PLOC

Documentation Technique V.1

Description du document

Titre	Documentation Utilisateur
Date	25/08/2014
Auteur	Caroline Bagnost
Email	bagnos_c@epitech.eu
Sujet	Documentation Technique du projet PLOC
Mots clés	DT– PLOC – Embedded – Arduino
Version du modèle	1.0

Table des matières

Introduction 3
Utilisation 4
Présentation du matériel 4
Le corps 4
Le cerveau 5
Interface 6
Fonctionnement 7
Hardware 7
Description du matériel 7
Schéma de câblage 7
Software 9
GPIO Manager 9
API Rest 13
Informations complémentaires 16
Contact 16
Documentation / sources complémentaires 16

Introduction

PLOC (PLant Online Connection) est un projet de monitoring de plante d'intérieur par le biais d'une carte Intel Galileo. Ce système récupère différents types d'informations afin de déterminer l'état de la plante surveillée et agir en conséquence, le tout en donnant accès à ces informations par le biais d'une interface web – accessible par wifi depuis un ordinateur, un smartphone ou encore une tablette. La plante s'abreuve toute seule, il suffit juste de remplir le réservoir d'eau quand l'interface le signale.

Il permet actuellement de vérifier et actionner l'alimentation en eau et de contrôler la luminosité. Ceci est possible grâce à de nombreux capteurs qui seront décrit rapidement dans la partie « Utilisation » de cette documentation. Leurs branchements et leur exploitation seront abordés dans la partie « Fonctionnement ».

Le but est de ne plus risquer de faire dépérir ses plantes vertes lors de vacances ou tout simplement par suites d'oublis et/ou de compensation (plante desséchée ou noyée).

Fig.1 : Présentation du système PLOC

Ce projet n'est bien sûr pas une innovation en soi, de nombreux projets semblables existent déjà :

- Garduino,
- Properduino,
- OpenSprinkler,

Mais la différence de PLOC réside dans l'utilisation de la Galileo. Celle-ci nous offre un grand nombre d'I/O comme l'Arduino avec, en plus, la possibilité d'avoir un serveur embarqué, dans le même principe qu'une Raspberry Pi.

Utilisation

Présentation du matériel

Ploc est composé de différents éléments :

Le « cerveau » :

1. Une carte Intel Galileo
2. Un module Wi-Fi,

Le « corps » :

3. Une carte de connexion, nous permettant de brancher tous les capteurs
    a. LDR ou photorésistance – capteur de luminosité
    b. Deux clous - capteur d'humidité
4. Un réservoir d'eau,
5. Des tuyaux d'alimentation
6. Un relais,
7. Une pompe d'aquarium

} *Peuvent être remplacés par une électrovanne*

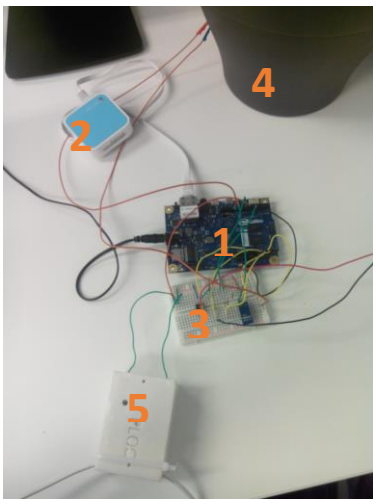

Fig.2 : Matériel

Le corps

Les capteurs

Le principe est donc en premier lieu de surveiller la « soif » d'une plante. Pour cela il suffit de connaître le taux d'humidité de la terre dans laquelle se trouve la plante. Dans le cas de PLOC, il s'agit de deux clous plantés dans la terre à environ 1 cm l'un de l'autre. On envoie un courant sur l'un des clous et on mesure en sortie sur l'autre clou : l'eau étant conductrice, plus le courant sera grand en sortie, plus le pot est humide.

Fig. 3 et 4 : Capteur d'humidité - Fig.5 : Capteurs de luminosité (LDR)

Un support en plastique a été imprimé pour isoler la partie haute des clous. Ceci afin de ne pas prendre en compte la surface du terreau. Notre programme va donc ponctuellement envoyer du courant et mesurer le retour de ces clous. Il en sera de même pour la mesure de luminosité (LDR : photorésitance).

Le réservoir d'eau

La solution mise en place est un peu rustique mais très efficace. Le réservoir a été percé à une hauteur correspondant au niveau minimal de fonctionnement de la pompe. On a ensuite dénudé partiellement deux fils électrique pour les glisser dans le trou en veillant à ce qu'ils soient proches sans pour autant se toucher. Le tout a été fixé hermétiquement au pistolet à colle. Un des fils servira de point d'entrée du courant envoyé par la Galileo, le second reverra le courant qui sera mesuré et interprété.

Le système d'arrosage

L'alimentation en eau se fait par le biais d'une simple pompe d'aquarium enclenchée par un relais, lui-même piloté depuis le programme hébergé sur la carte Galileo. Nous avons choisi un arrosage léger : un tuyau est disposé tout autour de la plante et percé, l'eau est alors distribuée par un goutte-à-goutte léger.

Dans le cas de l'électrovanne le principe est plus simple : plus besoin de relais. Le courant est directement envoyé par la Galileo pour enclencher la vanne. L'eau arrive par contre en bas du pot, par l'assiette – la plante est alimentée par capillarité.

Le cerveau

Les éléments précédemment présentés sont donc pilotés par un programme hébergé sur notre microcontrôleur Galileo. Pour faire simple, un microcontrôleur est une sorte d'ordinateur miniature. Il possède les mêmes types de composants (microprocesseur, RAM, ROM), mais limités au strict nécessaire, et moins gourmands en ressources. Galileo allie les possibilités de Linux avec celles d'Arduino dans une seule carte. Nous l'utilisons ici pour piloter nos différents capteurs mais également pour héberger le site qui informant de l'état de la plante.

Le site quant à lui est disponible par Wi-Fi grâce au nano routeur TP-Link, en se connectant au réseau « PlOC », le reste de la démarche sera décrit au paragraphe suivant.

Fig.6 : Microcontrôleur Intel Galileo

Interface

Une fois le système installé et alimenté, il faut se connecter au réseau Wi-Fi généré par le nano routeur. Dans notre cas le réseau se nomme « PlOC », et son mot de passe est « owidupoolay ».
Une fois connecté au réseau, vous pouvez accéder à l'interface depuis un ordinateur, un smartphone ou encore une tablette en entrant l'adresse http://192.168.42.42:4242 dans votre navigateur.

Vous accéder alors à la page d'accueil du site, où vous pouvez accéder au monitoring de votre plante.

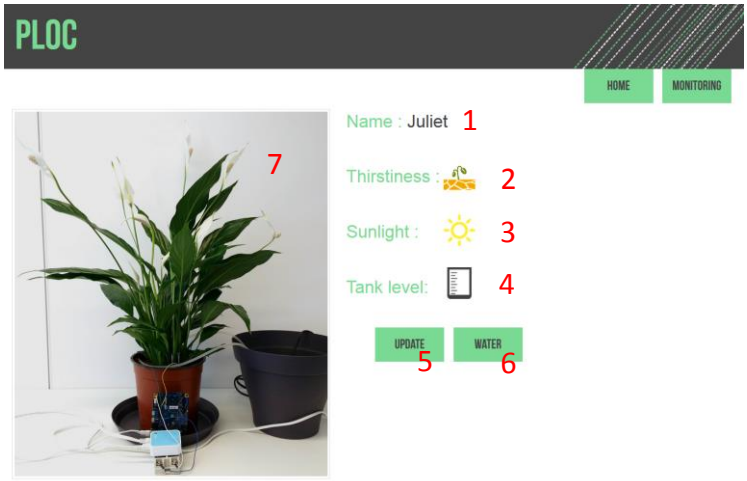

Fig.7 : Interface web

Sur cette page de monitoring on peut observer la photo de la plante concernée (7) et différentes informations la concernant. Ces informations sont rafraîchies automatiquement toutes les dix minutes, il est possible de forcer la mise à jour en cliquant sur le bouton « UPDATE » (5).
Nous avons donc accès :

- (1) Au nom de la plante, ici « Juliet »
- (2) A son niveau de soif (ou d'humidité) en trois niveaux :

Fraîchement arrosée  Correct  Besoin d'eau (sécheresse) 

- (3) A son niveau d'ensoleillement en quatre niveaux :

Très lumineux  Lumineux Couvert (faible)  Nuit (trop faible) 

- (4) Au niveau du réservoir

Correct  A remplir 

Bien que l'arrosage se fasse automatiquement quand la plante en a besoin, il est possible de le déclencher manuellement, en cliquant sur le bouton « WATER » (6).

Fonctionnement

Hardware

- Intel Galileo
- Pompe d'aquarium 12V DC ou électrovanne
- relais 12V
- alimentations 12V DC
- Composants électroniques (décrits dans le paragraphe *Schéma de câblage*)

Description du matériel

"Arduino made by Intel."
Les cartes Arduino sont équipées de microcontrolleurs Atmel AVR. Il s'agit d'une architecture de type Harward 8 bits (RISC[1]) modifiée. Cette carte est un peu spécifique vu qu'elle permet de faire fonctionner des programmes pour AVR ainsi que pour x86 tout en conservant la notion de shields pour le prototypage et les évolutions matérielles.

Pompes : Il y a deux possibilités pour l'arrosage,
- par pompe (pour les plantes ayant besoin de beaucoup d'eau)
- par électrovanne, l'eau arrivera au niveau de plateau – l'arrosage se fera alors par capillarité

Relais : Un relais 12V est utilisé pour manipuler la pompe (On/Off).

Alimentation 12V : Une alimentation 12V continue est requise. Pour nos tests, une alimentation 3 à 12V par pompe a été utilisée. **Bien penser à vérifier la tension totale des composants et dimensionner l'alimentation en conséquence si elle est commune !**

Schéma de câblage

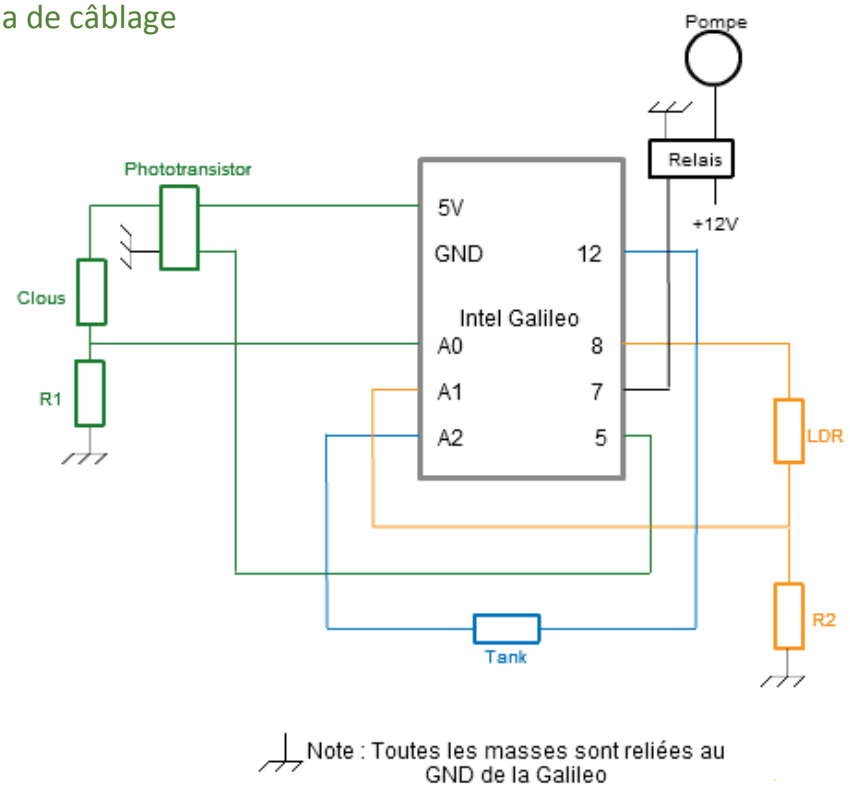

Fig.8 : Schéma électrique

Le schéma ci-dessus est composé de 4 parties :

- au centre la **carte Intel Galileo**, avec les pins utilisés,
- en noir, le système d'arrosage : **relais + pompe**
- en vert, la mesure d'humidité : un **phototransistor** pour envoyer le courant sur les **clous** (le 5V continu de la Galileo a apparemment plus d'ampérage que celui des pins normaux), et une **résistance de 18kΩ**.
- en jaune, la mesure de luminosité : une **LDR** (photorésistance) et une résistance de **1KΩ**
- et enfin en bleu, la mesure de niveau d'eau, où l'on connecte simplement les deux fils du réservoir au pin dédié et à la masse.

Software

GPIO Manager

L'objectif ici, va être d'utiliser notre Intel Galileo pour contrôler l'activation des différents capteurs et le déclenchement de l'alimentation en eau. Pour cela, nous allons avoir besoin d'interroger les GPIO de la carte pour envoyer le signal.

GPIO ?

Une GPIO, General Purpose Input / Output, est tout simplement un port d'entrée / sortie sans utilisation prédéfinie. Ces GPIO peuvent être associés à des connecteurs physiques (ou "pin"), disposés directement sur le circuit - ici, l'Intel Galileo. Dans notre cas, ces pins sont au nombre de 14, ce qui nous permettra largement de subvenir à nos besoins.

Ces GPIO sont utilisés en mode Output simple pour alimenter les différents éléments (ce qui revient à faire sortir des valeurs binaires (0 ou 1) sur les pins).
Pour lire une information, on utilise le mode input où l'on pourra lire soit des valeurs binaires, soit en analogique. Ce mode analogique a été intégré dans la Galileo avec le composant AD7298 ADC IC. Ce composant interne possède 8 canaux, mais seuls 6 d'entre eux sont connectés sur la Galileo. Chaque canal a une résolution de 12 bit. Les valeurs de sorties varient donc entre 0 et 4095, 0 correspondant à une tension d'entrée nulle et 4095, à 5V.

Certains peuvent également être utilisés en mode PWM, Pulse Width Modulation, permettant de faire facilement varier la valeur associée. Cette fonctionnalité n'est pas nécessaire au projet pour le moment, mais elle pourra être utile lors de futures améliorations; pour faire varier le débit de liquide, ou utiliser un débitmètre (en mode lecture) par exemple.

Première étape : accès via le système

Le système Linux embarqué permet, via son système de fichiers, d'accéder à toutes les fonctions nécessaires à la configuration et l'exploitation des connecteurs. Ce qui les concerne se trouve à l'emplacement **/sys/class/gpio**[1].

Tout d'abord, il faut ouvrir l'accès au connecteur avant de pouvoir en faire quoi que ce soit. Via une commande système, cela donne, pour exporter le connecteur numéro 7 (indiqué comme tel sur la Galileo) :

```
echo -n "27" > /sys/class/gpio/export
```

Attention : l'identifiant du connecteur physique ne correspond pas à celui du GPIO correspondant. Il faut se référer à la documentation Intel pour trouver les correspondances exactes.

La commande précédente a eu pour conséquence l'apparition de nouveaux nœuds dans le système de fichiers :

/sys/class/gpio/gpio27
/sys/class/gpio/gpio27/direction
/sys/class/gpio/gpio27/drive
/sys/class/gpio/gpio27/value

[1] **Références** [Programming GPIO from Linux][http://www.malinov.com/Home/sergey-s-blog/intelgalileo-programminggpiofromlinux]

La prochaine étape consiste à configurer le GPIO pour accéder en écriture au connecteur. Pour cela, il suffit de le préciser sur le nœud créé précédemment :

echo -n "out" > /sys/class/gpio/gpio27/direction

Pour utiliser les pins, il faut donc lui envoyer une information binaire. Soit envoyer un signal fort (1), soit aucun signal (0). On précise cela par la commande :

echo -n "strong" > /sys/class/gpio/gpio27/drive

La configuration est alors terminée. Pour l'exploitation, il suffit de procéder de la même manière pour envoyer un signal :

echo -n "1" > /sys/class/gpio/gpio27/value

Et pour arrêter d'émettre :

echo -n "0" > /sys/class/gpio/gpio27/value

Pour fermer le canal vers ce pin de cette manière :

echo -n "27" > /sys/class/gpio/unexport

En ce qui concerne la lecture analogique, les entrées seront accessible sur les fichiers :

/sys/bus/iio/devices/iio:device0/in_voltageX_raw

Les sorties analog sont multiplexées, et peuvent être dédiées à de l'output, il faut donc les configure avant de les utiliser. Les multiplexeurs sont contrôlables par le biais des pins. L'exemple suivant montre comment configure le pin A0. Attention, référez-vous au tableau d'entrées / sorties [2]avant de choisir un pin.

```
echo -n "37" > /sys/class/gpio/export
echo -n "out" > /sys/class/gpio/gpio37/direction
echo -n "0" > /sys/class/gpio/gpio37/value
```

De la même manière que ce qui a été fait précédemment, la première commande permet d'ouvrir le noeud 37 permettant de contrôler le multiplexer A0. La seconde configure la GPIO en OUTPUT, la dernière connecte A0 à la puce ADC. Notre analog est donc connecté, on peut lire sa valeur depuis Sysfs:

```
cat /sys/bus/iio/devices/iio\:device0/in_voltage0_raw
2593
```

Pour plus de commodité, un programme en JavaScript a été créé pour reproduire cette démarche par un service en Node.js.

Deuxième étape : Helper JavaScript

Pour simplifier le travail, un objet GalileoGpio a été créé :

```
var GalileoGpio = function(){}
```

Pour éviter toute erreur lors de la manipulation des GPIO, le numéro de pin est mappé à l'identifiant de son GPIO :

```
this.pin =
{
        7: "27",
        8: "26"
};
```

[2] http://www.malinov.com/Home/sergey-s-blog/intelgalileo-programminggpiofromlinux

Il suffit maintenant d'implémenter les méthodes correspondant aux commandes système précédemment décrites. En prenant comme exemple la commande permettant d'exporter, on obtient en généralisant:

```
GalileoGpio.prototype.openPin = function(pin_number)
{
        fs.writeFileSync("/sys/class/gpio/export", this.pin[pin_number]);
};
```

Ce code d'exemple peut être reproduit pour implémenter l'écriture sur direction, drive et value via des méthodes setPinDirection, setPinPortDrive et writePin.

Pour rendre le module accessible dans le contexte Node.js, il est nécessaire de l'exporter:

```
module.exports = new GalileoGpio;
```

PLOC Manager

Le manager va permettre de prendre le contrôle des différents composants (capteurs, pompe) et récupérer leurs données. On commence par instancier un GalileoGpio :

```
var galil = require('./gpio-galileo-helper');
```

Afin de mieux s'y retrouver, on labellise les différents connecteurs (pin), on caractérise l'état allumé ou éteint (state = envoi signal ou non) et enfin on initialise les informations à stocker :

```
var PlocManager = function()
{
        this.pin =
        {
                'pot': 'A0',
                'ldrOut': 'A1',
                'low': 'A2',
                'temp': 'A3',
                'probe': 13,
                'tank': 12,
                'clou': 5,
                'pump': 7,
                'ldrIn': 8
        }
        this.state =
        {
                 'ON':   "1",
                'OFF':  "0"
         };
        this.info =
        {
                 "Humidity": 0,
                 "Light": 0,
                 "Level": 42,
                "Temperature":0
        }
}
```

Gestion des pins

Comme il a été présenté dans le paragraphe précédent, le GPIO Manager s'occupe des instructions « bas niveau » à envoyer à la Galileo. Histoire d'éviter toute redondance, des fonctions génériques ont été implémentée pour l'ouverture en écriture des pins (open Alim), l'ouverture en lecture (openAnalog) et la fermeture (closeAlim) :

```
PlocManager.prototype.openAlim = function(pin_number)
{
 galil.openPin(this.pin[pin_number]);
 galil.setPinDirection(this.pin[pin_number], "out");
 galil.setPinPortDrive(this.pin[pin_number], "strong");
 galil.writePin(this.pin[pin_number], this.state['ON']);
};

PlocManager.prototype.openAnalog = function(pin_number)
{
 galil.openPin(this.pin[pin_number]);
 galil.setPinDirection(this.pin[pin_number], "out");
 galil.writePin(this.pin[pin_number], this.state['OFF']);
};

PlocManager.prototype.closeAlim = function(pin_number)
{
 galil.writePin(this.pin[pin_number], this.state['OFF']);
 galil.closePin(this.pin[pin_number]);
};
```

Pour utiliser nos différents capteurs, il suffit donc d'utiliser ces fonctions sur les pins concernés et de remplir le **this.info** correspondant :

Exemple pour l'humidité

```
PlocManager.prototype.getHumidity = function()
{
 this.openAlim('clou');
 this.openAnalog('pot');

 this.info["Humidity"] = galil.readAnalogPin(this.pin['pot'])

 this.closeAlim('clou');
 galil.closePin(this.pin['pot']);

 return this.info["Humidity"];
};
```

Le contrôle de la pompe est un peu plus complexe, en effet il est nécessaire de définir une durée d'arrosage, qui sera défini depuis le serveur :

```
PlocManager.prototype.setPump = function(delay)
{
 var that = this;
 galil.openPin(this.pin['pump']);
 galil.setPinDirection(this.pin['pump'], "out");
 galil.setPinPortDrive(this.pin['pump'], "strong");

 if (this.info["Humidity"] < 2000 && this.info["Level"] != 0)
 {
  galil.writePin(this.pin['pump'], this.state['ON']);
  setTimeout(function()
   {
    galil.writePin(that.pin['pump'], that.state['OFF']);
    galil.closePin(that.pin['pump']);
   }, delay);
 }
 else
 {
  galil.closePin(this.pin['pump']);
 }
};
```

Les commandes permettant d'exploiter une pompe devraient maintenant être familières. La méthode ci-dessus automatise son appel, et, via l'emploi d'un **setTimeout**, permet de déterminer le temps pendant lequel celle-ci va pomper via le paramètre **delay**. L'appel d'une callback permettrait de chainer nos commandes.

La gestion pure des composants est terminée, on peut passer au serveur.

API Rest

Le but est d'obtenir un système le plus ouvert possible et pouvant être utilisé par l'ensemble le plus large possible d'appareil/interface. Une couche dite d'API a été implémentée afin de pouvoir récupérer l'ensemble des plantes enregistrées. Cela donne alors accès aux informations les concernant mais aussi de déclencher les capteurs et la pompe.

A partir de ces deux actions, on peut alors obtenir une interface (simple) de monitoring, facile à utiliser. Le serveur reste pour le moment sans interface et accessible en ssh par le réseau (mode opératoire décrit au paragraphe précédent). Le client web sera alors accessible depuis n'importe quel appareil connecté au réseau Wi Fi de Ploc (cf. page.7, paragraphe Interface).

Explications

L'API offerte par PLOC est assez simple, elle se compose en deux parties:

- Une action [GET] /api/profiles permettant de récupérer la liste des plantes
- Un ensemble d'action [POST] /api/profiles/{name} permettant d'accéder aux informations de la plante nommée {name}

Il est donc possible d'effectuer un premier appel, par exemple au lancement de la page de monitoring pour récupérer la liste des plantes, le serveur va créer alors un flux JSON du type:
[{name: 'Juliet', endpoint: '/api/Juliet'}, {name: 'Juliet', endpoint: '/api/Juliet'}]

Chaque profil étant contenu dans un fichier .json dans le dossier /profiles de ploc avec le schéma suivant:

```
{
        "name": "Juliet",
        "api": "juliet",
        "data":
    {
      "time": 10000,
      "humidity":0,
      "light":0,
      "temperature":0,
      "tank":42
    }
}
```

Technique

Node.js était l'environnement le plus logique pour notre API car ce Framework est couramment utilisé sur les cartes Galileo. De plus il s'utilise très bien dans le cadre de la création d'API.

Pour plus de facilités au niveau de la création de sites et de services web avec Node.js, le package Express a été ajouté. A l'aide d'un script il est possible de charger les différents profils de plantes puis d'exposer les deux endpoints vu précédemment.

Afin de pouvoir travailler, il faut d'abord charger les différents modules à utiliser : fs pour créer, manipuler des fichier, async pour travailler de manière asynchrone et le fameux ploc manager.

```
var fs = require('fs');
var async = require('async');
var ploc = require('./ploc-manager');
```

Il faut ensuite mettre en place le serveur

```
var port = 4242;
var express = require('express');
var app = express();
var server = require('http').createServer(app, { log: false })
```

La partie express permet la manipulation des JSON **profiles** où sont stockées les informations.

```
app.use(express.json());
app.use(express.urlencoded());
app.use(express.compress());
app.use(express.static(__dirname + '/public'));

var profiles = [];
```

Il est alors possible de lire tous les profils de plantes dans le dossier **./profile**, et de les charges en base de données pour préparer la requête **POST** qui permettra d'afficher les informations sur le site.

```
fs.readdir('./profiles/', function (err, files)
{
	files.forEach(function (f)
	{
		if (f.endsWith('.json'))
		{
		  fs.readFile('./profiles/' + f,  function (err, data)
		  {
			if (err)
			{
				console.log('Error: ' + err);
				process.exit();
			}
			else
			{
				var r = JSON.parse(data);
				profiles.push(r);
				console.log('Plant ' + r.name + ' added');
			}
		  });
		}
	});
});
```

Le fichier JSON ne sauvegarde que le dernier relevé, on va donc créer un log pour garder un historique.

```
fs.appendFile('log.csv', 'Time,Plant,Humidity,Light,Temp,Tank\n', function (err)
		{
			if (err) throw err;
		});
```

La requête GET, démarrée depuis le site, va interroger pour chaque profil et récupérer les informations demandées dans la variable **re**.

```
app.get('/api/profiles', function (req,res)
{
	res.header("Access-Control-Allow-Origin", "*");
	res.header("Access-Control-Allow-Headers", "X-Requested-With");
	res.header("Content-Type", "application/json");

	var re;
	….
}
```

La récupération de données s'effectue simplement en appelant la fonction adaptée :

```
re.data.humidity = "" + ploc.getHumidity();
```

Une fois toutes les informations stockées dans **re**, on peut l'envoyer au client et attendre la prochaine commande

```
res.send(re);

server.listen(process.env.PORT || port);
console.log('server started on port ' + (process.env.PORT || port));
```

Informations complémentaires

Contact

bagnos_c@epitech.eu
contact@hub.epitech.eu

Documentation / sources complémentaires

https://github.com/Innovation-Hub/ploc