

Revue de Projet commune : Interface domotique pour Piscine

Théo Boucaud, Lucas Authier, Léo Barao

BTS Systèmes Numériques Option Informatique et Réseau

Période de projet 30 Janvier 2024 au 31 Mai 2024

Partie commune

Présentation du système :	3
Description du système:	4
Répartition des tâches et rôles du groupe:	6
Objectifs du projet	7
UML	8
Matériel utilisé	10
Diagramme de Gantt prévisionnels	12

Présentation du système :

Une piscine doit conserver un équilibre chimique particulier, sans quoi l'eau sera moins pure, irritante, colorée, le matériel se détériore.

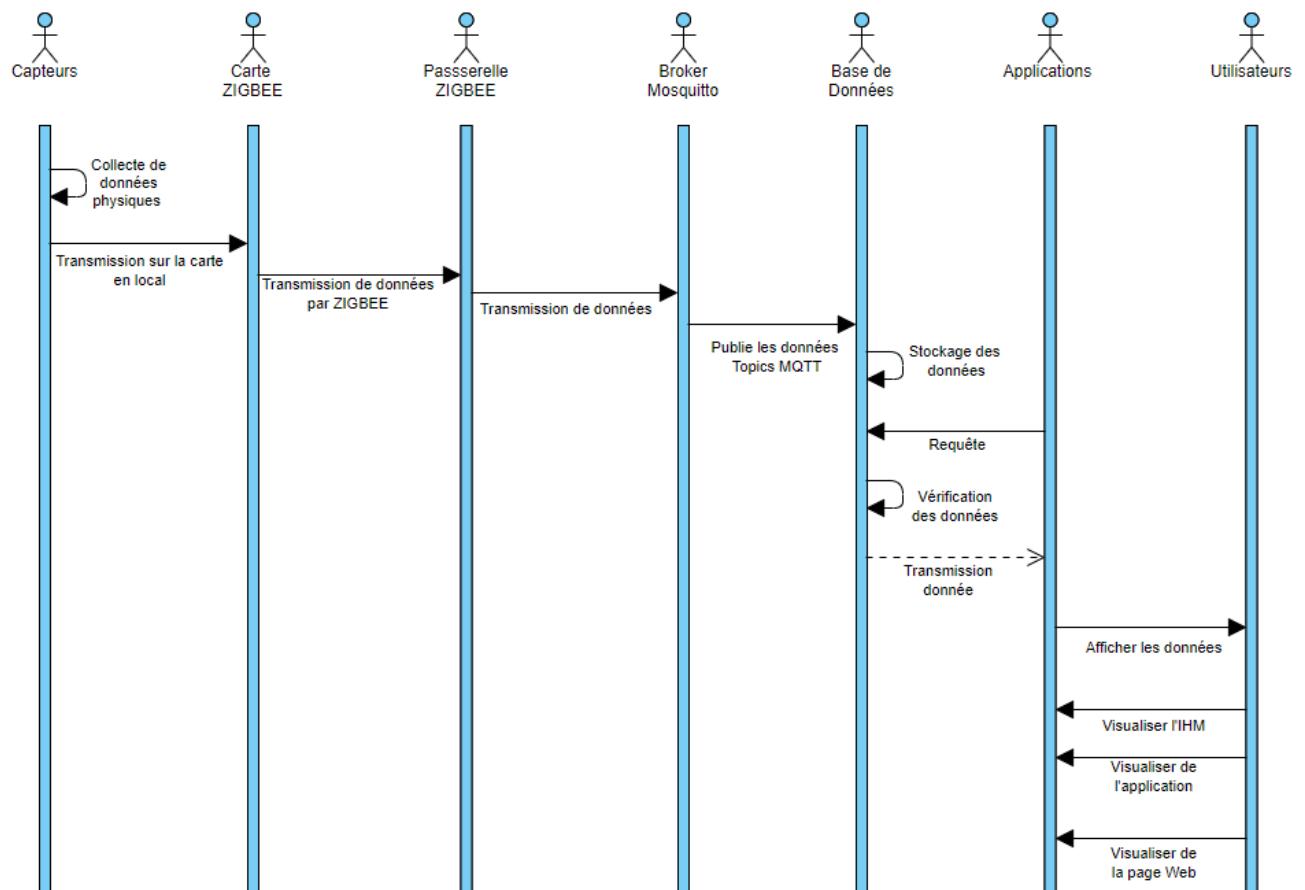
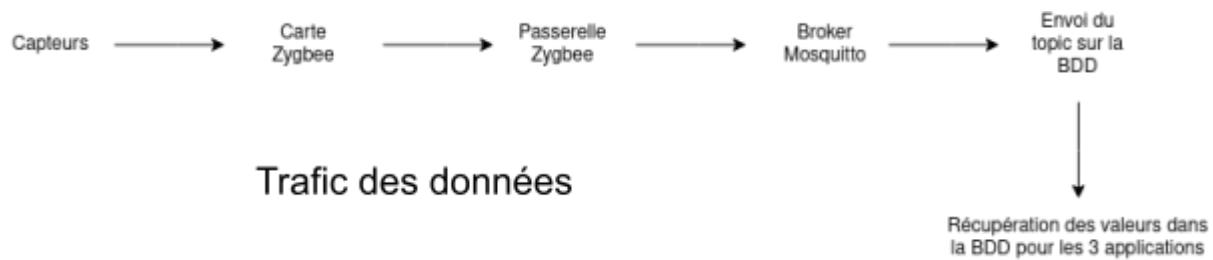
L'analyse de l'eau est une étape indispensable dans l'entretien d'une piscine. Il s'agit d'équilibrer son pH, mais aussi d'ajuster son traitement.

La centrale de mesure pour piscine permet la mesure du PH, du Redox ou ORP (Oxydo Reduction Potential), et de la température de l'eau, elle contrôle aussi le fonctionnement correct de la pompe de filtration. Les informations collectées sont envoyées à une centrale domotique permettant l'affichage et la gestion des paramètres.

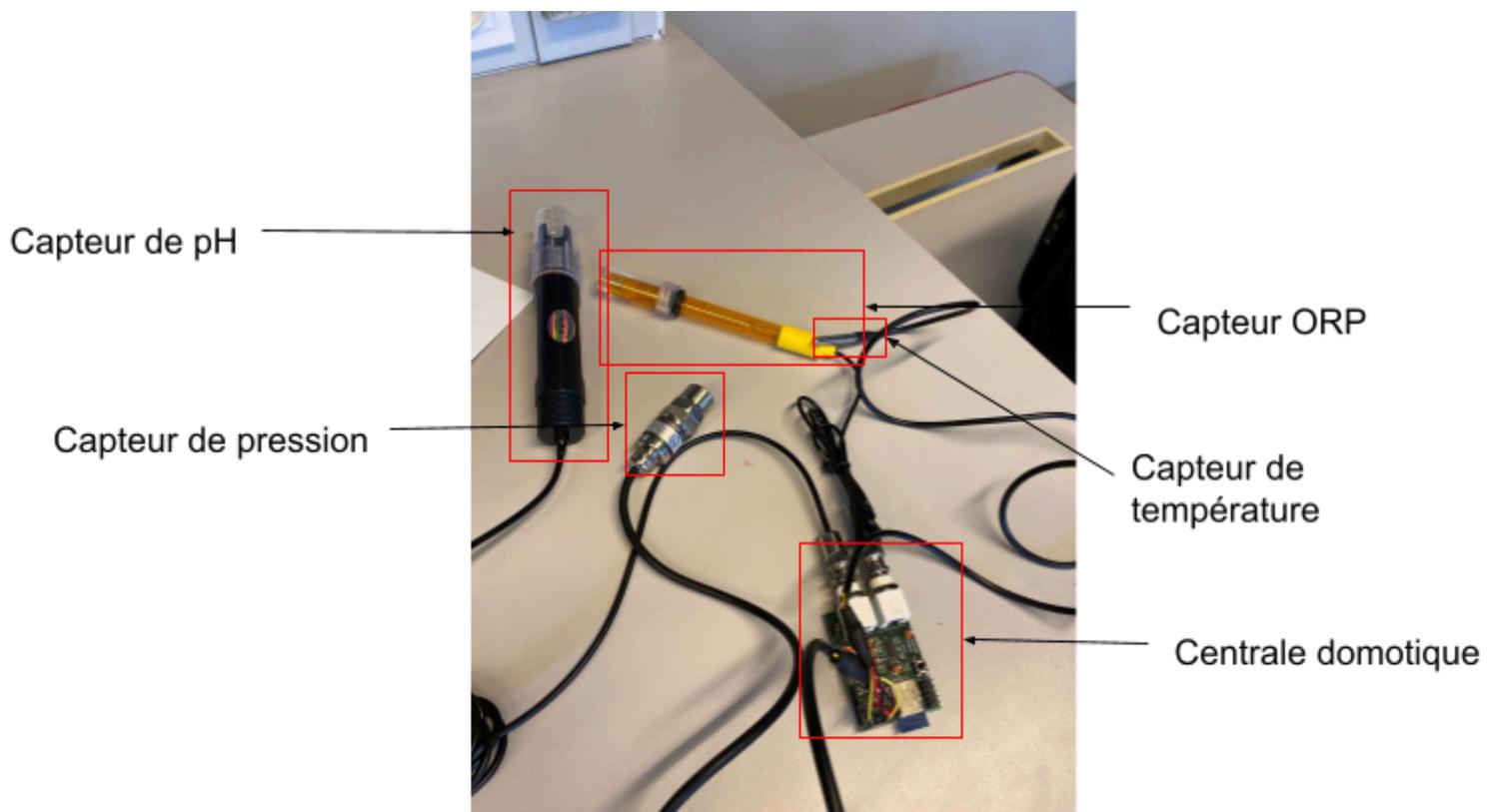
Le pH ou potentiel d'hydrogène est lié à la concentration en ions H⁺ qui représentent l'acidité de l'eau. L'échelle de pH s'étend de 0 (milieu très acide : concentration d'ions H⁺ importante) à 14 (milieu très basique : faible concentration d'ions H⁺). Un pH neutre est équivalent à 7. En piscine, le pH idéal se situe entre 7 et 7.2.

Le Redox ou ORP (Oxydo Reduction Potential) indique le pouvoir oxydant ou réducteur d'une substance. Il permet de juger de l'aspect général de l'eau d'un point de vue sanitaire, c'est-à-dire de la présence de la bonne quantité de chlore. L'ORP se mesure en millivolts. La température idéale en piscine doit se situer à environ 10 °Celsius en dessous de la température du corps soit environ 27°C. La température idéale pour profiter au maximum de sa piscine extérieure est comprise entre 26 et 28°C.

Description du système:



Centrale domotique et les 4 capteurs.



Répartition des tâches et rôles du groupe:

Le projet va être réparti en trois parties correspondant à un étudiant.

Étudiant 1 (Théo Boucaud):

L'étudiant 1 assurera la mise en œuvre de l'interface Zigbee, l'installation et le paramétrage de la passerelle Zigbee2MQTT, l'installation et le paramétrage du Broker MQTT.

Il devra aussi créer l'application de gestion pour ordinateur Windows permettant de configurer l'interface piscine et d'accéder aux mesures.

Étudiant 2 (Lucas Authier):

Cet étudiant s'occupe de la création de l'IHM de configuration embarquée sur la carte domotique Raspberry. Cette IHM permet de configurer l'interface et d'accéder aux mesures. Cette IHM utilise un écran tactile.

Étudiant 3 (Léo Barao):

Cet étudiant va créer une base de données, permettant d'enregistrer les valeurs mesurées (1an de sauvegarde) avec une détection d'alerte en cas de problèmes sur la carte Serveur Raspberry pi. Lorsqu'un problème est détecté une alerte sera émise par la carte Raspberry à destination des applications de gestion. Une application Web sera développée pour afficher les paramétrages, les données mesurées et les courbes.

Objectifs du projet

Utilisation d'une sonde de mesure de Ph Électrode pH ASP 200-2-1M », la précision devra être inférieure à 5% et la résolution de 0.1.

La mesure du Redox ou ORP (Oxydo Reduction Potential) est réalisée par une sonde ORP 110020293. La résolution sera de 10 mV.

La mesure de température est réalisée par une sonde DS18B20, résolution 0.5°C, précision 0,5%.

Contrôle du fonctionnement de la pompe de filtration par capteur de pression SEN0257 (0 à 1MPa), résolution et précision 500Pa.

La transmission des données Zigbee de l'interface piscine se fera vers un coordinateur Zigbee (clé USB SONOFF Zigbee 3.0) branché sur un mini-ordinateur faisant office de centrale domotique.

Le logiciel Zigbee2Mqtt servira de passerelle Zigbee/Mqtt et enverra les données de l'interface piscine vers un broker MQTT hébergé sur la centrale domotique. Une base de données permettra de stocker les valeurs de l'interface piscine sur 1 an minimum.

Une application Windows et une application web, permettant de gérer cette centrale.

UML

Diagramme des cas d'utilisation

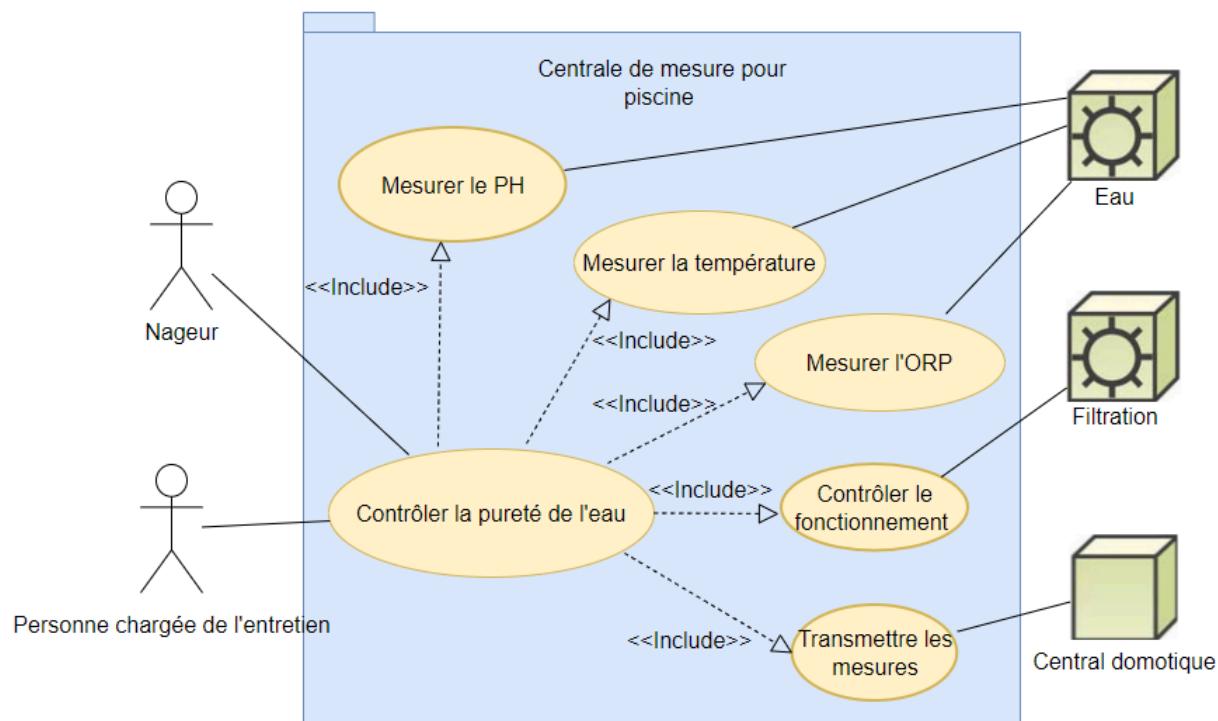
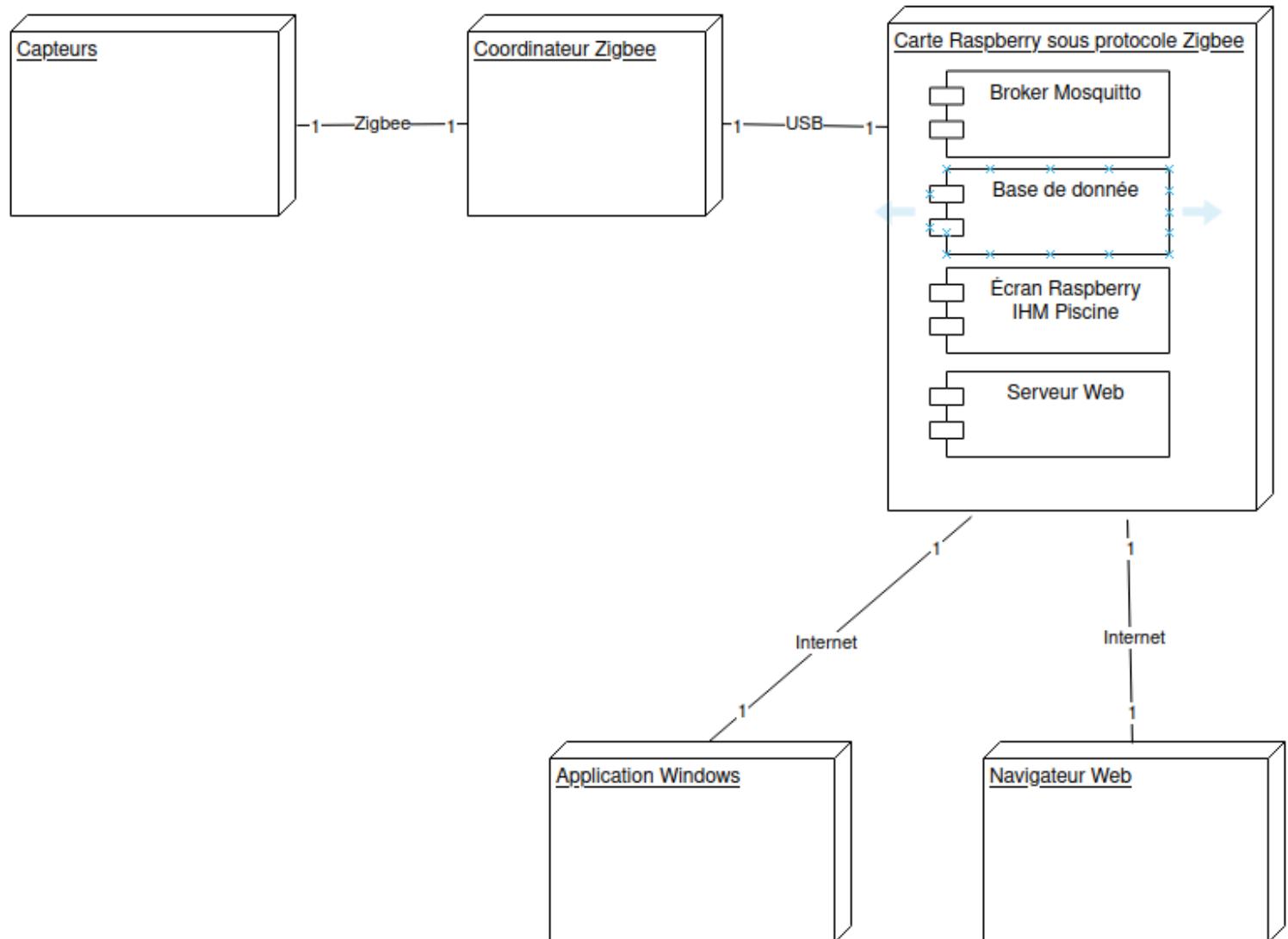


Diagramme de déploiement :



Matériel utilisé

Capteur de température D18B20 :

La sonde de température D18B20 est une sonde étanche fabriquée en acier inoxydable. Elle permet de mesurer avec précision la température de -55°C à +125°C en milieu humide.



Sonde pH (H-101 pH electrode) :

Cette sonde pH est une sonde professionnelle permettant de mesurer un pH entre 0 et 14 à une température comprise entre 0 et +60°C.



Capteur de pression SEN0257 :

Ce capteur permet la mesure de la pression de l'eau. Il communique avec un microcontrôleur via une liaison analogique.



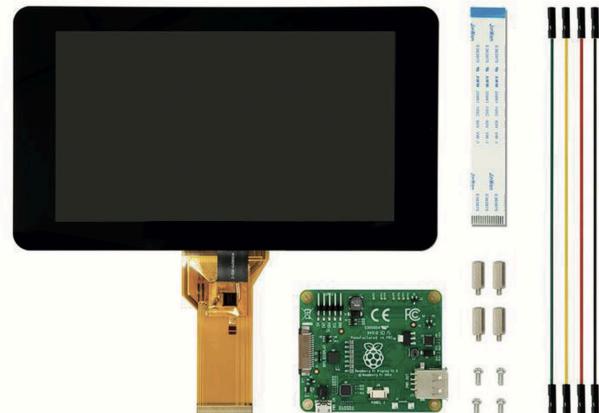
Sonde ORP (Oxydo-réduction):

Cette sonde permet de mesurer le potentiel d'oxydo-réduction (ORP).



Ecran Raspberry Pi 7":

Cet écran permettra d'afficher l'interface homme machine.



Coût du projet

Nom du composant	Prix
Capteur de température D18B20	8€
Sonde pH (H-101 pH electrode)	60€
Capteur de pression SEN0257	25€
Sonde ORP (Oxydo-réduction)	30€
Ecran Raspberry Pi 7"	80€
Carte Raspberry Pi 4 4GB RAM	90€
Clé sonoff USB ZigBee 3.0	25€
Centrale de mesure	100€
Total	418€

Diagramme de Gantt prévisionnels

Etudiant 1:

Nom	Date de dé...	Date de fin
Découverte du projet	09/01/2...	12/01/2...
Commencement de la revue ...	16/01/2...	19/01/2...
Découverte des documentati...	23/01/2...	24/01/2...
Installation du broker Mosquitto	25/01/2...	26/01/2...
Sécurisation du broker Mosq...	30/01/2...	31/01/2...
Découverte de zigbee2mqtt	01/02/2...	02/02/2...
Installation de zigbee2mqtt	06/02/2...	07/02/2...
Branchemet des capteurs s...	08/02/2...	08/02/2...
Récupération des données d...	09/02/2...	09/02/2...
1ère revue de projet	05/03/2...	05/03/2...
Configuration base de données	12/03/2...	15/03/2...
programme application	09/04/2...	12/04/2...
2ème revue de projet	02/04/2...	02/04/2...
3ème revue de projet	07/05/2...	07/05/2...
Fin du projet	31/05/2...	31/05/2...
Configuration des capteurs	19/03/2...	22/03/2...
Préparation de l'oral	30/04/2...	29/05/2...

Etudiant 2 :

Nom	Date de dé...	Date de fin
Découverte du projet	30/01/2...	31/01/2...
Branchemet de l'écran Ras...	01/02/2...	02/02/2...
Cross-compilation	06/02/2...	07/02/2...
Programmation des boutons...	08/02/2...	09/02/2...
Programmation page acceu...	13/02/2...	15/02/2...
Programmation jauge de pr...	16/02/2...	06/03/2...
Programmation graphique t...	07/03/2...	12/03/2...
Programmation jauge ORP	13/03/2...	15/03/2...
Programmation jauge pH	19/03/2...	21/03/2...
Affichage des alertes	22/03/2...	28/03/2...
Préparation de l'oral	30/04/2...	06/06/2...

Etudiant 3 :

Nom	Date de début	Date de fin
Analyse cahier des charge	08/01/2024	09/01/2024
Recherches solution	10/01/2024	12/01/2024
Préparation Installation sur VM	15/01/2024	19/01/2024
Installation officiel des soft sur ...	05/02/2024	09/02/2024
Développement template Site Web	12/02/2024	16/02/2024
Développement de la liaison PHP	04/03/2024	08/03/2024
Développement des scripts Java-s...	11/03/2024	15/03/2024
Correction des Problèmes lier à PH...	02/04/2024	12/04/2024
Sécurisation du site et ses identifiant	29/04/2024	30/04/2024
Création des paramètres	02/05/2024	03/05/2024
Création de l'historique	06/05/2024	07/05/2024
Compilation de la Beta	21/05/2024	21/05/2024
Présentation Revue de projet Perso	21/05/2024	21/05/2024
Compilation sur la carte et testes	10/06/2024	11/06/2024

Rapport de projet personnel



Sommaire

Remerciement:	15
Introduction.....	16
Cahier des charges:.....	16
Solutions choisies:.....	16
Présentation des logiciels choisis:.....	17
Présentation des outils utilisés:.....	17
Prévisions des tâches:.....	17
Espace de sauvegarde:.....	20
Diagramme de séquence:.....	21
Compilation croisée:.....	22
Pourquoi la compilation croisée est-elle utile ?.....	22
Comment fonctionne la compilation croisée ?.....	23
Conclusion:.....	24
Programmation:.....	25
QML:.....	25
C++:.....	30
Liaison entre le C++ et le QML:.....	33
Bash:.....	35
Documentation de fonctionnement.....	37

Remerciement:

Tout d'abord je tiens à remercier les professeurs d'informatique du lycée La Fayette Monsieur Bulcke, Monsieur Barthomeuf et Monsieur Degoute ainsi que Madame Chopin, notre professeur de sciences physiques pour leur suivi quotidien et leur grande aide.

Introduction

Cahier des charges:

J'ai été chargé dans ce projet de faire l'interface homme machine (IHM) sur l'écran tactile Raspberry. Elle permet de configurer l'interface et d'accéder aux mesures.

Elle permet l'affichage et le réglage des paramètres de filtration automatique, l'affichage des données de température, de pression, de pH et d'ORP ainsi que l'affichage des alertes.

Solutions choisies:

J'ai choisi d'utiliser un écran Raspberry Pi de 7 pouces et une carte Raspberry Pi 4 comme microcontrôleur et dont le système d'exploitation est Raspbian qui est basé sur Debian.

J'avais le choix entre faire mon application en utilisant l'outil graphique de Qt creator ou bien d'utiliser le langage QML. C'est la deuxième solution que j'ai utilisé car le QML est mieux pour une interface utilisateur fluide et moderne, cependant pour utiliser le QML, il faut passer par Qt Creator.

Je suis donc également passé par Qt Creator pour cela, et pour la programmation qui est faite en C++.

Présentation des logiciels choisis:

Qt est un framework de développement d'interfaces utilisateur (UI) open-source. Il offre un ensemble complet d'outils pour créer des applications graphiques riches et réactives pour de nombreuses plateformes, y compris les ordinateurs, les téléphones ou encore les cartes comme les Raspberry Pi. Le logiciel Qt est basé sur le langage C++.

QML (Qt Meta-Language) est un langage déclaratif pour la création d'interfaces utilisateur (UI) dans le framework Qt. Il s'agit d'un langage basé sur du texte, facile à apprendre et à utiliser, qui nous permet de décrire l'apparence et le comportement de notre interface utilisateur de manière concise et intuitive.

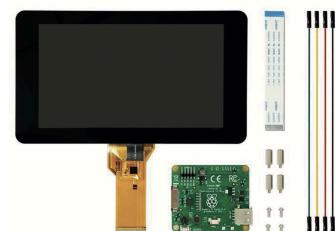


Présentation des outils utilisés:

J'ai eu besoin d'un ordinateur sous Linux avec Qt Creator version 5.15.2 d'installé afin de pouvoir réaliser la compilation croisée et mon IHM en QML.



L'écran Raspberry Pi de 7 pouces est un écran tactile qui a une durée de vie de 20 000h et consomme 200mA à 5V avec la luminosité maximale.



Prévisions des tâches:

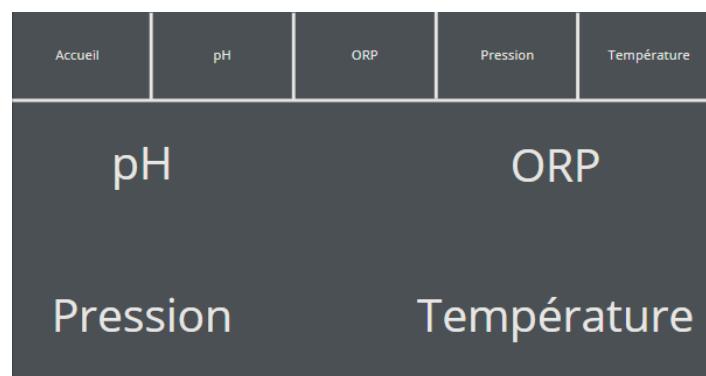
J'ai prévu de commencer en créant mon espace de travail sur la carte Raspberry grâce aux commandes qui créent un utilisateur puis en ajoutant cet utilisateur au groupe des super utilisateurs. Les commandes sont les suivantes: **sudo adduser lucas** et **sudo adduser lucas sudo**.

Ensuite je vais faire la compilation croisée pour pouvoir créer le kit de compilation sur Qt de la carte Raspberry et commencer la partie graphique de l'application.

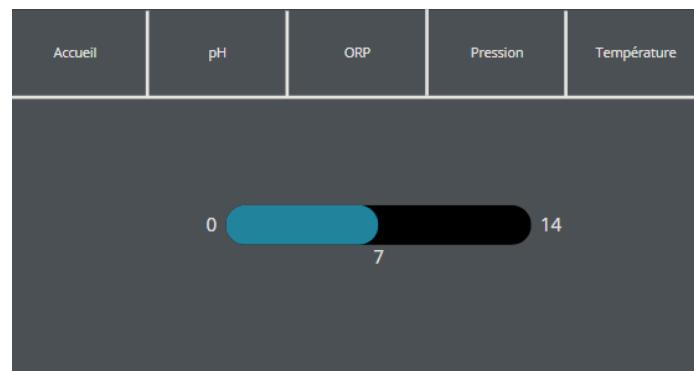
Nom :	RPI4 Piscine
Nom du système de fichiers :	
Type de périphérique sur lequel exécuter:	Périphérique Linux distant
Périphérique sur lequel exécuter:	reterminal (défaut pour Linux distant)
Compiler le périphérique:	Desktop (défaut pour Desktop)
Compilateur:	C: GCC linaro C++: G++ linaro
Environnement:	Aucune modification à appliquer.
Débogueur:	Aucune
Racine du système:	/home/eleve/reterminal/sysroot
Version de Qt:	Qt 5.15.2 (qt5.15)
mkspec de Qt:	
Paramètres supplémentaires du profil Qbs:	
Outil CMake:	CMake 3.27.7 (Qt)

Pour cette partie j'ai prévu de faire 5 boutons en haut de l'écran, un pour l'accueil, un pour la page pH, un pour la page ORP, un pour la page pression et un pour la page température.

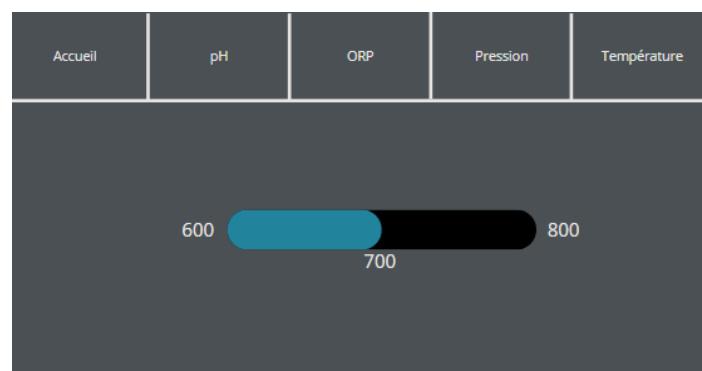
La page d'accueil va afficher les 4 données instantanées et les alertes s'il y en a.



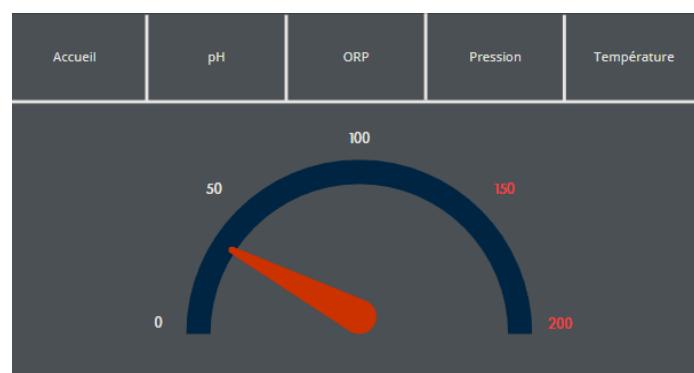
La page pH va afficher une jauge linéaire qui ira de 0 à 14 et si le pH n'est pas compris entre 7 et 7,2, il sera affiché en rouge.



La page ORP va afficher une jauge linéaire aussi, qui ira de 600mV à 800mV et l'ORP doit être compris entre 650mV et 750mV, sinon il sera affiché en rouge.



La page pression va afficher une jauge circulaire qui ira de 0 à 200 centibar. La pression ne doit pas dépasser 140 centibar, elle sera marquée en rouge sur la jauge si cela arrive.



La page température va afficher un graphique des températures de l'eau au cours de la journée.



Voici le diagramme de Gantt prévisionnel que j'ai préparé pour ma partie.

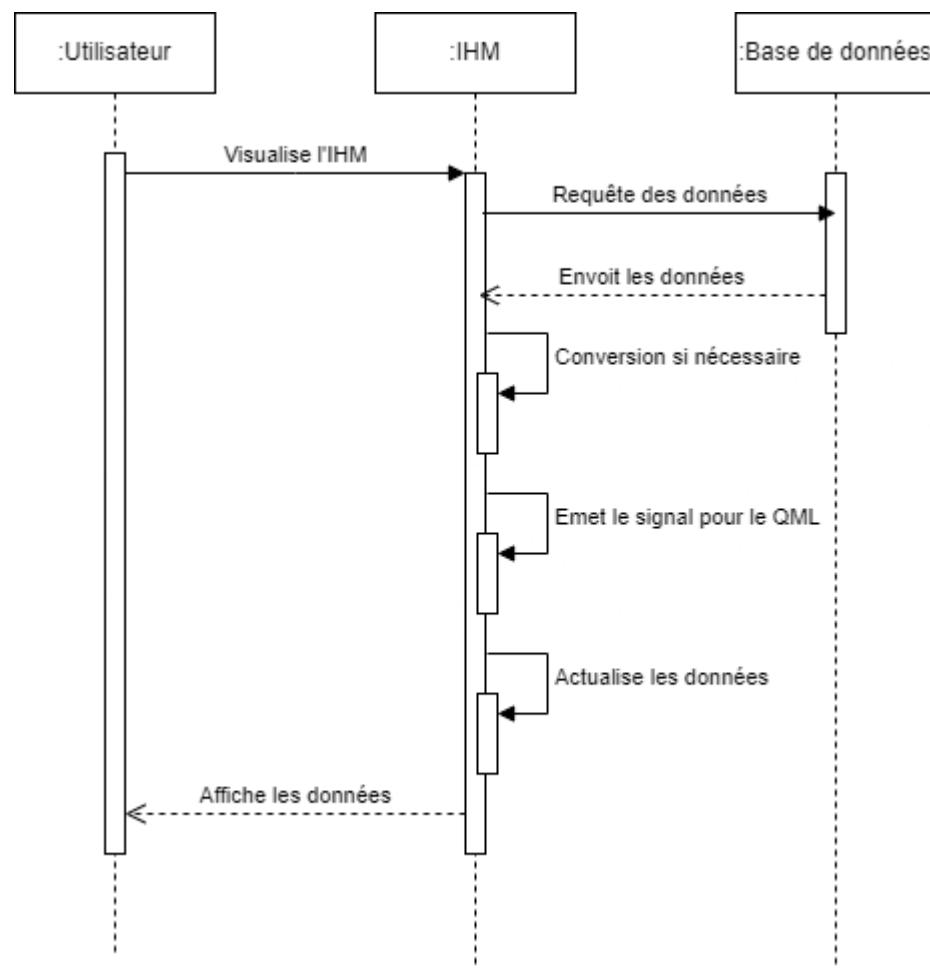
Nom	Date de début	Date de fin
Découverte du projet	30/01/2024	31/01/2024
Branchemet de l'écran Raspberry	01/02/2024	02/02/2024
Cross-compilation	06/02/2024	07/02/2024
Programmation des boutons QML	08/02/2024	09/02/2024
Programmation page accueil IHM	13/02/2024	15/02/2024
Programmation jauge de pression QML	16/02/2024	06/03/2024
Programmation graphique température	07/03/2024	12/03/2024
Programmation jauge ORP	13/03/2024	15/03/2024
Programmation jauge pH	19/03/2024	21/03/2024
Affichage des alertes	22/03/2024	28/03/2024
Préparation de l'oral	30/04/2024	06/06/2024

Espace de sauvegarde:

J'ai sauvegardé à chaque fois mon code sur Github, pour être sûr de ne pas le perdre et de pouvoir travailler chez moi si c'était nécessaire.

 Piscine	Delete Piscine/Journal_theo
 ProjetBeta	Add files via upload
 ecran	Add files via upload
 piscine	Add files via upload
 Journal_Lucas	Create Journal_Lucas
 Journal_theo	Create Journal_theo
 README.md	Initial commit

Diagramme de séquence:



Compilation croisée:

La compilation croisée, aussi appelée "cross-compilation" en anglais, est le processus de création d'un programme exécutable pour une plateforme cible différente de celle sur laquelle s'exécute le compilateur. En d'autres termes, le code source est compilé sur une machine (machine hôte) avec un ensemble d'outils de développement spécifiques, mais le programme résultant est conçu pour s'exécuter sur une autre machine (machine cible) avec une architecture et/ou un système d'exploitation différents.

Pourquoi la compilation croisée est-elle utile ?

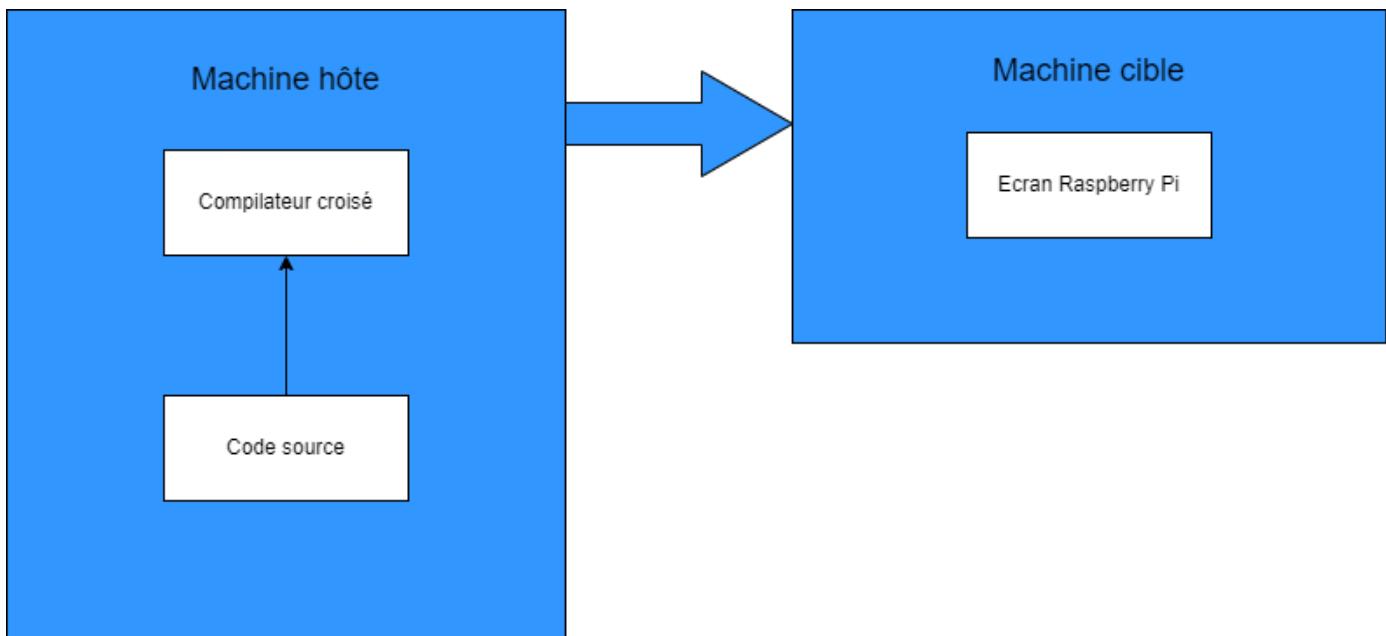
La compilation croisée est particulièrement utile dans les situations suivantes :

- **Développement pour des systèmes embarqués:** Les systèmes embarqués, tels que les microcontrôleurs et les routeurs, n'ont souvent pas les ressources nécessaires pour exécuter un compilateur complet. La compilation croisée permet de développer des programmes pour ces systèmes sur une machine plus puissante, puis de transférer le code exécutable sur la machine cible.
- **Portabilité des applications:** La compilation croisée permet de compiler un programme une seule fois, puis de le générer pour différentes architectures et systèmes d'exploitation. Cela peut être utile pour les développeurs qui souhaitent créer des applications multiplateformes.
- **Débogage et tests:** La compilation croisée peut être utilisée pour déboguer et tester du code sur une machine différente de celle sur laquelle il s'exécutera finalement. Cela peut être utile pour identifier les problèmes spécifiques à une plateforme particulière.

Comment fonctionne la compilation croisée ?

Le processus de compilation croisée implique généralement les étapes suivantes :

1. **Installation de la chaîne de compilation croisée:** Une chaîne de compilation croisée est un ensemble d'outils de développement spécifiques à la plateforme cible. Ces outils incluent généralement un compilateur, un assembleur, un éditeur de liens et d'autres utilitaires nécessaires à la création d'un programme exécutable.
2. **Configuration de l'environnement de compilation:** L'environnement de compilation doit être configuré pour spécifier la chaîne de compilation croisée à utiliser et les options de compilation appropriées.
3. **Compilation du code source:** Le code source du programme est compilé à l'aide de la chaîne de compilation croisée. Le compilateur traduit le code source dans un langage machine spécifique à la plateforme cible.
4. **Génération du programme exécutable:** L'assembleur et l'éditeur de liens lient les différents modules du programme compilés ensemble pour créer un programme exécutable.



Conclusion:

La compilation croisée est un outil puissant qui permet aux développeurs de créer des programmes pour des plateformes différentes de celle sur laquelle ils travaillent. Elle est particulièrement utile pour le développement de systèmes embarqués, la portabilité des applications et le débogage et les tests.

Programmation:

QML:

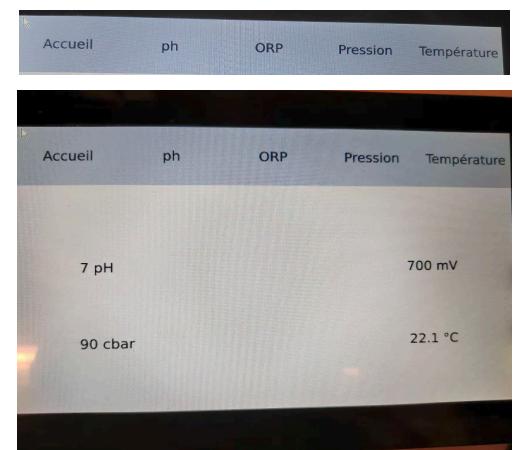
J'ai commencé la programmation par faire les 5 boutons de l'application en QML, donc un bouton pour la page d'accueil, un bouton pour le pH, un bouton pour l'ORP, un bouton pour la pression et un dernier bouton pour la température.

Si on prend l'exemple pour du bouton accueil, on déclare son identifiant, le texte qu'il aura donc "Accueil", sa position sur l'écran avec l'axe des abscisses et des ordonnées, la taille en largeur et en longueur. Enfin on ajoute une fonction lorsqu'on clique sur le bouton qui nous permet d'afficher d'afficher ce qu'on veut sur la page et de cacher le reste. Ici on montre les 4 valeurs instantanées et on cache les différents graphiques qui correspondent aux autres pages.

On fait en sorte de mettre les boutons en haut de l'écran.

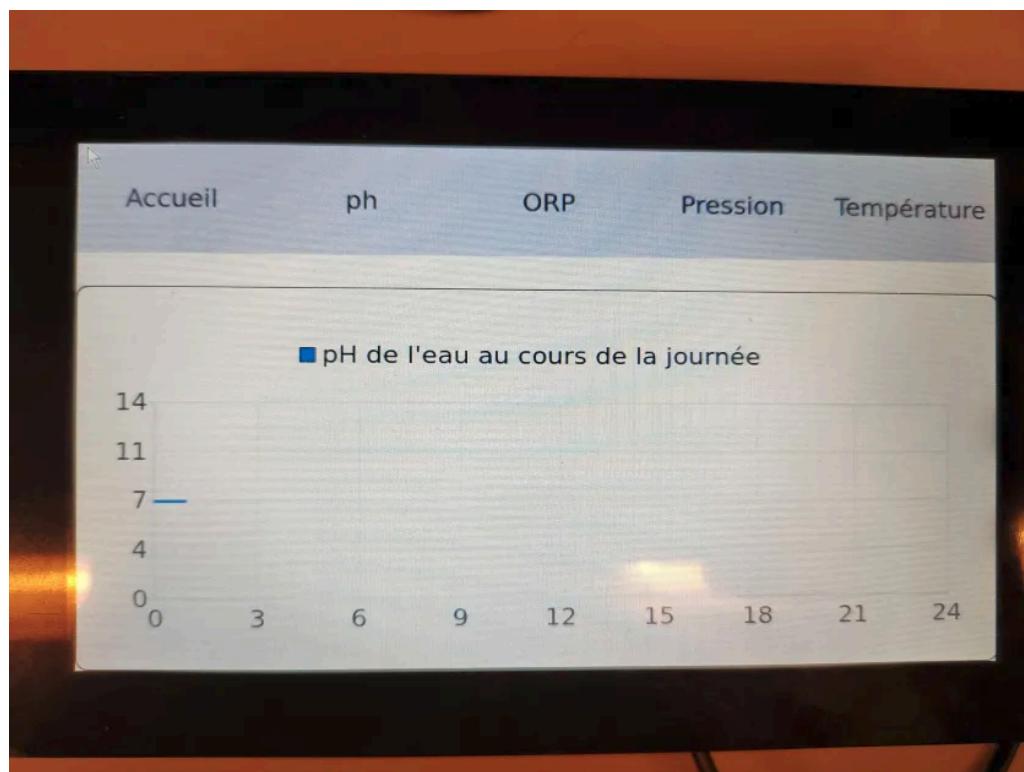
Ensuite pour la page accueil, on met les 4 valeurs instantanées des capteurs ainsi que les alertes s'il y en a pour chaque capteur, comme par exemple s'il y a une pression trop forte.

```
Rectangle {
    Button {
        id: boutonAccueil
        text : "Accueil"
        x: 0
        y: 0
        width: 160
        height: 100
        onClicked: {
            function onClick(){
                rectPression.visible = false
                rectTemp.visible = false
                rectPh.visible = false
                rectORP.visible = false
                textPression.visible = true
                textTemperature.visible = true
                textORP.visible = true
                textPh.visible = true
            }
            onClick();
        }
    }
}
```

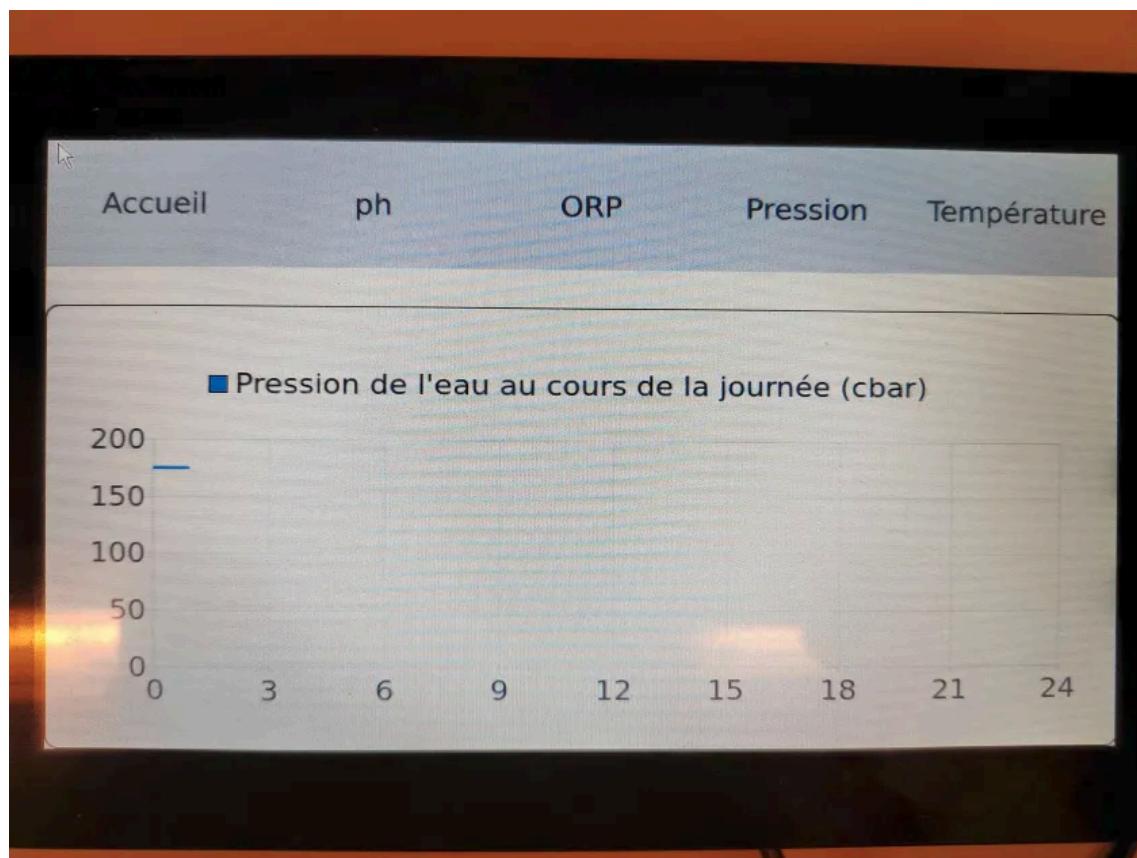


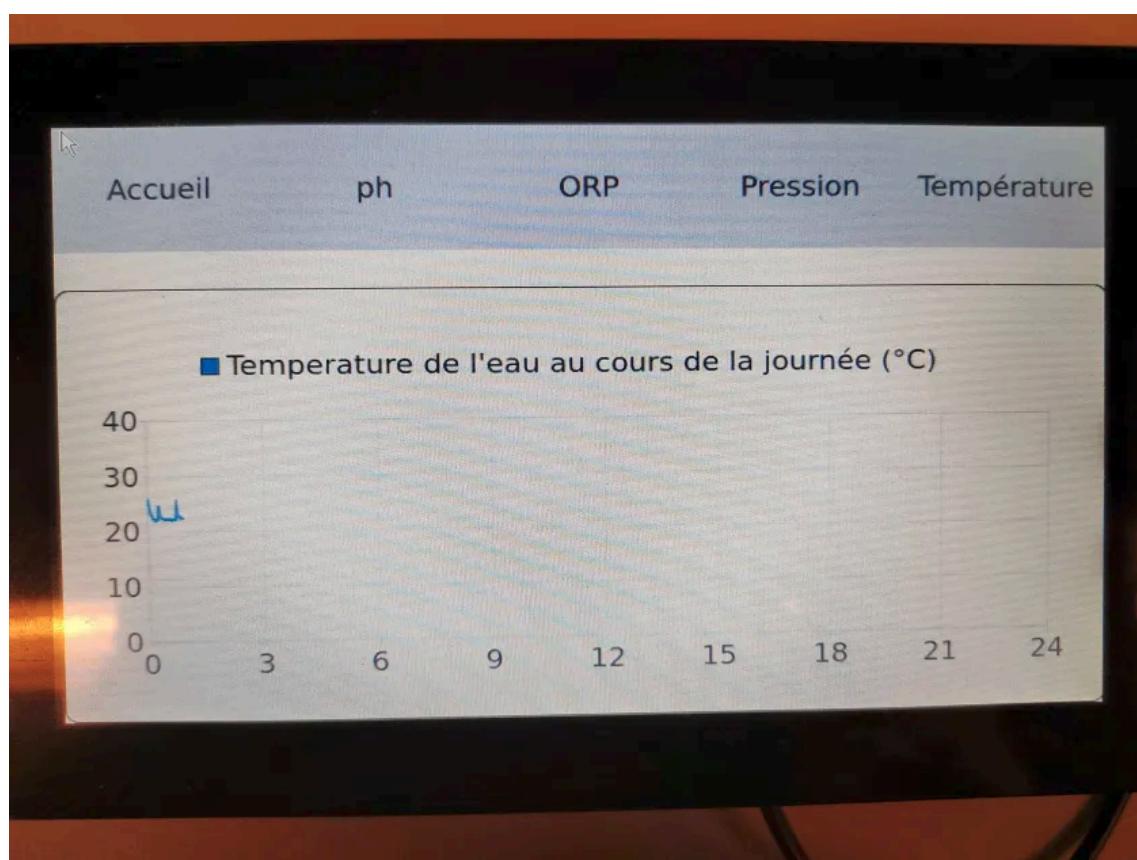
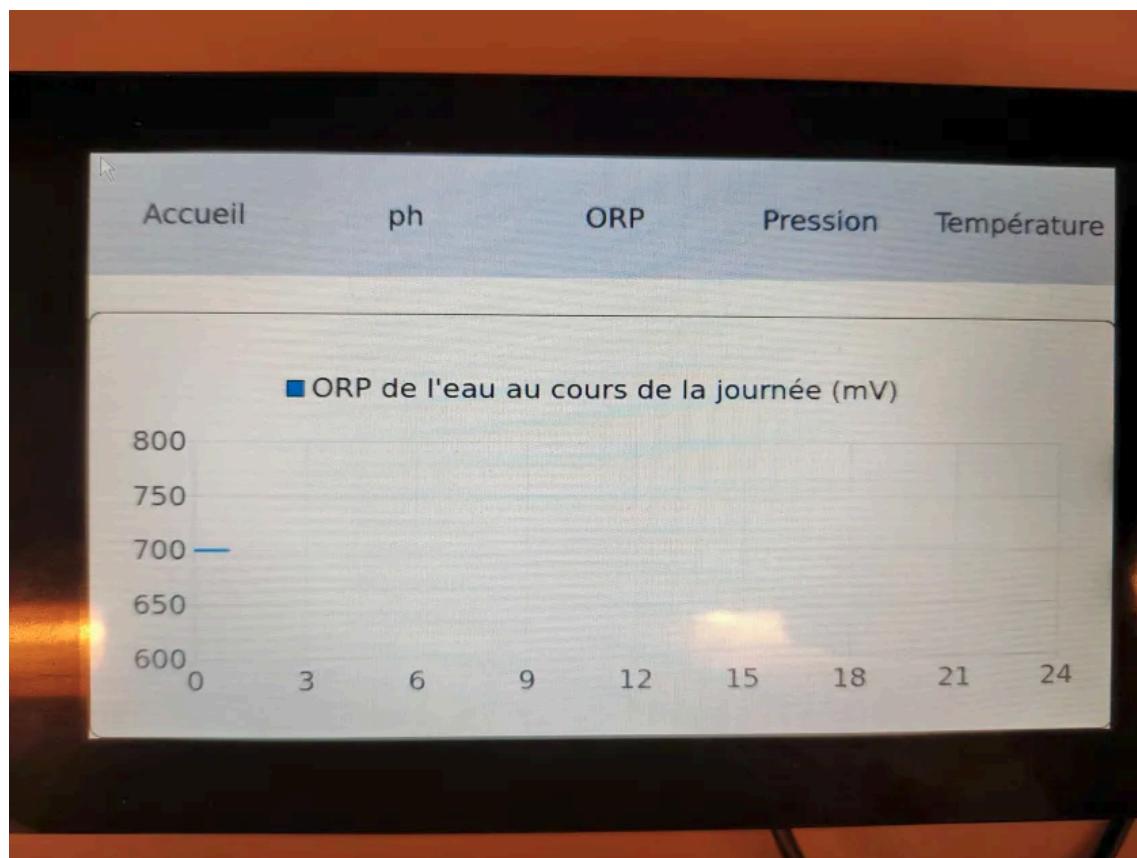
Pour la page pH, j'affiche un graphique qui montre l'évolution du pH du jour. Pour cela on crée un rectangle en QML, on lui donne un identifiant, on le place sur l'écran donc ici de sorte à ce qu'il prenne tout l'espace restant en dehors des boutons. On crée ensuite le graphique avec l'outil "ChartView" auquel on met un identifiant, les valeurs maximum et minimum de l'axe des abscisses et des ordonnées et enfin le nom du graphique qui s'affiche juste au-dessus. De base on met le graphique en non-visible car la page de base de l'application est celle d'accueil. Donc lorsque l'utilisateur appuie sur le bouton "pH" il pourra voir le graphique.

```
Rectangle {
    id: rectPh
    anchors.horizontalCenter: parent.horizontalCenter
    anchors.bottom: parent.bottom
    height: 350
    width: 800
    border.width: 1
    radius:10
    ChartView {
        id: graphPh
        Layout.fillHeight: true
        Layout.fillWidth: true
        anchors.fill: parent
        antialiasing: true
        LineSeries {
            id: ph
            axisX: ValueAxis{
                min:0
                max:24
                tickCount: 9
                labelFormat: "%.0f"
            }
            axisY: ValueAxis {
                min: 0
                max: 14
                tickCount: 1
                labelFormat: "%.0f"
            }
        }
        name: "pH de l'eau au cours de la journée"
    }
    visible: false
}
```

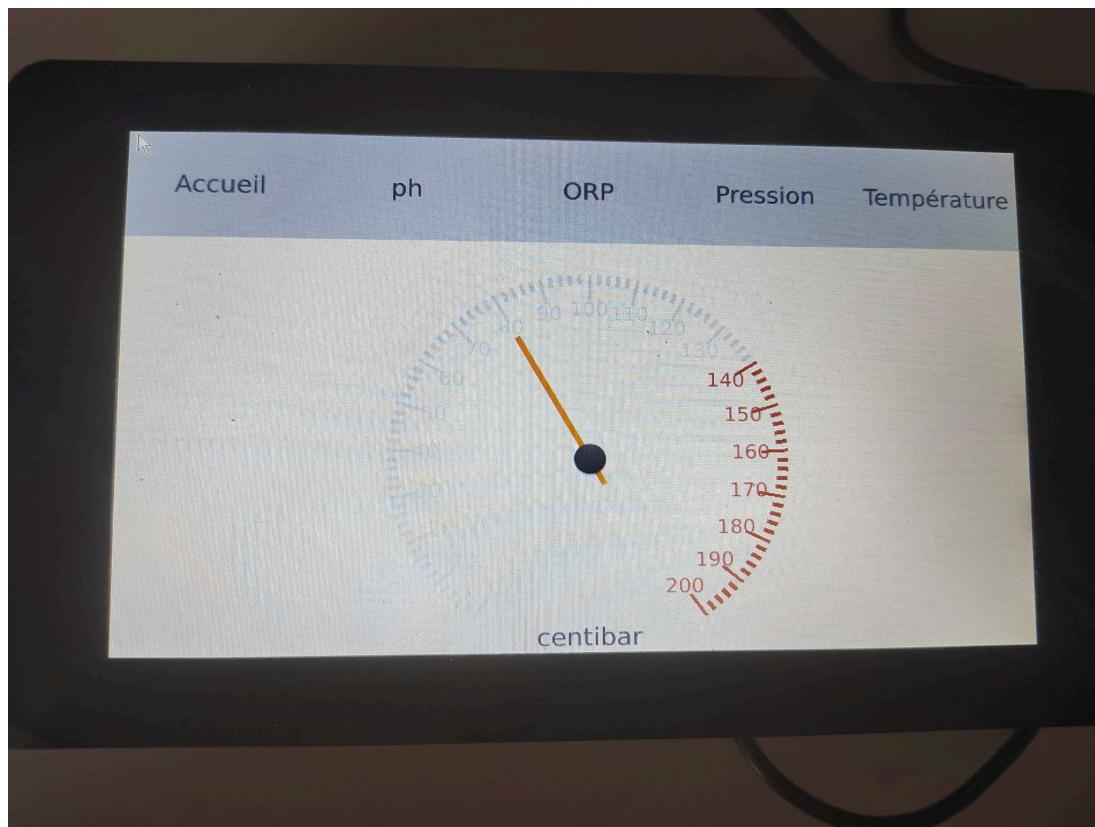


On fait la même chose pour les autres pages, donc un graphique pour l'évolution de l'ORP, de la pression et de la température au cours de la journée.





J'avais également fait une jauge circulaire pour la page pression seulement après concertation avec les autres membres du groupe, nous avons convenu qu'il était mieux de faire un graphique comme sur les autres pages.



C++:

J'ai eu besoin de faire des classes en C++ pour récupérer les données des capteurs qui sont stockées dans la base de données.

J'ai donc créé 5 classes différentes, 1 classe pour la base de données et 4 autres pour chaque capteur, celles-ci ont besoin de la classe de la base de données déclarée dans chacune pour fonctionner. Elles héritent toutes de la classe “QObject”. Je vais d'abord présenter les fichiers d'en-tête que j'ai faits.

La classe de la base de donnée (bdd) a besoin des bibliothèques “QSqlDatabase”, “QSqlQuery” et “qobject”. On déclare une variable privée qui contient notre base de donnée de la classe “QSqlDatabase” qui s'appelle *dbPiscine* et une variable privée pour chaque capteur.

On déclare ensuite le *constructeur*, le *destructeur*, la méthode *connect* qui est un booléen et 4 méthodes qui permettent de récupérer chacune la dernière donnée de leur capteur respectif qui est entrée dans la base de donnée. Enfin on déclare un signal pour chaque capteur ce qui permet d'accéder à ces données depuis le QML.

Ensuite, pour les classes de chaque capteur, elles sont toutes identiques. Si on prend la classe du capteur température par exemple, on déclare une variable privée qu'on appelle *temp*. On déclare ensuite le constructeur de la classe qui est *explicit*, ce qui signifie que le type de la donnée ne peut pas être converti en un autre. La méthode *getTemperature* est déclarée en *Q_INVOKABLE* ce qui signifie que la méthode peut être appelée à partir de signaux, ce qui permet donc de l'utiliser dans le QML. Enfin on déclare le signal *temperatureChanged* qui prend une variable *m_temperature* en paramètre.

```
#ifndef BDD_H
#define BDD_H
#include <QSqlDatabase>
#include <QSqlQuery>
#include <qobject.h>

class Bdd : public QObject
{
    Q_OBJECT
private:
    QSqlDatabase dbPiscine;
    double m_temperature;
    double m_ph;
    double m_pression;
    double m_orp;
public:
    Bdd();
    ~Bdd();
    bool connect();
    double getLastTemp();
    double getLastPh();
    double getLastPression();
    double getLastOrp();
signals:
    void temperatureChanged();
};

#endif // BDD_H
```

```
#ifndef TEMPERATURE_H
#define TEMPERATURE_H

#include <QObject>
#include "bdd.h"

class temperature : public QObject
{
    Q_OBJECT
private:
    double _temp;
public:
    explicit temperature(QObject *parent = nullptr);
    temperature(double temp);
    Q_INVOKABLE double getTemperature();
signals:
    void temperatureChanged(double m_temperature);
};

#endif // TEMPERATURE_H
```

Ensuite, je vais présenter le code source des classes.

On reprend donc la classe *bdd*. On fait notre constructeur dans lequel on appelle la méthode *connect*.

Ensuite on crée cette méthode qui est un booléen. On commence par déclarer notre variable QSqlDatabase

qui s'appelle *dbPiscine*. On dit ensuite à la base de données qu'elle se connecte à une base de données MySql. Ensuite, on donne les outils de connexion à la base de données que mon camarade a créé, donc son adresse IP, son nom, le nom d'utilisateur et le mot de passe. Ensuite, si la base de données s'ouvre correctement on fait une requête SQL qui récupère toutes les données de la table *mesures*, table dans laquelle sont stockées les valeurs des capteurs. On fait ensuite une boucle while qui nous permet de donner par exemple à la variable *m_temperature* la valeur de la température dans la requête MySql.

```
#include "bdd.h"
#include <QtSql>
#include <QDebug>
#include <QSqlDatabase>

Bdd::Bdd()
{
    connect();
}

bool Bdd::connect()
{
    QSqlDatabase dbPiscine;
    dbPiscine = QSqlDatabase::addDatabase("QMYSQL");

    dbPiscine.setHostName("172.21.28.60");
    dbPiscine.setDatabaseName("projet_bts");
    dbPiscine.setUserName("bts");
    dbPiscine.setPassword("projet");
    if(dbPiscine.open())
    {
        QSqlQuery requete;
        if(requete.exec("SELECT * FROM mesures")) {
            int i=0;

            while(requete.next()) {
                m_temperature = requete.value("temperature").toDouble();
                m_ph = requete.value("ph").toDouble();
                m_press = requete.value("pression").toDouble();
                m_orp = requete.value("ORP").toDouble();
                i++;
            }
        }
        else {
            qDebug() << "Echec de la requête";
            qDebug() << requete.lastError();
        }
    }
    return true;
}
```

Il suffit alors de retourner cette variable pour chaque méthode qui lui correspond, pour la température c'est donc *getLastTemp* qui retourne *m_temperature*.

On fait ensuite le destructeur, qui consiste à simplement fermer la base de données.

```
double Bdd::getLastTemp()
{
    return m_temperature;
}
```

```
Bdd::~Bdd()
{
    if(dbPiscine.isOpen())
    {
        dbPiscine.close();
    }
}
```

Enfin pour les classes des capteurs, par exemple celle de la température, on fait notre constructeur et la méthode *getTemperature* qui ouvre une instance de la classe base de données, puis on déclare une variable temporaire à laquelle on lui attribue la valeur renvoyée par la méthode *getLastTemp* de l'instance. Enfin, on émet le signal nommé *temperatureChanged* en passant la valeur de la variable temporaire comme argument. On retourne ensuite la valeur de la température. Les autres classes des capteurs fonctionnent de la même manière.

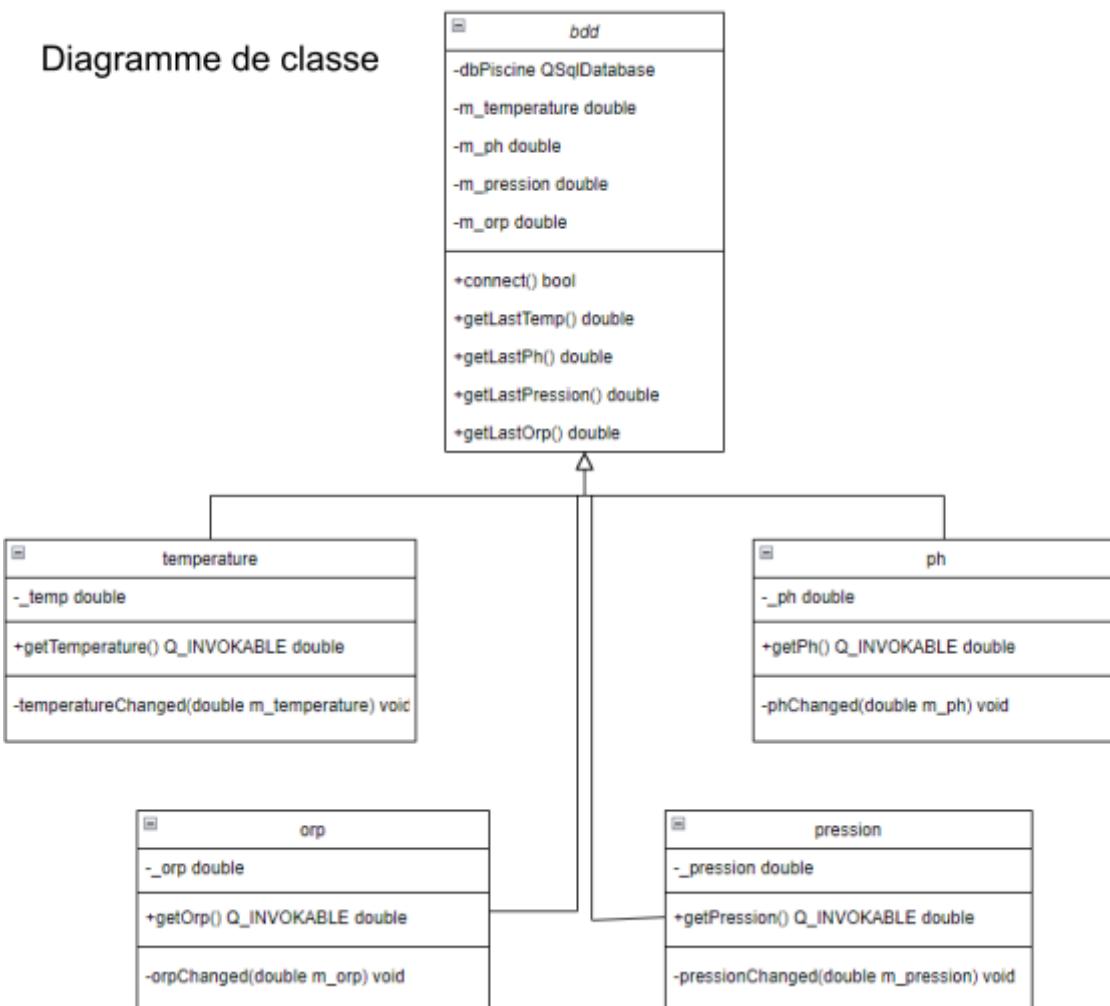
```
#include "temperature.h"
#include "bdd.h"
#include <QDebug>
using namespace std;

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

temperature::temperature(double temp)
{
    _temp = temp;
}

double temperature::getTemperature()
{
    Bdd test;
    double temp = test.getLastTemp();
    qDebug() << temp;
    emit temperatureChanged(temp);
    return temp;
}
```

On retrouve toutes les méthodes et les attributs dans le diagramme de classe ci-dessous:



Liaison entre le C++ et le QML:

Comme expliqué précédemment, pour faire le lien entre le C++ et le QML, on doit passer par des signaux. Pour les utiliser, il faut utiliser le composant *Connections* de QML et lui donner une cible. Par exemple pour la température, j'ai appelé la cible *tempSignal* et j'utilise le signal reçu *onTemperatureChanged* qui devient une fonction, dans laquelle je récupère la valeur de la dernière température puis je la rentre le graphique de la page QML de température.

```
Connections{
    target: tempSignal
    ignoreUnknownSignals: true
    function onTemperatureChanged(value)
    {
        textTemperature.text = value.toString() + " °C"
        var xValue1 = temperature.count/60; // Ici pour changer la distance entre chaque point du graphique température
        var yValue1 = value
        temperature.append(xValue1, yValue1);
    }
}
```

C'est ensuite dans le main.cpp que l'on finit de faire le lien, avec la ligne de code :

```
engine.rootContext()->setContextProperty("tempSignal", new temperature());
```

- *engine* représente l'instance du moteur QML en cours d'exécution.
- *rootContext* permet d'accéder au contexte racine du moteur. Il contient toutes les propriétés et fonctions accessibles aux fichiers QML.
- *setContextProperty* permet de définir une nouvelle propriété accessible dans les fichiers QML.
- Notre cible *tempSignal* qui est la propriété.
- *new temperature* permet de créer une nouvelle instance de la classe *temperature*.

```
engine.rootContext()->setContextProperty("tempSignal", new temperature());
engine.rootContext()->setContextProperty("phSignal", new ph());
engine.rootContext()->setContextProperty("pressionSignal", new pression());
engine.rootContext()->setContextProperty("orpSignal", new orp());
```

Donc on crée une nouvelle instance de la classe, on l'expose comme propriété dans le contexte racine et elle peut ensuite être utilisée dans les fichiers QML.

Enfin, on crée un composant Timer dans le QML qui a pour but d'appeler les fonctions getTemperature etc toute les minutes, pour ensuite les afficher sur la page d'accueil de manière instantanée.

```
Timer {
    interval: 60000
    repeat: true
    running: true
    triggeredOnStart: true //Pour commencer dès le démarrage
    onTriggered: {
        tempSignal.getTemperature()
        phSignal.getPh()
        pressionSignal.getPression()
        orpSignal.getOrp()
    }
}
```

Bash:

J'ai également utilisé des lignes de commandes sur le terminal Linux de la Raspberry.

La première avait pour but de lancer l'application de l'IHM directement au lancement de celle-ci. Pour cela, je vais dans le crontab. Celui-ci est un outil de Linux qui permet d'automatiser les tâches, de planifier l'exécution de scripts, de commandes ou de programmes à des moments précis ou selon des intervalles récurrents.

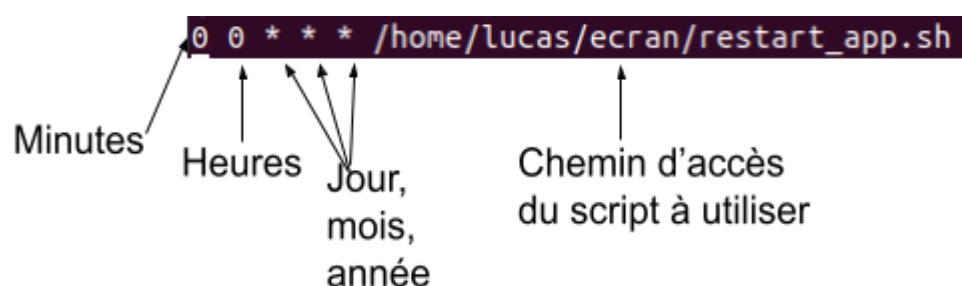
Je met alors cette ligne de commande à l'intérieur:

```
@reboot /home/lucas/ecran/bin/ecran
```

J'ai ensuite créé le fichier restart_app.sh qui permet de redémarrer l'application de l'IHM.

```
arg=$(ps -e | grep ecran | cut -c 1-6)
kill -9 $arg
~/ecran/bin/ecran
```

Ensuite, on retourne dans le crontab où je peux rajouter cette ligne de commande pour que le script s'exécute tous les jours à minuit.



Pour finir, voici mon diagramme de Gantt réel pour mon projet:

Nom	Date de dé...	Date de fin
Découverte du projet	30/01/2...	31/01/2...
Branchement de l'écran Ras...	01/02/2...	02/02/2...
Cross-compilation	06/02/2...	08/02/2...
Programmation des boutons...	13/02/2...	14/02/2...
Programmation jauge de pr...	16/02/2...	08/03/2...
Programmation graphique t...	12/03/2...	27/03/2...
Programmation jauge ORP	28/03/2...	28/03/2...
Programmation jauge pH	29/03/2...	29/03/2...
Changement de style de l'IHM	11/04/2...	12/04/2...
Programmation page accuei...	02/05/2...	07/05/2...
Programmation graphiques ...	21/05/2...	23/05/2...

Conclusion:

Pour conclure, ce projet m'a permis d'approfondir mes connaissances en QML et en C++ mais aussi dans les systèmes UNIX ainsi que le travail de groupe. Ce projet fut enrichissant au niveau des idées de chacuns car nous avions différent point de vue, ce qui nous a permis d'élargir nos idées.

Documentation de fonctionnement

Pour faire fonctionner l'écran Raspberry, il suffit de le brancher avec un câble RJ45 et de brancher le câble d'alimentation. Il faut également que la base de données soit lancée.

Si la connexion à la base de données ne se fait pas automatiquement, il suffit d'ouvrir le programme avec Qt Creator, aller dans le fichier bdd.cpp puis dans la méthode connect() et changer les informations de connexion à la base de données avec le setHostName, setDatabaseName, setUserName et setPassword.

```
bool Bdd::connect()
{
    QSqlDatabase dbPiscine;
    dbPiscine = QSqlDatabase::addDatabase("QMYSQL");

    dbPiscine.setHostName("172.21.28.60");
    dbPiscine.setDatabaseName("projet_bts");
    dbPiscine.setUserName("bts");
    dbPiscine.setPassword("projet");
    if (!dbPiscine.open())
        qDebug() << "ERREUR : Connexion impossible à la base de données";
}
```