



STAGE : SERVICE AMÉNAGEMENT NUMÉRIQUE

Dossier Bilan

Auteur :
Florian GRANTE

Superviseur :
David LAROSE

Table des matières

Introduction	1
1 Monitoring : Mise en place d'Please insert intro preambleune solution IoT avec Azure	2
1.1 Le cahier des charges	2
1.2 La base de la station	3
1.2.1 La <i>Raspberry Pi</i>	3
1.2.2 L' <i>Intel Edison</i>	4
1.2.3 Le choix de la base de la station	5
1.3 Les composants	6
1.3.1 Le Shield	6
1.3.2 Capteur de température et d'humidité	6
1.3.3 Capteur de luminosité	7
1.3.4 L'écran d'affichage	7
1.3.5 L'encodeur rotatif	7
1.4 Le langage de programmation	7
1.5 Le coût d'une station	9

Introduction

Au cours de mon mois de stage au service aménagement numérique de la ville de Drancy, j'avais pour objectif principal de réaliser une démonstration de faisabilité d'une station de monitoring pour bâtiments publics. Tout était à faire, tant par le choix matériel que par la construction logiciel qui en découlait. J'ai également eu l'occasion de participer à un second projet dont je parlerais dans un second temps. Avant de délivrer mes conclusions sur ces projets, je tiens à remercier mon superviseur, M. LAROSE David pour sa confiance totale sur ce projet.

Chapitre 1

Monitoring : Mise en place d'une solution IoT avec Azure

Le but de ma mission était une la réalisation d'une démonstration de faisabilité d'une station permettant de relevé la température, l'humidité ou encore savoir si la lumière d'une pièce est allumée. Cette station a pour but d'être installée dans des bâtiments public comme des écoles par exemple. Le type de bâtiment n'est pas anodin puisqu'il ajoute des contraintes non négligeable pour l'installation des stations.

1.1 Le cahier des charges

Même si le nombre de contrainte est limité, elles conditionnent cependant le système à tel point qu'elle limite énormément les choix possibles :

1. Le système ne doit nécessiter aucun pré-requis technique. J'ai réalisé un dossier d'installation du système que j'ai retenue pour cette mission. Ce dossier, devrait normalement remplir cette condition.
2. Le système ne doit communiquer uniquement par l'intermédiaire d'un câble Ethernet. Il n'est pas envisageable pour une mairie d'installer des systèmes sans fils dans une école primaire par exemple.
3. Pour offrir une liberté d'installation, la station doit s'affranchir d'une alimentation sur prise secteur.
4. les données doivent être consultable sur le *Cloud Microsoft Azure*.
5. les données doivent en plus, être visible localement.

Même si cela est succinct, je vais montrer au fur à mesure des choix que j'ai été amené à faire que ces conditions ont été déterminantes.

1.2 La base de la station

1.2.1 La *Raspberry Pi*



FIGURE 1.1 – *Une Raspberry Pi*

La *Raspberry Pi* est sans équivoque l'ordinateur du *Maker* par excellence. Open source, elle permet de réaliser des systèmes embarqués performant pour un coût relativement faible et simple d'utilisation. Regardons ensemble les points du cahier des charges :

1. Aucun pré-requis : Il s'agit ni plus ni moins que d'un ordinateur. J'ai détaillé cependant la façon dont on installe et configure l'OS pour la *RaspberryPi*
2. Communication Ethernet : La *Raspberry Pi*, comme tout bon ordinateur, possède une carte réseau et donc un port Ethernet. Noté toutefois, qu'à partir de la *Raspberry Pi 3*, elle embarque également le *Wi-Fi* et le *Bluetooth*.
3. L'alimentation de la *Raspberry Pi* se fait par un port micro-usb en 5V. La *Raspberry Pi* ne gère pas l'alimentation *PoE*. On peut alors recourir à une batterie ou bien à un *PoE Splitter* qui récupère l'alimentation par le câble Ethernet et le transforme dans un port micro-usb.
4. Disposant d'une carte réseau, il n'y a aucun problème pour envoyer les données dans le cloud.
5. La *Raspberry Pi* possède une sortie HDMI pour éventuellement brancher un écran. Il est également possible de mettre de petit écran LCD comme afficheur mais nous verrons ça plus tard.

La *Raspberry Pi* est alors un choix possible pour notre base de la station. Elle remplit effectivement toutes les conditions du cahier des charges.

1.2.2 L'Intel Edison

Intel a décidé de se lancer dans l'IoT (il semblerait que le projet soit abandonnée) en sortant une puce : *Edison*.



FIGURE 1.2 – Une puce *Edison*

Cette puce doit alors être placé sur une carte que fabrique également *Intel* afin de pouvoir y brancher toute sorte de composants :



FIGURE 1.3 – *Un intel Edison sur son board arduino*

Concrètement, il s'agit ni plus ni moins que d'un board Arduino amélioré. Amélioré par la performance de calcul de la puce *Intel Edison* et amélioré car elle embarque un *firmware* qui permet de compiler toute sorte de langage, dont le langage Arduino.

Le langage Arduino est parfait pour une mission comme la station car parfaitement adapté pour de l'électronique embarqué. Regardons les conditions du cahier des charges :

1. Aucun pré-requis technique : De la même façon que pour la *Raspberry Pi*, il suffit d'installer le *firmware* dans une micro-SD par le biais d'un utilitaire très simple fournis par *Intel*
2. L'*intel* ne dispose malheureusement pas ce carte réseau avec un port *RJ45*. Il embarque cependant le *Wi-Fi*. Comme ce dernier est embarqué sur un shield Arduino, on peut supposer qu'il est possible d'y installer un shield qui apporterait la fonctionnalité à notre système. Cela rendrait le système bien plus volumineux et complexe.
3. Ne possédant pas de port *RJ45*, nous pouvons oublier la technologie *PoE* sans l'utilisation d'un *PoE Splitter* comme pour la *Raspberry Pi*. L'alimentation pouvant se faire sur un micro-usb, il faudra donc opter pour la même solution.
4. Ayant la possibilité d'avoir un accès internet, il n'y aura donc aucun problème pour envoyer les données sur le *Cloud Azure*. Il existe d'ailleurs de la documentation sur le site de *Azure* concernant l'usage de *L'intel Edison*.

L'intel Edison est donc un second choix possible même s'il est plus compliqué à mettre en place car nécessite plus de pièces.

1.2.3 Le choix de la base de la station

Comme je l'ai dit plus tôt, les deux choix sont possible. Il va alors falloir trancher au niveau du coût d'une part, puis de la simplicité de mise en place. Comme *L'intel Edison* nécessite l'ajout d'un Shield pour pouvoir espérer un accès internet par câble, mon choix s'est porté sur la *Raspberry Pi*.

L'intel Edison se montre très intéressant de par sa puissance de calcul et ses possibilités, cependant, son coût est élevée et sa puissance se montre inutile pour l'usage que nous allons en avoir pour la station. Comme l'utilisation aurait été le même qu'un simple arduino uno, nous aurions pu envisager l'utilisation de ce dernier car il est un très bon choix en rapport qualité/prix. Il ne rivalise cependant pas en coût avec la *Raspberry Pi* dès lors que nous ajoutons le shield Ethernet.

Ainsi, j'ai tout naturellement choisis la *Raspberry Pi* pour réaliser la station. Il faut ensuite choisir les capteurs et autres composants que nous allons brancher à la *Raspberry Pi*.

1.3 Les composants

Il faut maintenant faire une sélection de composants qui vont nous permettre de récolter les données, les afficher puis les envoyer dans le Cloud. Même si mon âme de roboticien voudrait créer une carte électronique où il suffirait de souder quelques composants pour avoir une plate forme faite sur mesure, il ne faut pas perdre de vu notre conditions "Aucun pré-requis". C'est pourquoi j'ai choisis d'utiliser les composants suivants :

1.3.1 Le Shield

Pour réaliser une interface simpliste avec la *Raspberry Pi* et les capteurs, nous allons utiliser un shield. Ce dernier porte le nom de *GrovePi+*.

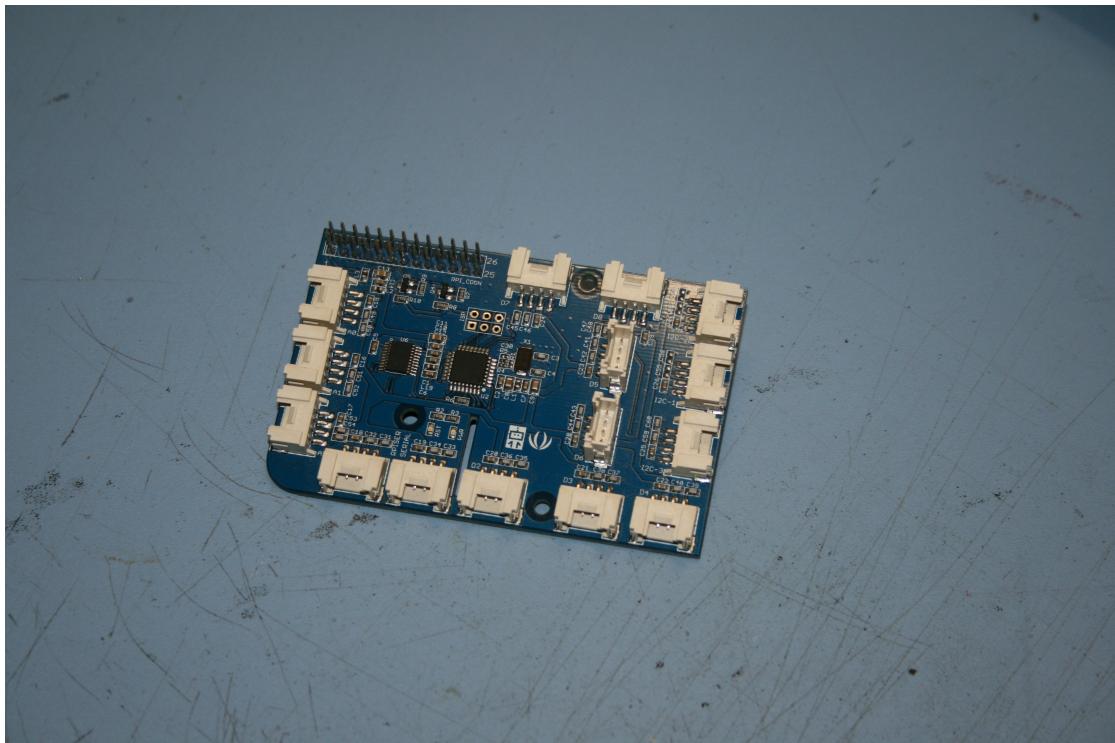


FIGURE 1.4 – le Shield *GrovePi+*

Pour l'*Intel Edison* je parlais qu'il embarquait un board Arduino et compilais le langage Arduino (qui est ni plus ni moins qu'une version adapté de C++). Et bien ce shield est en réalité un Arduino qui communique avec la *Raspberry Pi*. On notera le gros travail des ingénieurs qui ont réalisé les bibliothèques de tel sorte qu'il soit très facile de programmer pour l'utiliser mais nous détaillerons le langage de programmation dans une prochaine partie. Le seul bémol de ce Shield, c'est ses connecteurs 4-pin par lequel on branche les composants.

En effet, même s'il est simple et ludique de faire les branchements, cela oblige à acheter les composants qui sont déjà fabriqué pour être branché dessus. Du moins si l'on veut pas faire de soudure. Il serait possible de commander uniquement les connecteurs pour utiliser les composants que l'on souhaite mais ce n'est pas l'objectif ici.

1.3.2 Capteur de température et d'humidité

Le capteur que j'ai choisi ici est le DHT22.

Après plusieurs test avec son petit frère, le DHT11, j'ai décidé qu'il serait plus judicieux de monter en gamme car ce dernier manquait cruellement de précision. Ce genre de capteur sont cependant pas fait pour avoir une température instantanée. En effet, leur conception font qu'ils sont relativement lent pour réagir aux choc de température. Ces capteurs sont en effet là pour avoir une courbe de tendance de la température. Nous allons avoir l'allure de la variation de la température sur de longue durée de l'ordre de plusieurs heure. C'est pour ça qu'il n'est pas impossible de voir quelques valeurs faussée à quelques instants.

1.3.3 Capteur de luminosité

Les capteurs de luminosité sont très imprécis car la mesure de la luminosité avec précision coûte très chère. Cependant, nous n'avons pas besoin de mesurer de façon exacte la luminosité. D'une part car son unité, le luxmètre n'est pas très parlant pour le grand public et d'autre part, nous voulons juste avoir une information nous permettant de définir si la lumière d'une pièce serait allumée ou éteinte.

J'ai donc testé le capteur en le laissant tourner durant 2 jours complets pour voir un seuil à partir duquel on pouvait définir que la lumière était effectivement éteinte.

A noter cependant que ce seuil a été choisi arbitrairement et qu'il pourrait ne pas fonctionner selon les endroit où est placé la station (au bord d'une fenêtre typiquement).

1.3.4 L'écran d'affichage

Comme je l'ai spécifié dans le cahier des charges, il fallait pouvoir récupérer les valeurs localement. J'ai donc installé un écran afficheur 2x16 caractères. De cette façon, la station pourra afficher les valeurs.

1.3.5 L'encodeur rotatif

L'écran ayant une capacité d'affichage limitée, j'ai ajouté un encodeur rotatif pour naviguer entre différents menu sur l'écran pour afficher différentes valeurs utile.

1.3.6 PoE Splitter

Le PoE Splitter va permettre d'alimenter la *RaspberryPi* en PoE en découpant le câble RJ45 en 2 canaux : Un sur un câble micro - USB pour l'alimentation et un câble Ethernet RJ45 pour relier la station au réseau.

1.4 Le langage de programmation

Avec le *Shield*, il était possible d'utiliser une multitude de langage de programmation. Cependant il fallait un langage intuitif et simple car il y a des lignes qui devront être changé lors d'un déploiement comme le nom des ressources sur Microsoft Azure où envoyer les données. J'ai automatiquement pensé au langage Python. Il est le langage de haut niveau par excellence. De plus, il est relativement adapté pour des script embarqué. Il n'est pas rare de voir des code de pilotage de robot fait en Python.

Voici un petit aperçu du code :

```

1 ##### IMPORTATION DES BIBLIOTHEQUES#####
2 from grovepi import *
3 from grove_rgb_lcd import *
4 import time,datetime,json
5 from math import *
6 #from azure.servicebus import ServiceBusService
7 #from azure.servicebus import Message
8 ##### DECLARATION DES CONSTANTES GLOBALES#####
9 dht_sensor_port = 7 #capteur humidite et temperature (DHT11 ou DHT22) sur D7
10 dht_sensor_type = 0 #mettre 0 si bleu (DHT11) ou 1 si blanc (DHT22)
11 t_actuator = 799 #nombre de loop entre deux envoie de donnees
12 t_wait = 30
13 lum_seuil = 10 #seuil au dessus du quel on determine la lumiere comme allume
14 lum_sensor = 1 #capteur de lumiere sur A1
15 potentiometer = 2 #bouton de menu sur A2
16 ##### ACTIVATION DES PORTS DU SHIELD GROVEPI#####
17 ##### VARIABLES INTERNES#####
18 t_refresh = 800 # nombre de loop entre deux envoie a l instant t
19 compteur_echec_envoie = 0 #compteur d echec d envoie de donnee sur azure (se remet a 0 si reussite)
20 mode_value = 0 # les deux valeurs permettent d eviter une actualisation inutile de l ecran qui le faisait clignoter
21 mode_value_old = 0
22 ##### VARIABLES DE DONNEES ENVOYEEES#####
23 temp = 0
24 hum = 0
25 lum = 0
26 lum_statut = False
27 identifiant = "Identifiant" # a modifier
28 lum_envoie = 0
29 ##### FONCTIONS#####
30 def DHT() : #temperature et humidite numerique
31     global temp,hum
32     [temp,hum] = dht(dht_sensor_port,dht_sensor_type)
33     temp = round(temp,1)
34
35 def Luminosite() : #luminosite qui envoie True ou False avec le seuil
36     global lum_statut,lum_envoie
37     lum_value = analogRead(lum_sensor)
38     try :
39         resistance = (float)(1023 - lum_value)*10/lum_value
40         if resistance > lum_seuil :
41             lum_statut = False
42             lum_envoie = 0
43         else :
44             lum_statut = True
45             lum_envoie = 1
46     except : #si erreur au capteur, il va alterner true et false pour que cela soit visible
47         if lum_statut :
48             lum_statut = False
49             lum_envoie = 0
50         else :
51             lum_statut = True
52             lum_envoie = 1
53
54 def screen_administrator() : # permet de gerer lecran sans qu'il refresh a chaque iteration
55     global mode_value
56     mode_value_old = mode_value
57     if (encoder_value <=341 and encoder_value >= 0) and mode_value != 1 : #MODE 1
58         setText("Temperature : \n"+str(temp))
59         setRGB(0,128,255)
60         mode_value = 1
61     elif (encoder_value <= 682 and encoder_value > 341) and mode_value != 2 : #MODE 2
62         setText("Humidite : \n"+str(hum))
63         setRGB(255,0,128)
64         mode_value = 2
65     elif (encoder_value > 682 and encoder_value <=1023) and mode_value != 3 : # MODE 3
66         if lum_statut :
67             text = 'Allumee'
68         else :
69             text = 'Eteinte'
70         setText("Lumiere \n"+text)
71         setRGB(255,128,0)
72         mode_value = 3
73     if not ((mode_value - mode_value_old) != 0) : #si il y a pas eu un changement de mode sur l ecran
74         time.sleep(140.0/1000.0) #on attend 140 ms pour etre sur du temps a chaque loop
75
76
77 #def createSBS() : #permet de creer le canal de communication avec Azure
78
79 #     service_namespace = 'namespaceName' # a modifier
80 #     key_name = 'SharedAccessKeyName' # a modifier
81 #     key_value = 'SharedAccessKey' # a modifier
82
83 #     sbs = ServiceBusService(service_namespace, shared_access_key_name=key_name, shared_access_key_value=key_value)
84
85 #     return sbs
86 ##### SETUP #####
87 #sbs = createSBS()
88 setText("Bienvenue\ndans l'IoT Hub")
89 setRGB(128,255,0)
90 time.sleep(2)

```

```

91 ##### BOUCLE INFINI #####
92
93
94 while True :
95     if ( t_refresh >= t_actuator ) :
96         Luminosite()
97         DHT()
98         while (isnan(temp) or temp == 0) : # on essaie tant que le capteur n a pas de valeur valide
99             DHT()
100            dt = str(datetime.datetime.now())
101            d = { 'DeviceID' : identifiant, 'Time' : "France/Local Time here", 'Temperature' : temp, 'Humidity' : hum, 'Light' :
102                msg = json.dumps(d) #cree le message a envoyer
103            try :
104                #sbs.send_event('eventHubName',msg) #a modifier
105                compteur_echec_envoie = 0
106            except :
107                compteur_echec_envoie += 1
108                if compteur_echec_envoie == 720 : #echec d envoie de message depuis 24h
109                    setRGB(255,0,0)
110                    setText("Probleme envoie\nmessage azure")
111                    break # sort de la boucle, reboot necessaire du programme ou de la RPI
112            t_refresh = 0
113
114        if (t_refresh >= t_wait) : # on attend un peu avant de refresh l ecran
115            encoder_value = analogRead(potentiometer)
116            screen_administrator()
117            t_refresh += 1

```

1.5 Le coût d'une station

Regroupons dans un premier temps le coût matériel, puis nous discuterons ensuite du coût logiciel, notamment sur *Azure*.

Je vous propose donc le tableau ci-dessous réalisant l'inventaire des pièces nécessaire ainsi que de leur coût moyen en me basant sur deux sites : *Amazon* et *Seeed Studio*, qui est le distributeur de GrovePi+. Attention par contre à vérifier les provenance de Seeed car je me demande s'il n'est pas possible que des frais de douanes s'appliquent sur leur produits.

Composant	Prix	Amazon	Seeed
RaspberryPi 3	35 euros	Amazon	Seeed
Shield GrovePi+	33 euros	Amazon	Seeed
DHT22	17 euros	Amazon	Seeed
Luminosite	3 euros		Seeed
Ecran	16 euros	Amazon	Seeed
Bouton rotatif	5 euros	Amazon	Seeed
PoE Spliter	10 euros	Amazon	
Total	119 euros		