# Projet RICM4 SmartCitizen

# FRÉBY Rodolphe - LABAT Paul

# 9 avril 2014

# Table des matières

| 1  | Introduction                                     | 2             |  |  |  |  |
|----|--|---------------|--|--|--|--|
| 2  | Cahier des charges                               | 2             |  |  |  |  |
| 3  | MQ Telemetry Transport                           | 2             |  |  |  |  |
| 4  | Les capteurs                                     | 3             |  |  |  |  |
| 5  | Base de données  5.1 Choix de la base de données | <b>4</b><br>4 |  |  |  |  |
| 6  | Site web   | 5             |  |  |  |  |
| 7  | Le serveur NodeJs 7.1 Architecture               | <b>8</b><br>8 |  |  |  |  |
| 8  | Problèmes rencontrés                             | 9             |  |  |  |  |
| 9  | Les métriques                                    | 9             |  |  |  |  |
| 10 | Gestion de projet                                | 10            |  |  |  |  |
| 11 | 1 Évolutions possibles                           |               |  |  |  |  |
| 12 | Conclusion                                       | 11            |  |  |  |  |

## 1 Introduction

L'augmentation fulgurante du nombre d'objets connectés amène des nouveaux concepts comme celui des villes intelligentes et de l'internet des objets. Les objets ainsi connectés, représente potentiellement des milliers de capteurs qui fournissent des informations en temps réel sur notre environnement. Ces données peuvent être ensuite utilisées pour réaliser de la supervision dans la ville et adapter celle-ci à ses usagés. Ces derniers, en plus d'être des usagés, peuvent être également des contributeurs actifs à l'évolution et à la supervision de leur ville. Nous pouvons alors parler de citoyen responsable.

C'est dans ce contexte que se place le projet *SmartCitizen*. Il vise à rendre les usagés (les citoyens) responsables de leur ville en partageant des données avec les services municipaux et les autres citoyens.

## 2 Cahier des charges

Le but de ce projet est de superviser une ville grâce à un réseau de capteurs réparti dans celle-ci. Il est ensuite possible de visualiser ces données sur une carte interactive.

Premièrement, le site web doit permettre aux utilisateurs de consulter les données de chacun des capteurs sur une carte. Ensuite, en fonction des valeurs des capteurs, des zones de couleurs différentes pourront être affichées. Elles permettront aux utilisateurs de voir rapidement si, par exemple, le seuil de pollution de cette zone est sous le seuil autorisé par l'état ou non.

Pour chaque capteur, nous affichons dans une popup, des données relatives à sa localisation, son propriétaire ainsi que la dernière valeur enregistrée et un graphique représentant l'évolution de ses données au cours du temps.

Finalement, un utilisateur pourra, au moyen d'une application mobile dédiée ou de Twitter directement, envoyer des données, des images ou tout autre type d'information pouvant être intéressante pour les utilisateurs. Ces données seront également affichées sur la carte.

Pour qu'un détenteur de capteur puisse voir celui-ci sur la carte, il doit être présent dans la collection qui comprend tous les capteurs autorisés à émettre. Pour cela il faut contacter la personne en charge de la base de données. Celle-ci renverra un identificateur qui sera nécessaire pour publier les données.

## 3 MQ Telemetry Transport

MQ Telemetry Transport (MQTT) est un protocole de messagerie qui se base sur les protocoles TCP/IP. Il possède de nombreux avantages qui nous ont poussés à l'utiliser pour ce projet. Il est basé sur le patron de conception publish ?subscribe et se veux simple à implémenter. Ces caractéristiques font qu'il est adapté à des programmes s'exécutant dans des environnements à forte contrainte. Les capteurs réalisés pour ce projet sont réalisés sur des cartes Arduino, les contraintes matérielles sont donc bien présentes (peu de puissance de calcul et d'espace mémoire). Ils peuvent être utilisés dans des lieux avec une connexion à internet à faible débit. Cette dernière peut également être instable.

Le patron de conception sur lequel est basé MQTT fonctionne de la manière suivante :

- une machine envoie des données sur un topic particulier, par exemple sensor/humidity/id.
- une (ou plusieurs) machine s'abonne à ce topic. Elle peut maintenant recevoir les données envoyées par la machine précédente

## 4 Les capteurs

L'émergence de solution électronique *Do It Yourself* conçue à partir de cartes arduino nous a amenés à les utiliser. Elles possèdent également de nombreux avantages : elles sont très répandues avec une grande quantité de documentation et une communauté active très présente.

Pour réaliser nos capteurs, nous avons utilisé une carte Arduino Leonardo ainsi qu'un shield Ethernet ENC28J60. Ce dernier permet à la carte arduino d'envoyer des données sur le réseau. Nous avons cependant remarqué une instabilité de la connexion. En effet, au bout d'un certain temps, la carte devient incapable d'envoyer des données sur le réseau, ou bien les paquets arrivent en retard. Lorsque la connexion est perdue, nous forçons la reconnexion avec le serveur.

Il existe d'autres shield Ethernet qu'il serait bon d'utiliser pour comparer les taux de perte (un paquet perdu par minute avec notre carte).

Nous avons eu à disposition plusieurs capteurs que nous avons ajoutés sur la carte :

- MQ4 : détection de méthane
- DHT11 : capteur de température et d'humidité
- Compteur Geiger (pas totalement pris en charge)

Pour que les données envoyées sur internet soient récupérables par un serveur, nous avons utilisé le protocole MQTT. Ainsi, chaque capteur peut envoyer ses données vers un topic particulier pour qu'une machine tierce les récupère, les traite et les stocke dans une base de données.

La librairie *nanode*, qui est une implémentation de ce protocole pour les cartes arduino avec des shields Ethernet, nous a permis de réaliser l'envoi de message via ce protocole.

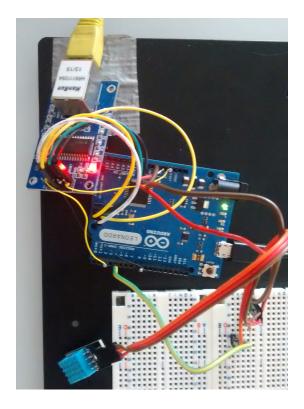


FIGURE 1 – Arduino leaonardo avec le shield Ethernet ENC28J60 et le capteur DHT11

### 5 Base de données

#### 5.1 Choix de la base de données

Notre première approche de la base de données a été de choisir parmi les deux disponibles dans le programme *node-red*. Celui-ci fournit des services *redis* et *mongoDB*.

Nous nous sommes donc renseignés sur le fonctionnement de redis et mongoDB. De ces recherches, plusieurs caractéristiques de mongoDB nous ont parus essentielles quant à la réalisation de notre projet.

Elle fait partie de la mouvance NoSQL, c'est-à-dire, des bases de données non plus limitées à des tables relationnelles, mais fonctionnant sur le principe de simples tableaux associatifs. N'ayant plus de schéma particuliers à suivre, les entrées dans la base peuvent évoluer au cours du temps : rajout ou suppression de champs sans avoir à effectuer une migration ou une modification du schéma des tables. Cela permet de gagner en souplesse. Ce système est particulièrement bien adapté aux systèmes géants, c'est-à-dire des systèmes utilisant des bases de données comprenant d'énormes quantités de données (des millions voir milliards d'entrées).

Notre projet, permettant de stocker toutes les valeurs envoyées par un capteur, est susceptible de prendre en charge une grande quantité de capteurs, sous-entendant de grandes quantités de données. MongoDB correspond donc bien à nos attentes concernant la gestion des données.

Autre point important, mongoDB prend en charge nativement la réplication et la répartition de charge. Ceci est important si on considère que l'on peut être amené à gérer un fort trafic avec des milliers voir des millions de requêtes. Cela permet d'augmenter la tolérance aux pannes (et ainsi augmenter la disponibilité), d'alléger la charge d'un serveur particulier et de disposer des mêmes informations sur plusieurs serveurs situés dans différents pays (réduction du temps de latence entre l'émission de la requête client et la réception et l'affichage du résultat).

## 5.2 Choix technique

Nous avons créé une base de données nommée dbSMartCitizen. Celle-ci contient deux collections : sensors et sensorsData. Les collections sont des équivalences aux tables de relations dans les bases de type SQL.

La première collection permet de décrire tous les capteurs qui sont autorisés à interagir avec notre serveur. Les caractéristiques des capteurs sont décrites par une structure particulière :

```
latitude: Number,
idKey: String,
type: String,
longitude: Number,
city: String,
owner: String
```

Ces données sont utilisées pour effectuer des recherches dans la base. Elles sont également utilisées pour l'affichage des résultats des recherches et dans les popup. De cette façon, nous pouvons distinguer les capteurs les uns des autres.

L'ajout des capteurs dans cette collection se fait au moyen d'un script python. Cela permet à l'administrateur de la base de gérer lui-même ces ajouts. Ce script vérifie également si le capteur est présent dans la base, cela permet d'éviter les conflits lors de la réception des données via MQTT ou lors de la recherche dans la base.

La seconde collection contient les valeurs des capteurs. Elle suit le schéma suivant :

```
idKey: String,
value : Number,
date:{type:Date, default:Date.now()},
```

Nous avons choisi de mettre un identifiant par capteur qui suit la norme *Universal Unique Identifier*. Cela permet d'identifier de manière unique un capteur et il sert également à faire une « jonction » entre les deux collections (pour récupérer les valeurs de tous les capteurs de type *Celsius* par exemple).

### 6 Site web

Chaque capteur envoie des données sur le serveur. Ce dernier les traites et les envoie dans la base de données. Pour que les citoyens puissent visualiser ces données, nous avons mis en place un site web. Celui-ci est divisé en deux pages principales.

La première, nommée map.html, permet de faire de la visualisation sur une carte. Chaque capteur est représenté par un point. Chaque point est cliquable et permet, via une popup, d'obtenir diverses informations concernant les capteurs : localisation (coordonnées GPS, ville), le propriétaire ainsi que la dernière valeur enregistrée. Un graphique permettant d'obtenir l'évolution des 30 dernières valeurs est également affiché dans la popup. Cependant, il n'est disponible que lorsque nous cliquons une deuxième fois sur le marqueur représentant le capteur. Cette limitation est due à un choix technique. Nous ne voulions pas charger les 30 dernières valeurs de tous les capteurs présents dans la base, si nous avions des milliers de capteurs, cela aurait été trop gourmand en ressources au niveau de la base et il aurait été probable que la machine cliente ne puisse pas gérer une aussi grosse quantité de données.

La carte et la gestion des événements marqueurs sont effectués par une librairie open source : Leaflet.js. Les tuiles qui représentent la carte en elle-même sont fournies par OpenStreetMap, un gestionnaire de carte libre.

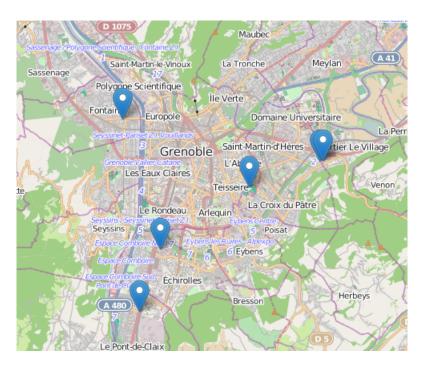


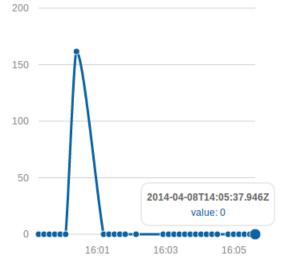
FIGURE 2 – Carte avec 6 capteurs

Les popups, qui affichent les informations sont visualisables par simple clic sur un marqueur. Chaque fois qu'il est cliqué, on effectue une requête dans la base pour récupérer les dernières valeurs et ainsi avoir en permanence des graphiques à jour.

Owner: Rodolphe Freby Latitude: 45.1794005 Longitude: 5.779343 City: Gieres Valeur: 0 ppm

Owner : Rodolphe Freby
latitude : 45.1338664
longitude : 5.7009845
City : Grenoble
value : 16.0545384343 percent

(a) Une popup au chargement de la carte



(b) Une popup avec un graphique

FIGURE 3 – Les popups des capteurs

Les graphiques, quant à eux, sont réalisés au moyen d'une libraire JavaScript : morris.js. Elle est simple d'utilisation et facile à prendre en main. Chaque objet graphique prend en paramètre un tableau de valeurs qui correspond à un couple [date, valeur]. On choisi ensuite comment doivent être affichées les valeurs (quelle valeur associée à quel axe).

Morris.js met à notre disposition quatre types de graphique, cependant, seuls deux d'entre eux sont utilisables pour nos besoins.

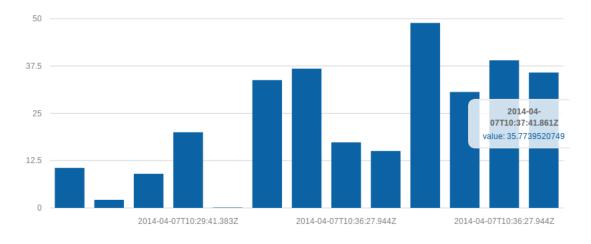


FIGURE 4 – Un histogramme morris.js

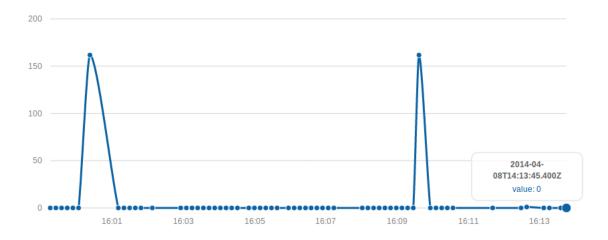


FIGURE 5 – Une courbe morris.js

La seconde page, about.html, permet de récupérer toutes les valeurs des capteurs. Il existe un système de filtrage qui permet de choisir quels capteurs affichés. Il est ainsi possible de choisir plusieurs options :

- Identifiant de capteur
- Type de capteur
- Ville
- Ville et type de capteur

La page affiche ainsi un graphique et toutes les informations relatives au capteur, se trouvant dans la collection *sensors*: localisation (coordonnées GPS, ville), type, propriétaire. Nous affichons également la date de la dernière mise à jour (dernière valeur reçue).

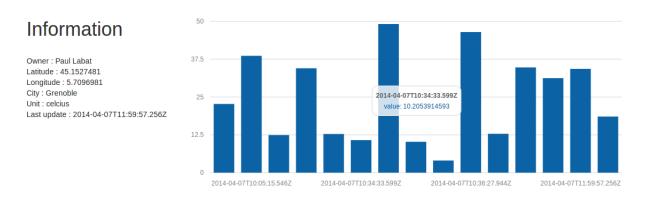


FIGURE 6 – Informations affichées dans la page about.html

### 7 Le serveur NodeJs

Le serveur est le coeur du projet. Il fait office d'intermédiaire entre les différentes fonctionnalités qui sont offertes par *SmartCitizen*.

#### 7.1 Architecture

Le diagramme de déploiement suivant résume son fonctionnement au sein du système.

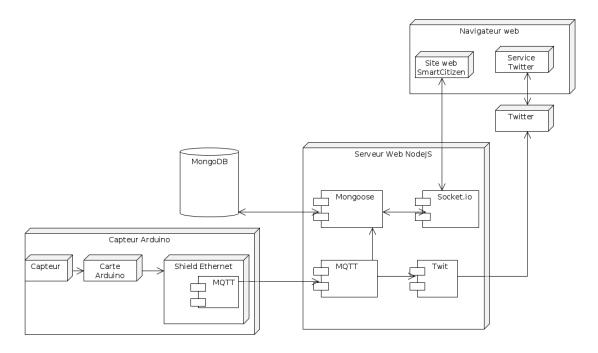


FIGURE 7 – Diagramme représentatif du fonctionnement de SmartCitizen

Le serveur est basé sur le framework libre *NodeJS*. Il permet d'écrire des applications réseau en JavaScript. Nous utilisons plusieurs plug-ins pour interagir avec les différents éléments qui constituent notre projet.

- Mongoose : Ce plug-in sert d'interface entre le serveur et la base de données mongoDB.
   Il nous permet d'injecter ou de récupérer des données depuis les différentes collections.
- MQTT : Il permet de s'abonner à des topics pour ensuite récupérer les données.
- Socket.io : Permet l'envoi de données par des websockets (typiquement entre le serveur et la page web).
- Twit: Ce plug-in nous permet d'interagir avec Twitter.

#### 7.2 Fonctionnement

Lorsqu'un capteur envoie des données via MQTT, le serveur les reçoit, vérifie que le capteur qui envoie ces données se trouve bien dans la collection *sensors*. Il le fait au moyen du message reçu qui se trouve sous la forme idCapteur#valeur. Nous pouvons ainsi récupérer l'identifiant du capteur et vérifier son existence dans la collection.

Le serveur vérifie ensuite si la valeur dépasse un seuil. Celui-ci est dépendant du type de capteur (température, pollution, humidité ...). Si cette valeur est dépassée, un tweet est envoyé, signalant que le capteur a dépassé le seuil. Nous précisons également dans le tweet, son propriétaire ainsi que sa localisation et sa valeur actuelle. Elle est ensuite insérée dans la collection par l'intermédiaire du plug-in mongoose.

Lorsque l'utilisateur demande l'affichage d'un graphique ou de données relatives à un capteur, il envoi une requête. Celle-ci transite sur le réseau au moyen de websockets. De cette façon, le code effectuant la communication avec la base de données se trouve uniquement sur le serveur (protection de celle-ci). Les réponses effectuent le chemin inverse et sont ensuite traitées de sorte à mettre à jour l'affichage.

### 8 Problèmes rencontrés

Lors de la réalisation de ce projet, nous nous sommes retrouvés confrontés à deux problèmes principaux.

Le premier concernait le matériel Arduino. Nous avions des problèmes de détection de la carte par port série lorsque nous utilisions l'IDE Arduino. Nous nous sommes rendus compte que cela venait de la version officielle de Java que nous avions installée sur Ubuntu. Un retour à l'utilisation de l'OpenJdk nous a permis de résoudre ce problème.

La mise en place du shield Ethernet pour la carte Arduino nous a également posé problème. Le manque de documentation et l'absence de connaissance sur Arduino nous ont beaucoup ralentis. Grâce à l'aide de M. Richard ainsi qu'un tutoriel trouvé sur Internet nous ont permis de réaliser le câblage. Nous avons également observé deux problèmes logiciels liés à ce shield. Au bout d'un certain temps, la carte perdait la connexion MQTT. Pour résoudre le problème, nous effectuons une reconnexion. Nous avons également observé un autre problème lié à ce shield. L'envoi des messages MQTT était marqué comme réussi par la carte, mais le serveur ne recevait rien. Nous avons donc mis en place un appel régulier à la méthode uip.poll() qui permet de résoudre ces envois de paquets fantômes.

Le second problème concerne l'interaction logicielle avec la base de données. La syntaxe de requête étant totalement différente de celle utilisée dans une base SQL, nous avons eu des difficultés à en comprendre le fonctionnement. Deux semaines ont été nécessaire pour prendre en main correctement la syntaxe qui peut être grandement variable. En effet, il existe plusieurs façons d'écrire une même requête. Une fois la syntaxe comprise, il est assez aisé d'interagir avec la base mongoDB.

## 9 Les métriques

Pour savoir le nombre de lignes de code que nous avons écrites durant ce projet, nous avons trouvé un plug-in développé pour nodeJS, nommé *SLOC*, qui permet de connaître cela. Bien entendu, nous n'incluons pas dans le comptage les librairies utilisées.

|                  | map.html | index.html | about.html | server.js | test-msg.py | manage-sensors.py |
|------------------|----------|------------|------------|-----------|-------------|-------------------|
| lignes physiques | 156      | 65         | 212        | 448       | 28          | 78                |
| sloc             | 116      | 44         | 174        | 371       | 22          | 49                |
| commentaires     | 8        | 4          | 5          | 24        | 1           | 14                |
| vide             | 32       | 17         | 33         | 53        | 5           | 15                |

Le fichier Arduino principal contient 65 lignes de code.

## 10 Gestion de projet

La gestion de projet s'est déroulée selon le diagramme de gantt suivant :

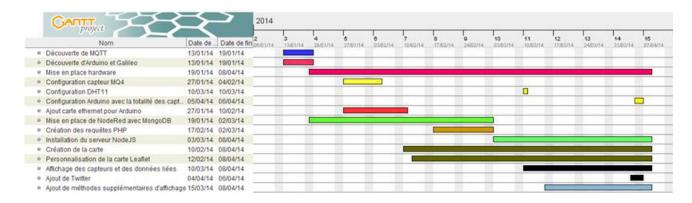


FIGURE 8 – Diagramme de gantt

# 11 Évolutions possibles

Le but de ce projet étant de rendre les citoyens responsables, une des évolutions majeures de ce projet serait de prendre en charge les capteurs des téléphones. En effet, ceux-ci embarquent de plus en plus de capteurs et les équipementiers fournissent eux aussi des fonctionnalités liées à l'interaction avec l'environnement. Nous pourrions ainsi mettre en place une application pour smartphone pour gérer cela. Elle pourrait également prendre en charge l'affichage de la carte, ainsi que les différentes informations liées aux capteurs.

Pour enrichir la base de données, une diversification des types et du format des sources est à envisager. En observant les médias sociaux, des images, des vidéos, du texte pourrait être récupérés puis affichés sur la carte. Il faudrait alors différencier l'affichage entre les capteurs et les autres types de sources.

La gestion du cache de la base de données serait à mettre en place. Cette optimisation serait notamment nécessaire lorsque le nombre d'utilisateurs commencerait à être important.

Dans l'optique de rendre les données accessibles par tous, une API peut être mise au point pour fournir l'intégralité des collections disponibles aux développeurs. Cela s'inscrit dans une optique d'*Open Data*.

La dernière évolution à laquelle nous avons pensé est la mise au point de sessions utilisateurs qui seraient accessibles via une page web. Ces sessions permettraient aux usagés de paramétrer des alertes ou d'ajouter des capteurs. Les alertes peuvent être sur les réseaux sociaux ou en utilisant un envoi de mail.

De plus, il est pour le moment possible de n'avoir qu'un seul capteur sur une position GPS précise. Si plusieurs capteurs se trouvent sur la même position GPS, ils doivent être légèrement espacé dans la abse de données pour ensuite apparaître sur la carte.

# 12 Conclusion

Lors de la réalisation de ce projet, nous avons eu à utiliser des technologies qui nous étaient inconnues, nous avons eu de nombreux problèmes auxquels nous avons pu apporter des solutions. Ces problèmes, bien qu'ils nous aient empêchés de réaliser l'entièreté du cahier des charges, ne nous ont pas pour autant bloqués au point de ne pas avoir un système fonctionnant comme nous le désirions (sans certaines fonctionnalités).

Ce projet nous a permis d'appréhender des concepts récents et en plein essor : l'Internet des objets, les villes intelligentes pour ne citer qu'eux. Nous avons ainsi appris à lier du matériel (des capteurs) à du logiciel et une base de données.