

# Rapport Projet Interaction Distribuée : ID-PHYCS

## I. Architecture générale

Dans ce projet, nous devons utiliser des slots composés d'agrégats. Un agrégat représente un capteur connecté en filaire renvoyant une ou plusieurs données. À travers MQTT sous python, nous allons implémenter des scripts permettant leur envoi vers un serveur. Celui-ci souscrit à tous les topics de tous les agrégats du système afin qu'il puisse recevoir leurs données. Ensuite, le serveur écrira ces données dans les fichiers correspondants afin de simuler une mémoire des données. Enfin, une API REST tournera en continu et mettra à jour toutes les minutes les données d'un slot composé de plusieurs agrégats.

## II. Une preuve de concept

Afin d'appliquer l'architecture et le cahier des charges du projet, les agrégats seront représentés par une Raspberry 3. Deux différents capteurs seront branchés sur la Raspberry 3 : un capteur de température/humidité (SEN-DHT22) et un capteur de présence (SEN-PIRSR501).

Sous MQTT, le capteur d'humidité (DHT22Tes2t.py) publie des données sur les topics suivants :

- "/Agregat1/temp" : température en degré Celsius,
- "/Agregat1/humid" : humidité de la salle

Ce capteur est branché dans les broches suivantes (cf figure 1) :

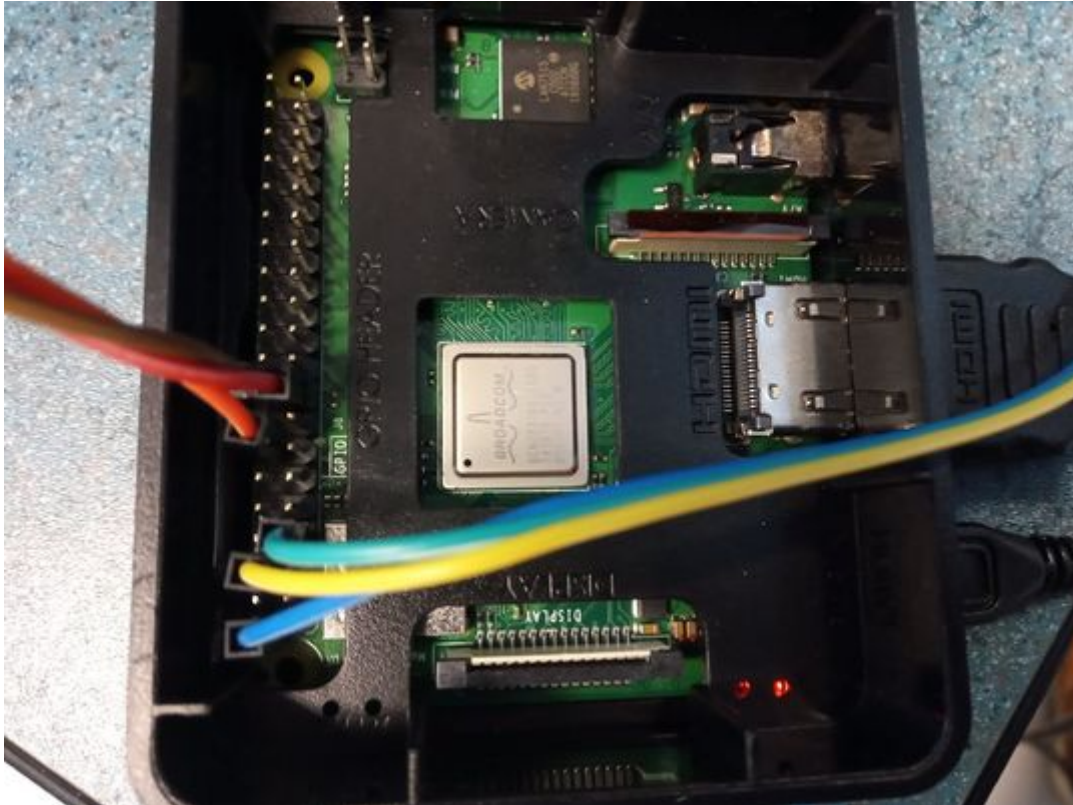
- 3.3V = 17<sup>e</sup> broche
- Résultat = GPIO 24 (18<sup>e</sup> broche)
- gnd = 14<sup>e</sup> broche

Sous MQTT, le capteur de présence (pirTest2.py) publie des données sur le topic suivant :

- "/Agregat1/pres" : horaire en seconde à laquelle le mouvement a été capté

Ce capteur est branché dans les broches suivantes (cf figure 1) :

- 5V = 2eme broche
- Résultat = GPIO 4 (7eme broche)
- gnd = 6eme broche



*Figure 1 : câblage des deux capteurs sur la Raspberry.*

### **III. Un peu plus sur des agrégats**

Les échanges de données entre les agrégats et les slots se font par l'API REST à l'aide de la librairie Flask. En utilisant un format JSON nous pouvons, lorsque le projet tourne, ouvrir une page internet en utilisant l'adresse IPV4 de l'appareil faisant tourner le serveur. Différentes pages internet peuvent être ouvertes représentant chaque agrégat du système. Il existe donc 2 pages possibles à afficher sur internet si nous ajoutons un deuxième slot. Par ailleurs l'adresse IPV4 de l'appareil serveur est très importante, car permet aux Raspberry 3 de se connecter via MQTT et d'envoyer des données aux bons agrégats sans erreur de connexion. Le port de connexion est également important, et dans notre exemple, voici les deux adresses possibles à afficher (Port 1883) :

- le 1er slot : <http://10.60.160.97:5000/api/bureau/>
- le 2<sup>e</sup> slot : <http://10.60.160.97:5000/api/foyer/>

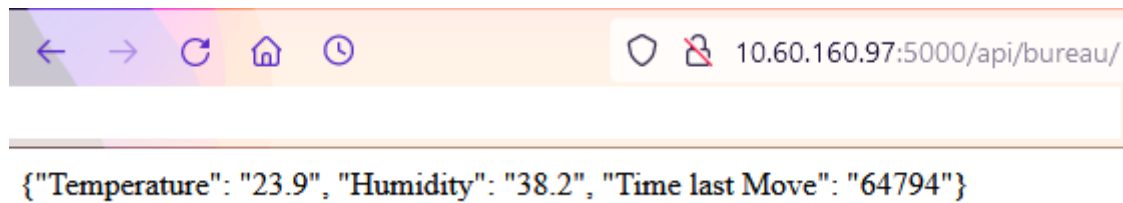


Figure 2 : Affichage du 1er slot sur une page web.

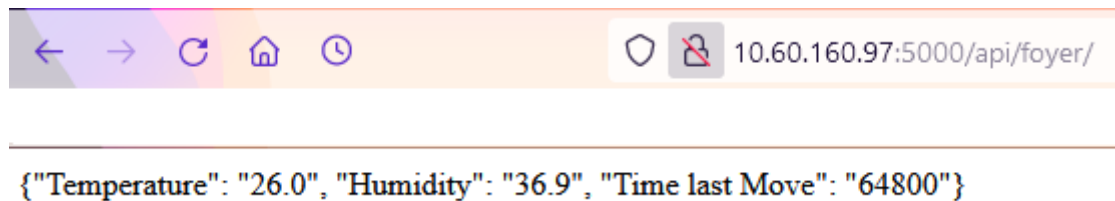


Figure 3 : Affichage du 2<sup>e</sup> slot sur une page web.

```
bytoxiik@DESKTOP-UV7Q502:/mnt/c/Users/cedri/Documents/ecole_inge/3A/ReseauID$ python3 ApiRest.py
* Serving Flask app 'ApiRest' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://10.60.160.97:5000/ (Press CTRL+C to quit)
10.60.160.97 - - [18/Jan/2022 14:33:18] "GET /api/foyer/ HTTP/1.1" 200 -
10.60.160.97 - - [18/Jan/2022 14:35:00] "GET /api/foyer/ HTTP/1.1" 200 -
10.60.160.97 - - [18/Jan/2022 14:36:04] "GET /api/foyer/ HTTP/1.1" 200 -
10.60.160.97 - - [18/Jan/2022 14:38:15] "GET /api/bureau/ HTTP/1.1" 200 -
```

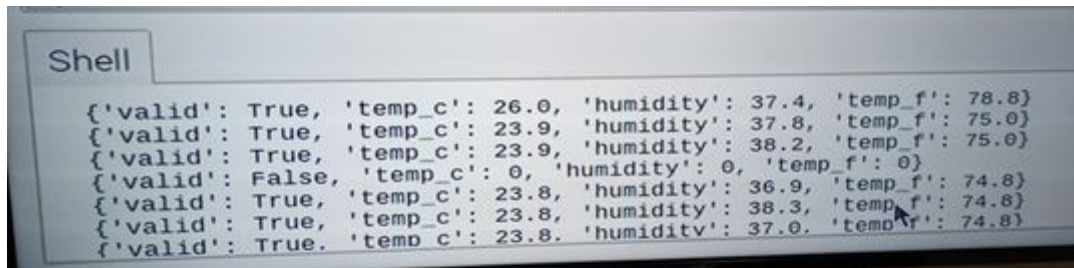
Figure 4 : Affichage de l'API REST sur le terminal.

De plus, l'ajout d'un deuxième slot, représente l'ajout d'une deuxième Raspberry avec les mêmes configurations que le 1er slot (mêmes capteurs, branchements et données envoyées), mais dans une pièce différente. Les données envoyées par ce 2<sup>e</sup> slot sont publiées sur les topics suivants :

- "/Agregat2/temp" : température en degrés Celsius,
- "/Agregat2/humid" : humidité de la salle
- "/Agregat2/pres" : horaire en secondes à laquelle le mouvement a été capté

Dans la même idée, le serveur souscrit également à ces topics et écrit les données dans des fichiers correspondants. En réalité, le serveur n'écrit de données dans ces fichiers qu'uniquement s'il reçoit des données des topics auxquels il a souscrit. En l'occurrence, chaque agrégat/capteur envoie des données toutes les deux secondes. Le serveur doit donc respecter cette exigence et écrire les données lors de leur réception.

Autre chose : concernant le capteur de température/humidité. Parfois, celui-ci apporte des données aberrantes et qui doivent donc être ignorées. Le package pigpio-dht permet tout d'abord de gérer ce capteur en particulier, mais surtout de gérer ce problème de réception de données aberrantes. Lorsque nous faisons appel à une mesure, le capteur envoie un dictionnaire comportant 4 informations. Voici un exemple :

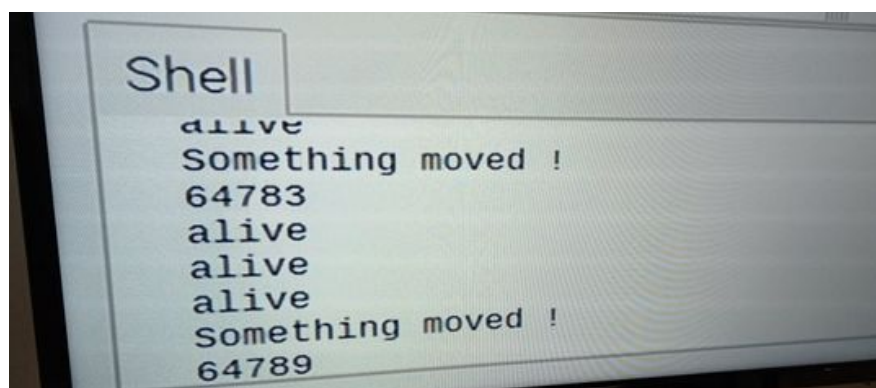


```
Shell
{'valid': True, 'temp_c': 26.0, 'humidity': 37.4, 'temp_f': 78.8}
{'valid': True, 'temp_c': 23.9, 'humidity': 37.8, 'temp_f': 75.0}
{'valid': True, 'temp_c': 23.9, 'humidity': 38.2, 'temp_f': 75.0}
{'valid': False, 'temp_c': 0, 'humidity': 0, 'temp_f': 0}
{'valid': True, 'temp_c': 23.8, 'humidity': 36.9, 'temp_f': 74.8}
{'valid': True, 'temp_c': 23.8, 'humidity': 38.3, 'temp_f': 74.8}
{'valid': True, 'temp_c': 23.8, 'humidity': 37.0, 'temp_f': 74.8}
```

Figure 5 : Affichage du script du capteur de température/humidité.

Pour notre application, nous nous intéressons aux données température et humidité lorsque le champ “valid” du dictionnaire est à ‘True’ (mesures non aberrantes). Le script ne publie ces deux données qu’uniquement si deux secondes sont passées et si elles sont valides. Il est très important que le démon pigpiod soit lancé, sinon le script ne marche pas (***sudo pigpiod***, dans le terminal).

Concernant le 2<sup>e</sup> capteur, celui de présence, aucune librairie particulière n'est nécessaire. Le capteur envoie un signal à la broche résultat lorsqu'il détecte un mouvement. En conséquence, seuls les changements de signaux dans la broche concernée (GPIO 4) sont à analyser. Lors d'un mouvement, le script publie donc l'heure à laquelle le mouvement a été capté en secondes. En revanche, le script affiche “alive” toutes les 2 secondes pour signifier que le capteur de présence est bien en vie.



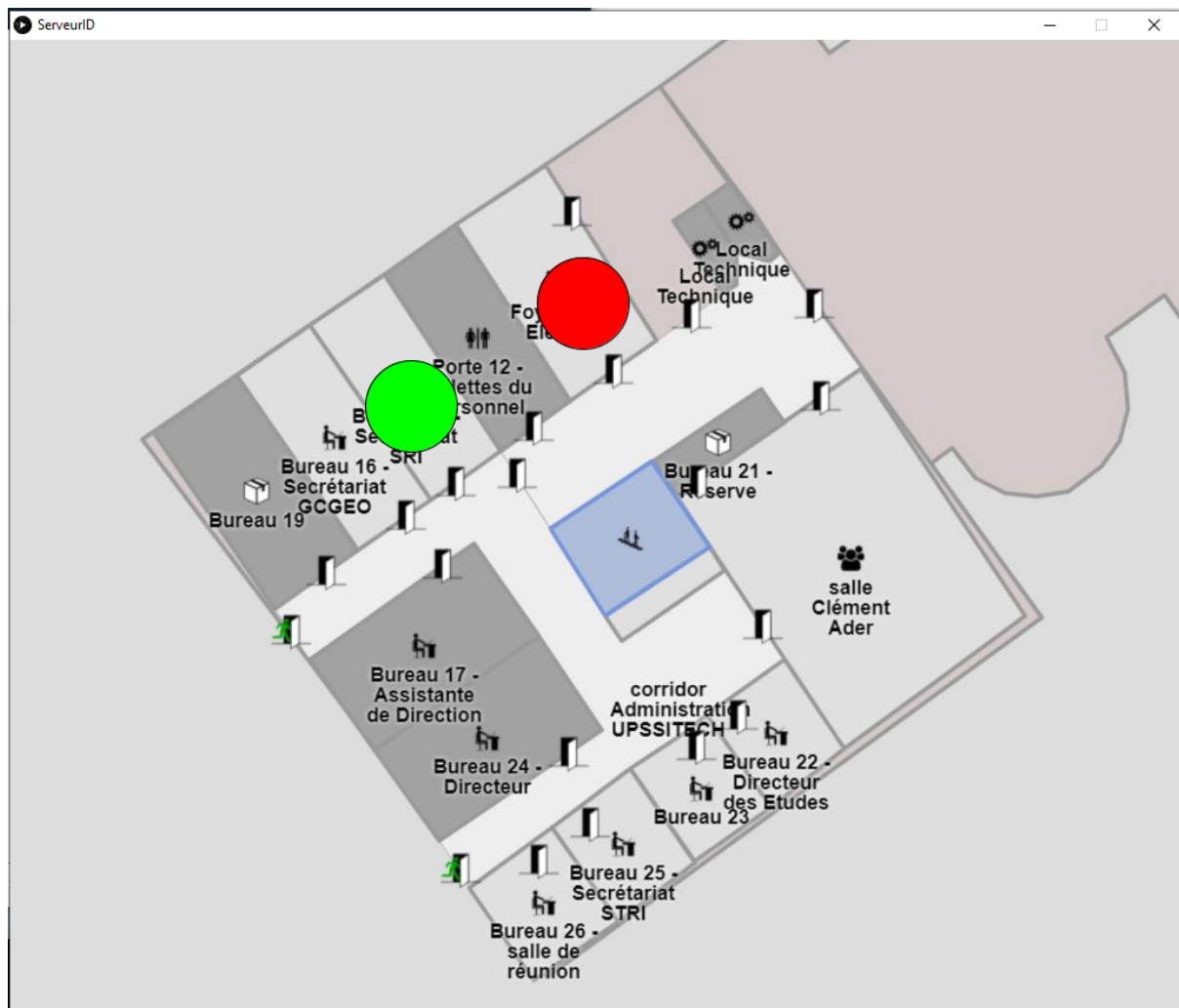
```
Shell
alive
Something moved !
64783
alive
alive
alive
Something moved !
64789
```

Figure 6 : Affichage du script du capteur de présence.

## IV. Un service opérationnel

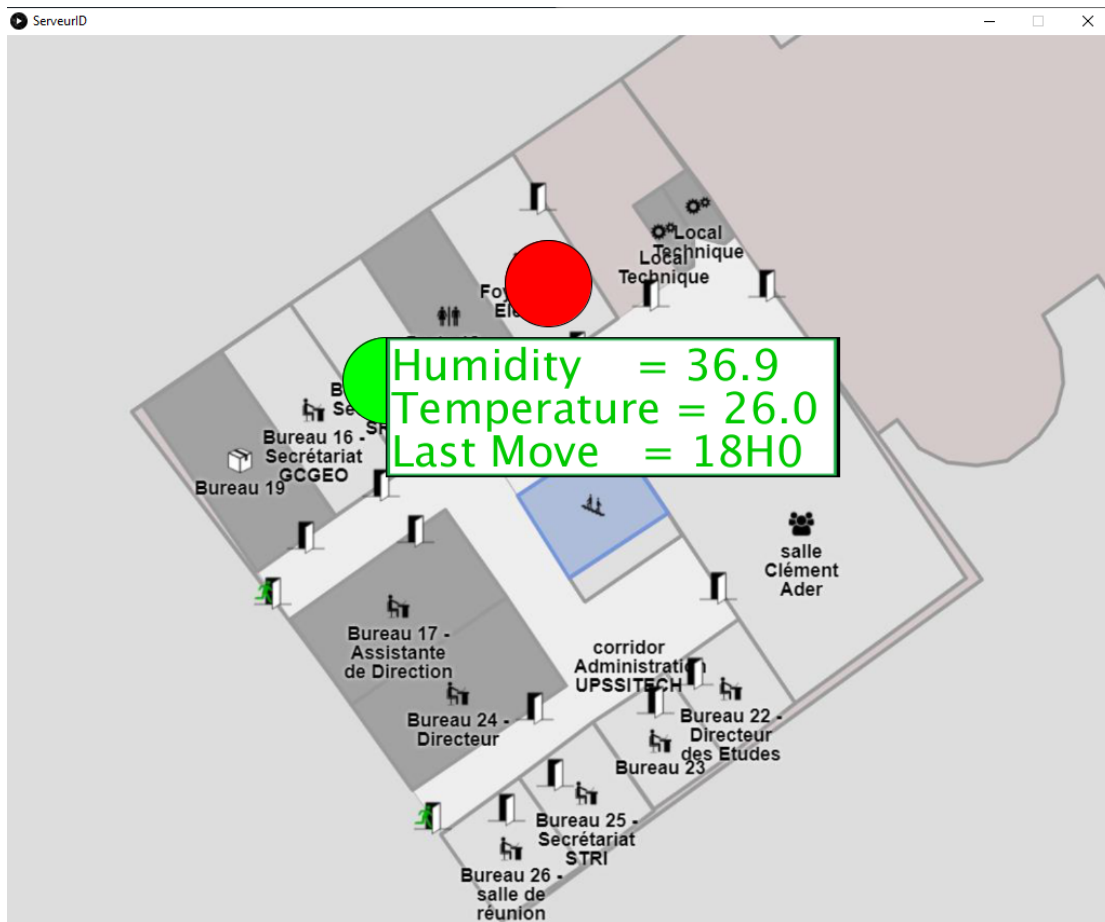
Les agrégats (représentés par une Raspberry 3) communiquent avec un serveur (PC) à l'aide de MQTT. Des données sont écrites dans des fichiers afin de les sauvegarder. En revanche, jusque-là nous ne voyons absolument rien sur les échanges de données entre ces systèmes hormis par API REST. En utilisant Processing et shiftr.io, nous allons pouvoir observer chaque changement de données ainsi que les échanges MQTT en temps réel.

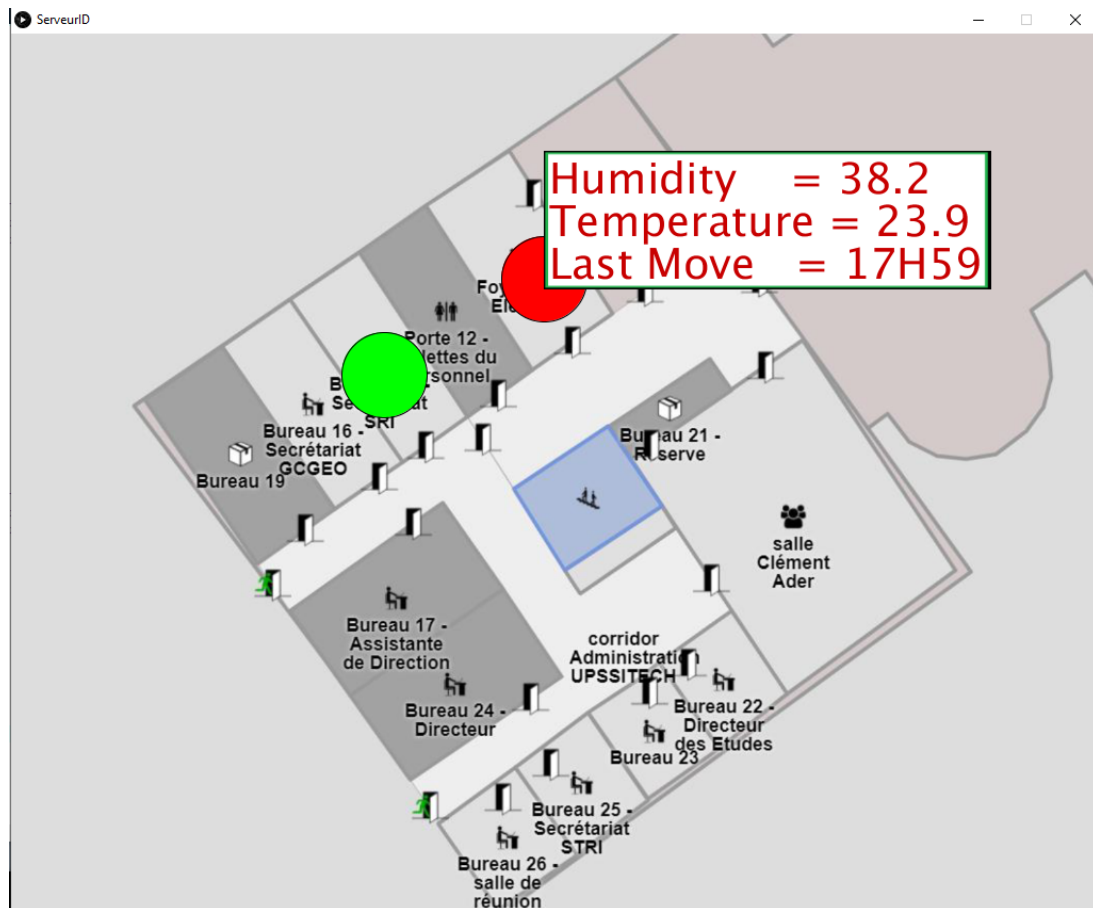
Sous Processing, une interface graphique permet de visualiser deux slots représentant deux Raspberry 3 avec les mêmes configurations (Figure 7).



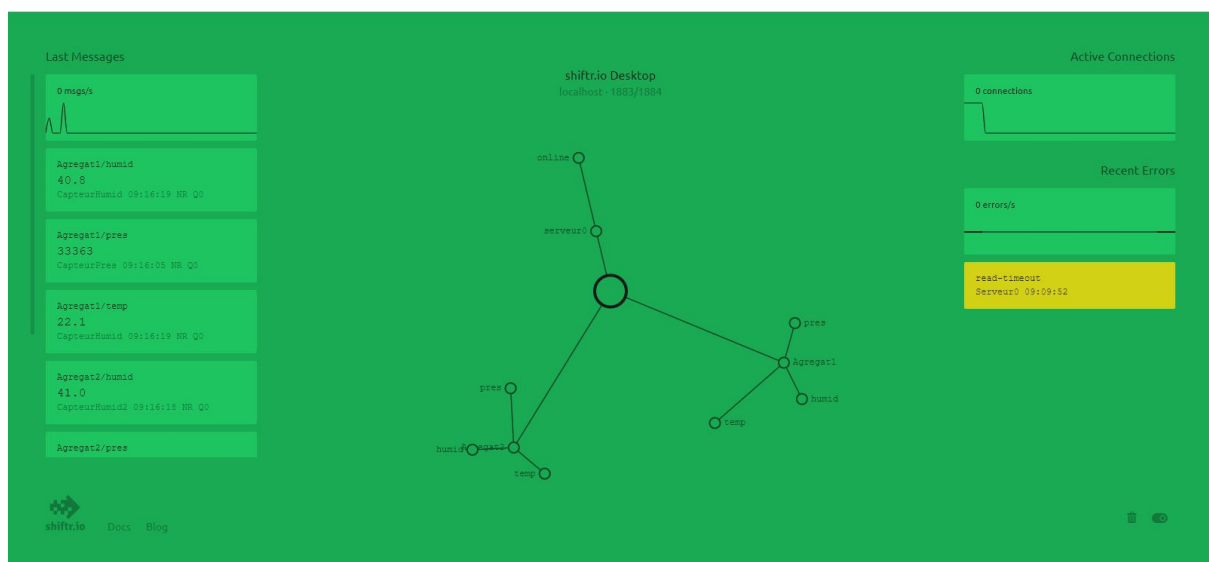
*Figure 7 : Interface sous processing*

Nous pouvons observer deux cercles de couleurs (représentant chacun 1 slot) et lorsque nous approchons une souris sur le cercle, nous pouvons alors afficher les informations des capteurs en temps réel :





Enfin sous shiftr.io, nous allons observer les données échangées par MQTT. À noter que le serveur envoie toujours une donnée "ON" pour signifier qu'il est en vie.



*Figure 8 : Système d'agrégats sous shiftr.io (inactif)*



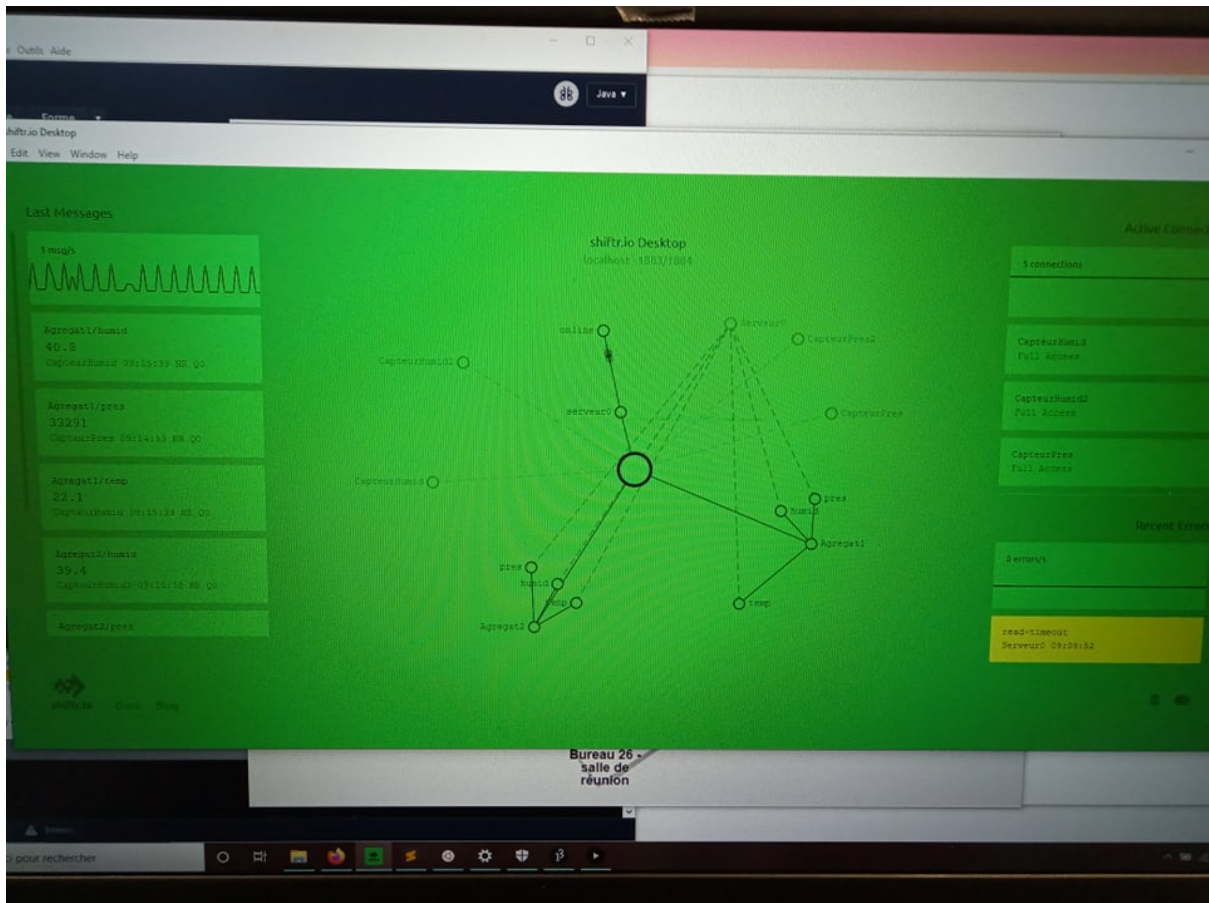


Figure 9 : Système d'agrégats sous shift.io (actif)

Une vidéo démonstrative du système en fonctionnement est disponible dans le répertoire github ServeurID. (Interface processing, interface shift.io ainsi que le terminal recevant les données des capteurs)

## V. Avantages, Inconvénients et limites

L'avantage principal de l'architecture choisie est qu'elle est très simple à mettre en place. Il suffit de câbler quelques capteurs aux Raspberry 3 et de lancer dans chacun des scripts pythons ainsi qu'un serveur et le tour est joué.

Quelques problèmes ont été rencontrés avec MQTT, en particulier avec les connexions entre les appareils client et serveur. Pendant l'appel à la commande `client.connect(ip,port)`, le script n'arrivait jamais à se connecter à l'appareil et affichait tout le temps "time out" pendant la connexion. L'inconvénient principal est donc qu'il faut être connecté au même accès Internet et qu'il soit suffisamment accessible pour supporter des connexions MQTT. Typiquement, les partages de connexion 4G et Eduroam n'ont pas marché.



Un autre problème avec le capteur de température DHT22 où il a fallu chercher plusieurs librairies avant d'en trouver une qui permette au capteur de fonctionner convenablement (c'est-à-dire pigpio-dht).

Chaque agrégat est représenté par un script python. Il serait peut-être plus intéressant d'avoir un script qui fasse tourner plusieurs agrégats, car si nous imaginons un problème où il y a des dizaines de pièces (ou slots) à gérer, on pourrait se perdre dans les scripts. Dans notre cas, ça marche plutôt bien.

L'accès aux informations des capteurs par l'interface graphique Processing se fait en lisant des fichiers textes. Le but était de séparer chaque tâche du système et précisément ne pas avoir Processing faisant tourner le serveur MQTT ainsi que l'interface graphique.