

Smart Building

Rapport de TP – 5 ISS INSA Toulouse

Nom 1 : **Laurens**

Nom 2 : **Ferrandi**

Prénom 1 : **Pierre**

Prénom 2 : **Emmanuel**

Groupe : **B2**

Tuteur : **Nawal Guermouche**

Dans ce rapport, il vous sera présenté les différentes étapes que l'on a réalisé pour créer cette architecture IoT. En effet, notre application a été développée pour simuler la supervision d'un bâtiment équipé avec différents actionneurs comme le chauffage, la lumière ainsi que des capteurs comme la températures et la luminosité. Avec les données échangées avec les IoTs, et une logique implémentée en java, il nous a été possible de créer des scénarios pour faciliter la vie des utilisateurs. Ainsi quand la température descend, le chauffage peut s'allumer sans intervention humaine.

Organisation du projet	1
Méthode agile	2
Utilisation d'Icecrum	2
Notre application "Smart Building"	4
Choix d'architecture	4
Installation / Utilisation	4
Résultats	4
Améliorations Possibles	5
Backlog Fonctionnel	5
Backlog Dette Technique	5
Backlog Sécurité	6
Conclusion	6

Organisation du projet

Méthode agile

La méthode agile est souvent utilisée dans le domaine du développement logiciel pour sa rapidité à fournir une première version fonctionnelle pour le client. De plus, il permet d'avoir un planning des tâches précis pour chacun des développeurs afin qu'ils sachent ce qu'ils doivent faire et leur future tâche.

Nous avons donc pu expérimenter cette méthode dans ce projet ainsi que découvrir ses nouveaux avantages. Pour cela, nous avons été invité à utiliser le logiciel Icrecum pour organiser nos sprints. Un sprint correspond à une période de temps défini par le manager où l'on doit effectuer des tâches.

Utilisation d'Icecum

Dans un premier temps, nous avons réalisé l'architecture de notre service pour avoir une vue plus précise des tâches que nous allons faire.

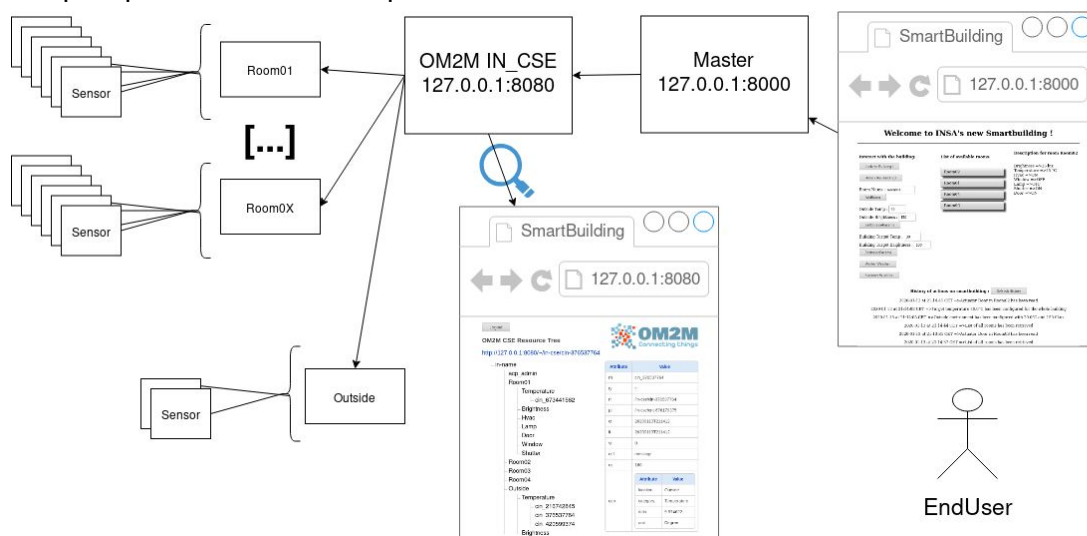


Figure 1 : Architecture de notre application SmartBuilding

Ainsi, avec cette vision, on a pu découper les tâches à faire et les répartir en fonction de nos connaissances. Voici un exemple d'un de nos sprints, qui est la partie centrale et la plus grosse. Elle n'a pas été évidente mais nous avons réussi à tout réaliser.

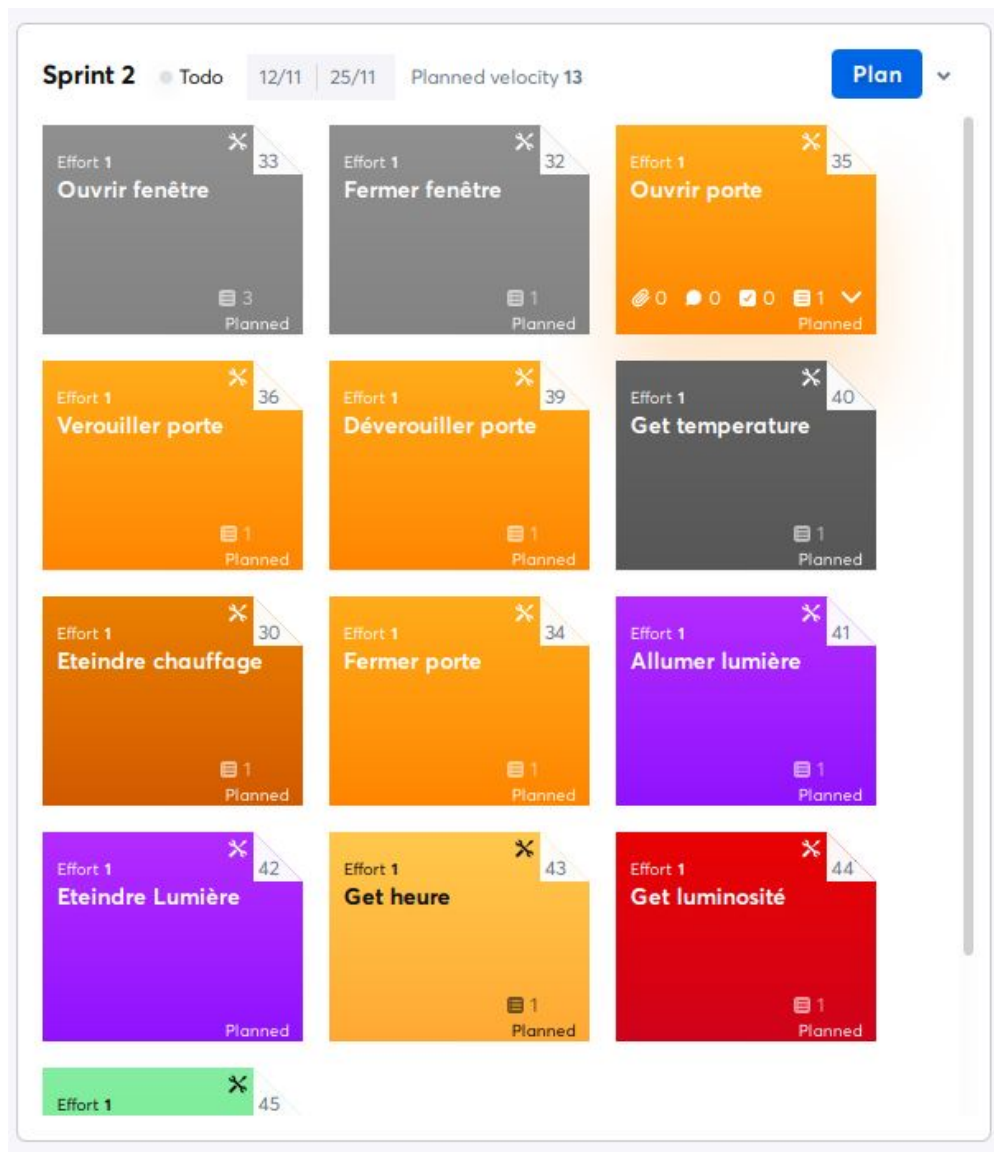


Figure 2 : Sprint 2 pour notre projet SmartBuilding sur Icecream

Nous avons à la fin réalisé 5 sprints au cours de notre projet pour lesquels j'ai réajusté le contenu, comme ci-dessous, respectant ainsi notre ordre de développement:

Sprint 1 : Simuler les capteurs et les actionneurs.

Sprint 2 : Construire un orchestrateur qui va s'interfacer avec les capteurs/actionneurs simulés et simuler un bâtiment.

Sprint 3 : Refactorer la partie capteurs/actionneurs pour le faire avec OM2M.

Sprint 4 : Créer une API REST pour manipuler l'orchestrateur qui simule le bâtiment.

Sprint 5 : Créer une GUI qui implémente l'API REST pour manipuler le bâtiment.

Notre application “Smart Building”

Notre architecture est orientée services car nous avons réalisé un couplage lâche entre l'utilisateur final qui souhaite manipuler un bâtiment et le système qui sert à piloter ce bâtiment à travers ses IoTs.

. Nous avons ajouté une couche d'abstraction entre les fonctions unitaires fournies par l'API OM2M et les fonctions offertes à l'utilisateur final par l'intermédiaire d'un orchestrateur (le master) qui se pilote via une API lui-même.

Cette couche permet à l'utilisateur final de manipuler notre bâtiment “virtuel” sans avoir la moindre connaissance d'OM2M.

Notre architecture orientée service se conforme à divers principes de gestion des services influençant directement le comportement intrinsèque d'une solution logicielle et le style de sa conception:

- L'encapsulation des services.
- Le faible couplage des services avec la maintenance d'une relation réduisant les dépendances.
- Le contrat de service adhère à un accord de communication, collectivement défini avec un ou plusieurs documents de description.
- L'abstraction des services dissimulant la logique du service à l'extérieur.
- La réutilisation des services partageant la logique entre plusieurs services avec l'intention de promouvoir la réutilisation.
- L'autonomie des services
- L'optimisation des services
- La découverte des services depuis leur description extérieure.

L'encapsulation des services:

La gestion de fonctions métiers représentées par des services REST unitaires (gestion de température/luminosité).

Le faible couplage des services avec la maintenance d'une relation réduisant les dépendances:

Possibilité de changer de framework (ne pas utiliser OM2M) sans changer l'API offerte à l'utilisateur final.

Le contrat de services adhère à un accord de communication, collectivement défini avec un ou plusieurs documents de description:

L'API REST exposée à l'utilisateur final formalise la manière d'interagir avec le bâtiment (formalisme qui doit normalement être documenter dans une documentation).

L'abstraction des services dissimulant la logique du service à l'extérieur:

L'utilisateur final n'a pas conscience du framework sous-jacent.

La réutilisation des services partageant la logique entre plusieurs services avec l'intention de promouvoir la réutilisation:

Le système n'est pas assez complexe pour avoir couvert ce concept.

L'autonomie des services:

“SmartBuilding” n'a pas besoin d'avoir une architecture OM2M pour fonctionner. Cependant, le découpage OM2M et “SmartBuilding” est aujourd'hui trop fort, il faudrait refactorer le code et gérer correctement les erreurs pour rendre le projet résilient.

L'optimisation des services:

Pas d'optimisation nécessaire pour l'heure

La découverte des services depuis leur description extérieure:

Nous n'offrons pas de possibilités d'introspection de nos services exposés à l'utilisateur final.

Choix d'architecture

Dans la figure 1, on peut voir les différentes parties nécessaires à notre système. On a choisi de gérer centralement le bâtiment via le composant "Smartbuilding" (master), piloté par une interface javascript, et pilotant un noeud d'infrastructure OM2M à démarrer séparément. L'API créée via notre code JAVA permet d'interagir avec tous les capteurs.

Ensuite, viens le Master qui lui à une vue d'ensemble des salles et peut simuler de manière plus ou moins précise un bâtiment en fonction de l'implémentation souhaitée. Pour visualiser et contrôler tous ces équipements, une interface web est disponible où il est possible d'allumer / éteindre des actionneurs ou récupérer des données de capteurs. Ainsi l'utilisateur va avoir la possibilité de gérer le bâtiment comme bon le semble.

De plus, pour assurer des tâches redondantes, des scénarios ont été implémentés. Nous avons choisi d'implémenter deux scénarios:


- La régulation de la température avec la comparaison entre la température extérieur et la température de la pièce et la température voulue. En fonction du résultat de la comparaison, nous actionnons le chauffage et nous fermons les fenêtres sinon nous fermons le chauffage et ouvrons la fenêtre.
- La régulation de la luminosité de la pièce. Nous mesurons la luminosité de la pièce, la luminosité extérieure et comparons ces luminosités à la luminosité voulue. En fonction, du résultat des comparaisons, nous actionnons ou pas les volets et nous allumons ou éteignons les lampes.

Nous simulons aussi le temps extérieur (météo) en envoyant des données à travers notre GUI.

Logout

OM2M CSE Resource Tree

<http://127.0.0.1:8080/~in-cse/cin-376537764>



– in-name

- acp_admin
- Room01
 - Temperature
 - cin_673441562
 - Brightness
 - Hvac
 - Lamp
 - Door
 - Window
 - Shutter
- Room02
- Room03
- Room04
- Outside
 - Temperature
 - cin_216742845
 - cin_376537764
 - cin_420599374
 - Brightness

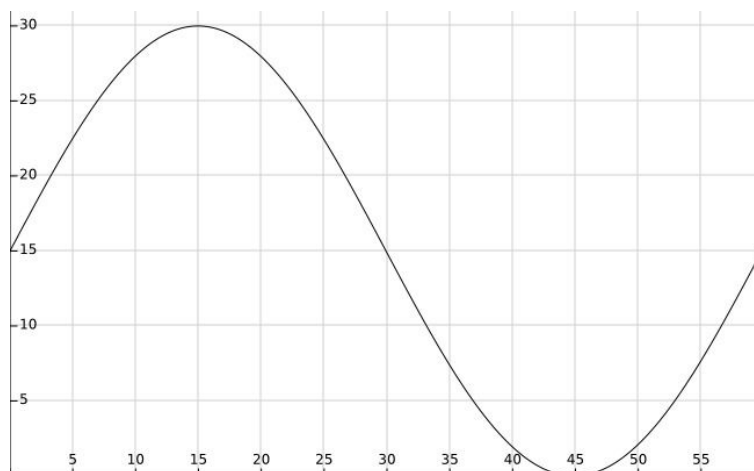
Attribute	Value										
rn	cin_376537764										
ty	4										
ri	/in-cse/cin-376537764										
pi	/in-cse/cnt-676179375										
ct	20200113T211412										
lt	20200113T211412										
st	0										
cnf	message										
cs	160										
con	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr style="background-color: #d9e1f2;"> <th>Attribute</th> <th>Value</th> </tr> </thead> <tbody> <tr><td>location</td><td>Outside</td></tr> <tr><td>category</td><td>Temperature</td></tr> <tr><td>data</td><td>9.974622</td></tr> <tr><td>unit</td><td>Degree</td></tr> </tbody> </table>	Attribute	Value	location	Outside	category	Temperature	data	9.974622	unit	Degree
Attribute	Value										
location	Outside										
category	Temperature										
data	9.974622										
unit	Degree										

Installation / Utilisation

Voici le [lien](#)¹ pour accéder à notre projet qui est stocké sur Github où un README est disponible détaillant l'installation ainsi que l'utilisation.

Résultats

Nous avons écrit la formule, $30 * (\sin(x * 0.105) + 1) / 2$ avec 30°C l'amplitude voulut en degré, permettant de simuler la variation de la température. La courbe , ci-dessus, est le résultat de cette variation.



Courbe 1: Profil de Température simulé sur 1 journée de 60min

¹ lien vers notre Repertoire Git: "<https://github.com/Pierrot31/SmartBuilding>"

L'image de l'interface, montre que nous avons créé 4 pièces, simulé une température extérieure de 20°C et une valeur de luminosité de 150 lux, avec les valeurs de consignes pour le thread qui pilote les actionneurs du bâtiment afin d'ajuster la température et la luminosité des pièces par rapport à ce que les capteurs mesurent dans les pièces.

Concernant la partie extérieure, nous avons entré des valeurs de température et luminosité arbitraires pour simuler des mesures.

Au moment de la capture ci-dessous, nous avons cliqué sur "winter", qui a lancé un thread simulant l'évolution des températures et luminosités extérieures en fonction d'un "programme" dont les valeurs ont été générées avec la courbe 1 ci-dessus.

Nous observons ainsi que le bâtiment, lorsque la température ou la luminosité extérieure évoluent, adapte les consignes sur les actionneurs pour réguler la température en prenant en compte les facteurs extérieurs.

Ainsi, nous mettons en évidence lien entre l'évolution du temps extérieur et l'évolution des positions d'actionneurs pour garantir le suivi de la consigne.

Welcome to INSA's new Smartbuilding !

Interact with the building:	List of available rooms:	Description for room Room02
<div>Lock the Building !</div> <div>Unlock the Building !</div> <div>Room Name : <input type="text" value="Room04"/></div> <div>AddRoom</div> <div>Outside Temp : <input type="text" value="20"/></div> <div>Outside Brightness : <input type="text" value="150"/></div> <div>SetOutsideParams</div> <div>Building Target Temp : <input type="text" value="30"/></div> <div>Building Target Brightness : <input type="text" value="100"/></div> <div>SetInsideParams</div> <div>Winter Weather</div> <div>Summer Weather</div>	<div>Room02</div> <div>Room01</div> <div>Room04</div> <div>Room03</div>	Brightness =>25 lux Temperature =>25 °C Hvac =>ON Window =>OFF Lamp =>OFF Shutter =>ON Door =>ON

History of actions on smartbuilding :

Refresh History

2020-01-13 at 21:14:41 CET =>Actuator Door in Room02 has been read

2020-01-13 at 21:14:08 CET =>Target temperature 30.0°C has been configured for the whole building

2020-01-13 at 21:14:05 CET =>Outside environment has been configured with 20.0°C and 150.0Lux

2020-01-13 at 21:14:44 CET =>List of all rooms has been retrieved

2020-01-13 at 21:13:31 CET =>Actuator Door in Room03 has been read

2020-01-13 at 21:14:37 CET =>List of all rooms has been retrieved

Améliorations Possibles

Nous proposons ci-dessous une liste (Backlog) des tâches que nous pourrions réaliser au cours d'un hypothétique sprint 6 comme améliorations de notre projet. Il se découpe en 3 partie, fonctionnelle, technique et sécurité.

Backlog Fonctionnel

Améliorer l'UI pour plus d'intuitivité dans la manipulation du bâtiment.

Offrir la possibilité de régler la température ou luminosité cible dans le bâtiment indépendamment l'un de l'autre.

Modifier la gestion des logs administrateurs pour les afficher par ordre chronologique, et updaten l'UI de manière événementielle.

Améliorer les actuateurs simulés pour intégrer une notion de course (ouvrir les volets à 50% par exemple).

Améliorer les capteurs simulés pour faire évoluer leurs mesures en fonction des inputs "extérieurs" (ex: le chauffage allumé doit réchauffer la pièce).

Donner la possibilité de gérer les pièces indépendamment les unes des autres.

Découper le bâtiment par étage et proposer des gestions par étages.

Réaliser une suite de Tests fonctionnels.

Persistance des logs.

Backlog Dette Technique

Vérifier si toutes les valeurs mesurées sur les capteurs sont correctement conservées de la lecture à l'affichage (pas d'erreurs dues aux conversions de format).

Uniformiser/Refactorer les URI manipulées dans l'API REST.

Uniformiser les sections de code manipulant des objets JSONs.

Refactorer le format des JSONs manipulés dans l'API.

Refaire les sections de code utilisant des méthodes obsolètes.

Documenter le code (javadoc) pour améliorer sa maintenabilité.

Externaliser dans un package dédié les sections de code statique côté master (calls OM2M par exemple).

Externaliser dans des fonctions dédiées les sections de code statique côté client javascript (calls REST par exemple).

Réaliser une suite de tests unitaires.

Backlog Sécurité

Réaliser une analyse de risque sur le projet pour alimenter ce backlog.

Valider les inputs utilisateurs.

Authentifier l'accès à l'API pour éviter les manipulations non désirées.

Authentifier l'accès à la plateforme OM2M.

Conclusion

Pour conclure, dans ce cours, nous avons créé une application Smart Building permettant de gérer un grand nombre de salle, de capteurs et d'actionneurs. Pour faciliter sa mise en place, nous avons utilisé la technique Architectures Orientées Services qui consiste à utiliser un service pour plusieurs autres applications ou services. Notre application Smart Building est donc utilisable dans un bâtiment avec la possibilité de rajouter des salles autant que l'on veut.