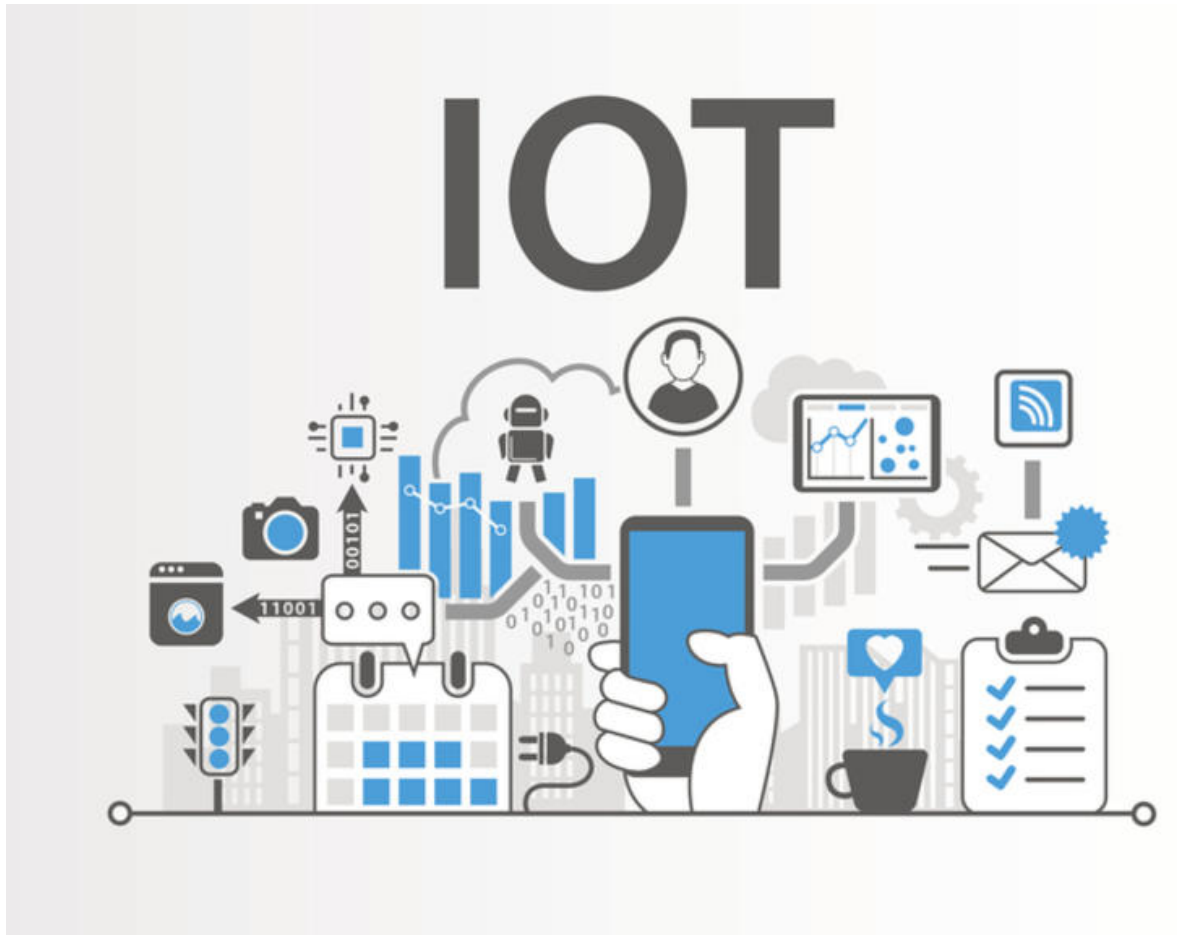


Straputicari Luca

Dourlent Maxime

Fernandez Mickael

Signal Alexandre



# Rapport de IOT

*Documentation de la partie IOT de la SAE dev APP*

---

<b>Introduction</b>	<b>2</b>
<b>Objectifs :</b>	<b>2</b>
Explication du code :	4
Récupère les données de capteurs :	4
Récupère la configuration depuis un fichier tiers :	6
Ecris dans le fichier releve.txt :	8
Affiche une alerte en fonction du maximum atteint :	9
Retour du code :	10
Contenu du fichier releve.txt :	10
Contenu du fichier configuration :	11
Alerte affiché :	12
<b>Conclusion :</b>	<b>13</b>

---

## Introduction

Ce document à pour but d'expliquer la partie IOT dans la SAE. Nous allons d'abord rappeler les objectifs, s'en suivra des captures d'écran du code avec les explications concernant ce dernier. Enfin, nous montrerons le résultat final, montrant que nous avons atteint les objectifs fixés pour cette partie IOT.

## Objectifs :

Nous devons écrire un programme qui permet de :

- Récupérer les données de plusieurs capteurs situés dans des entrepôts fictifs;
- Avoir un fichier de configuration contenant les noms des capteurs qui nous intéressent, le nom du serveur et le port du serveur;
- Faire en sorte que les données des capteurs s'écrivent dans un autre dossier nommé *releve.txt* qui contiendra toutes les données des capteurs sélectionnés et les possibles alertes;
- Implémenter un timer dans le programme afin que les données soient récupérées à intervalles réguliers.

## Explication du code :

En voici le contenu de la fonction on\_alarm :

```
def on_alarm(nos, frame):

    #TODO
    # recharger 'config'
    # parcourir 'donnees' en prenant que les valeurs correspon
    # envoyer payload à "on_write"
    # relancer "alarm"
    print("\nTentative de récupération des données...")
    global donnees, timer

    donneesFiltrees = {}
    config = get_config()

    if len(donnees) != 0 :

        for key, value in config.items():
            if value == "0":
                donneesFiltrees[key] = donnees["object"][key]

        on_write(donneesFiltrees)

        donnees = {}

    signal.alarm(timer)

    #test pour voir dans la console
    #jsonMsg = json.loads(msg.payload)

    #print(jsonMsg["rxInfo"][1]["name"])

#on récupère les paramètres de connexion en variables globales
#on récupère dans config le dictionnaire des données à récupérer
config = get_config()
connect()

client.on_message = on_message
signal.signal(signal.SIGALRM, on_alarm)
signal.alarm(timer)

client.loop_forever()
```

On affiche d'abord *"Tentative de récupération des données..."*, puis on récupère la configuration grâce à **"config = get.config()"** (au passage, on crée aussi un dictionnaire **"donneesFiltrees"**).

Avec un if, on vérifie si la longueur de **"donnees"** n'est pas nulle. On parcourt les données inscrites dans le fichier de configuration grâce à un for, et grâce au **"if value == '0'"**, on récupère uniquement les valeurs des objets correspondant à 0 (True) dans le dictionnaire, afin de les ajouter dans **"donneesFiltrees"** via **"on\_write"**.

Ainsi, le timer est défini dans le fichier de configuration et l'alarme, via **"signal.alarm(timer)"** est réalisée à chaque intervalle du timer défini.

Au final, on récupère dans config le dictionnaire de données à récupérer dans les messages et le programme s'exécute indéfiniment.

En voici la fonction récupérant la configuration depuis un fichier tiers :

```
def get_config():
    global mqttserver, mqttport, devices, dictAlert, timer

    fileRead = open('config.txt', 'r')

    lines = fileRead.readlines()
    dict = {}

    mqttserver = lines[0].strip().split(":")[1]
    mqttport = int(lines[1].strip().split(":")[1])

    timer = int(lines[2].strip().split(":")[1])

    keyVal = lines[3].strip().split(":")
    listDevices = lines[4].strip().split(":")

    for i in range (len(keyVal)-2):
        if ((i%3)==0):
            dict[keyVal[i]] = keyVal[i+1]
            dictAlert[keyVal[i]] = keyVal[i+2]

    for j in listDevices[1:]:
        devices.append(("application/1/device/" +j+ "/event/up", 1))

    fileRead.close()

    return dict
```

---

On crée d’abord une variable **“fileRead”**, avec laquelle on ouvre **“config.txt”**.

On crée une autre variable **“lines”**, celle-ci nous permet de lire **“config.txt”** grâce à **“fileRead”** et la méthode **“readLines()”**. Concrètement, **readLines()** s’occupe de lire les lignes du fichier une à une. Par ailleurs, on crée aussi un dictionnaire que nous allons voir plus tard, quelle en est son importance.

Les méthodes **“...strip().split(“:”)...”** dans les variables **mqttserver**, **mqttport**, **timer**, **keyval**, **listDevices** permettent grossièrement de séparer une ligne de texte afin de rendre lisible les différentes valeurs. Le **“(“:”)”** permet de définir quand la méthode **split()** doit séparer le texte, c’est-à-dire au moment où un **“:”** apparaît. **“lines[]”** permet de sélectionner la ligne du fichier que l’on veut séparer en liste.

Dans le premier for, on parcourt la longueur de **keyVal-2**, le **“-2”** sert à ne pas dépasser et ainsi pouvoir récupérer le dernier et l’avant dernier élément, sans risquer l’out of range.

Vient ensuite le if, dans lequel on réalise l’opération **(i%3)** en vérifiant qu’elle soit égale à 0. Cette opération nous permet de récupérer le nom des objets souhaités (température, humidité, CO2...). Sans cette opération, on retrouve le nom des objets dans les deux dictionnaires (chose qu’on ne veut pas). Les deux dictionnaires en question ont chacun une utilité propre. Le premier sert à récupérer la valeur des attributs tandis que le second s’occupe de prendre l’existence des attributs (0 ou 1). Dans le if, nous retrouvons l’existence du **[i+1]** et **[i+2]** et c’est pour cela qu’il est nécessaire de mettre **“-2”** dans le for, afin de, comme précisé, ne pas dépasser la lecture de la ligne.

Le deuxième for sert simplement à récupérer un device sous forme de tuple.

Enfin, on ferme le fichier grâce à la méthode **“close()”**.

En voici la fonction qui écrit dans le fichier `releve.txt` :

```
def on_write(donneesFiltrees):  
  
    #TODO  
    # ouvrir fichier 'releve'  
    # écrire 'payload' dans le fichier  
    # fermer fichier 'releve'  
  
    print("\nDonnées récupérées. \nEcriture sur le fichier releve.txt...")  
  
    fileWrite = os.open('releve.txt', os.O_WRONLY | os.O_TRUNC | os.O_CREAT, 0o666)  
  
    os.write(fileWrite, json.dumps(donneesFiltrees).encode("UTF-8"))  
  
    os.close(fileWrite)  
  
    print("\nEcriture terminée !")
```

On affiche d'abord *"Données récupérées. Écriture sur le fichier releve.txt..."*.

Grâce à **"fileWrite"** on ouvre le fichier **"releve.txt"** avec la permission d'écrire, on le crée s'il n'existe pas et on le vide de tout contenu s'il y en a.

La commande **"os.write"** permet d'écrire dans le fichier **releve.txt** via **fileWrite** les données filtrées, données en paramètres. (à ne pas confondre avec la variable `donneesFiltrees`).

On ferme ensuite le fichier grâce à la **"os.close"** et on affiche *"Écriture terminée !"*.

Enfin, voici la fonction qui affiche une alerte en fonction du maximum atteint :

```
def on_message(client, userdata, msg):

    global donnees, alertFiles, dictAlert
    config = get_config()
    dateTime = datetime.fromtimestamp(int(time.time()))

    donnees = json.loads(msg.payload)

    for key, value in config.items():
        if int(donnees["object"][key]) > int(dictAlert[key]):
            seuil = str(dictAlert[key])
            valeur = str(donnees["object"][key])
            capteur = str(donnees["rxInfo"][4]["name"])
            timeStamp = str(dateTime)
            fileAlert = os.open(alertFiles[key], os.O_WRONLY | os.O_APPEND | os.O_CREAT, 0o666)
            os.write(fileAlert, str("\n["+timeStamp+"] : Seuil d'alerte dépassé ! \n Capteur : "+capteur+"\n Seuil : "+seuil+"\n Valeur : "+valeur+"\n").encode("UTF-8"))
```

On récupère la configuration grâce à “**config = get\_config()**”.

**dateTime** permet de récupérer la date actuelle.

**json.loads(msg.payload)** sert à convertir le payload en un dictionnaire python.

Le for permet de parcourir les données et de vérifier si le maximum a été atteint via le **if**, dans lequel on attribue à **seuil**, **valeur**, **capteur** et **timeStamp** un string contenant les valeurs correspondantes aux variables au moment où le seuil a été dépassé (date pour timeStamp, blagnac\_iut pour le capteur...).

**fileAlert** permet d'ouvrir **alertFiles** avec la permission d'écrire, en préservant le contenu actuel grâce à **os.O\_APPEND** et en en créant un s'il n'existe pas.

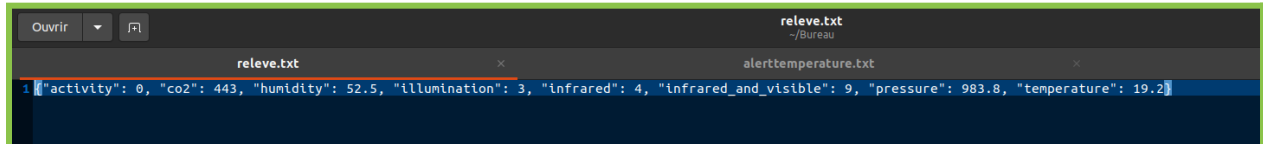
c'est grâce à **fileAlert** qu'on peut ensuite écrire dans **alertFiles** via la commande “**os.write**”, dans laquelle on écrit “*Seuil d'alerte dépassé !*” avec toutes les valeurs récupérées dans le **if**.



## Retour du code :

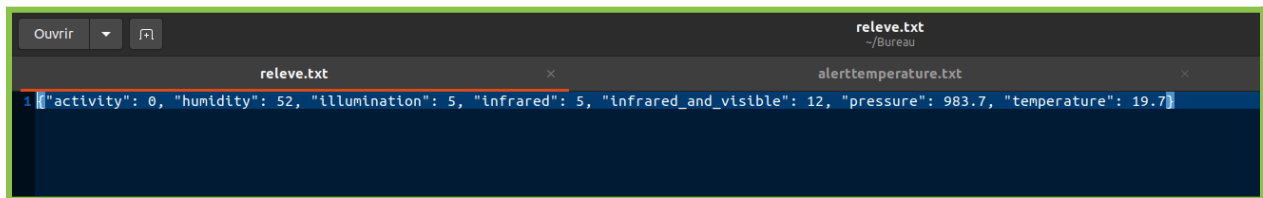
### Contenu du fichier releve.txt :

Ci-dessous, voici le contenu du fichier *releve.txt* après la première récupération des données (toutes les données sont récupérées) :



```
1 {"activity": 0, "co2": 443, "humidity": 52.5, "illumination": 3, "infrared": 4, "infrared_and_visible": 9, "pressure": 983.8, "temperature": 19.2}
```

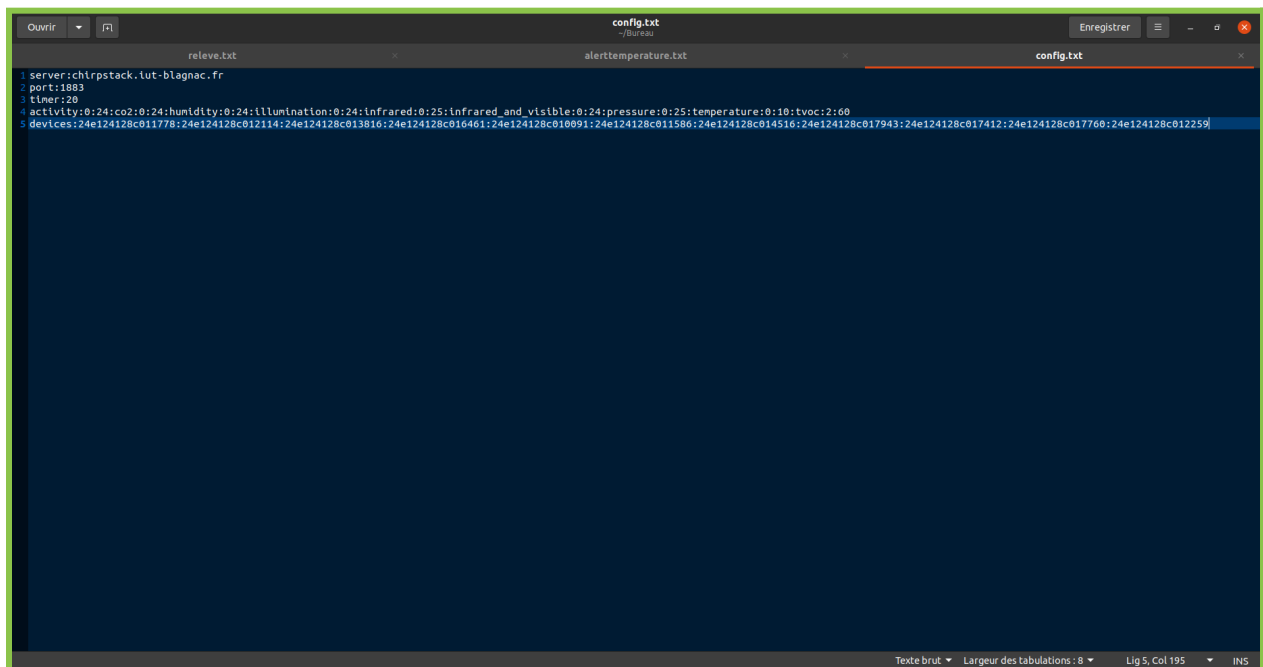
*releve.txt* après la deuxième récupération des données (toutes les données sont récupérées excepté le le CO2) :



```
1 {"activity": 0, "humidity": 52, "illumination": 5, "infrared": 5, "infrared_and_visible": 12, "pressure": 983.7, "temperature": 19.7}
```

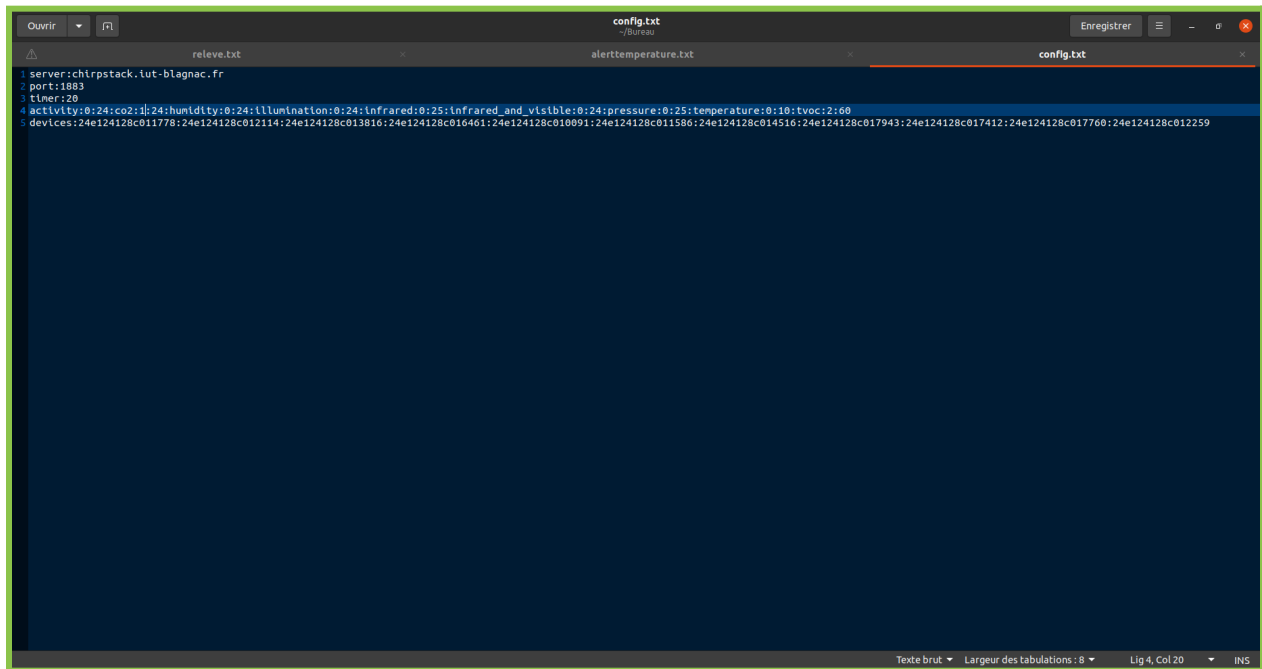
### Contenu du fichier configuration :

Fichier configuration avec la sélection de l'ensemble des données (via le 0) :



```
1 server:chirpstack.lut-blagnac.fr
2 port:1883
3 timer:20
4 activity:0:24;co2:0:24;humidity:0:24;illumination:0:24;infrared:0:25;infrared_and_visible:0:24;pressure:0:25;temperature:0:10;tvoc:2:60
5 devices:24e124128c011778:24e124128c012114:24e124128c013816:24e124128c016461:24e124128c018091:24e124128c011586:24e124128c014516:24e124128c017943:24e124128c017412:24e124128c017768:24e124128c012259
```

En voici le fichier de configuration avec la sélection de l'ensemble des données (sauf CO2 car 1) :

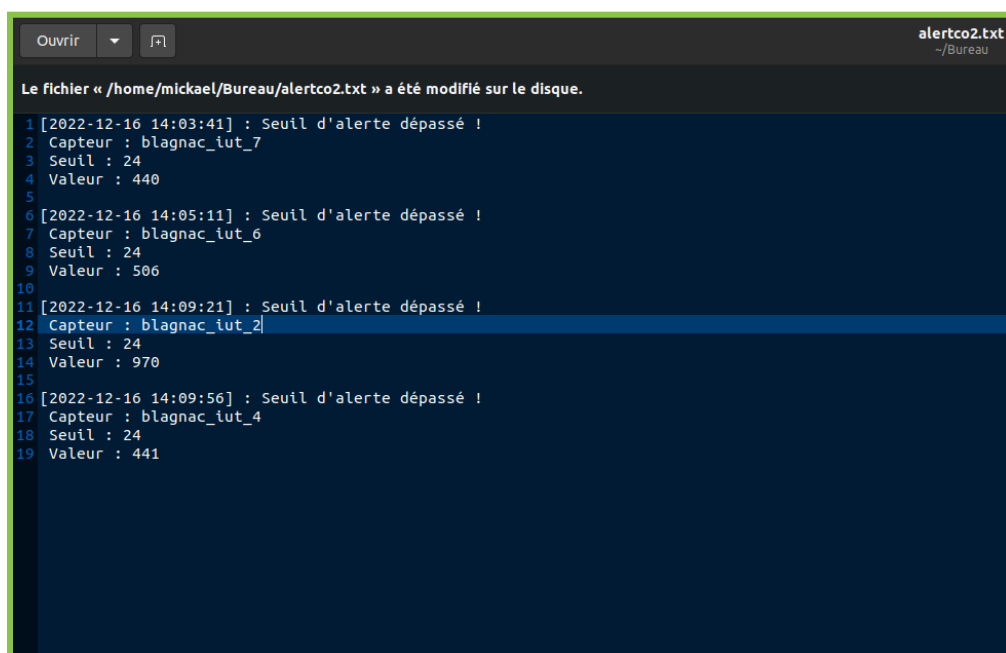


The screenshot shows a text editor window titled 'config.txt' with the following content:

```
server:chirpstack.lut-blagnac.fr
port:1883
timer:20
activity:0:24:co2:1:24:humidity:0:24:illumination:0:24:infrared:0:25:infrared_and_visible:0:24:pressure:0:25:temperature:0:10:tvoc:2:60
devices:24e124128c011778:24e124128c012114:24e124128c013816:24e124128c016461:24e124128c010091:24e124128c011586:24e124128c014516:24e124128c017943:24e124128c017412:24e124128c017768:24e124128c012259
```

### Alerte affichée :

Enfin, voici l'affichage de l'alerte avec le timestamp (correspondant à l'affichage de la date en temps et en heure) :



The screenshot shows a text editor window titled 'alertco2.txt' with the following content:

```
Le fichier « /home/mickael/Bureau/alertco2.txt » a été modifié sur le disque.
1 [2022-12-16 14:03:41] : Seuil d'alerte dépassé !
2 Capteur : blagnac_iut_7
3 Seuil : 24
4 Valeur : 440
5
6 [2022-12-16 14:05:11] : Seuil d'alerte dépassé !
7 Capteur : blagnac_iut_6
8 Seuil : 24
9 Valeur : 506
10
11 [2022-12-16 14:09:21] : Seuil d'alerte dépassé !
12 Capteur : blagnac_iut_2
13 Seuil : 24
14 Valeur : 970
15
16 [2022-12-16 14:09:56] : Seuil d'alerte dépassé !
17 Capteur : blagnac_iut_4
18 Seuil : 24
19 Valeur : 441
```

---

## Conclusion :

Pour la partie IOT, tous les objectifs ont été accomplis. Nous avons eu quelques difficultés lors du sprint 3, mais celles-ci ont été réglées pour le sprint suivant. Nous avons réalisé la partie IOT dans l'objectif de nous alléger la tâche pour la partie Java, et nous en sommes satisfait du résultat.