Capteur/Nuc : Manuel utilisateur

Antoine LEVAL

May 3, 2018

Abstract

Le projet Nuc/Capteurs se compose de 5 modules avec des dépendances entre eux. Tout le projet est développé majoritairement en Java et à l'aide de Maven. Le code source du projet est téléchargeable à l'adresse suivante : https://github.com/AQROPOL/aqropolNUC.

1 Les modules

Le projet parent **aqropolNUC** se compose des quatres sous modules suivant :

- `aqropolNUC` › `aqropolrest` Responsable de la partie rest, communication avec Android (via http). Le mobile peut faire des requêtes GET sur les capteurs, les mesures et connaitre l'identifiant du Nuc, et peut également faire une requête POST avec un tableau json clé(identifiant du nuc):valeur(dernier hash connu) afin de supprimer les données déjà collectée et postée sur le serveur.
- `aqropolNUC` › `aqropolmomProducer` Librairie java permettant très simplement d'instancier un producteur amqp pour poster les données des capteurs sur le broker rabbitMQ.
- `aqropolNUC` › `aqropolmomConsumer` Librairie java permettant d'instancier un consommateur amqp pour recevoir les données publiées par les capteurs et les faire persister en base de donnée.
- `aqropolNUC` › `measurepmsensor` Module permettant de lire les données sortant de l'Arduino via l'interface USB.

et d'un module à part en C++ destiné uniquement à l'Arduino :

- `aqropolNUC` › `Arduino` › `sensor` › `novaPMsensor` Module Arduino permettant de lire les données du capteurs SDS011, et de les envoyer via l'interface USB en json. Permet également d'écrire sur un écran LCD les données du capteur en temps réel.

2 Mise en place de l'environnement

Afin de lancer les modules du projet, celui-ci nécessite quelques prérequis. Assurez vous d'avoir le JDK Java 1.8 minimum ainsi que Maven 4.0. Un serveur postgre ainsi qu'un broker RabbitMQ sera nécessaire. Le reste des dépendances sera téléchargé grâce à Maven.

2.1 Base de donnée : Postgre

Si ce n'est pas déjà fait, démarrer un serveur postgres. Créer une base de donnée (ici 'aqropol', mais cela est modifiable dans les fichiers properties des modules). Pour instancier la base de donnée, le schema se trouve à la racine du projet `aqropolNUC` › `init.sql`.

```
CREATE USER api PASSWORD 'password';

GRANT UPDATE, SELECT, INSERT, DELETE ON sensor, measure, nuc TO api;
GRANT SELECT ON measure TO PUBLIC;
GRANT SELECT, UPDATE, INSERT, DELETE ON measure, sensor, nuc TO api;
```

L'utilisateur créé est 'api', si vous souhaitez changer le nom et le mot de passe de cet utilisateur, les changements devront être répercuté dans le pom parent du projet :

- aqropolNUC > pom.xml

aux lignes suivantes :

```
<properties>
     <aqropol.datasource.username>api</aqropol.datasource.username>
       <aqropol.datasource.password>password</aqropol.datasource.password>
       <aqropol.datasource.database>aqropol</aqropol.datasource.database>
       <aqropol.datasource.host>localhost</aqropol.datasource.host>
       <aqropol.datasource.port>5432</aqropol.datasource.port>
       ...
</properties>
```

A la fin du fichier `init.sql` se trouve les entrées nécessaire au fonctionnement du projet dans la base de donnée :

```
insert into nuc (token) values ('token_de_mon_nuc_dev_001');
insert into sensor (name, type, unity) values ('SDS011', 'pm10', 'ug/m3');
insert into sensor (name, type, unity) values ('SDS011', 'pm2.5', 'ug/m3');
```

La première ligne est l'identifiant du Nuc, il doit être unique par rapport au serveur principal. Les deux lignes suivantes sont deux valeurs différentes pour un même capteurs, le capteur Nova PM (SDS011). Si le capteur n'est pas présent dans la table sensor alors il ne sera pas possible d'entrer des mesures pour celui-ci.

2.2 Broker AMQP : RabbitMQ

Lancer un serveur rabbitMQ avec les paramètres suivant :

```
default_vhost=aqropol
default_user=aqropol
default_pass=64GbL3k7uc33QCtc
management.listener.port=8081
listeners.tcp.default=5672
```

Ou modifiez les paramètres dans le pom parent du projet : `aqropolNUC` > `pom.xml`.
Les paramètres de connexion au broker rabbitMQ :

```
<properties>
       <aqropol.amqp.host>10.42.0.1</aqropol.amqp.host>
       <aqropol.amqp.port>5672</aqropol.amqp.port>
       <aqropol.amqp.vhost>aqropol</aqropol.amqp.vhost>
       <aqropol.amqp.user>aqropol</aqropol.amqp.user>
       <aqropol.amqp.pass>64GbL3k7uc33QCtc</aqropol.amqp.pass>
       ...
</properties>
```

Les paramètre commun au producteur et au consommateur (même exchange, même queue, même route) :

```
     <aqropol.amqp.channel.exchanger>aqropol_sensors</aqropol.amqp.channel.exchanger>
       <aqropol.amqp.channel.route>aqropol_sensors_data</aqropol.amqp.channel.route>
       <aqropol.amqp.queue>sensors</aqropol.amqp.queue>
```

Et les paramètres de la queue, propre au producer :

```
       <aqropol.ampq.queue.durable>true</aqropol.ampq.queue.durable>
       <aqropol.amqp.queue.exclusive>false</aqropol.amqp.queue.exclusive>
       <aqropol.amqp.queue.autodelete>false</aqropol.amqp.queue.autodelete>
```

3 Exécution des modules

L'éxecution des modules peut être fait dans un ordre quelconque, à l'exception du module `measurepmsensor` qui doit être éxecuté seulement après avoir branché l'arduino à un port USB de votre machine.

Une fois le serveur postgre en route on peut lancer le service web `lstinlineaqropolrest` puisque celui-ci se contente de mettre en ligne les ressources stocké dans la base.

Lancer le producteur avant le consomateur amqp importe peu puisque les messages seront stocké dans une queue tant qu'ils ne sont pas consommé.

Avant de lancer les modules il faut demander à Maven de les construire. Placez vous à la racine du projet `AqropolNUC` (le pom parent). et éxecutez la commande suivante :

```
mvn clean package
```

3.1 Service REST

Pour démarrer le serveur web, rien de plus simple, il suffit d'éxecuter le jar suivant : `aqropolNUC` > `aqropolrest` > `target` > `aqropol-rest-1.0-SNAPSHOT.jar` : `java -jar aqropolrest/target/aqropol-rest-1.0-SNAPSHOT.jar`.

Ensuite rendez vous à l'adresse : `http://localhost:8080/` :

```
{
  "_links" : {
    "sensors" : {
      "href" : "http://localhost:8080/sensors{?page,size,sort,projection}",
      "templated" : true
    },
    "measures" : {
      "href" : "http://localhost:8080/measures{?page,size,sort,projection}",
      "templated" : true
    },
    "nucs" : {
      "href" : "http://localhost:8080/nucs{?page,size,sort}",
      "templated" : true
    },
    "profile" : {
      "href" : "http://localhost:8080/profile"
    }
  }
}
```

Spring Data Rest utilise le modèle HAL (Hypertext Application Language) pour faire des liens entre toutes les ressources proposé par le service REST. Le paramètre "projection" n'est pas utile ici car il n'en existe qu'un seul et c'est également celui par défaut, le paramètre est donc implicite.

Vous pouvez également vous rendre à l'adresse suivante : `http://localhost:8080/admin` pour accéder à diverse fonctionlitées. Les identifiants pour se connecter aux ressources d'administration se trouve dans la classe : `aqropolNUC` > `aqropolrest` > `src` > `main` > `java` > `config` > `WebSecurityConfig.java`.

```
@Bean
@Override
public UserDetailsService userDetailsService() {
   UserDetails user =
    User.withDefaultPasswordEncoder()
      .username("admin")
      .password("admin")
        .roles("ADMIN", "USER")
        .build();
    return new InMemoryUserDetailsManager(user);
}
```

3.2 RabbitMQ : Producteur

Le module aqropolmomProducer sert à instancier un nouveau producteur pour le broker rabbitMQ avec des paramètres par défaut instancier grâce à maven à la compilation. Si les bon paramètres sont entré dans le pom du projet parent aqropolNUC alors le producteur et le consomateur seront correctement réglé et il ne restera qu'à instancier un nouveau producteur comme ceci :

```
RoutingProducer.RoutingProducerFactory fact = new
    RoutingProducer.RoutingProducerFactory();
RoutingProducer producer = fact.build();
producer.send(jsonMessage);
```

Si certain paramètres doivent être changé la factory offres quelques méthodes :

```
public void setExchanger(String exchanger);
public void setRoute(String route);
public void setEncoding(String encoding);
```

Ce module doit donc être instancié par tout nouveau capteur souhaitant publier des données (récolté par le consommateur, et stocké dans la base de donnée).

3.3 RabbitMQ : Consommateur

Le module aqropolmomConsumer a pour rôle de récolter les données du ou des producteurs. Comme pour le producteur, si les paramètres par défaut du pom parent sont réglé comme il faut, il n'y aura pas besoin de toucher à quoique ce soit, simplement le démarrer et il fera persister dans la base de donnée toutes les mesures envoyés par le producteur.

Voir partie 2.1 et 2.2 pour les paramètres de la base de donnée et du broker rabbitMQ.

Pour démarrer le consommateur il suffit donc d'éxecuter le jar correspondant : aqropolNUC > aqropolmomConsumer > target > mom_consumer-0.0.1-SNAPSHOT.jar

```
java -jar aqropolmomConsumer/target/mom_consumer-0.0.1-SNAPSHOT.jar
```

3.4 Capteur : Measure PM Sensor

Le module measurepmsensor est un module implémentant un producteur rabbitMQ et qui lis les données sortant de l'Arduino via l'interface USB pour les envoyer via le producteur.

Avant d'éxecuter ce module il vous faut donc brancher en USB l'Arduino et s'assurer que les paramètres du module aqropolmomProducer sont correcte.

Pour l'éxecution le module a besoin d'au moins un paramètre. (Le module parse les paramètres grâce à commons-cli de apache). Vous pouvez par exemple afficher l'aide comme ceci :

```
java -jar target/measure-pmsensor-1.0-SNAPSHOT-jar-with-dependencies.jar -h
usage: [-h] | -l | -i <name>
 -h,--help             print this message.
 -i,--interface <name> Ouvre l'interface USB specifie. Par exemple :
                       /dev/tty.usbmodem1421
 -l,--list             Liste les interfaces USB disponibles.
```

L'aide étant assez explicit et exhaustive sur les options existantes, il n'y a pas beaucoup d'explication à rajouter. Il vous suffit donc de chercher l'interface USB correspondant à Arduino. Le nom de l'interface et le répertoire ou les interfaces USB sont situé dépend de l'environnement.

3.5 Arduino : Le capteur

Le code nécessaire à la partie électronique (Arduino) du projet se trouve à cet endroit du projet : aqropolNUC > Arduino > sensor > novaPMsensor > novaPMsensor.ino.

Ce projet utilise un ArduinoMega ADK mais peut être adapté à tout type d'arduino.

Dans le même dossier se trouve un README qui liste les librairies utilisée dans le code ainsi que tout les liens utile à la compréhension du code.

Pour aider à reconstituer le circuit Arduino voici comment était cablé l'écran LCD (bien que non indispensable en phase de prod) :

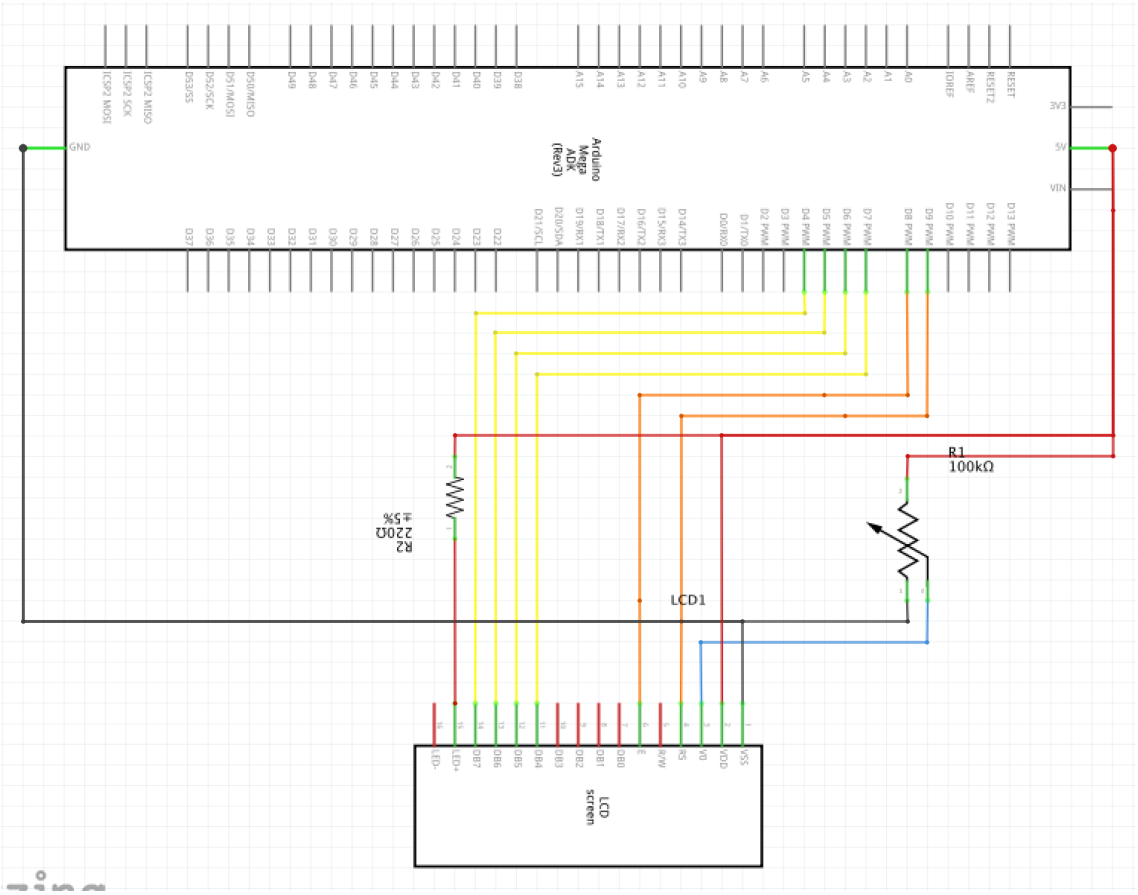

Figure 1: Schéma écran LCD

Ces pins devraient correspondre à un majorité d'Arduino. Néanmoins si vous désirez les changer afin de faire votre propre cablage cela devra être répercuté dans le code à cette ligne :

```
LiquidCrystal lcd(9, 8, 4, 5, 6, 7);
```

La partie capteur SDS011 (Nova PM Sensor) est configuré dans le code à cet endroid :

```
SDS011 my_sds;
...
void setup() {
  my_sds.begin(11, 10);
```

Les paramètres de la méthode sont définis ainsi :

```
begin(uint8_t pin_rx, uint8_t pin_tx);
```

Il suffit donc de cabler le pin rx du SDS011 au pin 11 puis le pin tx au pin 10 de l'Arduino ou alors de modifier le code en conséquence. Il faudra également relier le pin Vin (5V) au pin Vin du capteur et le GND du capteur au GND de l'arduino.

Pour plus d'information, vous pouvez vous référer à la documentation du capteur situé à cet endroit : aqropolNUC > Arduino > sensor > novaPMsensor > SDS011_laser_PM2.5_sensor_specification-V1.3.pdf