

BTS Systèmes Numériques**Option : IR****E 6-2 – PROJET TECHNIQUE****Dossier de présentation et de validation du projet**

Groupement académique :	Créteil Paris Versailles	Session : 2023
Lycée : JOSEPH GAILLARD		
Ville : FORT-DE-FRANCE		
N° du projet :	Nom du projet : Réseau Télémétrique de Mesures en Temps Réel. (R.T.M.T.R n°2) (Mesures de pollutions)	

Projet nouveau	Oui <input checked="" type="checkbox"/>	Non		Projet interne	Ou <input type="checkbox"/>	Non <input checked="" type="checkbox"/>
				Statut des étudiants	Formation initiale	Apprentissage
Spécialité des étudiants	EC <input type="checkbox"/>	IR <input checked="" type="checkbox"/>	Mixte <input type="checkbox"/>	Nombre d'étudiants : 4		
Professeurs responsables :						

I_2_Présentation du projet

Il s'agit de réaliser un réseau télémétrique de mesures en temps réel de stations qui fourniront un flot de données, pour la production d'informations, d'alertes de pollutions. Les polluants à mesurer sont : le dioxyde de soufre, le dioxyde d'azote, l'ozone, le benzène, le plomb, les particules, le monoxyde de carbone, le sulfure d'hydrogène (H₂S) émis par la décomposition des algues (sargasses).

I_4_Expression du besoin

L'entreprise Vision Informatique souhaite réaliser un réseau Télémétrique de surveillance et d'alertes de pollutions sur deux zones (urbaine: Mesure des pollutions inhérentes à l'activité humaine, Côtière: Mesure de la pollution émanant des algues Sargasse, le sulfure d'hydrogène H₂s). Le système devra être facile à déplacer afin d'être déployé, installé dans les zones à observer, à analyser.

Le réseau sera équipé d'analyseurs spécifiques en continu afin de mesurer certains polluants contenus dans l'air ambiant. Les données seront transmises périodiquement afin d'être centralisées sur un serveur (dans un service technique). Les mesures seront ensuite analysées et traitées *in situ*.

Les polluants analysés sont le dioxyde de soufre (**SO₂**), les oxydes d'azote (**NO/NO₂/NOX**), l'ozone (**O₃**), le monoxyde de carbone (**CO**), le mercure gazeux (**Hg**), les particules en suspension (**PM10/PM2.5/PM1**), le sulfure d'hydrogène (**H₂s**). Les paramètres météorologiques seront également relevés comme la température, l'humidité relative, la pression atmosphérique, les précipitations, la vitesse et la direction du vent.

Les stations embarqueront une base de données InFluxDB_Oss pour enregistrer les relevés. *Les TSDB (time series databases), ou bases de données de séries chronologiques, sont des systèmes logiciels optimisés pour trier et organiser des informations mesurées de manière temporelle. Une série chronologique est une série de points de données regroupés à des intervalles successifs et par ordre d'arrivée. Dans le jargon associé aux TSDB, un point de données correspond à une paire heure-valeur ou clé-valeur.*

Chaque nœud est spécialisé pour la mesure d'un polluant qui sera transmis par Radiofréquence à destination d'une passerelle qui réalise la collecte des mesures de pollutions. Cette passerelle (un nano pc) embarque une base de données qui permet l'enregistrement des données (les données de pollutions sont issues de plusieurs systèmes d'acquisition) qui seront exploitées grâce à une supervision locale.

En cas d'interruption de communication avec la supervision, chaque station sera capable d'enregistrer les données des périodes d'observation d'activités.

Le projet se divise en 4 sous-systèmes: **Acquisition, Collecte-Externalisation, Supervision locale, Supervision distante.**

Objectifs :

En tant qu'Etudiant_1 sur le *RTMTR*(Réseau Télémétrique de Mesure en Temps Réel)projet de mesure de pollution, je dois m'assurer que les données des *Noeud* passe dans les capteurs et se rejoignent dans le *Raspberry Pi 3*.

Source de données :

On utilise un module E90-DTU

(<https://www.ebyte.com/en/product-view-news.html?id=409>)

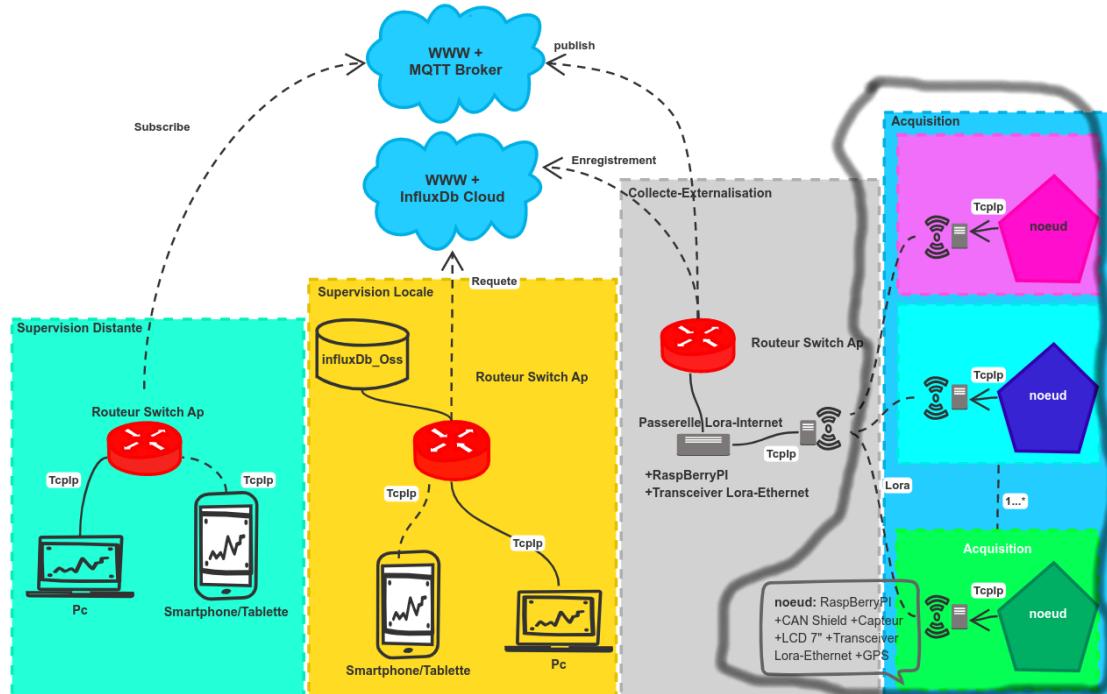
Ce module est branché en prise réseau et relié en câble RJ45 au client (PC).

Les documents technique sont joignable vers ce lien :

(<https://www.ebyte.com/en/data-download.html?page=2&id=365&cid=31#load>)

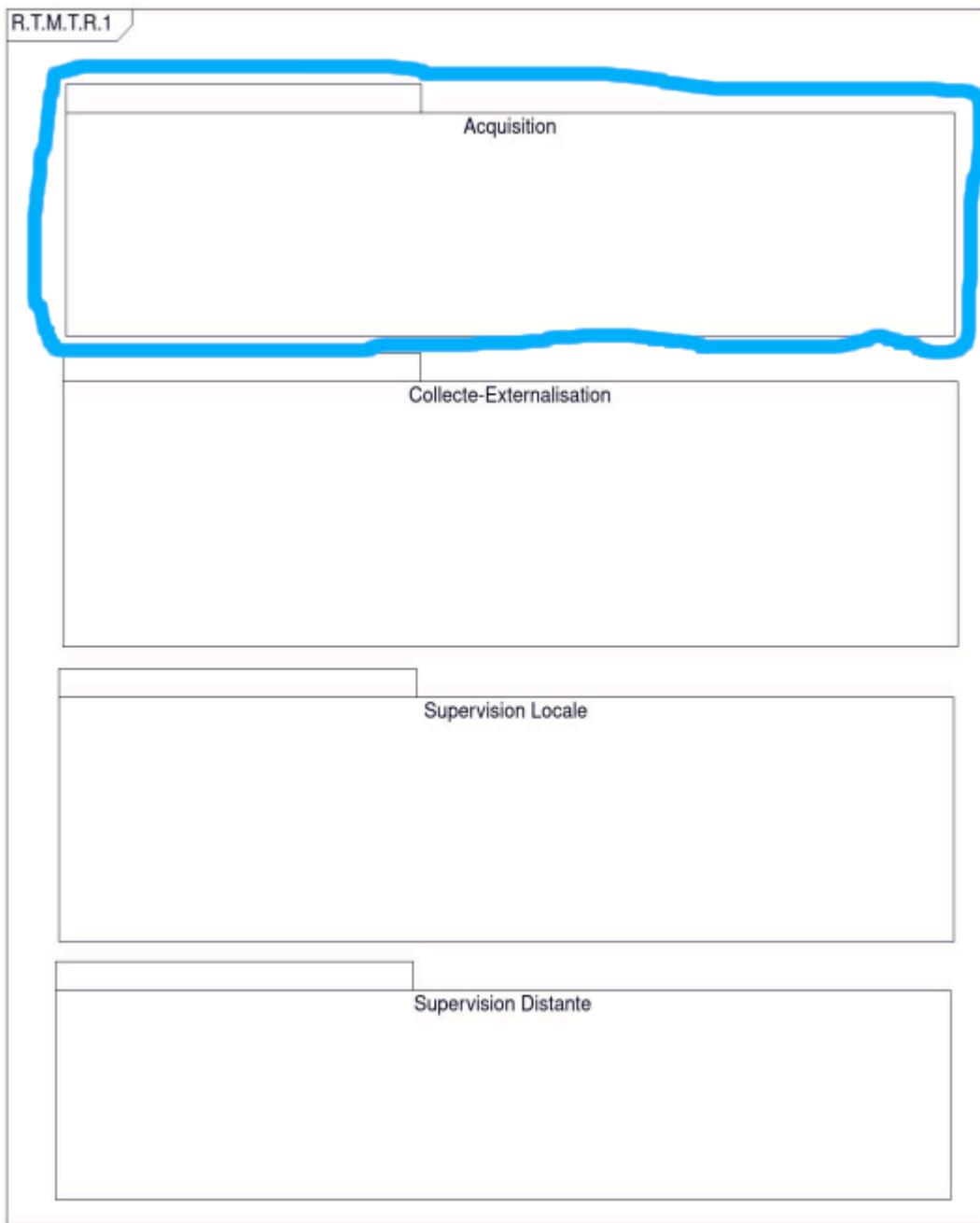
Programmation :

La partie encadrée en bleu est ma partie :



- **noeud :** Correspond aux matériels qui transmettent les données.

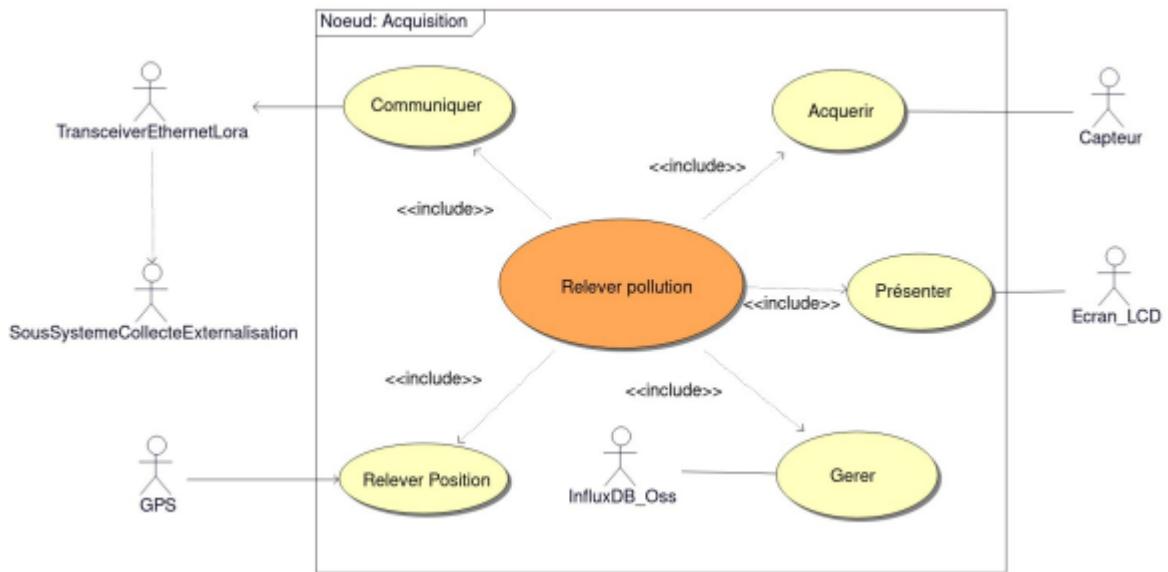
En tant qu'étudiant n° 1 je dois m'occuper du sous-systèmes **Acquisition** :



II_1 SPÉCIFICATION

Acquisition de Mesures (nœud de mesures) de pollutions.

1_Diagramme des cas d'utilisation



2_Description des acteurs

- Capteur
- GPS
- TransceiverEthernetLora
- InfluxDB_Oss
- Ecran_LCD

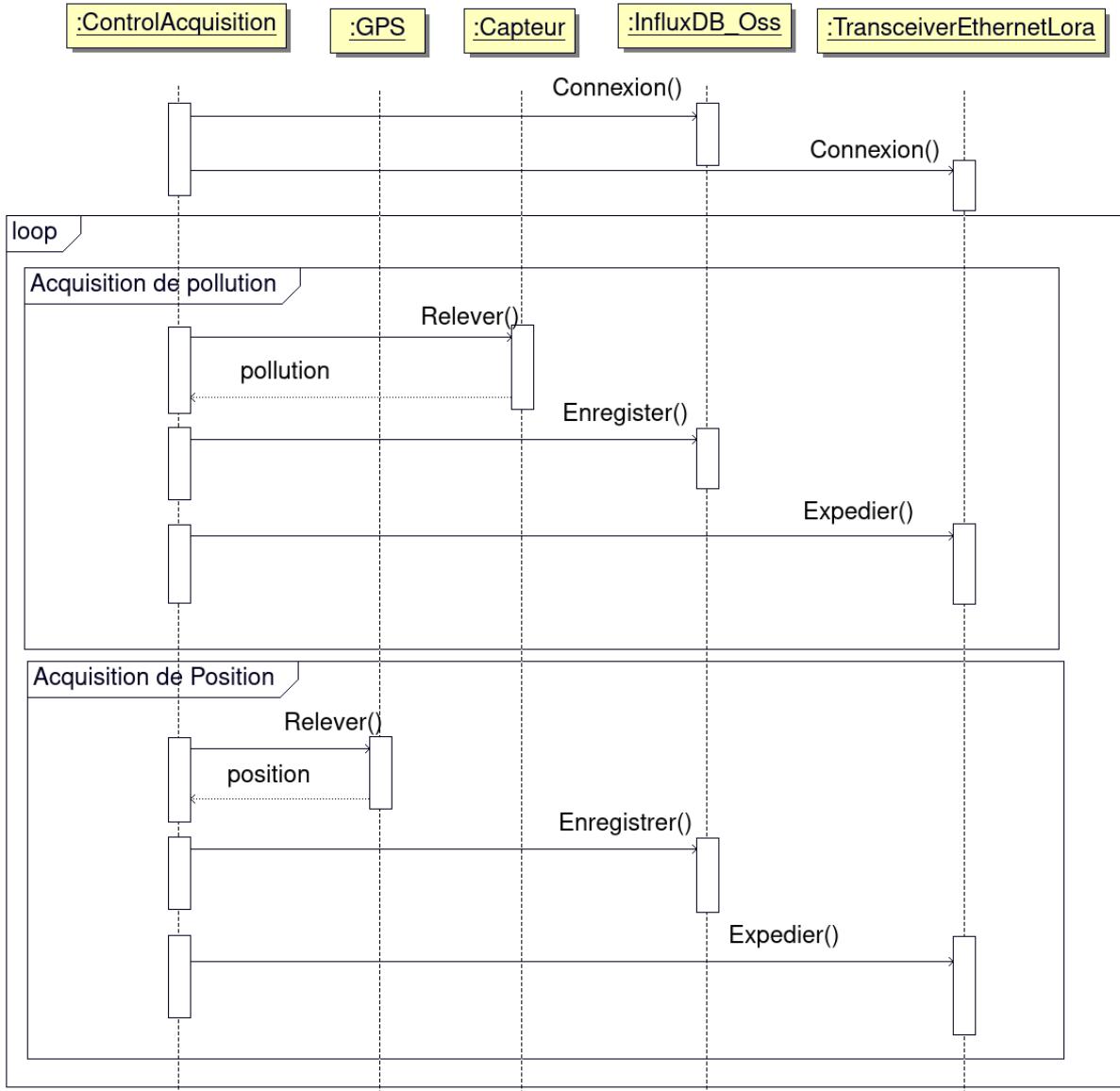
Acteur	Action
Capteur	Dispositif qui fournit la valeur représentative de la pollution à mesurer.
GPS	Relevé de la position du lieu de prélèvement des données.
TransceiverEthernetLora	L'expédition des données est assurée par le TransceiverEthernetLora, et permet la transmission au destinataire : Sous-système «Collecte-Externalisation»
InfluxDB_Oss	Base de données Time Series pour l'enregistrement des données. Le serveur de Base de Données Time Series InfluxDB_Oss intègre des outils de présentation de graphes depuis un dashboard et qui sont accessibles depuis un Browser (Affichage d'un intervalle d'observation, ou affichage en continu).
Ecran_LCD	Permet de tester le bon fonctionnement de la station, de visualiser les données in situ.

3>Description des cas d'utilisation

Cas d'utilisation	Rôle
Acquérir	Conversion analogique numérique puis lecture de la tension $v(t)$ au borne capteur. Cette tension est représentative d'un niveau de pollution
Relever Pollution	La mesure représentative de la pollution est relevée sur le capteur.
Communiquer	Les données acquises doivent être expédiées vers le sous-système Collecte-Externalisation grâce au transceiver Ethernet-Lora.
Gérer	Les données acquises sont aussi enregistrées dans la base de données locale InfluxDB Oss. En cas d'interruption momentanée de communication (perte de connexion), l'enregistrement dans la base de données embarquée InfluxDB Oss permettra de différer l'expédition à destination du sous-système « Collecte-Externalisation »
Présenter	Présentation des données relevées localement depuis la base de données (cas d'utilisation « Gérer ») InfluxB_Oss sur l'écran LCD.
Relever Position	Relever les coordonnées GPS.

Diagramme de Séquence

On convertit le diagramme de classe en diagramme de séquence :



4 Diagramme de classe

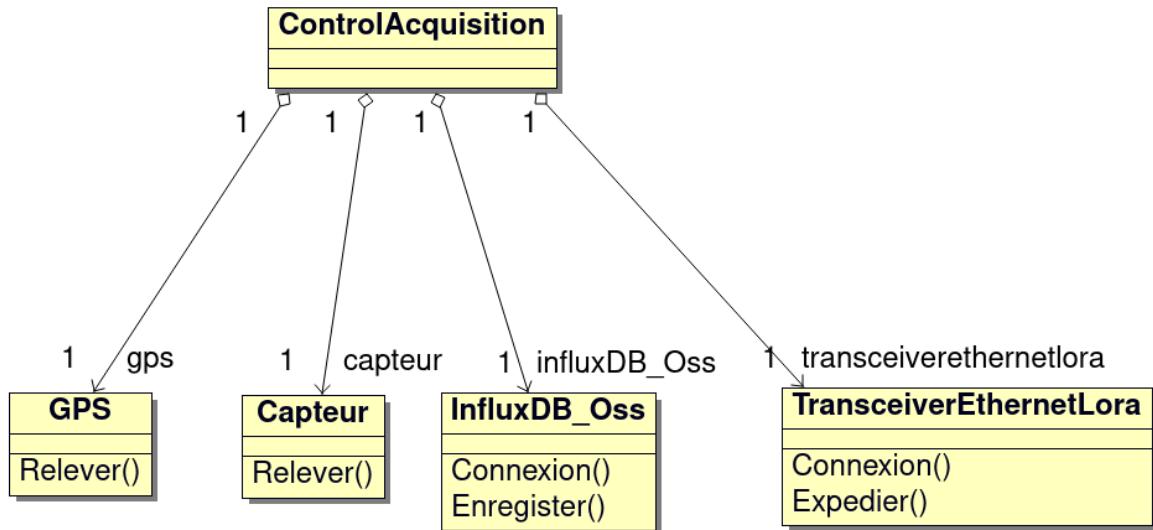
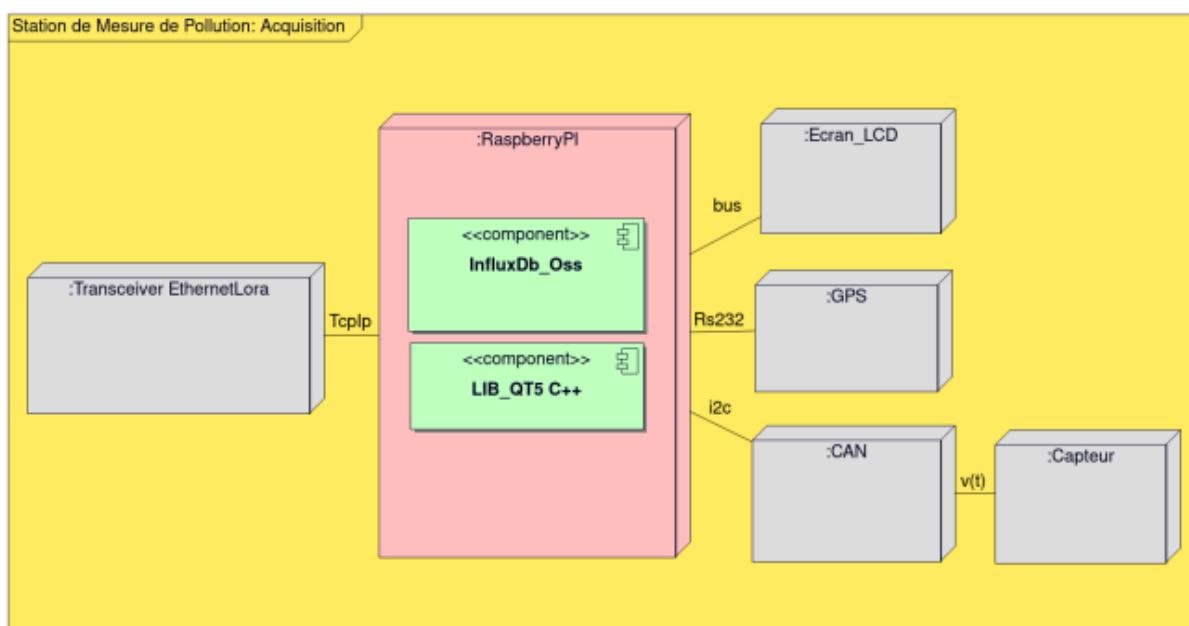


Diagramme de Déploiement



2b Fiche d'activité de l'étudiant 1 (Tâches individuelles) - Sous-système «Acquisition»

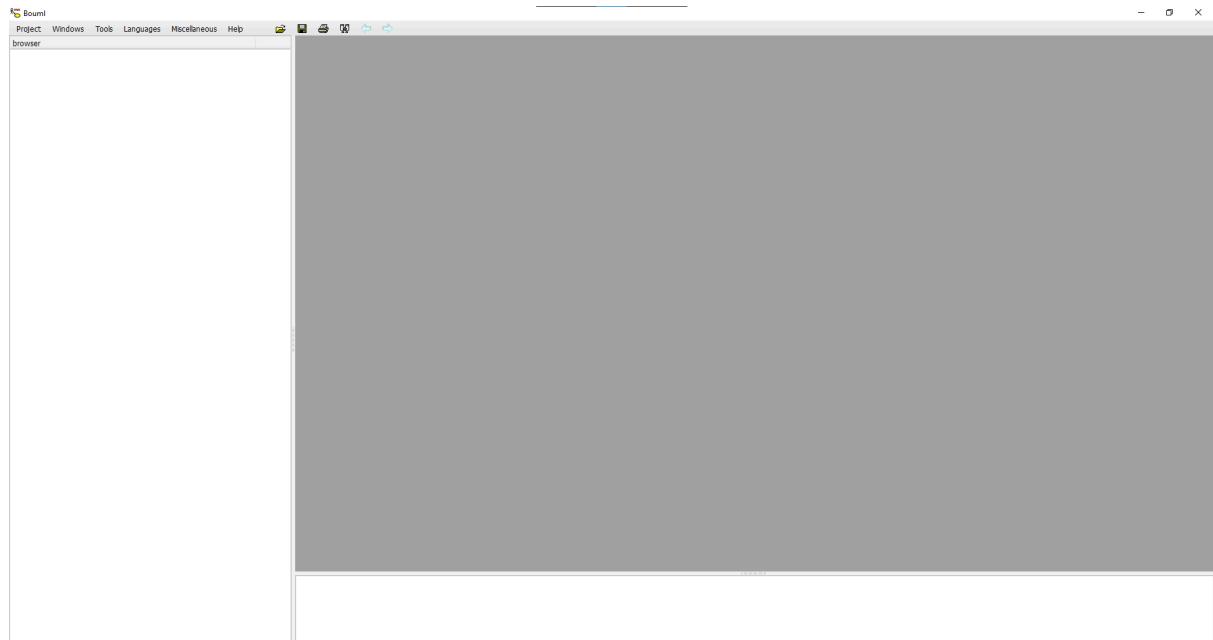
voilà la mission que l'on m'a confiée:

Coder un module logiciel test d'Acquisition (Raspberry +CAN+ Capteur + InfluxDB_Oss) permettant la lecture périodique d'une valeur de pollution du capteur puis son enregistrement dans la base de données locale InfluxDB_Oss.

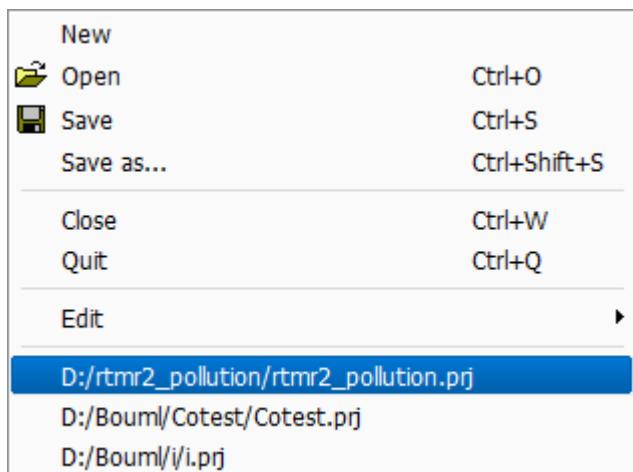
Coder en collaboration l'étudiant n°2 une Ihm permettant de configurer le sous-système pour son premier déploiement (Taux échantillonnage d'acquisition, position Gps, Adresse du serveur InfluxDB_Cloud etc..). Configurer la réPLICATION automatique des données entre influxDB_Oss et InfluxDB_Cloud.

J'utilise le logiciel *BOUML* pour faire des diagrammes avec les informations du projet :

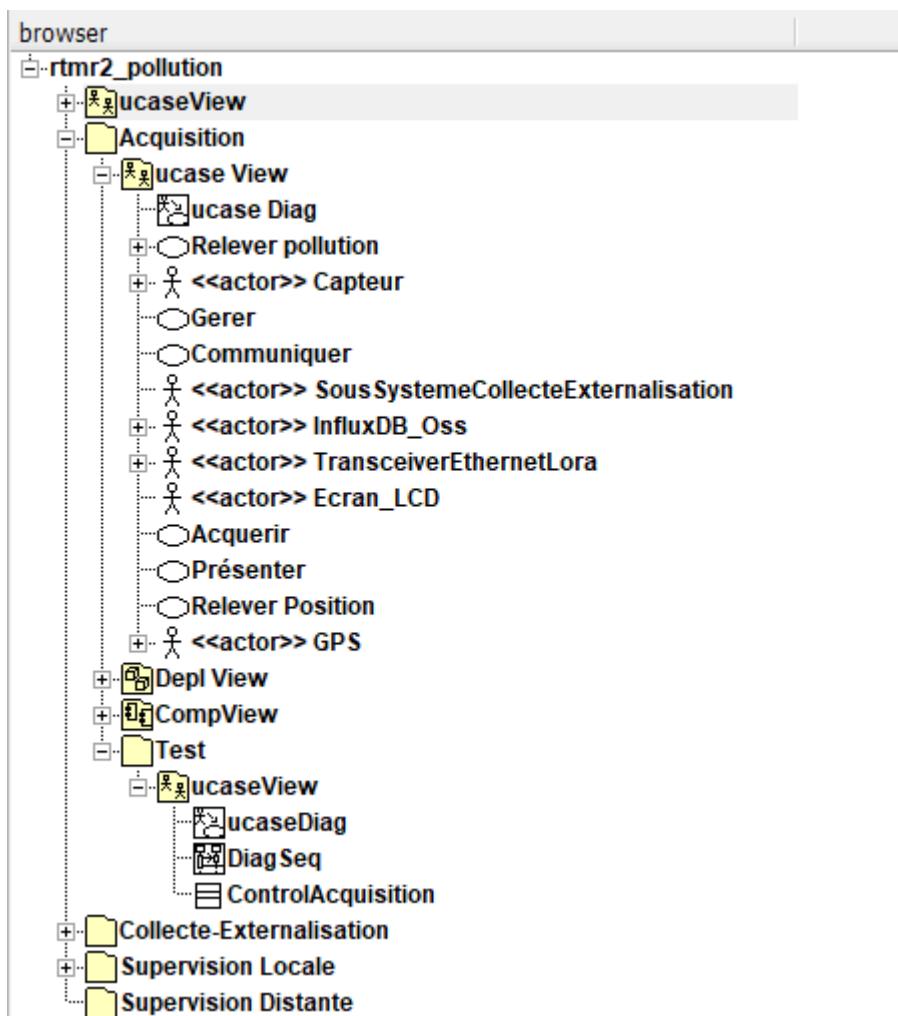
Ici le logiciel est vide, il n'y a aucune classe qui apparaît.



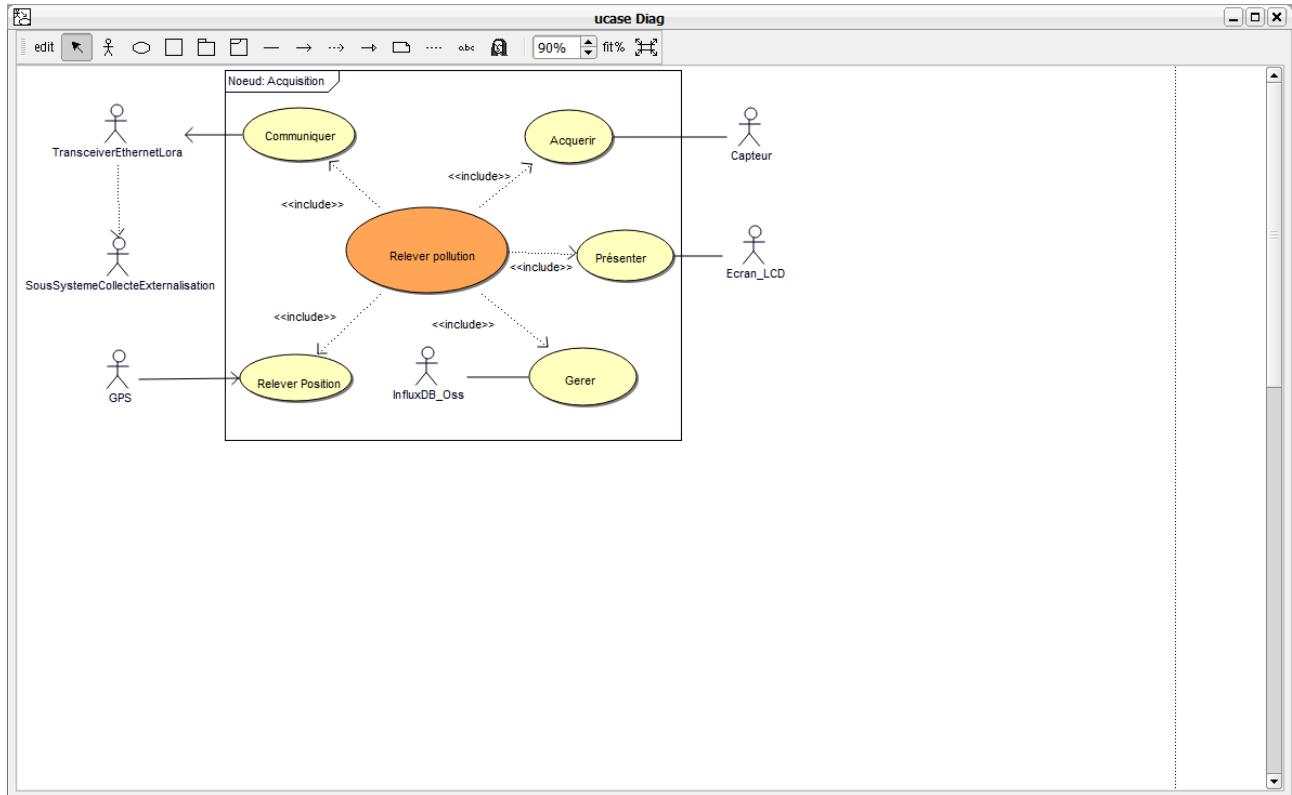
Puis, on ouvre le dossier récent :



Voici les classes générées automatiquement :

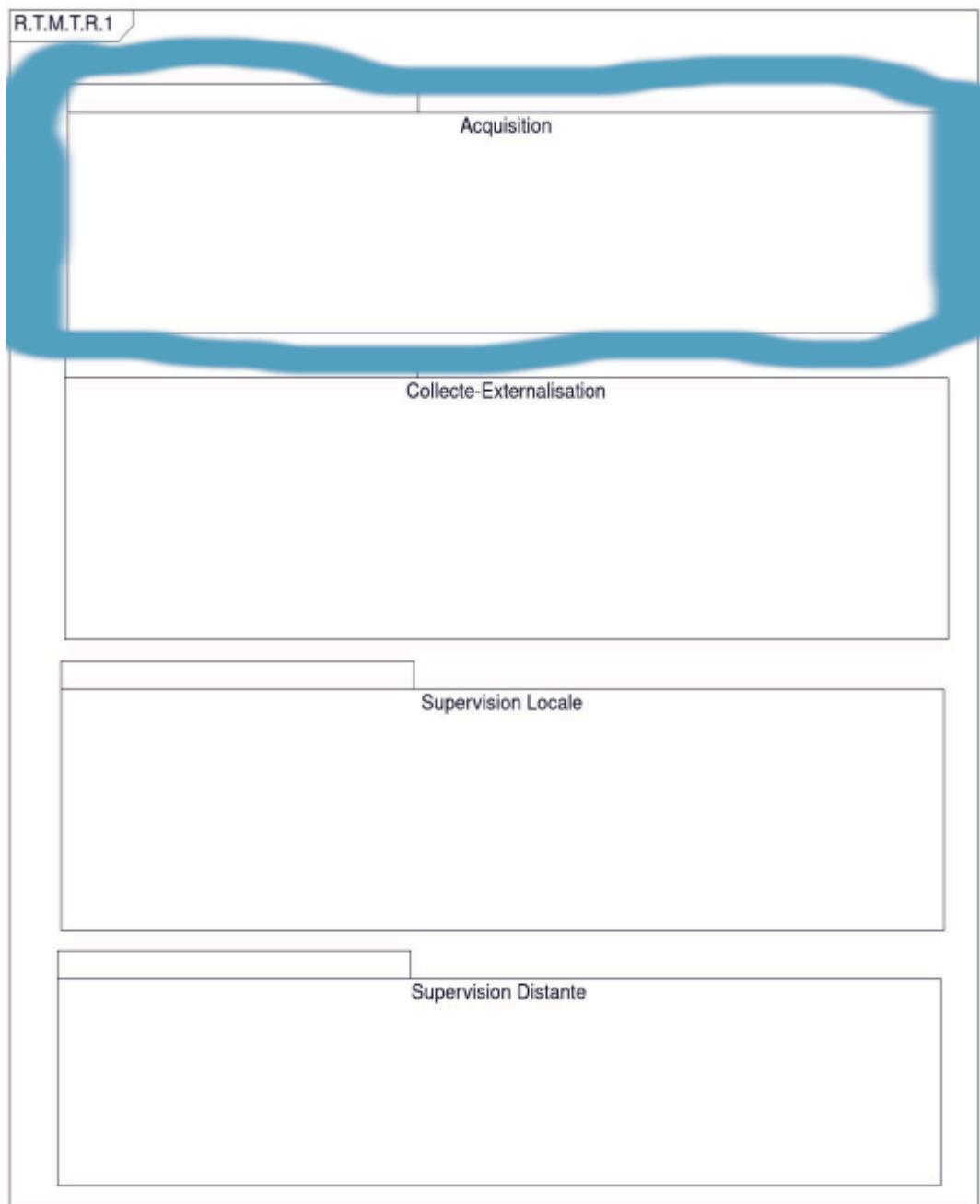


Le diagramme de cas d'utilisation avec les différentes descriptions des acteurs :



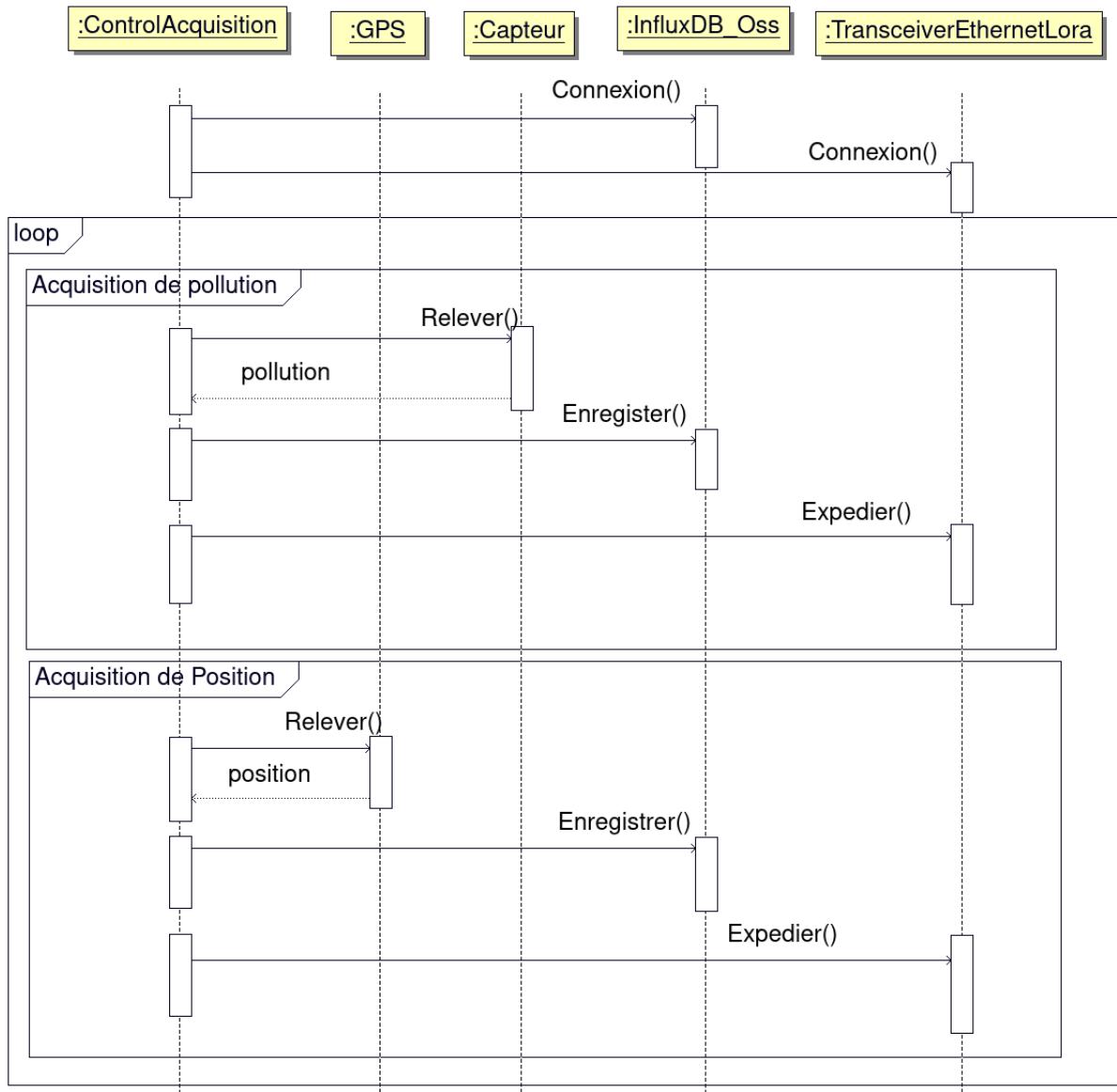
- **TransceiverEthernetLora** : Correspond aux capteurs.
- **GPS** : Correspond aux capteurs qui relève la position.
- **InfluxDB_Oss** : Correspond à la base de données qui gère la pollution.
- **Capteur** : Correspond qui acquiert la pollution.
- **Ecran_LCD** : Correspond au matériel qui présente les valeurs.
- **SousSystemeCollecteExternalisation** : Correspond au sous-système qui collecte les données des capteurs.

J'entoure la partie du sous-système qui me correspond en bleu :



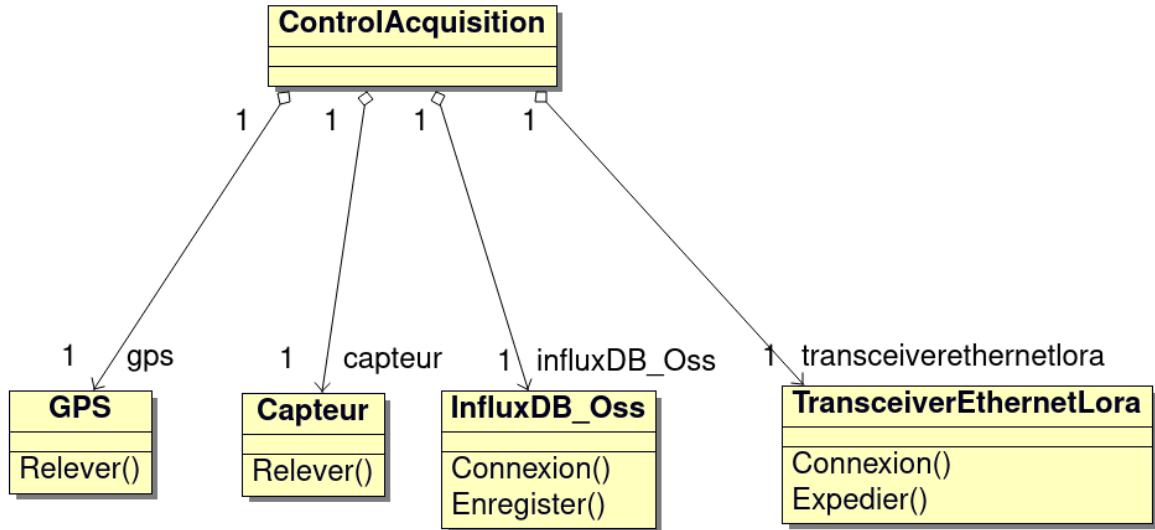
- **Acquisition** : dans cette partie entrant les descriptions qui nous concernent.

On crée un diagramme de séquence avec les informations du cahier des charges :



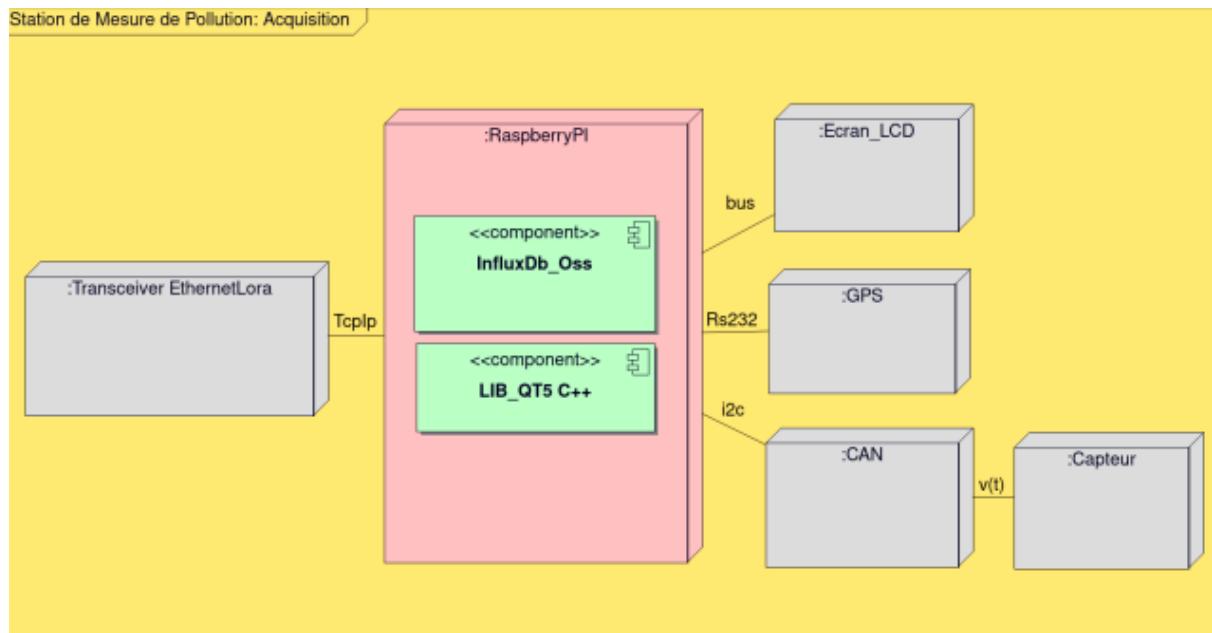
- **Contrôle acquisition** : Correspond à l'acquisition.
- **GPS** : Correspond au capteur de position.
- **Capteur** : Correspond au Raspberry Pi.
- **InfluxDB_Oss** : Correspond à la base de données locale
- **TransceiverEthernetLora** : Correspond à l'expédition des données.

On crée un diagramme de classe avec les informations données :



C'est une retranscription du diagramme de séquence.

Je conçois un diagramme de déploiement pour le matériel qui va être utilisé :



Je dois chercher les informations sur le **RaspberryPi 3** dans la documentation :

On recherche particulièrement la partie **Air Quality Sensor** :

<https://wiki.seeedstudio.com> › Grove... · Traduire cette page · :

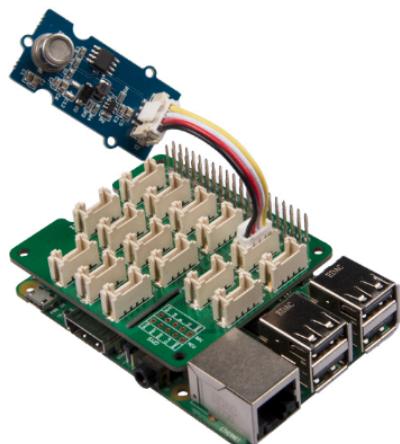
Grove - Air Quality Sensor v1.3 - Seeed Wiki

18 nov. 2014 — This sensor is designed for comprehensive monitor over indoor air condition.

It's responsive to a wide scope of harmful gases, as carbon ...

Vous avez consulté cette page 2 fois. Dernière visite : 18/01/23

Image du RaspberryPi 3 :

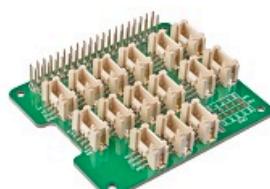


On n'a différente partie en image pour le **RaspberryPi 3** :

Tarte aux framboises



Chapeau de base Grove pour RasPi



Grove - Capteur de qualité de l'air

entrez la description de l'image ici

[Obtenir UN maintenant](#)

[Obtenir UN maintenant](#)

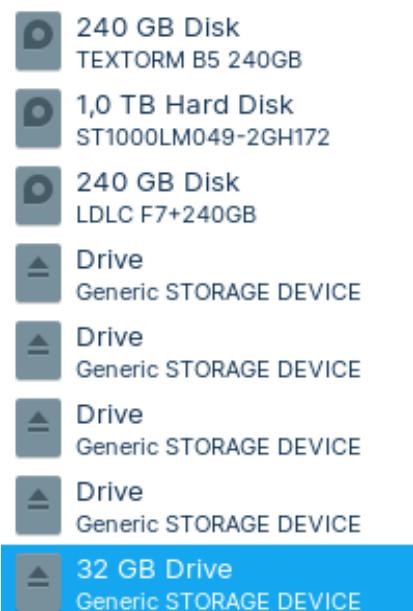
[Obtenir UN maintenant](#)

J'ai les différentes étapes chapitrés :

Matériel

- **Étape 1 .** Choses utilisées dans ce projet:
- **Étape 2 .** Branchez le Grove Base Hat sur Raspberry Pi.
- **Étape 3 .** Connectez le Grove - Capteur de qualité de l'air au port A0 du chapeau de base.
- **Étape 4 .** Connectez le Raspberry Pi au PC via un câble USB.

Ensuite, je sélectionne le nom du disque :



Je flash la **microSD** de 32 Go :



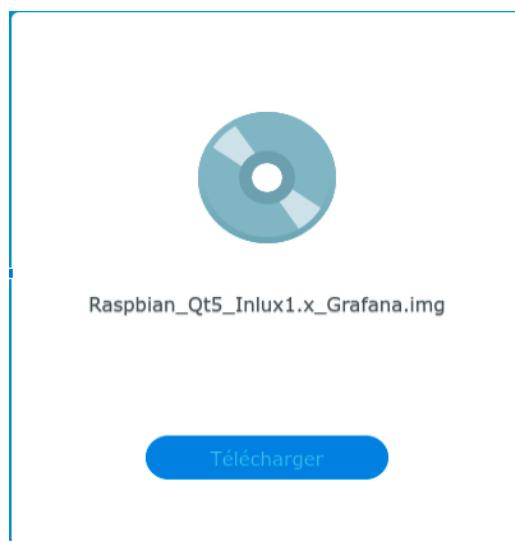
Résultat du Flash :



Je tape l'adresse que mon professeur m'a donné :

<http://192.168.1.200:5000/sharing/De0bNRmQo>

Et je télécharger l'image :



Raspbian_Qt5_....img ^

Après je restaure l'image du disque :

Formater le disque...

Créer une image disque...

Restaurer l'image disque...

Test de performance du disque...

Données SMART et auto-tests...

Paramètres du disque...

Mettre en veille maintenant

Sortir de veille

Éteindre

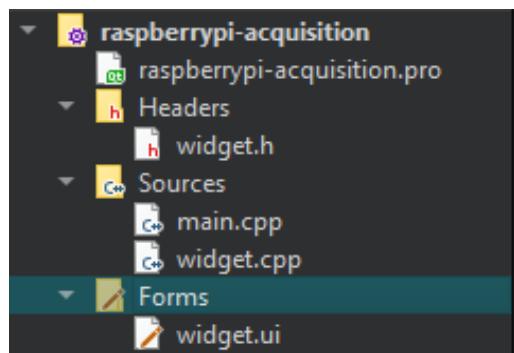
Le résultat du téléchargement :



Fin de téléchargement :

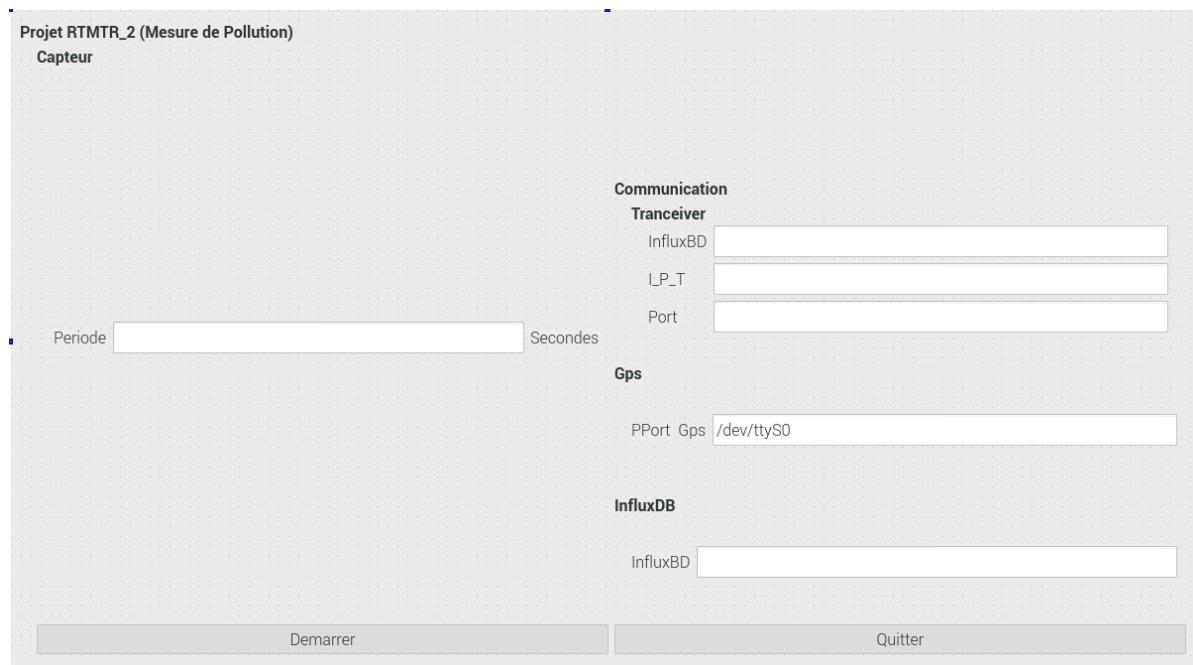


Je créer un projet raspberrypi-acquisition :



Je conçois une IHM(Interface Homme Machine) pour tester la classe **Capteur** et **InfluxDB** :

a) Réalisation de la classe Capteur



- **Periode** : Correspond aux temps d'affichage des valeurs du **Capteur**.

Extrait de code pour la classe **Capteur** :

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTimer>
#include <QDebug>
#include <influxdb.hpp>
#include "capteur.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

public slots:
    void releverCapteur();
    void on_bquit_clicked();

void Widget::on_bquit_clicked()
{
    this->close();
}
```

- this->close : permet de fermer l'IHM.

La commande suivante permet de voir les fichiers des disques :

```
pi@raspberrypi:~ $ df
Sys. de fichiers blocs de 1K Utilisé Disponible Utile Monté sur
/dev/root          30126764 10160616   18721416  36% /
devtmpfs            306372        0    306372   0% /dev
tmpfs              439300     7224    432076   2% /dev/shm
tmpfs              439300    11768    427532   3% /run
tmpfs                5120        4      5116   1% /run/lock
tmpfs              439300        0    439300   0% /sys/fs/cgroup
/dev/mmcblk0p1       258095    30415    227680  12% /boot
tmpfs              87860         4     87856   1% /run/user/1000
```

On recherche le **Grove Hat for Raspberry Pi** :

Je dois chercher les informations sur le **Grove Hat** dans la documentation :

https://wiki.seeedstudio.com/Grove-Base_Hat_for_Raspberry_Pi.html · Traduire cette page ::

Grove Base Hat for Raspberry Pi - Seeed Wiki

The **Grove Base Hat for Raspberry Pi** provide Digital/Analog/I2C/PWM/UART port to meet all your needs. With the help of build-in MCU, a 12-bit 8 channel ADC is ...

[Specification](#) · [Hardware Overview](#) · [Getting Started](#)

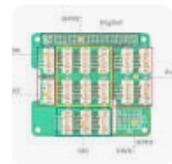
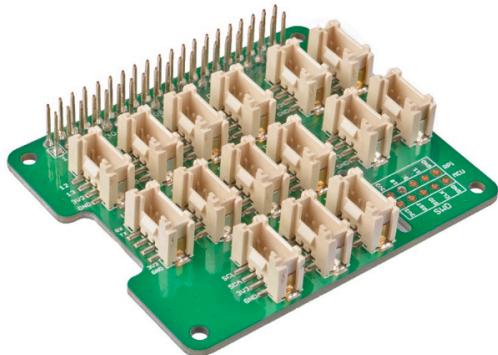


Image du **Grove Hat** :



Je vais sur la documentation pour le **ADS1115** pour configurer le **Raspberry Pi 3** avec le protocole **I2C** :

<https://learn.adafruit.com/ads1015-ads1115-i2c-lesson> · Traduire cette page ::

ADS1015 / ADS1115 | Raspberry Pi Analog to Digital ...

9 févr. 2016 — The ADS1015 and **ADS1115** are great analog to digital converters that are easy to use with the **Raspberry Pi** using its I2C communication bus.

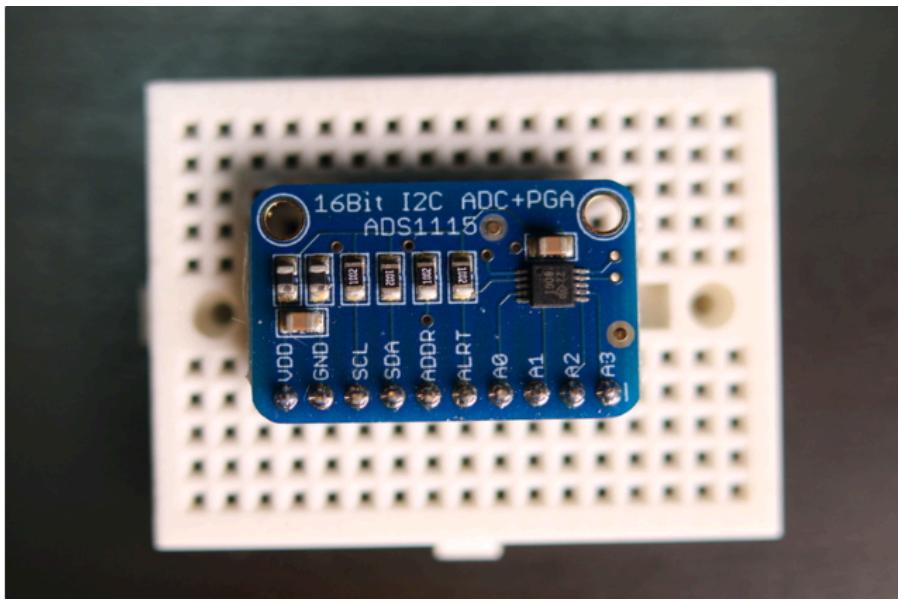
Lien pour l'**I2C** :

<https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c>

Lien de la documentation pour l'**ADS1115** :

<https://www.adafruit.com/datasheets/ads1115.pdf>

Image du **ADS1115** :



Je rentre les commandes pour installer les outils de l'**I2C** ci-dessous :

```
sudo apt-get install -y i2c-tools
```

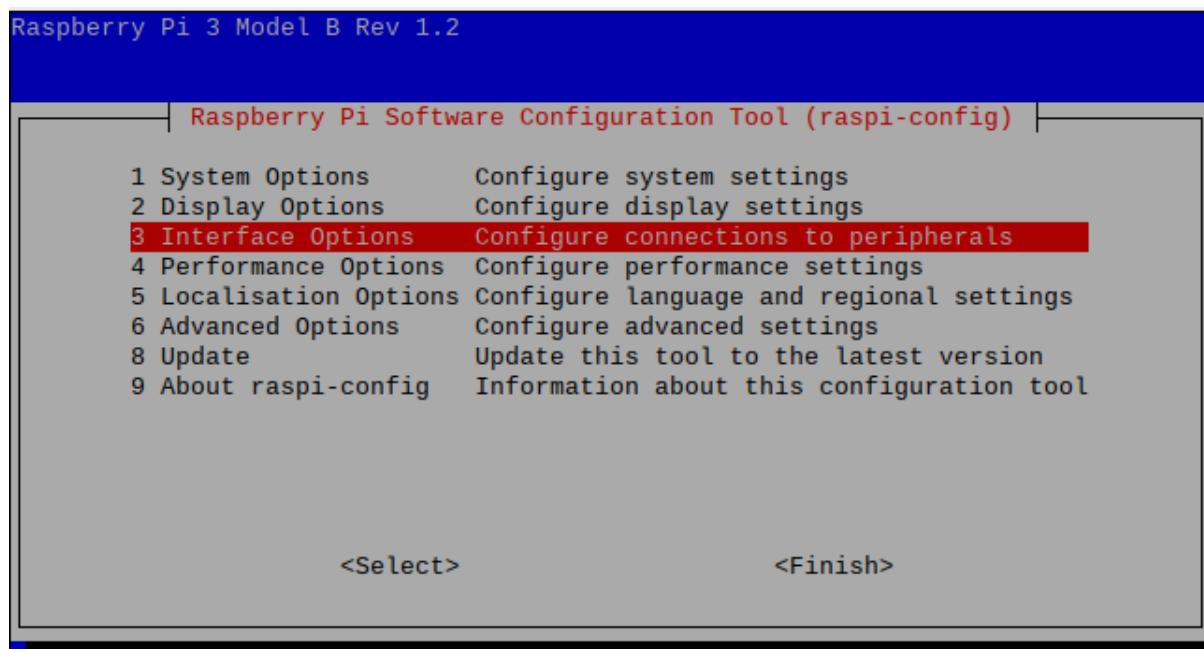
```
sudo apt-get install -y python-smbus
```

Je tape cette commande pour configurer le **Raspberry Pi 3** :

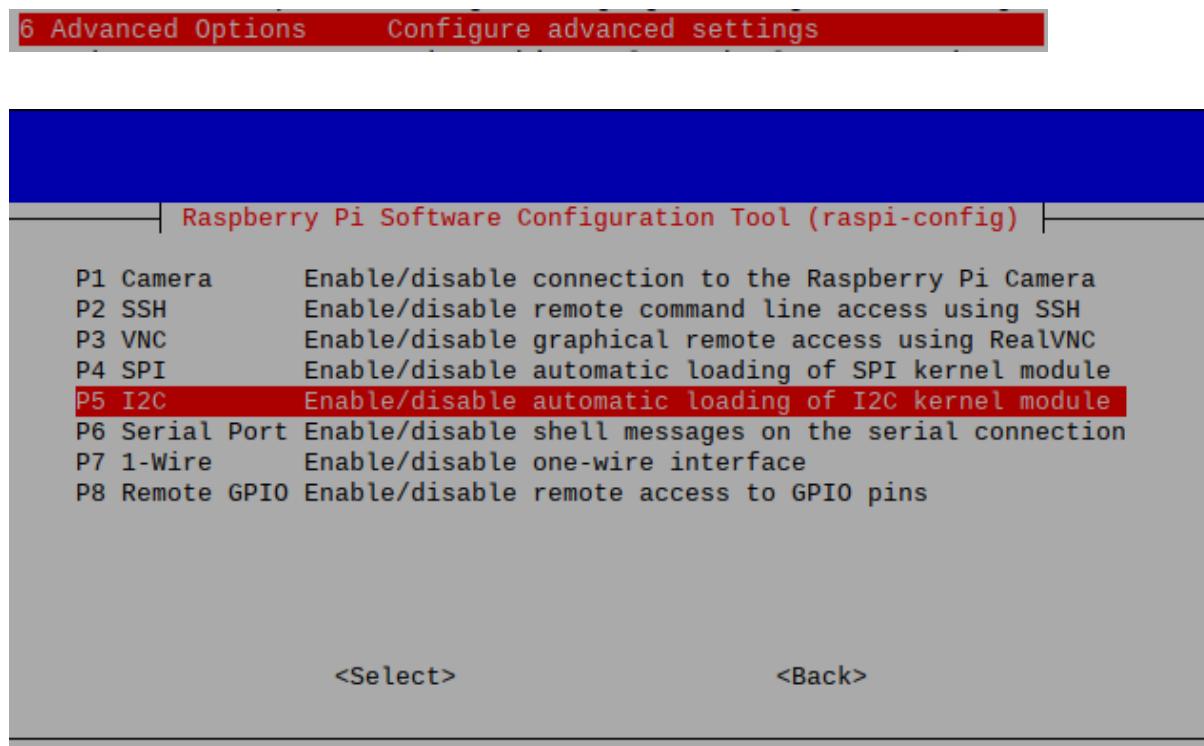
```
sudo raspi-config
```

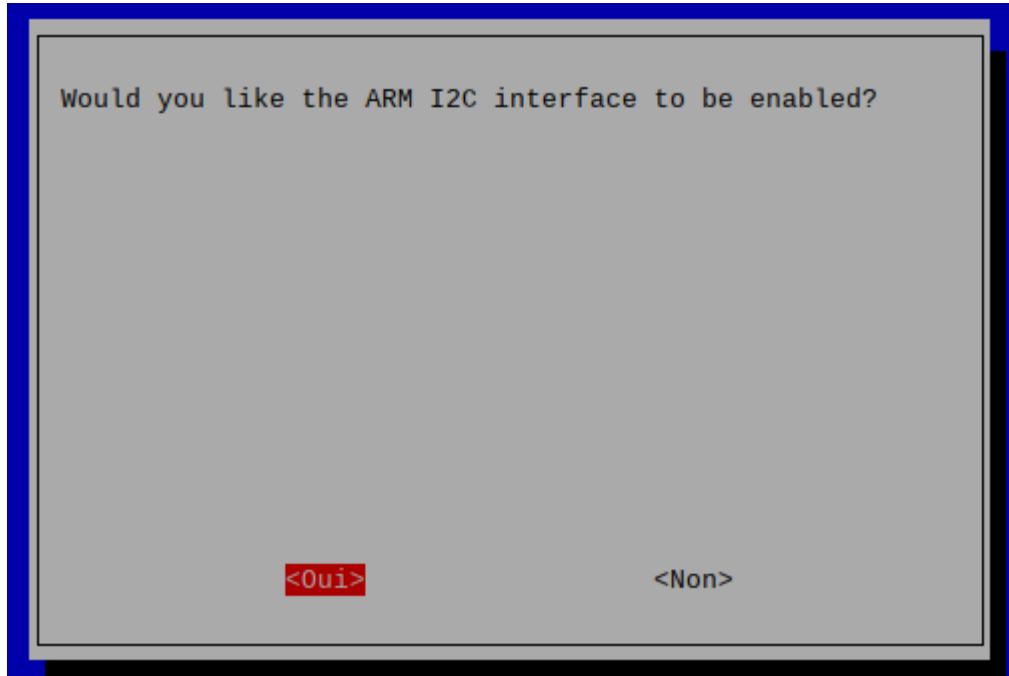
Après avoir entrer ces commandes j'ai une interface graphique qui apparaît :

Je sélectionne la section **Interface Options (Options de l'Interface)** :



En deuxième temps, j'appuie sur **Advanced Options (Options Avancées)** :





Puis, je redémarre le **Raspberry Pi 3** :

```
sudo reboot
```

Je teste les appareils connectés :

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- - - - - - - - - - - - - - - - - - - - - - -
10: - - - - - - - - - - - - - - - - - - - - - - -
20: - - - - - - - - - - - - - - - - - - - - - - -
30: - - - - - - - - - - - - - - - - - - - - - - -
40: - - - - - - - - - - - - - - - - - - - - - - -
50: - - - - - - - - - - - - - - - - - - - - - - -
60: - - - - - - - - - - - - - - - - - - - - - - -
70: - - - - - - - - - - - - - - - - - - - - - - -
```

Lien du site du **GPS** :

<https://wiki.seeedstudio.com/Grove-GPS/>

- **Step 1.** Follow [Setting Software](#) to configure the development environment.

- **Step 2.** Navigate to the demos' directory:

```
1 cd ~  
2 git clone https://github.com/DexterInd/GrovePi.git  
3 cd GrovePi/Software/Python/grove_gps
```

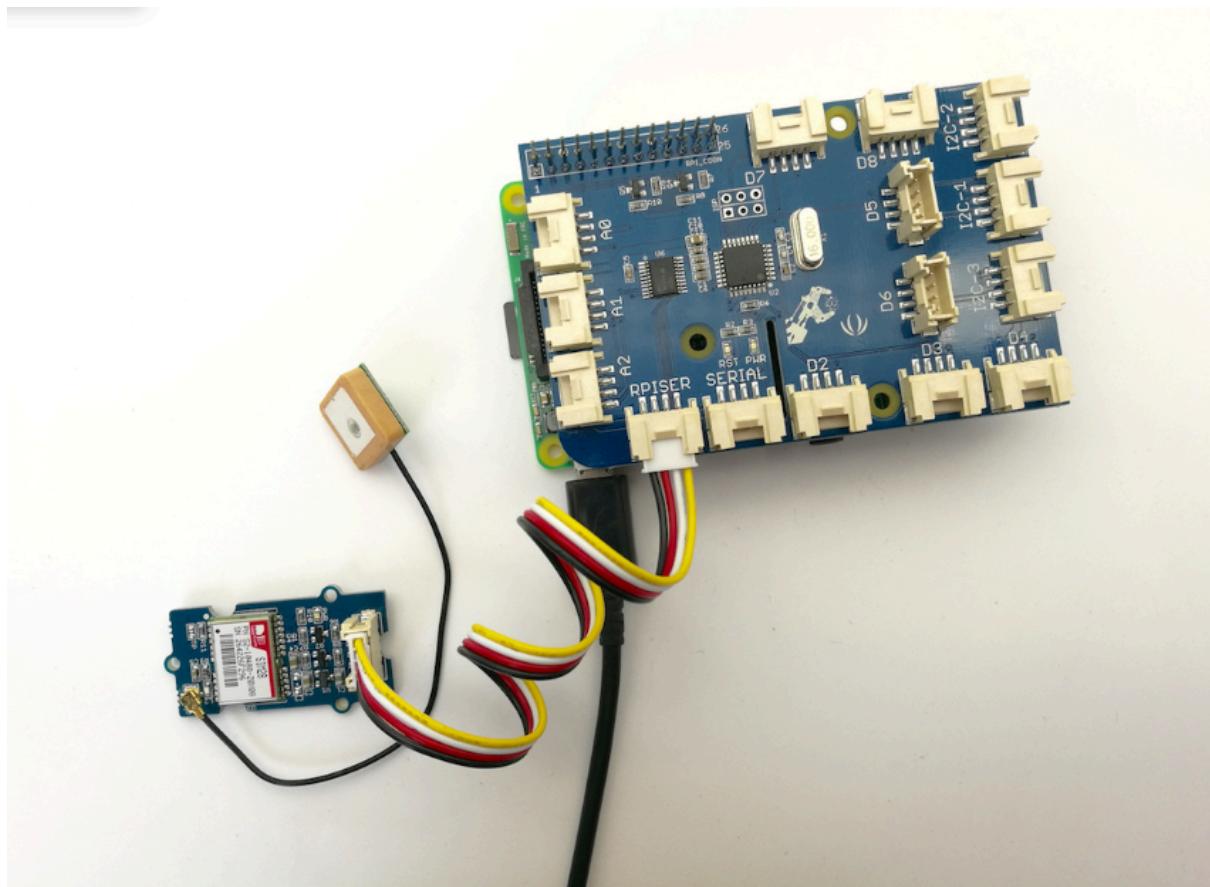
- **Step 3.** To see the code

```
nano grove_gps_data.py # "Ctrl+x" to exit #
```

- **Step 4.** Run the demo.

```
sudo python grove_gps_data.py
```

Image Capteur Grove GPS :



- **Step 3.** Open U-center.
- **Step 4.** Click Receiver -> Port and select the COM port that the Arduino is using.
- **Step 5.** Click Receiver -> Baudrate and make sure 9600 is selected.
- **Step 6.** Click View -> Text Console and you should get a window that will stream NMEA data.
- **Step 7.** Open the serial monitor, You can see as show below:

- **Step 1.** Install [u-center](#) software.
- **Step 2.** Copy the code into Arduino IDE and upload. If you do not know how to upload the code, please check [how to upload code](#).

- **Step 3.** Open U-center.
- **Step 4.** Click Receiver -> Port and select the COM port that the Arduino is using.
- **Step 5.** Click Receiver -> Baudrate and make sure 9600 is selected.
- **Step 6.** Click View -> Text Console and you should get a window that will stream NMEA data.
- **Step 7.** Open the serial monitor, You can see as show below:

Tableau des caractéristiques du **Grove GPS** :

Seeeduino	Grove - GPS
5V	Red
GND	Black
D3	White
D2	Yellow

Image **Grove Hat** et capteur **Grove GPS** :

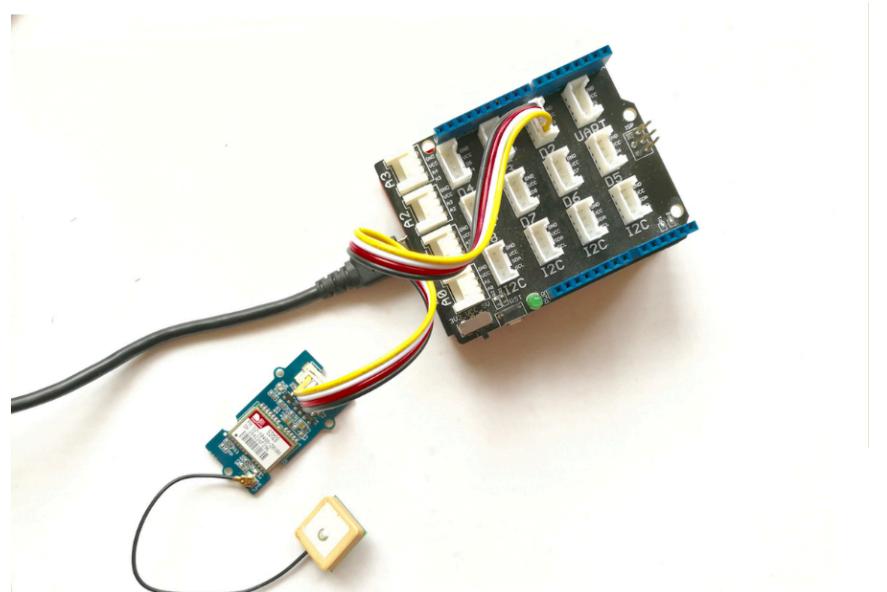
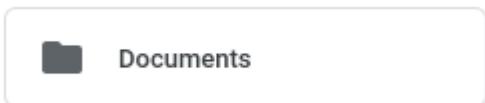


Image seul du **Grove Hat** :



Je branche le **Raspberry Pi 4** et je télécharge les fichiers requis pour le projet :

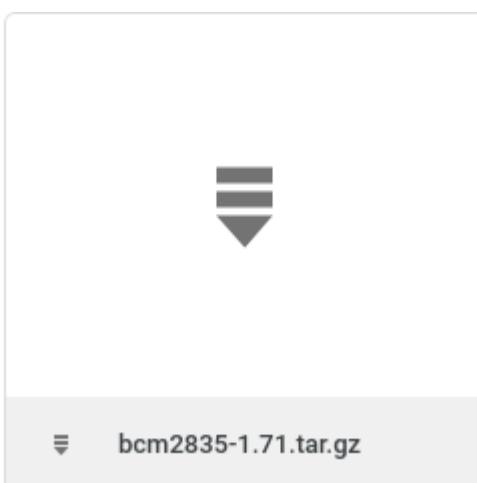
Dans le dossier **Documents** on trouve différents dossiers contenant des fichiers :



Je rentre ensuite dans **bI2c_BCM** pour télécharger le fichier :



Ensuite, le fichier :



Je clique sur le dossier **Codes** :



Et j'ai à nouveau plusieurs **dossiers** :

█ I2c_Test_grove_Hat

█ Classe_GroveBaseHat

█ InfluxDb_C++

█ influxdb-cxx-master

█ DexterAcquisition

et fichiers :



▀ groveAdcTest.tar.gz

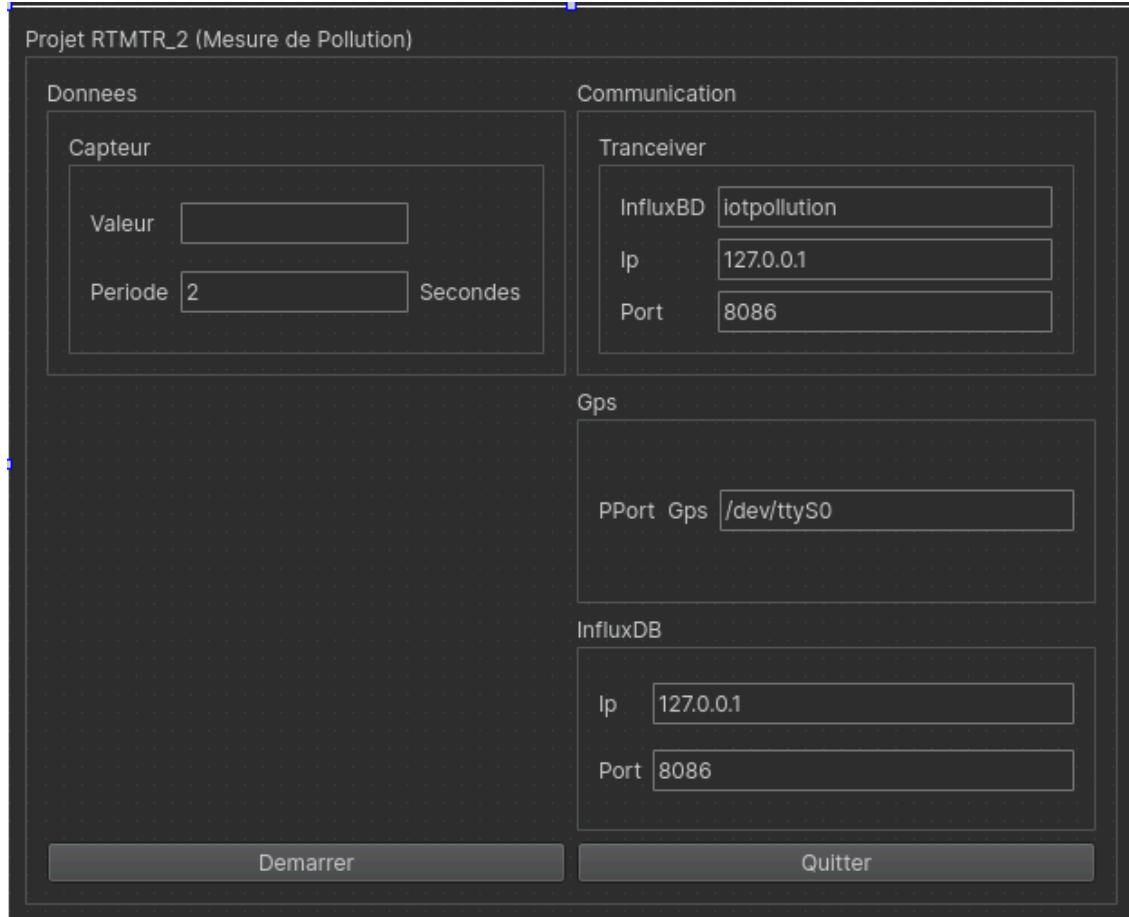


i2cTest.tar.gz



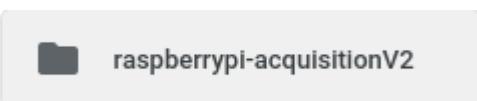
influxdb_Influx_V0.9_1.7.zip

Je modifie l'IHM en ajoutant une Valeur et un LineEdit (Champ de valeur) :



- **InfluxBD** : Correspond à l'adresse du **Serveur du Transceiver**.
- **Ip** : Correspond à l'adresse du **Transceiver**.
- **Port** : Correspond à la valeur du **Transceiver**.
- **Valeur** : Correspond à la valeur.
- **PPort Gps** : Correspond au répertoire du **Gps**.
- **InfluxBD** : Correspond à l'adresse du **Serveur**.

Je créais une copie classe qui porte le nom de **raspberrypi-acquisitionV2** :



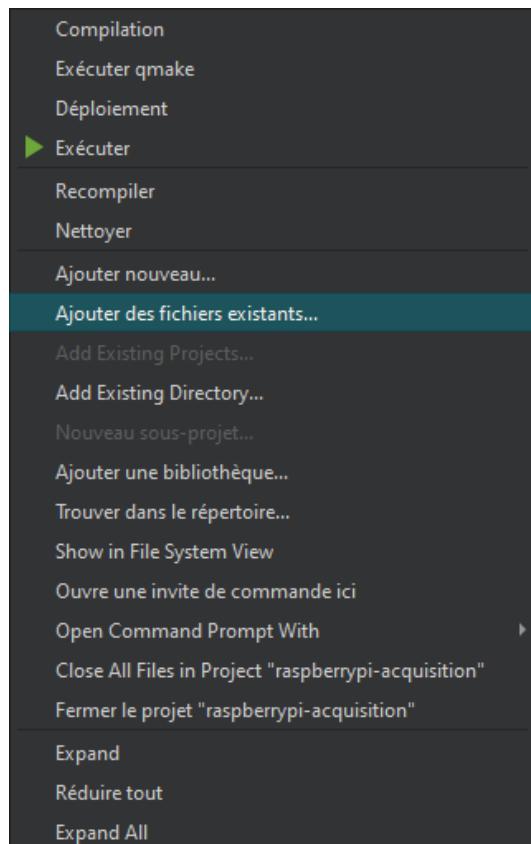
J'ajoute les lignes de code suivante pour implémenter l'objet **GrovePi** :

```
INCLUDEPATH += ./include
```

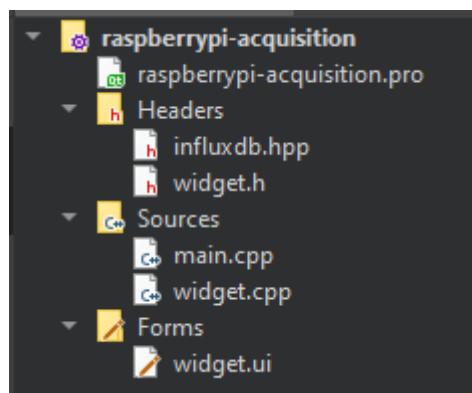
```
unix:!macx: LIBS += -L$$PWD/../../DexterAcquisition/lib/ -lgrovepicpp  
INCLUDEPATH += $$PWD/../../DexterAcquisition/include  
DEPENDPATH += $$PWD/../../DexterAcquisition/include
```

Je travaille sur cette copie :

J'ajoute un fichier **influxdb.hpp** :



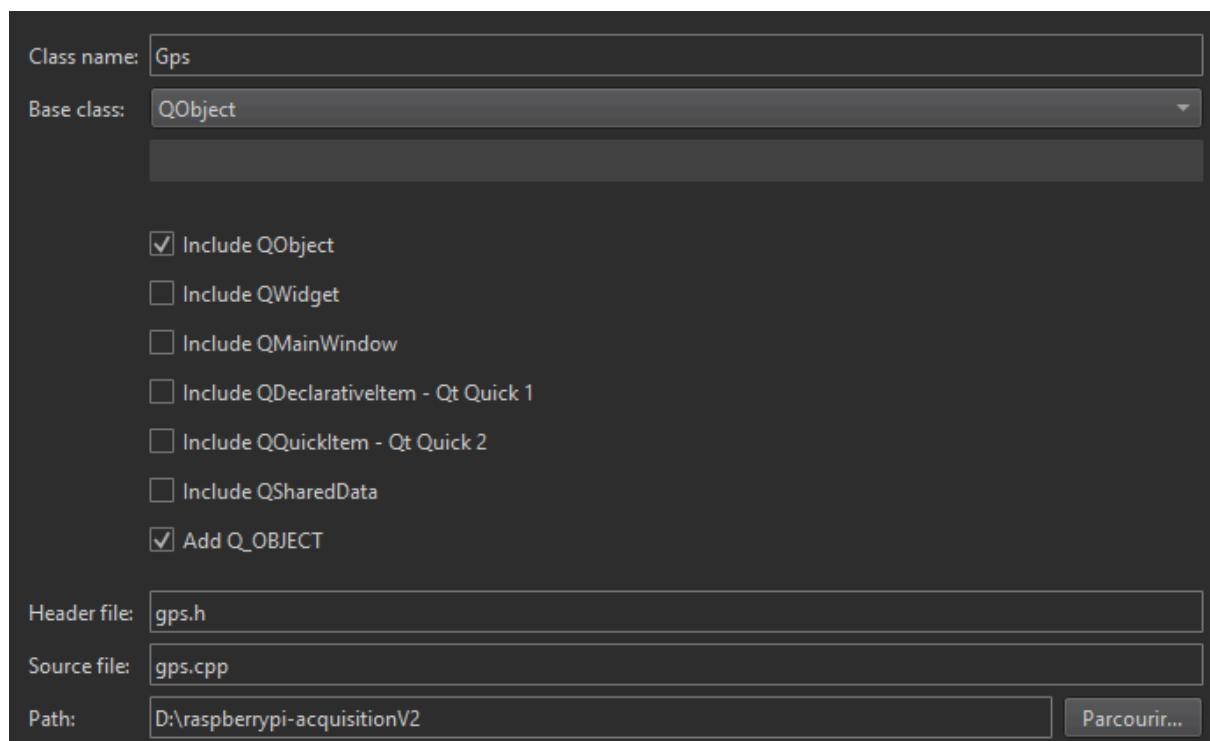
Le fichier ajouter :



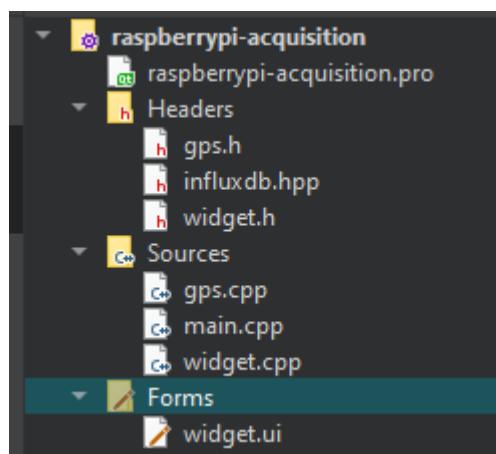
Langage utiliser pour faire l'objet :

C/C++

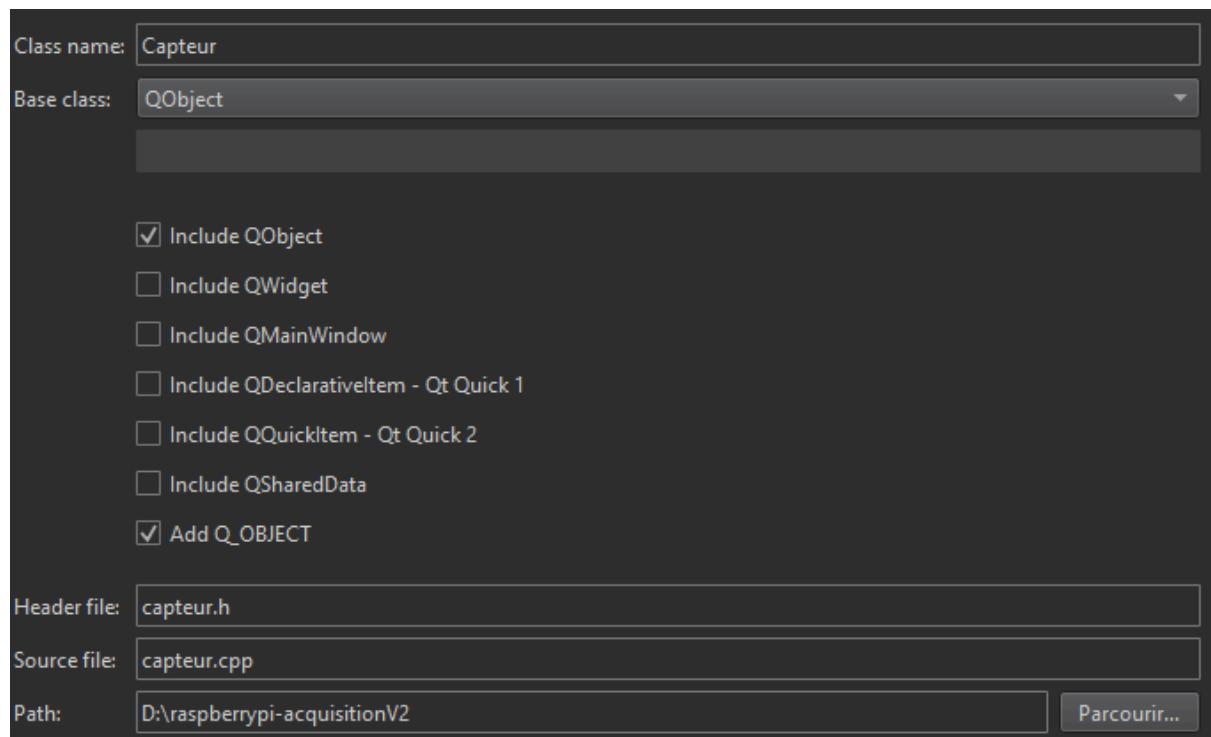
Puis, je rajoute une nouvelle classe pour l'objet **GPS** :



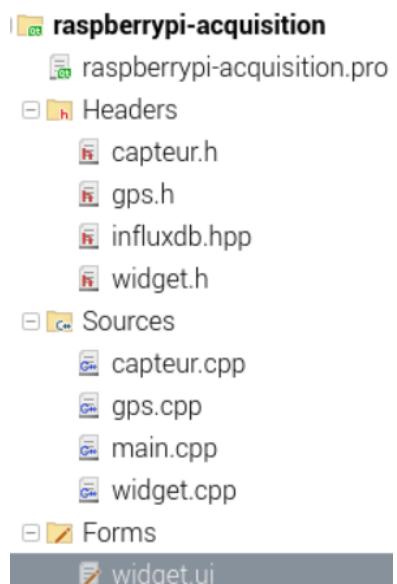
Le fichier ajouter au projet :



et ensuite l'objet **Capteur** :



Le fichier ainsi implémenter au projet :



Je rajoute des lignes de codes pour les classes précédemment créer :

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTimer>
#include <QDebug>
#include <influxdb.hpp>
#include "capteur.h"

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

public slots:
    void releverCapteur();
    void on_bquit_clicked();

private:
    Ui::Widget *ui;
    QTimer * timer;
    Capteur *capteur;
};

#endif // WIDGET_H
```

- **releverCapteur()** : Correspond au donnée du **Capteur**.

Dans la partie déclaration j'implémente les lignes de code pour configurer les valeurs dans la **Console** et mettre une limite au temps d'affichage :

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::Widget)
{
    ui->setupUi(this);
    timer = new QTimer(this);
    capteur = new Capteur;

    connect(timer, SIGNAL(timeout()), this, SLOT(releverCapteur()));
    timer->setInterval(500);
    ui->lperiod->setText("5");
    timer->start();
    qDebug() << "Acquisition";
    qDebug() << "Echantillonage";
}

void Widget::releverCapteur()
{
    ui->lval->setText(QString::number(capteur->Relever()));
}

... . . . . .
```

La partie **déclaration** du GPS :

```
#ifndef GPS_H
#define GPS_H

#include <QObject>

class Gps : public QObject
{
    Q_OBJECT
public:
    explicit Gps(QObject *parent = nullptr);

signals:

public slots:
};

#endif // GPS_H
```

Image de la ligne de code de la partie **déclaration** :

```
|     float Relever();
```

La partie **définition** du **GPS** :

```
#include "gps.h"

Gps::Gps(QObject *parent) : QObject(parent)
{
}
```

La partie déclaration du **Capteur** :

```
#ifndef CAPTEUR_H
#define CAPTEUR_H

#include <QObject>
#include <QDebug>

#include "grovepi.h"
using namespace GrovePi;

class Capteur : public QObject
{
    Q_OBJECT
public:
    explicit Capteur(QObject *parent = nullptr);
    float Relever();

signals:

public slots:

};

#endif // CAPTEUR_H
```

- **Relever()** : Permet de relever les données.

Dans la partie définition j'implémente les lignes de code pour afficher les valeurs dans la **Console** et mettre une limite au temps d'affichage :

```
#include "capteur.h"

Capteur::Capteur(QObject *parent) : QObject(parent)
{
}

float Capteur::Relever()
{
    qDebug() << "Relever";
    int pin = 0; // we use port A0
    int incoming; // variable to hold data for reading

    try
    {
        qDebug();
        initGrovePi(); //initialize communication w/ GrovePi

        // set the pin mode for reading
        pinMode(pin, INPUT);
        // continuously read data

        // read the data
        // receives a 10-bit value which maps to
        // 0V -> VCC, where VCC is the supply voltage of GrovePi
        incoming = analogRead(pin);
        // ui->lval->setText(QString::number(incoming));
        qDebug() <<"analog read = " << incoming;

        // wait 10 ms so we don't overflow the terminal
        qDebug() << "Relever3";
        return incoming;
    }
}
```

Permet de faire fonctionner le code si ce dernière dans la fonction **try** ne fonctionne pas :

```
    catch (I2CError &error)
    {
        printf(error.detail());
    }
}
```

Pour installer **InfluxDbv2** sur **RaspberryPi** voici les commandes ci dessous :

Image montrant la commande pour télécharger le fichier :

```
wget -qO- https://repos.influxdata.com/influxdb.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/influxdb.gpg > /dev/null
```

```
pi@raspberrypi:~ $ export DISTRIB_ID=$(lsb_release -si);
```

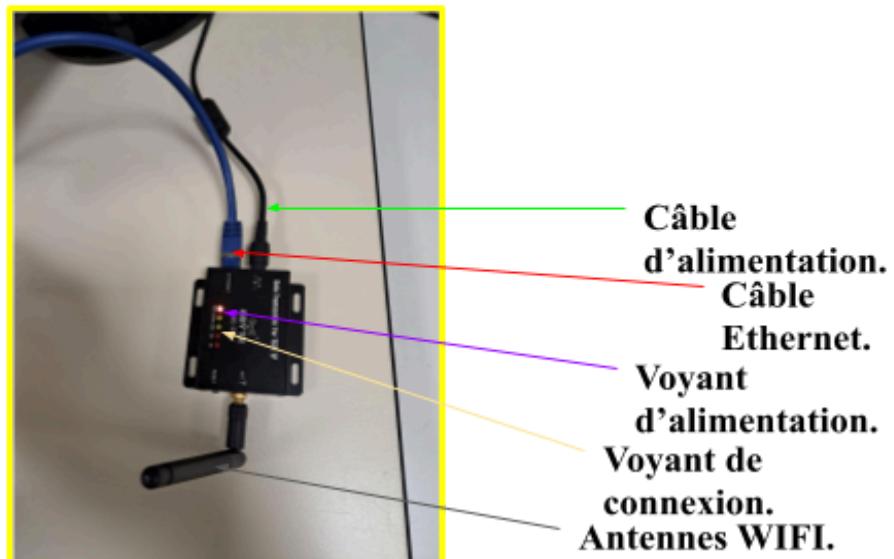
```
export DISTRIB_CODENAME=$(lsb_release -sc)
```

```
pi@raspberrypi:~ $ echo "deb [signed-by=/etc/apt/trusted.gpg.d/influxdb.gpg]
> https://repos.influxdata.com/${DISTRIB_ID},,
> ${DISTRIB_CODENAME} stable" | sudo tee
deb [signed-by=/etc/apt/trusted.gpg.d/influxdb.gpg]
https://repos.influxdata.com/
stable
```

Image montrant la commande pour installer et mettre à jour le **Serveur** :

```
pi@raspberrypi:~ $ sudo apt-get update && sudo apt-get install influxdb2
```

Image du TransceiverEthernetLora :



Par la suite j'ai rajouté des lignes de codes dans la partie **définition** et **déclaration** pour les classes **TransceiverEthernetLora**, **InfluxDB_Oss**, **Widget** :

Image de la partie **déclaration** de la classe **TransceiverEthernetLora** :

```
#ifndef TRANSCEIVERETHERNETLORA_H
#define TRANSCEIVERETHERNETLORA_H

#include <QObject>
#include <QDebug>

#include "grovepi.h"
using namespace GrovePi;
class TransceiverEthernetLora : public QObject
{
    Q_OBJECT
public:
    explicit TransceiverEthernetLora(QObject *parent = nullptr);
    float Connexion();
    float Expedier();

signals:

public slots:
};

#endif // TRANSCEIVERETHERNETLORA_H
```

Image de la partie **déclaration** de la classe **Widget** :

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QTimer>
#include <QDebug>
#include <capteur.h>
#include <gps.h>
#include <influxdb_oss.h>

#include <transceiverethernetlora.h>
#include <clienttcp.h>

QT_BEGIN_NAMESPACE
namespace Ui { class Widget; }
QT_END_NAMESPACE

#include "grovepi.h"
using namespace GrovePi;

class Widget : public QWidget
{
    Q_OBJECT

public:
    Widget(QWidget *parent = nullptr);
    ~Widget();

public slots:
    int relever();
    void on_bquit_clicked();

private:
    Ui::Widget *ui;
    QTimer * timer;
    Capteur * capteur;
    Gps * gps;
    InfluxDB_Oss * influxdb_oss;
    TransceiverEthernetLora * transceiverethernetlora;
    ClientTcp * client;
    float data;
};

#endif // WIDGET_H
```

Image de la partie **définition** de la classe **InfluxDB_Oss** :

```
#include "influxdb_oss.h"

InfluxDB_Oss::InfluxDB_Oss(QObject *parent) : QObject(parent)
{
}

void InfluxDB_Oss::Enregister(float data)
{
    influxdb_cpp::server_info si("127.0.0.1", 8086, "iotpollution", "RTMTR2", "pollution");
    // post_http demo with resp[optional]
    int ret = influxdb_cpp::builder()
        .meas("idStation1")
        .field("mesure", data)
        // .field("latitude",16)
        // .field("longitude", 20)
        // .timestamp(1512722735522840439)
        .post_http(si);
}
```

Image de la partie **déclaration** de la classe **InfluxDB_Oss** :

```
#ifndef INFLUXDB_OSS_H
#define INFLUXDB_OSS_H

#include <QObject>
#include <QDebug>

#include <influxdb.hpp>

#include "grovepi.h"
using namespace GrovePi;

class InfluxDB_Oss : public QObject
{
    Q_OBJECT
public:
    explicit InfluxDB_Oss(QObject *parent = nullptr);
    void Enregister(float data);

signals:

public slots:
};

#endif // INFLUXDB_OSS_H
```

Je change l'**interval de temps** et les valeurs de la **période** :

```
timer->setInterval(2000);
ui->lperiod->setText("20");
```

Je copie colle les lignes de code depuis le lien ci dessous :

Lien de partage du fichier **InfluxDB** depuis **GitHub** :

<https://github.com/orca-zhang/influxdb-cpp>

Je remplace le fichier influxdb.hpp par le même et je remplace le fichier main.cpp :

📁 raspberrypi-acquisition

- 📄 raspberrypi-acquisition.pro
- ⊖ 📁 Headers
 - 📝 capteur.h
 - 📝 gps.h
 - 📝 influxdb.hpp
 - 📝 influxdb_oss.h
 - 📝 transceiverethernetlora.h
 - 📝 widget.h
- ⊖ 📁 Sources
 - 📝 capteur.cpp
 - 📝 gps.cpp
 - 📝 influxdb_oss.cpp
 - 📝 main.cpp
 - 📝 transceiverethernetlora.cpp
 - 📝 widget.cpp
- ⊖ 📁 Forms
 - 📝 widget.ui

Je modifie les valeurs par défaut du **.meas**, **.field**, **influxdb_cpp::server_info** :

```
int main(int argc, char const *argv[])
{
    influxdb_cpp::server_info si("127.0.0.1", 8086, "iotpollution", "RTMTR2", "pollution");
    // post_http demo with resp[optional]
    string resp;
    int ret = influxdb_cpp::builder()
        .meas("idStation1")
        .field("mesure", 10)
        .field("latitude", 16 )
        .field("longitude", 20)
        // .timestamp(1512722735522840439)
        .post_http(si, &resp);

    cout << ret << endl << resp << endl;

    // create_db
    influxdb_cpp::create_db(resp, "x", si_new);
    cout << resp << endl;
    return 0;
}
```

Lien du serveur **InfluxDB_Oss** :

https://docs.influxdata.com/influxdb/v1.8/administration/authentication_and_authorization/#

Je rentre des commandes depuis le **RaspberryPi** ci-dessous :

```
> influx
```

```
create database iotpollution
```

```
> show measurements
```

```
show databases
```

Image des noms des **measurements** :

```
> show measurements
name: measurements
name
-----
idStation1
```

```
name: measurements
name
-----
idStation1
latitude
longitude
mesure
```

```
> Grant all to RTMTR2
```

```
> CREATE USER "RTMTR2" WITH PASSWORD 'pollution'
```

```
pi@raspberrypi:~ $ influx -username RTMTR2 -password pollution
Connected to http://localhost:8086 version 1.6.4
InfluxDB shell version: 1.6.4
```

Images montrant les **utilisateurs** et les droit d'accès **administrateur** :

```
> show users
user admin
-----
```

```
user    admin
-----  -----
RTMTR2 false
```

```
> drop database iotpollution
```

- Influx : permet de se connecter à la base de données.
- show measurements : Montre les mesures (**mesure**, **latitude**, **longitude**).
- create database **iotpollution** : permet de créer une base de données locale.
- show databases : permet de montrer la liste de données dans la base de données.
- drop measurement : permet de supprimer les mesures.
- drop database **iotpollution** : permet de supprimer la base de donnée locale.
- Grant all to **RTMTR2** : permet d'ajouter des droits **administrateurs**.
- CREATE USER “**RTMTR2**” WITH PASSWORD ‘**pollution**’ : permet de créer un identifiant et un mot de passe.

- influx -username **RTMTR2** -password **pollution** : permet de se connecter à la base de données avec l'identifiant et le mot de passe.
- show users : permet de montrer les utilisateurs.

Les adresses pour associer à la base de données locale sont :

- 127.0.0.1:8086/**idStation1 mesure**.
- 127.0.0.1:8086/**idStation1 latitude**.
- 127.0.0.1:8086/**idStation1 longitude**.

Je modifier le code dans la **déclaration** et **définition** de widget :

J'ajoute dans la **déclaration** d' **influxdb_oss** du code :

Je prends ce qui était dans la **définition** de widget et je le mets dans la **définition** d' **influxdb_oss** :

Je **compile** le code pour voir si il est bon :

Je supprime la base de données **iotpollution** pour la recréer à nouveau et pour voir les valeurs du **Capteur** s'affiche.

je remplace le code de la **définition** du main :

J'exécute le code pour si l'**IHM** apparaît :

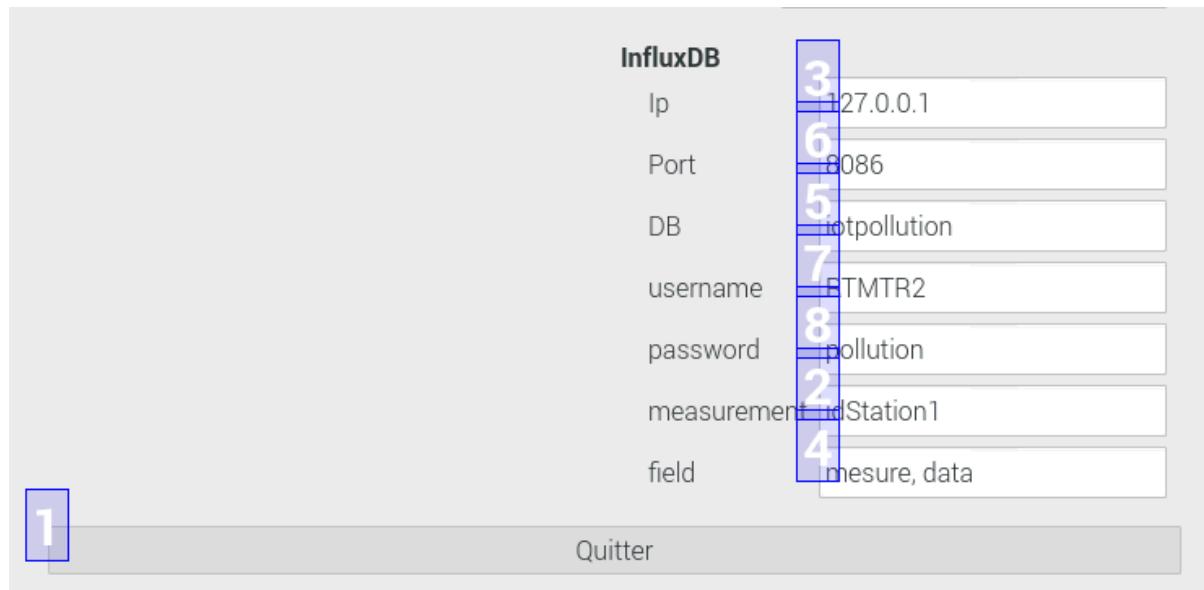
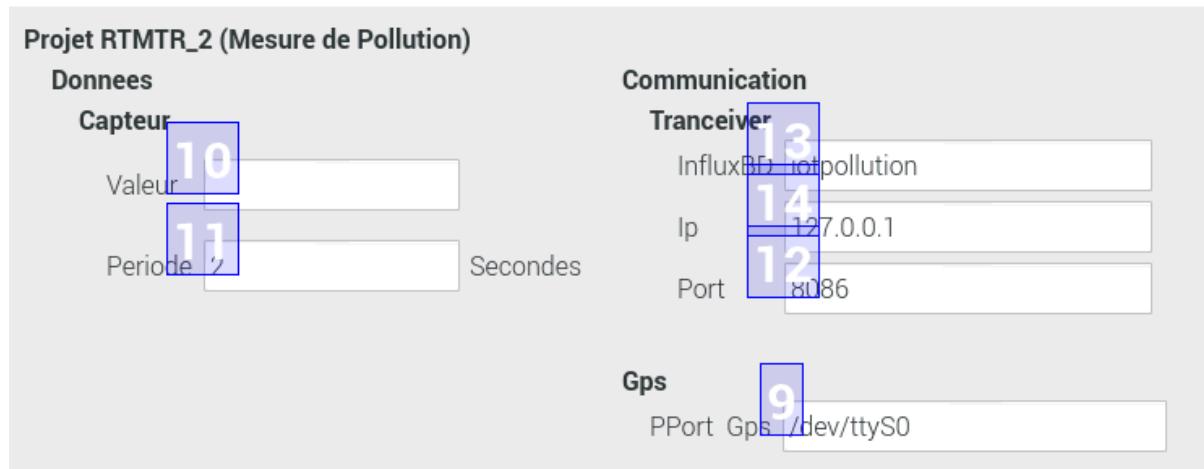
Ensuite, j'ajoute des fichiers et un dossier supplémentaire pour que le **TransceiverEthernetLora** puisse expédier les données :

Je rajoute encore du code dans le main.

Je modifie le **setInterval** pour que les données apparaissent moins vite et que les trois **digits** apparaissent dans le **serverTCP** :

```
timer->setInterval(4000);
ui->lperiod->setText("4");
```

Je rajoute d'autres parties dans l'**IHM** pour valider la classe **GPS** :



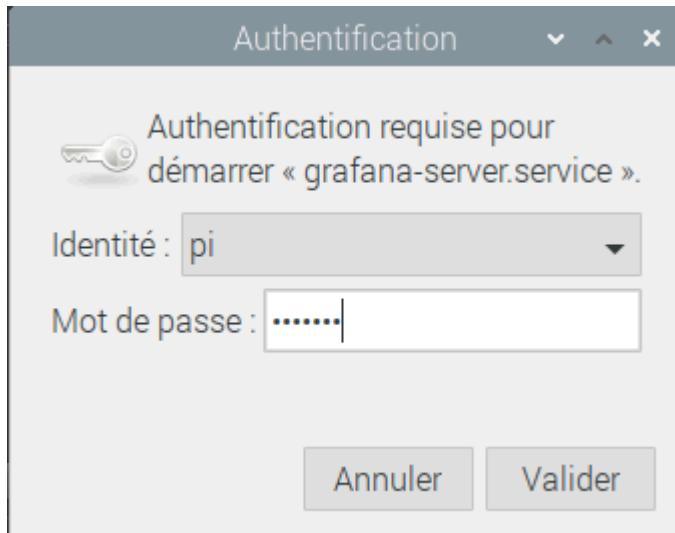
- Fermer l'application.
- Nom du **measurement**.
- Adresse locale du **Serveur**.
- Nom du **field**.
- Nom de la base de données.
- Numéro de port.
- Identifiant de la base de données.

- Mot de passe de la base de données.
- Adresse du **GPS**.
- Valeur du **Capteur**.
- Valeur de la **Période**.
- Numéro de port du **ClientTCP**.
- Nom de la base de données.
- Adresse du **ClientTCP**.

Je rentre la commande ci-dessous pour lancer **Grafana** :

```
systemctl start grafana-server
```

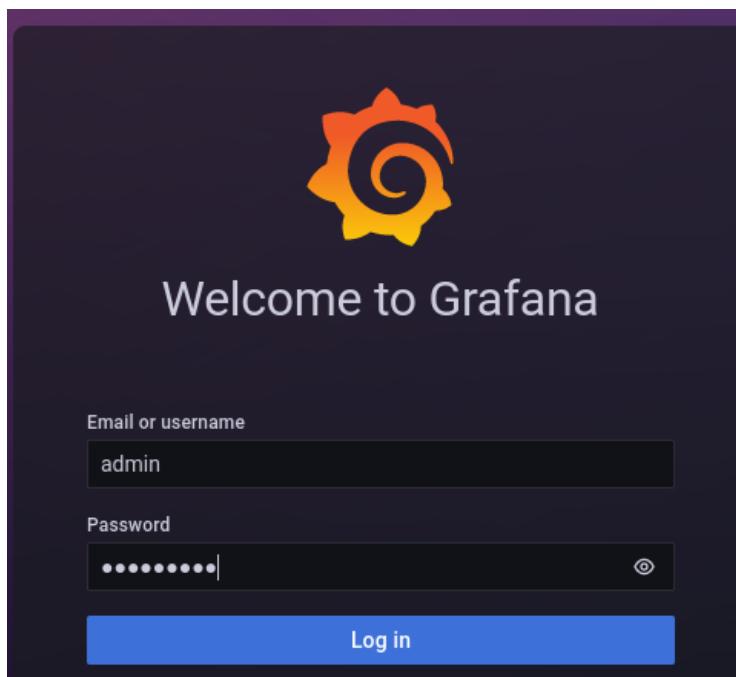
Je rentre mes **Identifiants** :



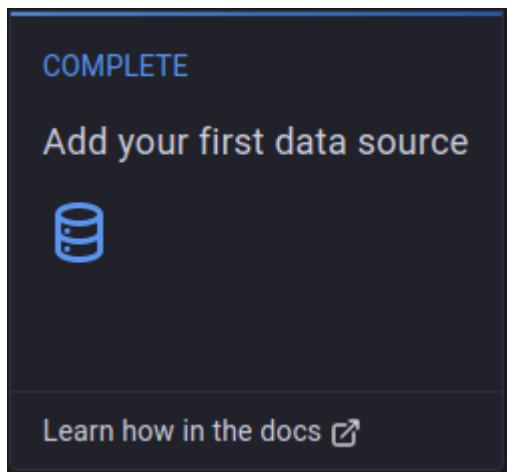
J'accède à l'interface de **Grafana** à l'aide de l'adresse ci-dessous :

<http://192.168.1.185:3000/login>

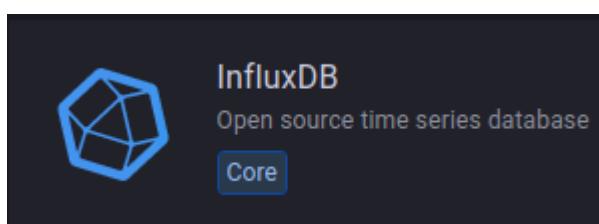
Je rentre mes **Identifiants** pour me connecter à **Grafana** :



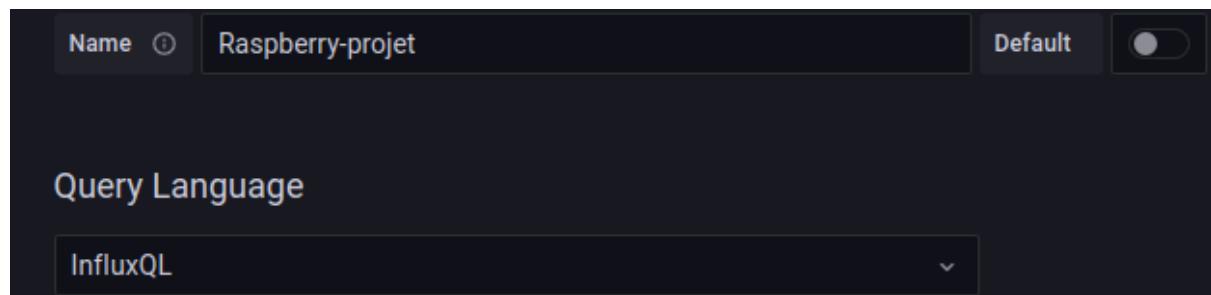
J'ajoute une base de données :



Je sélectionne la base de données :



Je rentre les informations ci-dessous :



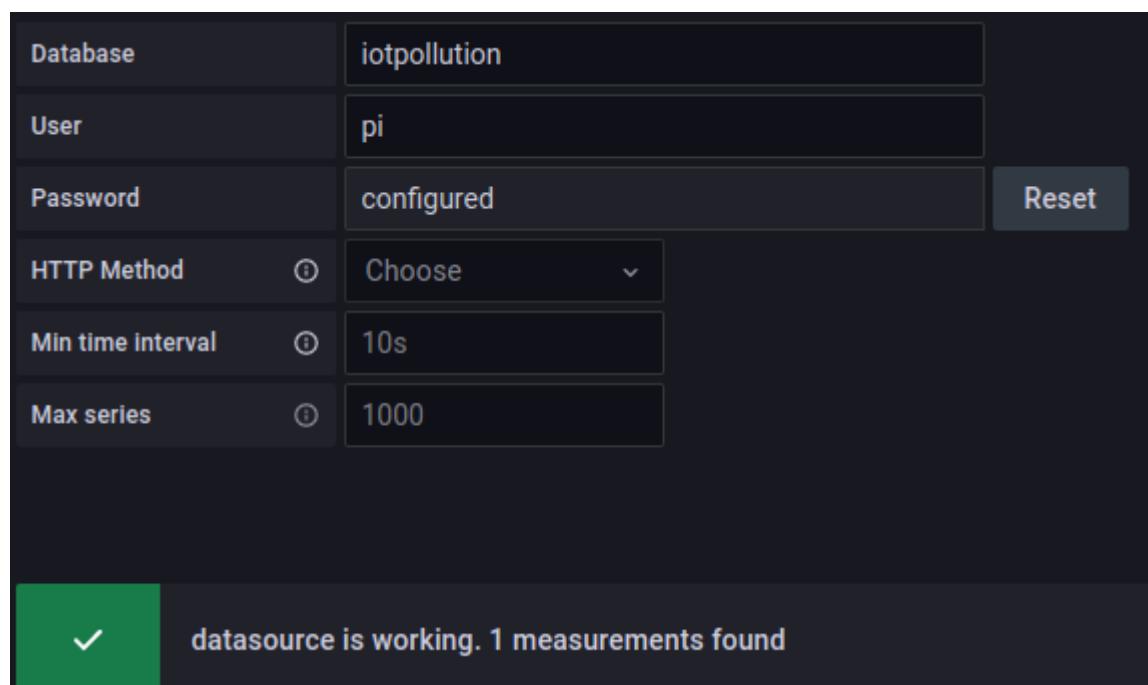
- **Name** : Correspond au nom du projet.
- **Query Language** : correspond au type de langue de programmation qu'on souhaite utiliser.

Ensuite, l'adresse ci-dessous :



- **URL** : Correspond à l'adresse d'**InfluxDB_Oss**.

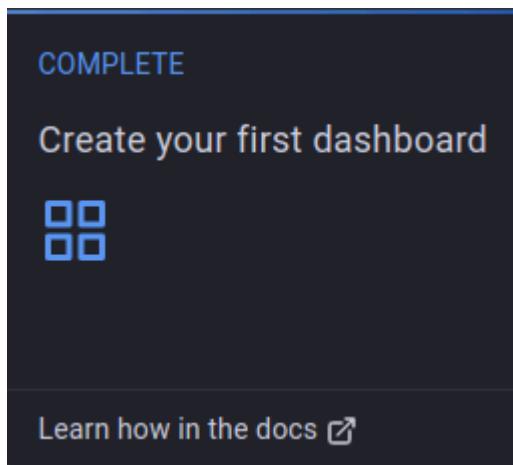
Je configure les informations pour la base de données :



- **Database** : Correspond au nom de la base de données.
- **User** : Correspond au nom de l'utilisateur.
- **Password** : Correspond au mot de passe.

- **HTTP Method** : Correspond au permet de choisir différentes méthodes.
- **Min time interval** : permet de rentrer un interval minimal pour chaque valeurs relevées.
- **Max series** correspond à la limites de **Tables**.

Je créer un Tableau de Bord(**Dashboard**) :



Je remplie les zones ci-dessous :

The image shows the Grafana query editor for the "iotpollution" database. The query is as follows:

```

    FROM default WHERE
    SELECT field(mesure) mean() +
    field(latitude) mean() +
    field(longitude) mean()
  
```

- **iotpollution** : Correspond au nom de la base de données.
- **FROM** : Correspond à la **Table**.
- **SELECT** : permet de sélectionner un **Champ**.

Je sélectionne les valeurs ci-dessous :

The image shows a configuration dialog for the "Min" value. It has a label "Min" and a note "Leave empty to calculate based on all values". Below is a text input field containing the value "0".

- **Min** : Rentrer une valeur minimale.

```
Max  
Leave empty to calculate based on all values  
786
```

- **Max :** Rentrer une valeur maximale.

Commande pour avoir la représentation du code du **GPS** :

```
sudo cutecom
```

Commande pour télécharger le paquet **Serialport** version développeur :

```
pi@raspberrypi:~ $ sudo synaptic
```

Je recherche le paquet sur le gestionnaire :



Ensuite, je télécharge le paquet :

```
[ ] libqt5serialport5-dev 5.11.3-2 5.11.3-2 Qt 5 serial port development files
```

Commande pour reconnaître le code du **Grove-GPS** :

```
QT      += core gui serialport
```

Je rentre la commande ci-dessous pour trouver le bon **Device** :

```
pi@raspberrypi:~ $ dmesg
```

Voici les différentes descriptions sur le **GPS** :

```
usb 1-1.4: Product: u-blox GNSS receiver
usb 1-1.4: Manufacturer: u-blox AG - www.u-blox.com
cdc_acm 1-1.4:1.0: ttyACM0: USB ACM device
```

Je sélectionne la bonne section :

Je peux fermer lorsque l'exécution est finie :

Les différentes trames dans **CuteCom** :

```
[12:14:07:383] $GNGGA,,,,,0,00,99.99,,,,,*56
[12:14:07:383] $GNGSA,A,1,,,,,,99.99,99.99,99.99*2E
[12:14:07:383] $GNGSA,A,1,,,,,,99.99,99.99,99.99*2E
```

Extrait de la **déclaration** du Grove-GPS :

```
#ifndef GPS_H
#define GPS_H

#include <QSerialPort>
#include <QSerialPortInfo>
#include <QIODevice>
#include <QDebug>
#include <QString>

class GPS : public QSerialPort
{
    Q_OBJECT
private:
    QList<QByteArray> spltGpsData_;
    QList<QByteArray> spltGNGGA_;
    double lat_;
    double long_;
public:
    GPS(QString device);

public slots:
    void Relever();
signals:
    void EmettrePosition(QString latitude, QString longitude);
};

#endif // GPS_H
```

Extrait de la **définition** du Grove-GPS :

```
#include "gps.h"

GPS::GPS(QString device)
{
    setPortName(device);
    setBaudRate(QSerialPort::Baud9600);
    setParity(QSerialPort::NoParity);
    setDataBits(QSerialPort::Data8);
    setStopBits(QSerialPort::OneStop);
    open(QIODevice::ReadOnly);

    connect(this,SIGNAL(readyRead()),this, SLOT(Relever()));
}

void GPS::Relever()
{
    QByteArray data;
    data = readAll();
    if (data.indexOf("$GNGGA") >= 0)
    {

        spltGpsData_ = data.split('\n');
        for (int i = 0; i < spltGpsData_.count(); ++i)
        {
            if (spltGpsData_[i].indexOf("$GNGGA") >= 0)
            {
                qDebug() << "-- Ligne de QString concernée = " << i;
                spltGNGGA_ = spltGpsData_[i].split(',');
                lat_ = spltGNGGA_[2].toDouble();
                long_ = spltGNGGA_[4].toDouble();

                emit EmettrePosition(QString::number(lat_),QString::number(-long_));
                qDebug() << data.toStdString().c_str();

            }
        }
    }
}
```

Extrait de la **déclaration** du Widget :

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QtDebug>
#include <QString>
#include <gps.h>
#include <QList>
//#include <influxclientv1.h>
//#include <influxclientv2.h>

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();

private:
    Ui::Widget *ui;
    double lat_;
    double long_;
    // InfluxClientV1 client1;
    // InfluxClientV2 client2;
    GPS *gps;
    QList<QByteArray> splt_;

private slots:
    void recevoirPosition(QString latitude, QString longitude);

    void on_bstop_clicked();
    void on_bclear_clicked();

};
```

Extrait de la **définition** du Widget :

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

    gps = new GPS(ui->ldevice->text());
    connect(gps,SIGNAL(EmettrePosition(QString, QString )),this, SLOT(recevoirPosition(QString, QString )));
}

Widget::~Widget()
{
    delete ui;
    gps->close();
}

void Widget::recevoirPosition(QString latitude, QString longitude)
{
    //client1.Enregistrer(latitude, longitude);
    //client2.Enregistrer(latitude.toDouble(), longitude.toDouble());
    qDebug() <<"Pour ui" << latitude;
    qDebug() <<"Pour ui" << longitude;

    ui->textlat->setText(latitude);
    ui->textlong->setText(longitude);
}

void Widget::on_bstop_clicked()
{
    gps->close();
}

void Widget::on_bclear_clicked()
{
    ui->textlat->clear();
    ui->textlong->clear();
}
```

Image du **Grove-GPS** :



La fiche qui se connecte à la sortie **USB** du **Raspberry Pi**.

Je m'occupe par la suite de l'**ECRAN_LCD**.

Image du **Ecran_LCD** :



Arrière de l'Ecran LCD :

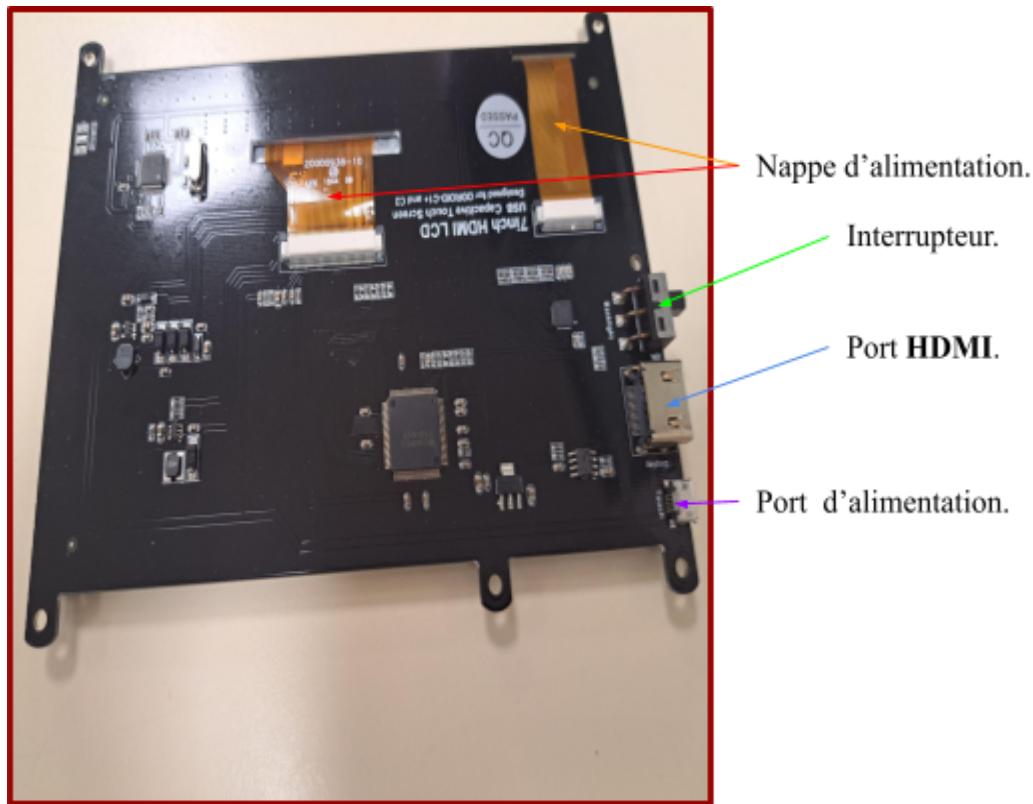
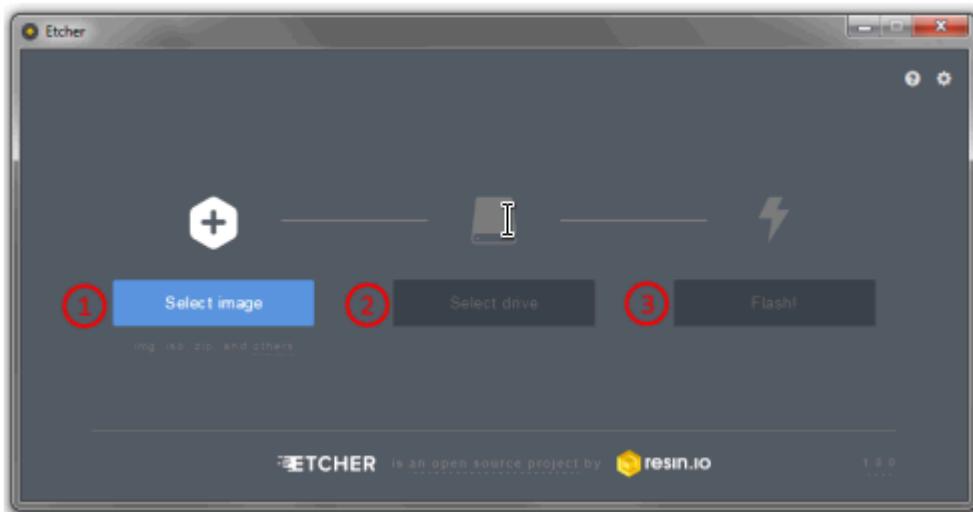


Image du logiciel **Etcher** pour transférer l'image système dans la **Carte SD** :



Site **Joy-it** :

<https://joy-it.net/en/?goods=7-hdmi-lcd-v7-2>

Commande pour configurer l'écran :

```
sudo nano /boot/config.txt
```

```
dtparam=audio=on
max_usb_current=1
hdmi_force_hotplug=1
config_hdmi_boost=7
hdmi_group=2
hdmi_mode=1
hdmi_mode=87
hdmi_drive=1
display_rotate=0
hdmi_cvt 1024 600 60 6 0 0 0
```

Résultat :

Image montrant les données du **champ longitude** de la **table idStation1** :

```
> select longitude from idStation1
name: idStation1
time          longitude
----          -----
1678820211871003981 20
1678820249513822032 20
1678820766206204931 20
1678820768823529367 20
```

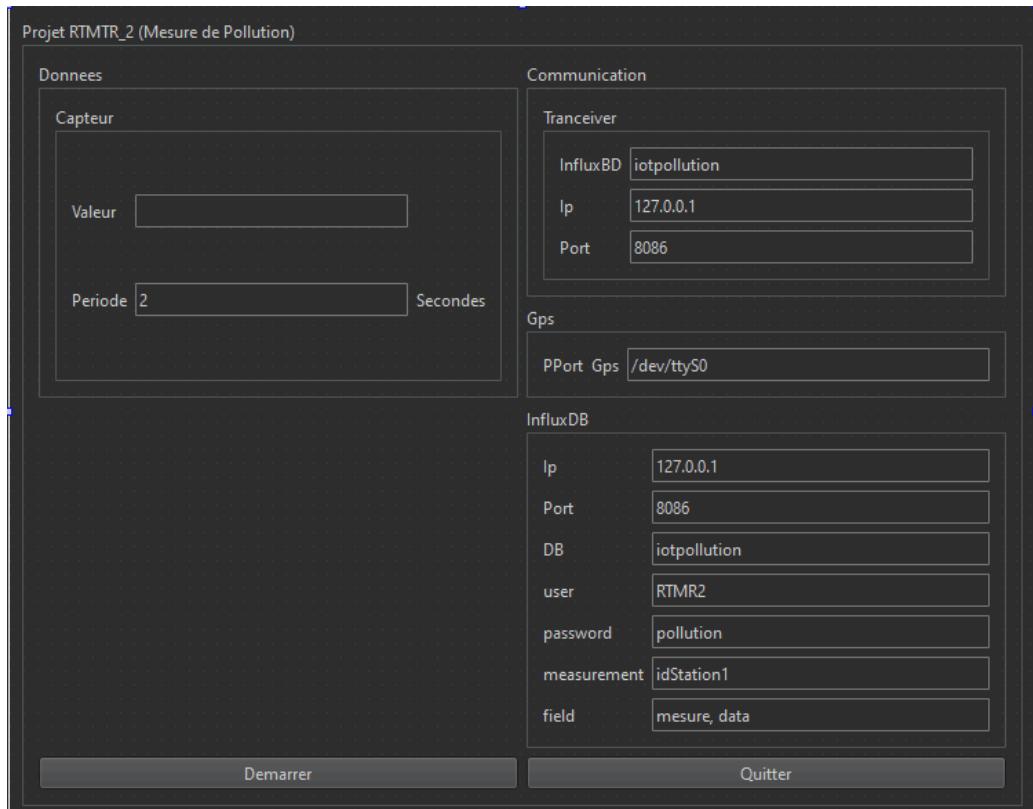
Image montrant les données du **champ latitude** de la **table idStation1** :

```
> select latitude from idStation1
name: idStation1
time          latitude
----          -----
1678820211871003981 16
1678820249513822032 16
1678820766206204931 16
1678820768823529367 16
```

Image montrant les données des **champs** de la table **idStation1** :

```
* select * from idStation1  
name: 'idStation1'  
time      latitude longitude mesure  
-----  
1678820211871003981 16    20     10  
1678820249513822032 16    20     10  
1678820766206204931 16    20     10  
1678820768823529367 16    20     10  
1678821931953251594 16    20     10  
1678823434164842564 16    20     10  
1678823438370165169 16    20     10  
1678823452586232039 16    20     10  
1678823485239354965 16    20     10  
1678823504995077204 16    20     10  
1678823523284426095 16    20     10  
1678823557587974369 16    20     10  
1678823567217584904 16    20     10
```

Image finale de l'IHM :



Envoie des résultats via le **Transceiver** :

```
qt5ct: using qt5ct plugin
-- Succès
-- ClientTCP Connexion
```

```
Relever
analog read =  30
Relever3
Relever
analog read =  30
```

Envoie des résultats via le **GPS** :

```
-- Ligne de QString concernée =  0
Pour ui "0"
Pour ui "0"
$GNGGA,,,,,,0,00,99.99,,,,*56
$GNGSA,A,1,,,,,,99.99,99.99,99.99*2E
$GNGSA,A,1,,,,,,99.99,99.99,99.99*2E
```

Image des données des mesures relevées via **InfluxDB_Oss** sur **Chronographe** :

