UQÀM **INF4375 - Paradigmes des échanges Internet**
Projet de session

Session: Automne 2018
Nom de l'enseignant: TRÉPANIER MONTPETIT, Yohan
Adresse Courriel: trepanier_montpetit.yohan@uqam.ca
Local de cours: SB R-740
Site Web: http://ytrepanier.com/inf4375

Versionnement de l’énoncé

Version 4 (5 octobre 2018):

- Ajout de 2 conditions dans la logique pour activer le verrouillage des portes selon une heure définie de l’horloge. L’évaluation reste la même (-0.2 par ligne non-fonctionnelle).
- Pied de page
- Détails de remise de code (section évaluation)

Version 3 (3 octobre 2018):

- Version initiale

1- Informations générales

Introduction

L'objectif du projet de session est de développer une application de gestion de maison intelligente.

Le projet est divisé en plusieurs étapes. Chaque étape a sa date d'échéance et qui sera évaluée indépendamment des autres.

Objectifs

L'objectif du projet est de se familiariser avec les concepts suivants:

- Les mécanismes de communication *Publish-Subscribe* et *Request-Response*
- Le développement d'un API REST
- L'utilisation d'un API REST
- L'authentification web

Groupes

Le travail peut être fait individuellement ou en équipe de 2.

Périodes de travail

Les périodes de temps pour réaliser le projet sont les suivantes:

- Durant les périodes de 2h de laboratoire par semaine, durant lesquels du support sera fourni par le démonstrateur de laboratoire;
- En dehors des périodes de cours et laboratoires.

Échéances et évaluation

Le projet est divisé en cinq: quatre étapes de développement et un rapport. Les dates d'échéance et les pointages sont les suivantes:

Étape	**Date**	**Points (total de 30)**
Communication avec des capteurs et actuateurs	19 octobre	**5 points**
API web	9 novembre	**6 points**
Alertes courriel	23 novembre	**4 points**
Authentification	14 décembre	**5 points**
Rapport	21 décembre	**10 points**

Le projet en entier est noté sur 30 points et représente 30% de la note finale du cours de INF4375. Chaque point représente effectivement 1% de la note finale.

L'évaluation se fera de manière **interactive** durant le laboratoire en testant les entrées et les sorties de l'application. Les détails de chaque évaluation sont décrites dans leurs sections respectives dans les pages qui suivent.

Le **code** de l'application devra aussi être soumis avant chaque début de laboratoire d'évaluation par Moodle.

Langage de programmation

Le langage de programmation recommandé est le **Java**. Des cadriciels et outils seront présentés pour utiliser ce langage et faciliter le développement. Le chargé de laboratoire sera en mesure de supporter des problèmes techniques pouvant survenir avec ces outils et langage.

Un autre langage peut être utilisé si désiré mais aucun outil, cadriciel ou support ne pourra être fourni par le chargé de laboratoire ou le chargé de cours. Les formats de sérialisation des communications et des API devront tout de même respecter les contraintes externes (principalement en JSON) peu importe le langage choisi.

Rapport

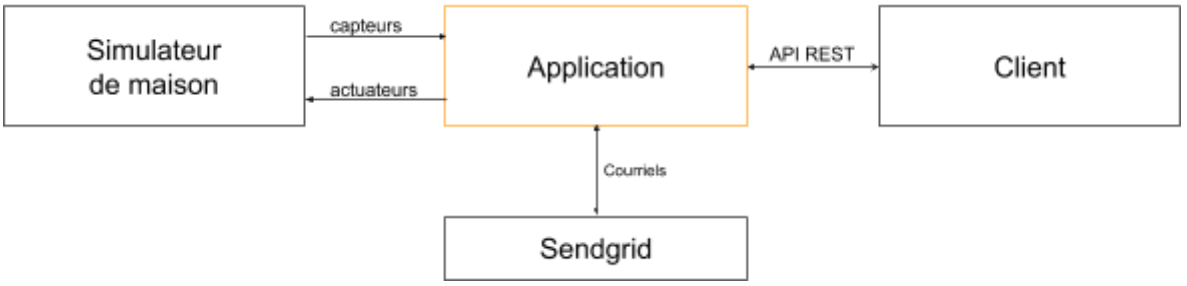
Un rapport fera partie de la remise finale du projet. Plus de détails sur le rapport sont décrits à la fin du document.

2 - Le projet

Le projet est divisé en 4 étapes:

1. Communication avec des capteurs et actuateurs
2. API web
3. Alertes courriel
4. Authentication

L'architecture finale sera la suivante. Le module «Application» représente ce qui sera développé dans le projet et qui rendra la maison «intelligente». Le module «Simulateur de maison» représente l'application fournie pour simuler une maison avec ses capteurs (thermomètre, etc.) et actuateurs (chauffage, etc.). Le module «Client» servira à contrôler l'application par un API et le module Sendgrid permettra d'utiliser un API pour envoyer des courriels.



2.1 - Communication avec des capteurs et actuateurs

La première étape du projet consiste à communiquer avec les capteurs de la maison intelligente. Ces fonctionnalités seront la base du projet et permettront d'ajouter des fonctionnalités plus évoluées par la suite.

La communication se fait selon le patron *Publish-Subscribe* avec la libraire ZeroMQ. La librairie recommandée en Java est JeroMQ. Le projet de base disponible avec ce document contient déjà la librairie.

Les capteurs et actuateurs de la maison intelligente sont intégrées dans le Simulateur, fourni avec ce document. La première étape consiste donc à lancer cette application avec la commande suivante:

```
java -jar inf4375-house-v1.jar
```

Le message suivant devrait apparaître, alors que les capteurs et actuateurs se mettent en marche:

```
House simulator running [...]
```

Le tableau suivant décrit les capteurs. Ceux-ci sont des *Publishers* ZeroMQ et il est possible de s'y abonner en utilisant leur *topic* respectif.

Capteur	**Topic**	**Format et type de données**	**Exemple**
Thermomètre	temperature	Degrés Celcius (float)	21.5
Détecteur de présence	activity	0 ou 1 (int)	1
Horloge	time	HH:MM:SS (string)	15:24:02
Portes	door_lock_sensor	“on”, “off” (string)	on

Le tableau suivant décrit les actuateurs, soit les modules sur lesquels il est possible d'effectuer des actions. Ceux sont des *Subscribers* et attendent de recevoir des messages sur leur topic respectif.

Actuateur	**Topic**	**Messages**
Chauffage	heater	“on”, “off”
Air climatisé	ac	“on”, “off”
Lumières	lights	“on”, “off”
Verrouillage des portes	door_lock	“on”, “off”

Logique de base

Pour être «intelligente», l'application doit prendre des actions par elle-même. Voici les actions qui doivent être prises en compte par l'application. Notez que dans les étapes suivantes du projet, les valeur des conditions seront déterminées par un API REST. Pour cette première étape, il suffit de se fier uniquement aux valeurs du tableau.

Capteur(s)	**Condition**	**Actuateur**	**Action**
Thermomètre	< 19	Chauffage	Démarrer
Thermomètre	> 19	Chauffage	Arrêter
Thermomètre	> 23	Air climatisé	Démarrer
Thermomètre	< 23	Air climatisé	Arrêter
Détecteur de présence	1	Lumières	Ouvrir
Détecteur de présence	0	Lumières	Éteindre
Horloge	= 23:00:00	Verrouillage des portes	Activer
Horloge	= 7:00:00	Verrouillage des portes	Désactiver

La même commande peut être envoyée plusieurs fois. Par exemple, s'il fait 17 degrés, la commande pour allumer le chauffage peut être envoyée continuellement.

Détails sur le simulateur

Le simulateur envoie les données des capteurs approximativement une fois par seconde. Les valeurs des capteurs peuvent être observées et modifiées avec les commandes suivantes. Celles-ci doivent être entrées dans la fenêtre du terminal de l'application Java, suivi d'un retour de ligne (*enter*).

Capteur	**Effet désiré**	**Commande**
Thermomètre	Température à 25.2 degrés	T25.2
Détecteur de présence	Aucune présence	P0
Détecteur de présence	Présence	P1
Horloge	Définir l'heure	H14:56:00
	Afficher les valeurs des capteurs	S
	Afficher les valeurs des actuateurs	A

Les valeurs des actuateurs peuvent uniquement être modifiées à l'aide du mécanisme *Subscribe*. Aucune commande ne peut changer leur valeur manuellement.

L'application Java s'exécute en ligne de commande avec la commande décrite ci-haut.

Les *Publishers* publient sur le **port 5555**. Pour s'y abonner, il faut donc utiliser un *Subscriber* qui se connecte sur le port 5555.

Les *Subscribers* du simulateur tentent de se connecter sur le **port 6666** de votre application. Il faut donc que votre application implémente des *Publishers* sur le port 6666.

Il n'est pas nécessaire de redémarrer le Simulateur si vous redémarrez votre application. ZeroMQ supporte les reconnections automatiques.

Note sur les fils d'exécution

Afin d'éviter certaines complexités, il est recommandé d'implémenter l'application avec un seul fil d'exécution (thread). Voici la logique recommandée:

1. Attendre les données avec un *Subscriber* (bloquant)
2. Lorsque les données arrivent:
    a. traiter les données;
    b. faire des *Publish* au besoin;
    c. faire d'autres opérations au besoin.
3. Retour à l'étape 1

Évaluation

Cette étape sera évaluée en modifiant les valeurs des capteurs sur le Simulateur. Par exemple, la température sera mise à 17 degrés dans le simulateur, et l'application devra envoyer la commande pour démarrer le chauffage.

Afin de valider le fonctionnement, votre application doit:

- **Afficher un message lorsqu'elle voit un changement dans les capteurs.** Par exemple, si la température passe de 21 à 22, l'application doit afficher «température: 22»
- **afficher un message lorsqu'elle effectue une action.** Par exemple, si elle tente de démarrer le chauffage, elle doit afficher "Démarrage du chauffage".

L'évaluation sera faite en utilisant les commandes sur le simulateur pour changer les valeurs des capteurs et afficher les valeurs des actuateurs.

Pointage (total de **5 points**):

L'application reçoit des données des capteurs	**2 points (+0.5 par capteur)**
L'application contrôle des actuateurs	**2 points (+0.5 par actuateur)**
Toute la logique respecte le tableau de la section "Logique de base"	**1 point (-0.2 par ligne du tableau non fonctionnelle).** Si 2 lignes sont non fonctionnelles, 1-(2x0.2) = 0.6

2.2 - API Web

La seconde étape du projet consiste à intégrer un API REST pour contrôler certains aspects de l'application. L'application doit permettre de modifier des valeurs par un API REST.

En Java, l'utilisation de com.sun.net.httpserver.HttpExchange est recommandée. Votre serveur REST doit écouter les requêtes sur le **port 4444** afin de faciliter l'évaluation.

Pour supporter l'API, le concept de "thermostat" est intégré dans l'application. Il permet de définir une température que l'on désire viser. Si la température est **2 degrés** plus basse ou plus haute que la température visée, l'application doit lancer le chauffage ou l'air climatisée. Cette valeur sert donc de condition dans le tableau de la section *Logique de base*.

Le tableau qui suit décrit les *endpoints* à implémenter:

Endpoint	**Méthode**	**Paramètres**	**Description**	**Codes de retour**
/thermostat	GET	-	Connaître la température actuelle du thermostat	200
/thermostat	PUT	*temperature: float	Définir la température visée	200
/temperature	GET	-	Connaître la température **actuelle** du capteur	200
/door_lock	GET	-	Connaître l'état du verrouillage	200
/door_lock	PUT	*lock: bool	Verrouiller ou déverrouiller les portes	200
/presence	GET	-	Savoir si quelqu'un est présent	200

* Paramètre obligatoire

Autre note sur les fils d'exécution

Afin d'éviter encore une fois des conflits avec des fils d'exécution, il est recommandé de lancer un fil d'exécution séparé qui écoute les requêtes REST mais d'utiliser des fichiers pour conserver les valeurs importantes. La logique suivant peut être appliquée:

Fil 1	**Fil 2**
1. Attendre les données avec un *Subscriber* (bloquant) 2. Lorsque les données arrivent **a. Lire un fichier *thermostat_request.json*** ***b.* Lire un fichier *door_request.json*** **c. Appliquer les *requests* au besoin** d. Traiter les données e. Faire des *Publish* au besoin f. Faire d'autres opérations au besoin **g. Enregistrer les valeurs dans un fichier *status.json*** 3. Retour à l'étape 1	**1. Attendre des commandes REST (bloquant)** **2. Lorsque les données arrivent** **a. Si c'est un GET, lire le fichier thermostat.json, ou status.json** **b. Si c'est un PUT, écrire dans *thermostat_request*.json ou *door_request.json***

Note: Si vous préférez, vous pouvez aussi utiliser une vraie base de données (MySQL, PostgreSQL) pour garder les donnée en mémoire. C'est la solution qui serait utilisée dans un projet réel pour gérer la persistence des données.

Évaluation

Cette étape est évaluée en envoyant des commandes REST à votre application et en observant l'impact sur le simulateur. Par exemple, on pourrait mettre le thermostat à -100 degrés par un PUT sur /thermostat et s'assurer que le chauffage démarre. L'application doit répondre à la requête HTTP et doit appliquer la valeur sur le simulateur

Pointage (total de **6 points**):

/thermostat GET	**0.5 point (si le serveur répond et que la valeur est juste)**
/thermostat PUT	**2 point (+0.5 si le serveur répond, +0.75 si le thermostat lance le chauffage au bon moment, +0.75 s'il lance l'air climatisé au bon moment)**
/temperature GET	**0.5 point (si le serveur répond et que la valeur est juste)**
/door_lock GET	**0.5 point (si le serveur répond et que la valeur est juste)**
/door_lock PUT	**2 points (+0.5 si le serveur répond, +1.5 si l'action est appliquée sur l'actuateur)**
/presence GET	**0.5 point (si le serveur répond et que la valeur est juste)**

2.3 - Alertes courriel

La troisième étape du projet est de lancer des alertes par courriel s'il fait trop chaud. Le service recommandé est SendGrid. L'objectif de cette intégration est d'utiliser un API REST existant.

Requis de cette étape:

1. Lorsque la température monte au-dessus de **30 degrés**, votre application doit envoyer un courriel (un seul!).
2. Suite à l'envoi, l'application doit afficher le nombre total de courriels envoyés par la plateforme.

Détails sur SendGrid:

- Vous devrez vous créer un compte sur la plateforme et suivre les étapes pour accéder à une clef d'API
- Le plan "Free" permet sans problème d'envoyer assez de courriels pour les besoins du projet.
- Vous devez utiliser le service d'API REST, et non le service de relais SMTP
- Vous pouvez utiliser les librairies disponibles pour votre langage (Java ou autre)
- La documentation d'API sera utile. Voir les sections MAIL SEND et STATS (le champ *stats/metrics/requests* fournit le nombre de requêtes de courriels envoyés)

Évaluation

Note: Dans le sujet du courriel, indiquez le nom de l'un des membres de l'équipe.

Pointage (total de **4 points**):

L'application peut envoyer un courriel par l'API REST	**1 point**
L'envoi de courriel se produit au-dessus de 30 degrés	**1 points**
L'application envoie un seul courriel par incident (pas plusieurs courriels consécutifs)	**1 point**
L'application affiche le nombre de courriels envoyés	**1 points**

2.4 - Authentification

La dernière étape du projet consiste à mettre en place un processus d'authentification qui permettra de sécuriser l'accès à l'API REST. Le processus requis est le suivant:

1. Le client fait une requête POST sur le endpoint */auth* en spécifiant un nom d'utilisateur et un mot de passe sous le format de HTTP Basic Authentication.
2. Le serveur retourne un des codes suivants:
    a. 400 si des champs sont manquants
    b. 401 si le nom d'utilisateur ou le mot de passe n'est pas bon
    c. 200 si le nom d'utilisateur et le mot de passe sont valides
3. Dans le cas d'une réponse valide, le serveur retourne aussi un témoin (*cookie)* dans l'en-tête HTTP.

Une fois ce processus effectué, le client possède un témoin de session et pourra l'envoyer dans toutes les requêtes suivantes pour accéder à l'API. Le serveur devra refuser la requête avec un code 401 si le témoin est erroné ou s'il n'est pas présent.

Pour faciliter l'évaluation, le système doit accepter les paramètres suivants:

- **Nom d'utilisateur: inf**
- **Mot de passe: 4343**

Les comptes peuvent être sauvegardés dans une base de données, dans un fichier texte, ou simplement inscrit dans le code. Par contre, le mot de passe doit avoir passé dans une fonction de hachage et **il doit être impossible de voir le texte "4343" dans le code.**

Évaluation

Pour évaluer les réponses avec/sans cookie, une requête sur un endpoint aléatoire du tableau de la section API Web sera envoyée.

Pointage (total de **5 points**):

Serveur retourne un cookie sur /auth si le nom d'utilisateur et le mot de passe sont justes.	**1 point**
Serveur ne retourne pas de cookie sur /auth si le nom d'utilisateur et le mot de passe sont invalides.	**1 point**
Serveur retourne des réponses aux appels d'API si le cookie est valide	**1 points**
Serveur retourne un code 401 aux appels d'API si le cookie est invalide	**1 points (+0.5 pour cookie absent, +0.5 pour cookie invalide)**
Utilisation du hachage	**1 point si le mot de passe est validé en le comparant à la valeur hachée**

3 - Le rapport

Le rapport doit permettre de comprendre d'abord comment s'est déroulé le développement du projet, puis évaluer l'état actuel de l'application tel que développé suite à toutes les étapes. Il doit aussi contenir la documentation de l'API REST qui suit les bonnes pratiques de ce genre de documentation (voir Twitter). Utilisez le tableau de pointage pour structurer et détailler votre rapport.

Pointage (total de **10 points**):

Déroulement du projet	
Difficultés rencontrées	**1 point**
Éléments pertinents appris	**1 point**
État de l'application (qu'est-ce qui est bien et qu'est-ce qui manque si on voulait réellement lancer cette application?)	
Sécurité de l'authentification	**1.5 points**
Sécurité du reste de l'application	**1.5 points**
Fiabilité, robustesse, etc.	**2 points**
Scalabilité	**1.5 points**
Documentation d'API complète et détaillée	**1.5 points**