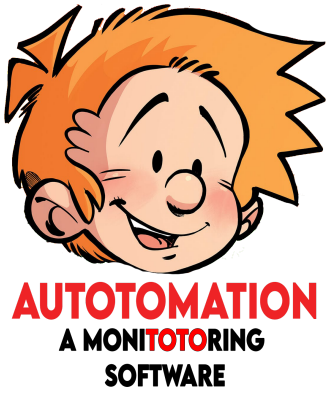
Enzo CONTINI
Hugo FERRER
Dorian DUGUE
Brian GASPARINI

IRC
4ème année

CPE Lyon



Rapport Projet

Logiciel de monito[to]ring Micro:bit
Module IOT

~

Automne 2020

Retour sur le travail effectué

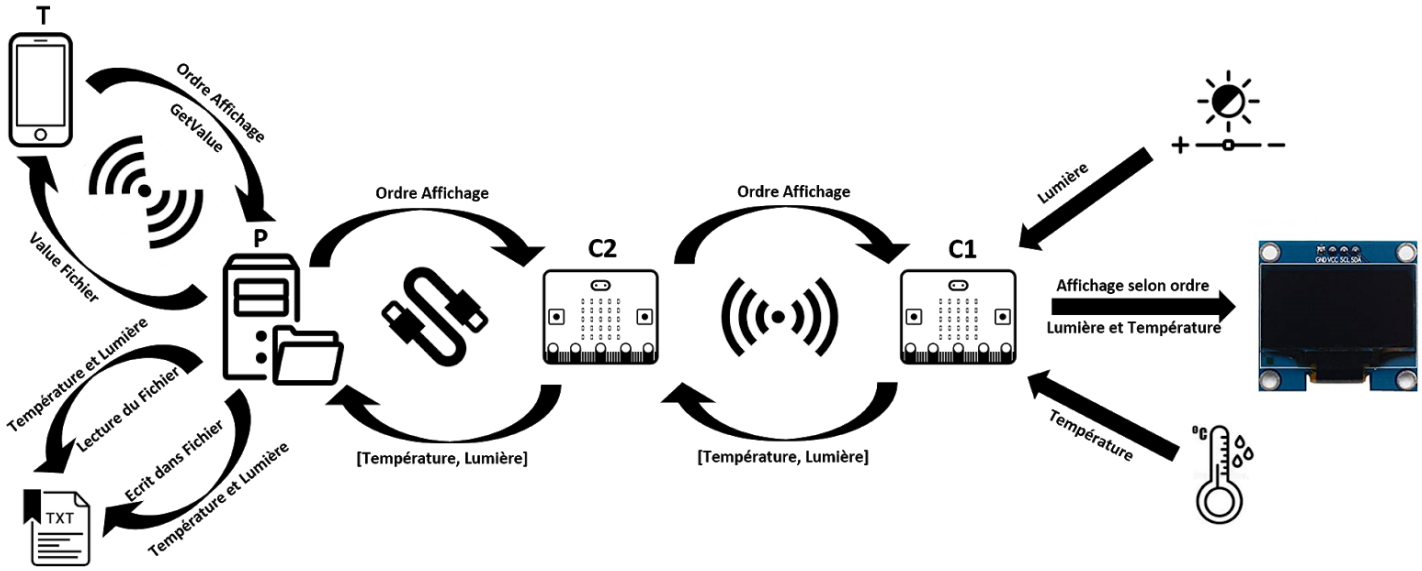
Technologies : Androïd, Python, Sockets

TABLE DES MATIÈRES

Présentation globale

Dans le cadre de notre formation d'ingénieur Informatique & Réseaux de Communication (IRC) dispensée(IRC) dispensée à CPE, nous devons réaliser un mini projet d'IOT. Le but de ce projet est de mettre en place une architecture informatique orientée IOT composée de deux cartes BBC micro:bit, l'une jouant le rôle de capteurs (lumière et température), l'autre de passerelle connectée à internet. Le réseau de capteurs peut être commandé à distance depuis un smartphone Android.

Voici un schéma de l'implémentation de notre solution :



Organisation et mise en route

Pour commencer nous nous sommes réunis pour organiser nos travaux. Nous avons établi que la communication dans le système se fera intégralement en JSON, entre tous les acteurs. Ce choix a été motivé par des raisons de facilité d'implémentation et de rapidité au sein de ce module, d'uniformité, et pour assurer la sécurité de notre système par plusieurs facteurs d'authentification :

- Un mot de passe communiqué en dur à chaque instance et stocké localement sur les différents intervenants (<PASSWORD>)
- Un chiffrement / hachage sur le JSON, processus défini en amont et nécessaire à l'interprétation des données
- La validation de la structure du JSON

Pour s'assurer du bon traitement à effectuer sur les acteurs du système un duo source/destination a été mis en place.

La définition des acteurs associés :

ID	Description
C1	Carte Micro-bit 1, capteur thermomètre et luminosité
C2	Carte Micro-bit 2, centre de contrôle du système, lien entre C1 et P
P	Passerelle, serveur manipulant la base de donnée (fichier dans le cadre de ce module)
T	Application Android mise en place sur le téléphone (ou émulateur)

La dualité mise en place *(dans un système mono-capteur C2 est simplement un intermédiaire)* :

Source	Destination	Description
C1	P	Envoi des données de Température + Luminosité
P	C1	Transmission de l'ordre d'affichage de Température + Luminosité
T	P	Envoi de l'ordre d'affichage de Température + Luminosité OU demande de valeurs
P	T	Transmission des données de Température + Luminosité

Pour définir les échanges dans le système un champ < data > est mis en place dans le JSON. Il s'agit d'un tableau de 2 entiers dont les détails d'implémentation sont précisés ci-dessous :

S	D	Indice 0	Indice 1
C1	P	int : Température	int : Luminosité
P	C1	string : "TL" ou string : "LT"	"Null"
T	P	string : "TL" ou string : "LT" (pour inverser l'ordre) OU string : "getValues()" (pour récupérer les valeurs)	"Null"
P	T	int : Température	int : Luminosité

La structure du JSON choisie est la suivante :

```
{
        source : < ID >,
        destination : < ID >,
        password : < PASSWORD >,
        data : [< DATA >]
}
```

Exemples de requêtes :

Communication des valeurs des capteurs (capteur vers passerelle) :


```
{
        source : "C1",
        destination : "P",
        password : "MDPtrèsSécurisé9998",
        data : [22,53]
}
```


Communication des valeurs des capteurs (passerelle vers téléphone) :


```
{
        source : "P",
        destination : "T",
        password : "MDPtrèsSécurisé9998",
        data : [22,53]
}
```


Demande du changement de l'ordre des capteurs :


```
{
        source : "T",
        destination : "P",
        password : "MDPtrèsSécurisé9998",
        data : ["TL","Null"]
}
```


Communication du changement de l'ordre des capteurs :


```
{
        source : "P",
        destination : "C1",
        password : "MDPtrèsSécurisé9998",
        data : ["TL",Null]
}
```


Demande des valeurs des capteurs à la passerelle :


```
{
        source : "T",
        destination : "P",
        password : "MDPtrèsSécurisé9998",
        data : ["getValues()",Null]
}
```

Distribution des rôles

Dorian Dugue	Code C2
Enzo Contini	Android
Hugo Ferrer	Code C1
Brian Gasparini	Serveur

Documentation technique

Code contrôleur C1

Le contrôleur C1 est celui connecté à l'écran OLED. C'est ce contrôleur qui envoie les données perçues par les capteurs de température et de luminosité. Il se charge également d'afficher ces informations à l'écran.

Les imports

microbit	*
radio	*
ssd1306	initialize, clear_oled()
ssd1306_text	add_text()

Les fonctions

displayToScreen(order: string)	return void

Affiche sur l'écran OLED la température ainsi que la luminosité.

getJSONToSend(void)	return string

Retourne une chaîne de caractères qui contient les données au format JSON.

parseData(json: string)	return Array

Retourne le champ “data” des données JSON qui est un tableau d’une taille de deux chaînes de caractères.

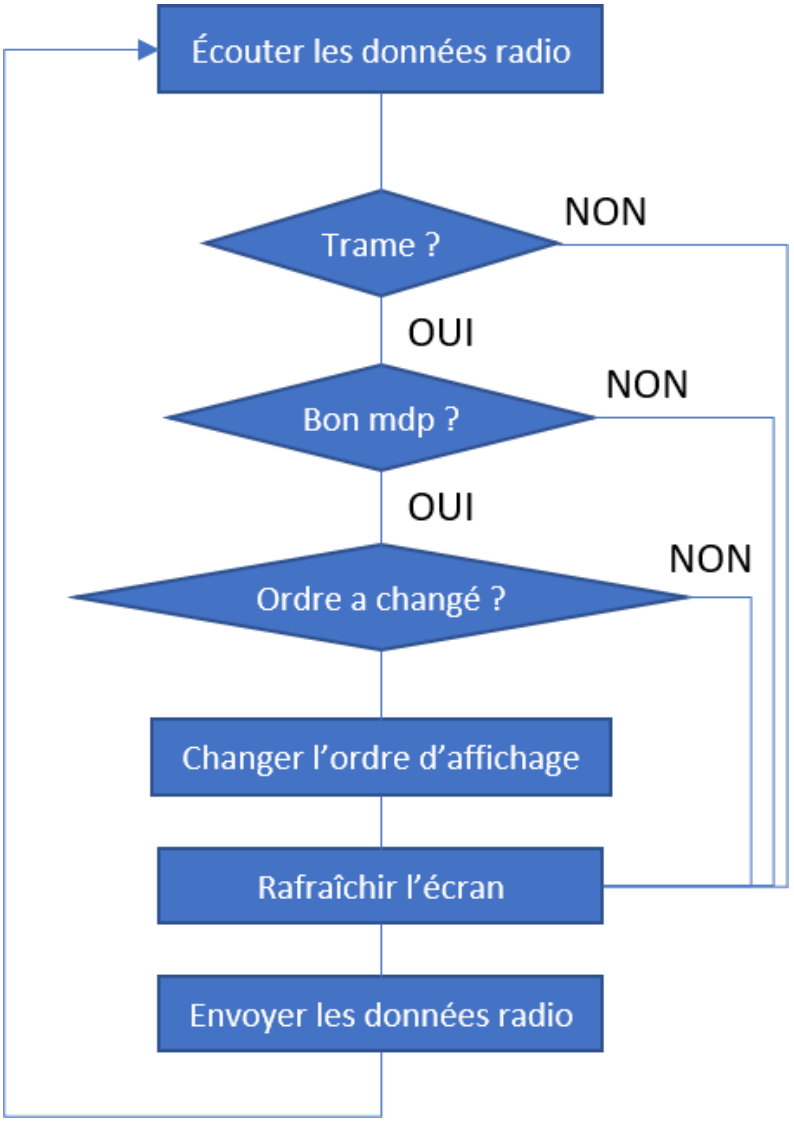
parsePassword(json: string)	return string

Retourne le champ “password” des données JSON.

checkPassword(passwd: string)	return boolean

Évalue si le mot de passe est correct.

Le déroulé du programme



Code contrôleur C2

Rôle :

Le rôle de ce contrôleur est de faire la liaison entre le Serveur et le second microcontrôleur. Il est donc connecté en série au serveur (câble) et communique en radio avec le microbit distant. Il va donc écouter son UART, va modifier son instruction qu'il va envoyer continuellement en radio sur le groupe défini. Tout en écoutant ce même channel radio afin d'envoyer en UART les valeurs de température et de lumière reçues. Bien évidemment il y a un contrôle de structure, de mot de passe et de chiffrement dans chaque transmission de données.

Les imports

microbit	*
radio	*

Les fonctions

parsepasword(json: string)	return string

Il va retourner le mot de passe récupéré dans le json passé en argument selon la structure mise en place.

checkPassword(passwd: string)	return boolean

Évalue si le mot de passe est correct.

crypte(msg: string)	return string

Crypte le string passé en argument selon la méthode de Vigenère par une clé défini.

decrypte(msg: string)	return string

Décrypte le string passé en argument selon la méthode de Vigenère par une clé défini.

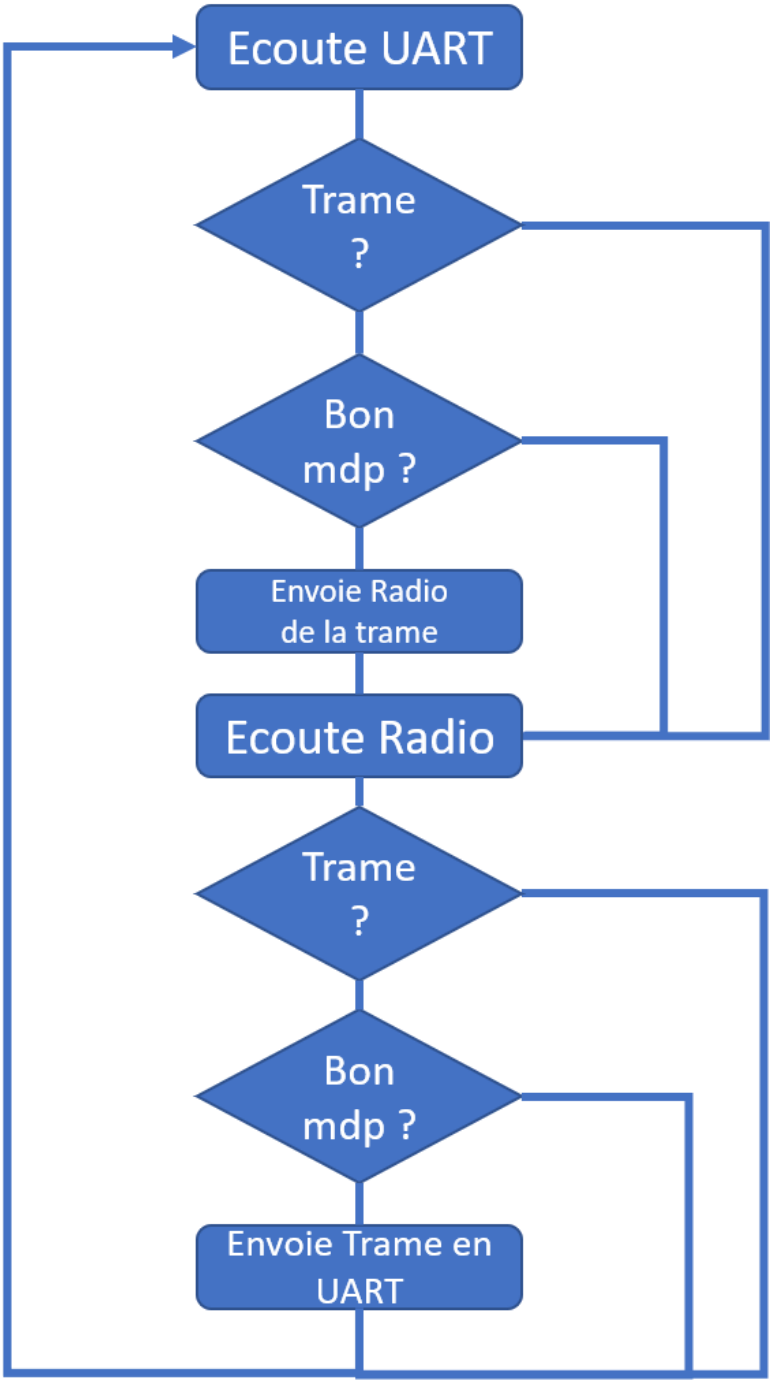
send(m: string)	return void

Envoie le message en argument par UART définie.

receive(void)	return void

Lit le message passé par UART.

Déroulé du programme



Code serveur python

Code application MainActivity

Le détail de la documentation concernant toutes les classes de l'application est disponible dans la JavaDoc fournie.

Les imports

android.os.Bundle	*
android.util.Log	*
android.view	OnClickListener()
android.widget	*
android.net	*

Les fonctions

onCreate(Bundle savedInstanceState)	return void

Initialisation de l'interface IHM, des listeners sur les boutons, de la socket UDP

sendData(String data)	return void

Méthode appelée à l'envoi de données vers la passerelle

initSocket(String data)	return boolean

Méthode de création d'une socket

initConnexionIHM()	return void

Initialisation du contenu des champs

initIHM()	return void

Initialisation des éléments IHM

formattingDataSent(String data)	return String

Formatage en JSON des données envoyées à la passerelle

encodeCesar(String data)	return String

Méthode de chiffrement de César

decodeCesar(String data)	return String

Méthode de déchiffrement de César

Problèmes rencontrés

- Microbit ne proposant pas de multi threads, mais fonctionnant uniquement de façon séquentielle, le raisonnement est différent. On ne pouvait pas réaliser un fonctionnement optimal en disant : 1 thread d'écoute, 1 thread d'envoie. Ce qui provoque un alourdissement du fonctionnement. Le microbit n'est pas forcément en écoute lorsque le deuxième est en envoie (et inversement), donc il est nécessaire d'envoyer continuellement les informations pour être sûr de leur réception à un moment, accentuant la latence dans le système.

- Le microbit C1 (avec écran) ne possédant que très peu de mémoire disponible, il est impossible de réaliser un protocole d'échange plus complexe. Nous avions l'idée de réaliser un protocole d'envoie avec accusé de réception afin de n'envoyer les informations seulement pendant un laps de temps suffisant afin d'être sûr de la bonne réception des informations, ce problème produit une latence certaine, voire de mauvais échange d'informations dans la boucle.

- Lors de la réception de trame json en UART par C2, le readline ne prenait pas en compte la ligne entière. Nous avions remplacé par read(), mais le problème était qu'ensuite le microbit lisait sur le port série des strings incorrects. Nous nous sommes rendu compte qu'ultérieurement il était nécessaire de mettre un "\n" dans la trame envoyée par le serveur sur le port afin que C2 réalise une bonne lecture du Json envoyé.

- Le microbit C1 ne "flashe" pas quand C2 et serveur sont en activation, il faut lancer C1 en premier dans la chaîne d'exécution. Le problème est inconnu à l'heure actuelle, on suppute que c'est à cause de la transmission radio activé sur C2, mais lorsque l'on flash C1 devrait réussir à réceptionner les trames lorsqu'il est prêt.

Évolutions Futures Possibles

- Mettre en place des cartes plus puissantes, prenant en compte le fonctionnement en multi threads, des sémaphores et des MQTT.
  Cause : Matériel
  Changement : RaspBerry par exemple.

- Chiffrement des échanges non opérationnels
  Cause : La mémoire n'est pas suffisante sur le microbit C1 à cause des dépendances de l'écran OLED.
  Changement : Mettre en place les fonctions "crypte" et "decrypte" qui se trouvent sur C2 sur C1, respectivement à l'envoi et à la réception du JSON.

- Ne pas envoyer les données continuellement
  Pourquoi : une question de sécurité, d'économie d'énergie
  Cause : Mémoire insuffisante de C1 (dépendances écran).
  Changement : Mettre en place un protocole d'échange avec accusé de réception afin d'envoyer seulement jusqu'à réception.

- Ne pas définir en dur la clé de chiffrement / Nouvelle carte dans le réseau
  Pourquoi : Pour augmenter la sécurité, la clé peut être générée dynamiquement par le serveur et envoyée à tous les périphériques reconnus dans le réseau.
  Changement : Mettre en place un protocole qui lorsque l'on rajoute une carte le microprocesseur demande sur le réseau la clé, qui remonte jusqu'au serveur avant de l'enregistrer pour le chiffrement

- Application web, avec l'ajout d'une base de données :
  Pourquoi : Ne pas enregistrer dans un fichier texte mais dans une base de données afin d'augmenter les possibilités d'évolutions (statistiques sur une durée, …).

Changement : Mettre en place un SGBD, modifier la façon d'enregistrer, lancer une requête au lieu d'écrire dans le fichier.