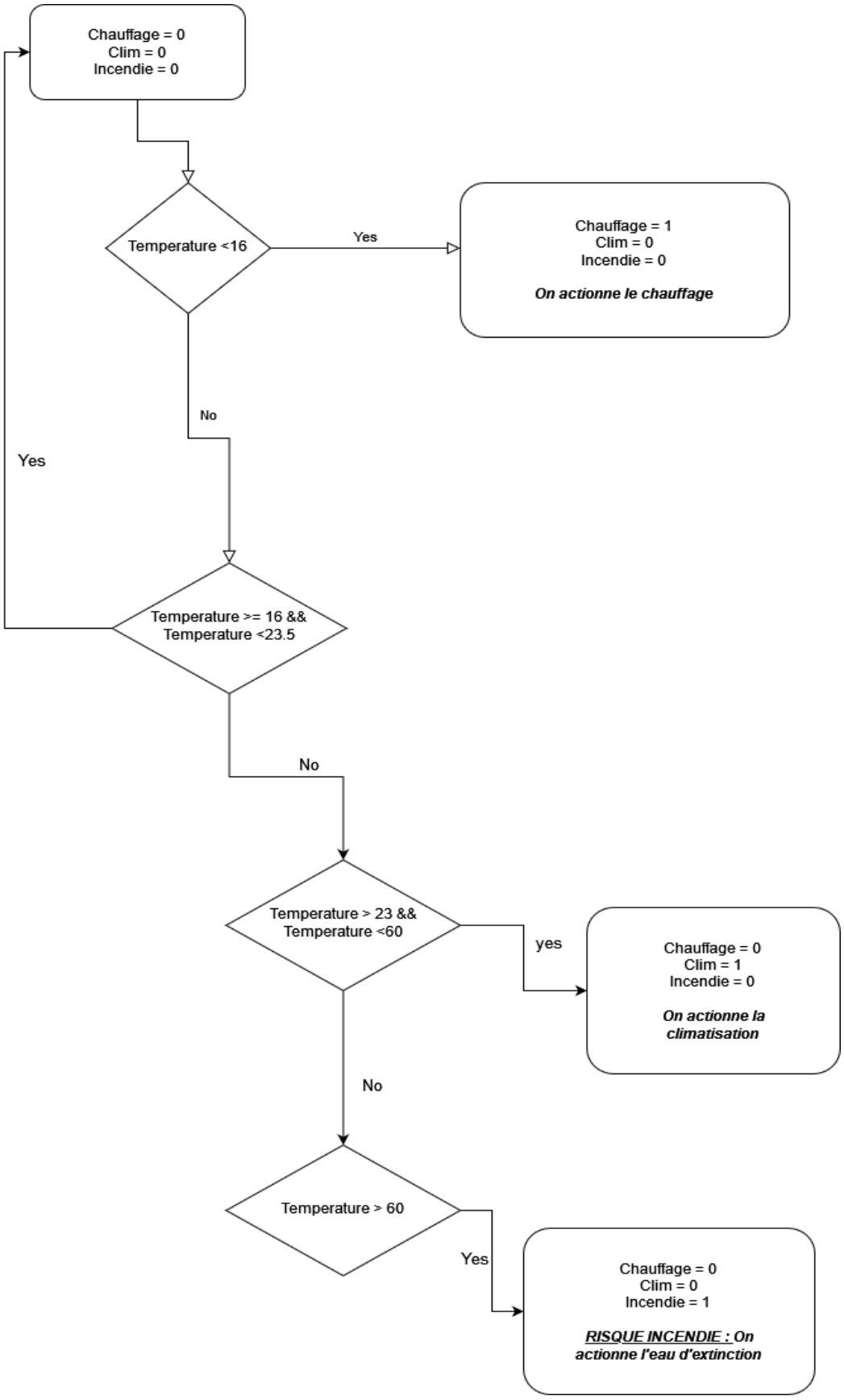

Alarme = 0
Lumière = 0
Porte = 0
Movement > 5 &&
Movement <30
Yes
Alarme=0
Porte=1
Lumière=1
on déclenche la lumière et on vient vérifier, on actionne pas l'alarme mais on verrouille la porte
No
(Movement >= 30)
Yes
Alarme=1
Porte=1
Lumière=1
on allume la lumière, on actionne l'Alarme et on verrouille la porte
No
(Movement <= 5)
Yes
Alarme=0
Porte=0
Lumière=0
on ne fait rien
No

iii) Scénario 3 – Maintenir une température ambiante dans une salle

L'objectif de ce scénario est d'actionner, en fonction de la température reçue :

- Le **système de chauffage** si la température est en dessous d'un certain seuil
- Le **système de climatisation** si la température est au-dessus d'une certaine température
- Ou bien carrément le **système d'eaux d'extinction** si la température dépasse largement 60° degré Celsius

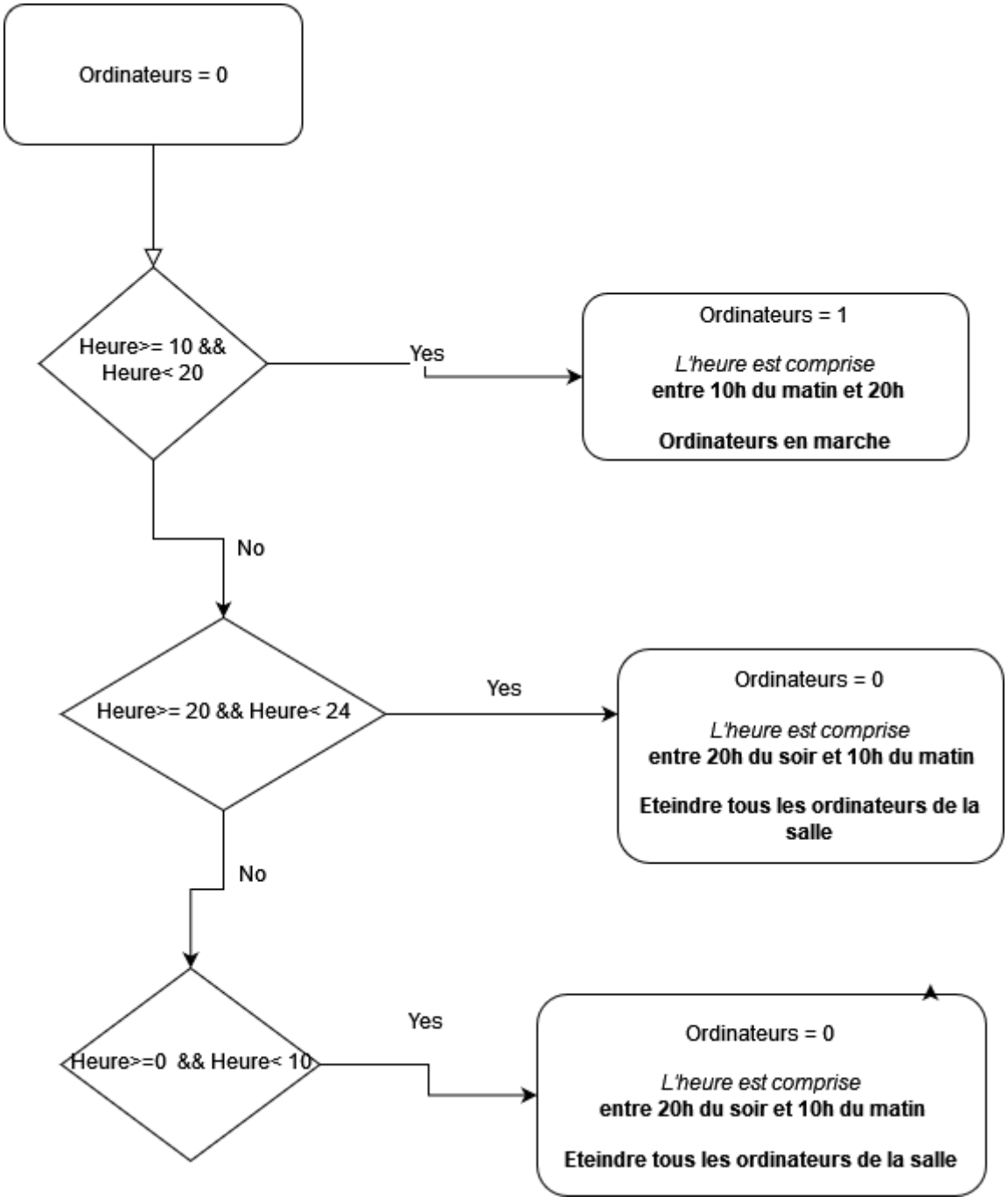
Ci-dessous, l'algorithme que nous avons implémenté :



iv) Scénario 4 - Déconnectez les ordinateurs pendant la nuit

L'objectif de ce scénario est **d'éteindre automatiquement les ordinateurs de la salle** à partir d'une certaine heure.

Ci-dessous, l'algorithme que nous avons implémenté :



III) Tests de l'architecture et des micro-services

Pour cette partie, nous avons décidé de réaliser une vidéo illustrant les tests de chacun de ces scénarios. Voici le lien de la vidéo : https://youtu.be/wGEVAgJIgwM

Conclusion

Ce projet a été une excellente occasion de créer une application Restful qui peut être mise en œuvre dans un scénario réel. Nous avons réussi à développer des services qui peuvent orchestrer l'automatisation d'une salle de travail. Nous avons utilisé Java pour développer les services et OM2M pour organiser nos ressources.

Nous avons utilisé Git pour le contrôle de version et Jira pour la gestion de projet agile. Pour améliorer ce projet, nous pouvons développer une interface utilisateur pour surveiller toutes les valeurs et l'historique des actionneurs.