

Rapport de Projet:

Piscine tranquille



SOMMAIRE

SOMMAIRE	2
1. Introduction	4
1. Besoins et contexte	4
2. Diagramme des cas d'utilisations	6
3. Solutions et technologies utilisées	7
4. Diagramme de déploiement	13
5. Répartition des tâches	14
6. Diagramme des cas d'utilisations	15
7. Planning prévisionnel	16
8. Recette partielle	17
L'objectif final de ce projet est de pouvoir contrôler la qualité de l'eau à travers une interface à distance ou en local à l'aide d'un écran tactile.	17
Actuellement, nous pouvons contrôler la piscine uniquement avec le mode manuel. À ce point si, on peut utiliser les pompes ainsi que recevoir les données des différents capteurs. L'interface est terminée et elle est simple d'utilisation. Les différents capteurs envoient les informations directement sur l'interface et l'utilisateur peut changer de mode comme il le souhaite depuis l'IHM.	17
2. Partie individuelle (DUCAROUGE Valentin)	18
1. Introduction	18
2. Familiarisation avec le logiciel Arduino	18
3. Protocole de communication	19
4. Capteur de température	19
5. Analyse de la classe CapteurORP.PH	19
6. Mise en oeuvre de la classe CapteurORP.PH	20
7. Difficultés rencontrées	24
8. Conclusion personnelle	24
3. Partie individuelle (OTT Mathieu)	26
1. Introduction	26
2. Développement et transfert de programme pour ESP32	27
3. Analyse des classes Pompe et regultateurPeristaltique	28
4. Fonctionnement des pompes péristaltiques	29
5. Mise en oeuvre des classes Pompe et regulateurPeristaltique	30
6. Difficultées rencontrées	32
7. Conclusion personnelle	32
4. Partie individuelle (CELLIER Nathan)	33
1. Introduction	33
2. Étude de Solution	33
3. La liaison série	33
4. Nextion Editor	34
5. Possibiliter sur nextion	34
6. Fonction pour calculer les angles	35

7. Diagramme de classe	36
8. Difficultés rencontrées	37
9. Conclusion	37
5. Partie individuelle (JOB Alexis)	38
1. Introduction	38
2. C'est quoi MQTT	38
3. Mise en place	38
4. Difficultés rencontrées	47
5. Conclusion	48
6. Conclusion globale	49
7. Annexes	50
1. Main GestionnairePiscine	50
2. CapteurORPPH.h	57
3. CapteurORPPH.cpp	58
4. RegulateurPeristaltique.h	60
5. RegulateurPeristaltique.cpp	61
6. pompe.h	63
7. pompe.cpp	63
8. ecran.h	64
9. ecran.cpp	64

1. Introduction

1. Besoins et contexte

L'entreprise CORTES PISCINE & SPAS qui installe des piscines et des spas dans le Cantal veut étoffer son offre en proposant à ces clients un service d'aide à l'entretien des piscines.

Ce service comportera 2 options :

- Option 1 : Piscine tranquille basique :
 - Automatisation du taux de chlore, du PH et du temps de filtration.
 - Commandes et accès aux données de la piscine en local sur un écran tactile.

- Option 2 : Piscine tranquille connectée :

option 1 plus :

- Envoies de notifications en cas de problème (mauvais PH ou taux de chlore).
- Gestion de la piscine à distance via une application.

Dans cette option le serveur qui communiquera avec l'application pourra soit :

- - être installé chez le particulier.
- être le serveur de l'entreprise.

Pour cela, nous avons deux options.

Option 1:

- Automatisation du taux de chlore, du PH et du temps de filtration.
- Commandes et accès aux données de la piscine en local sur un écran tactile.



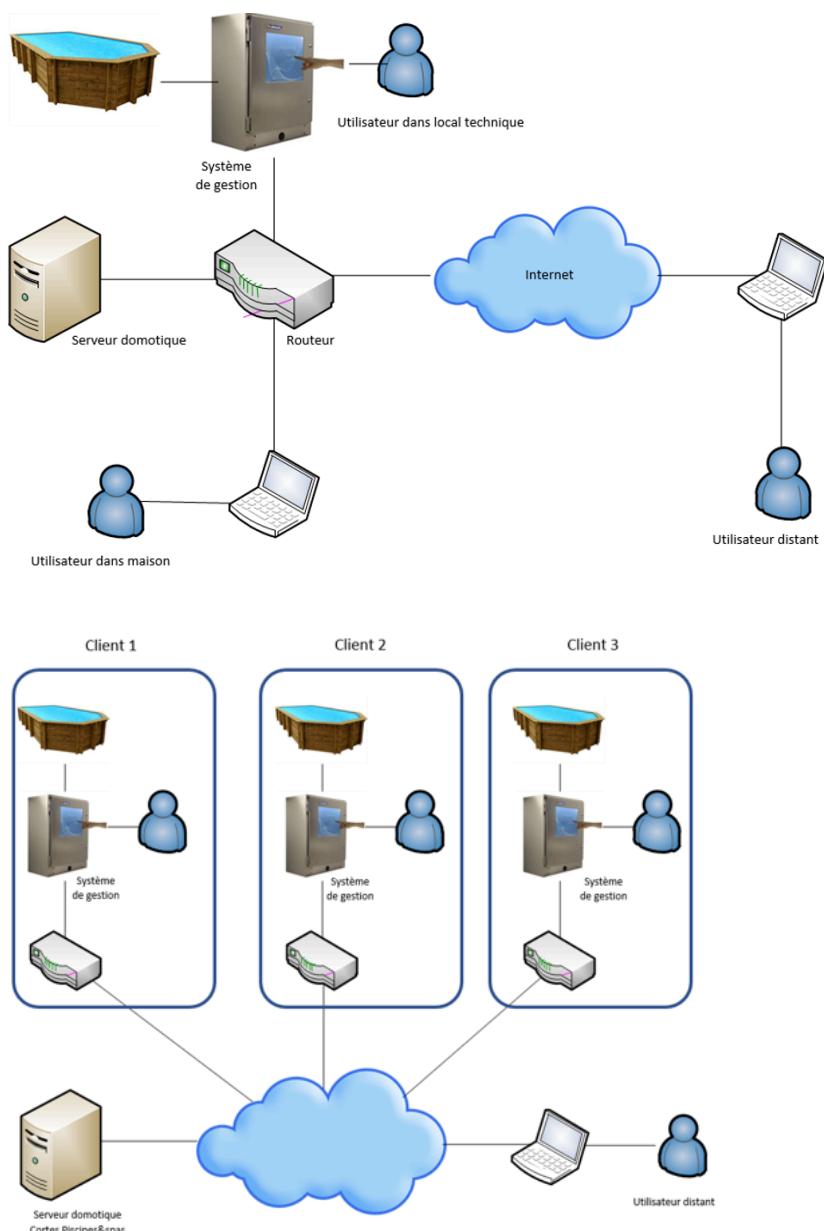
Option 2 :

Option 1 plus :

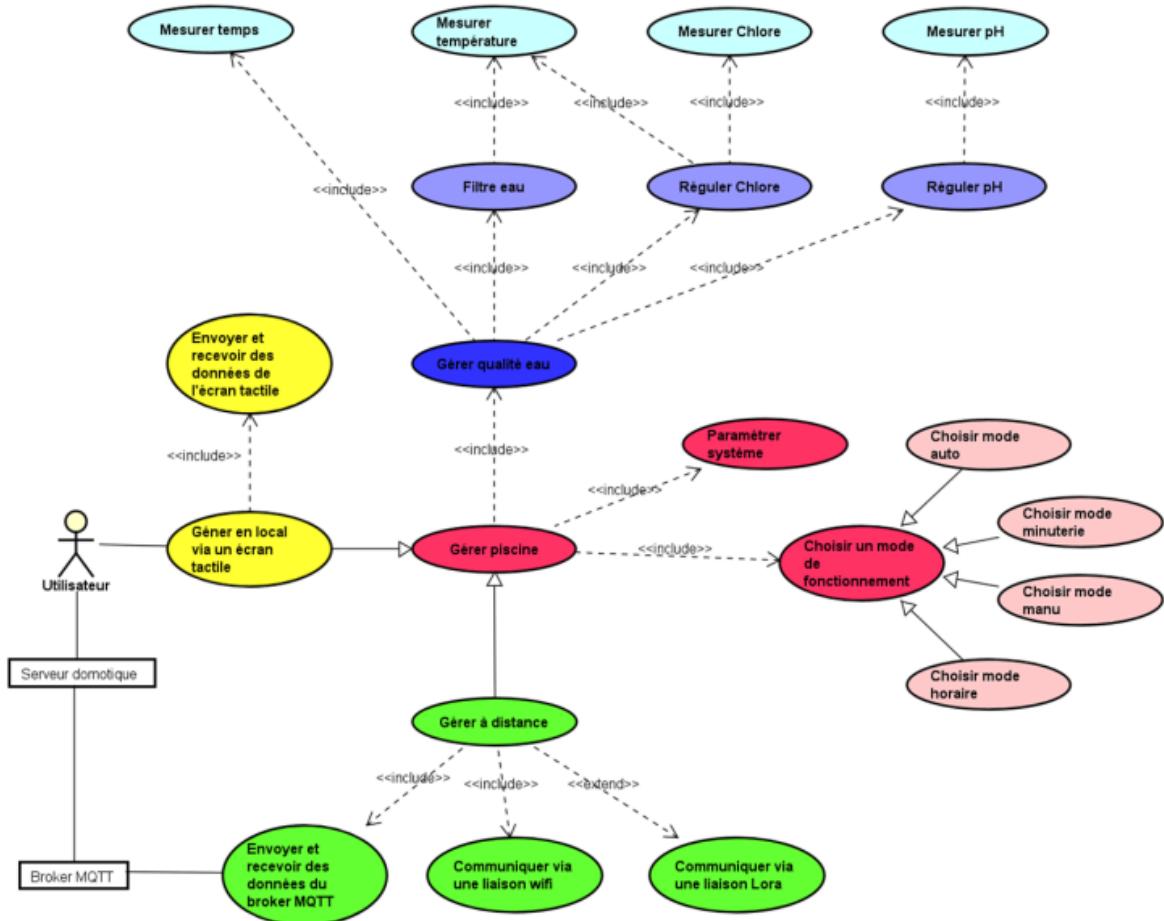
- Envoie de notification en cas de problème (mauvais PH ou taux de chlore).
- Gestion de la piscine à distance via une application.

Dans cette option le serveur qui communiquera avec l'application pourra soit :

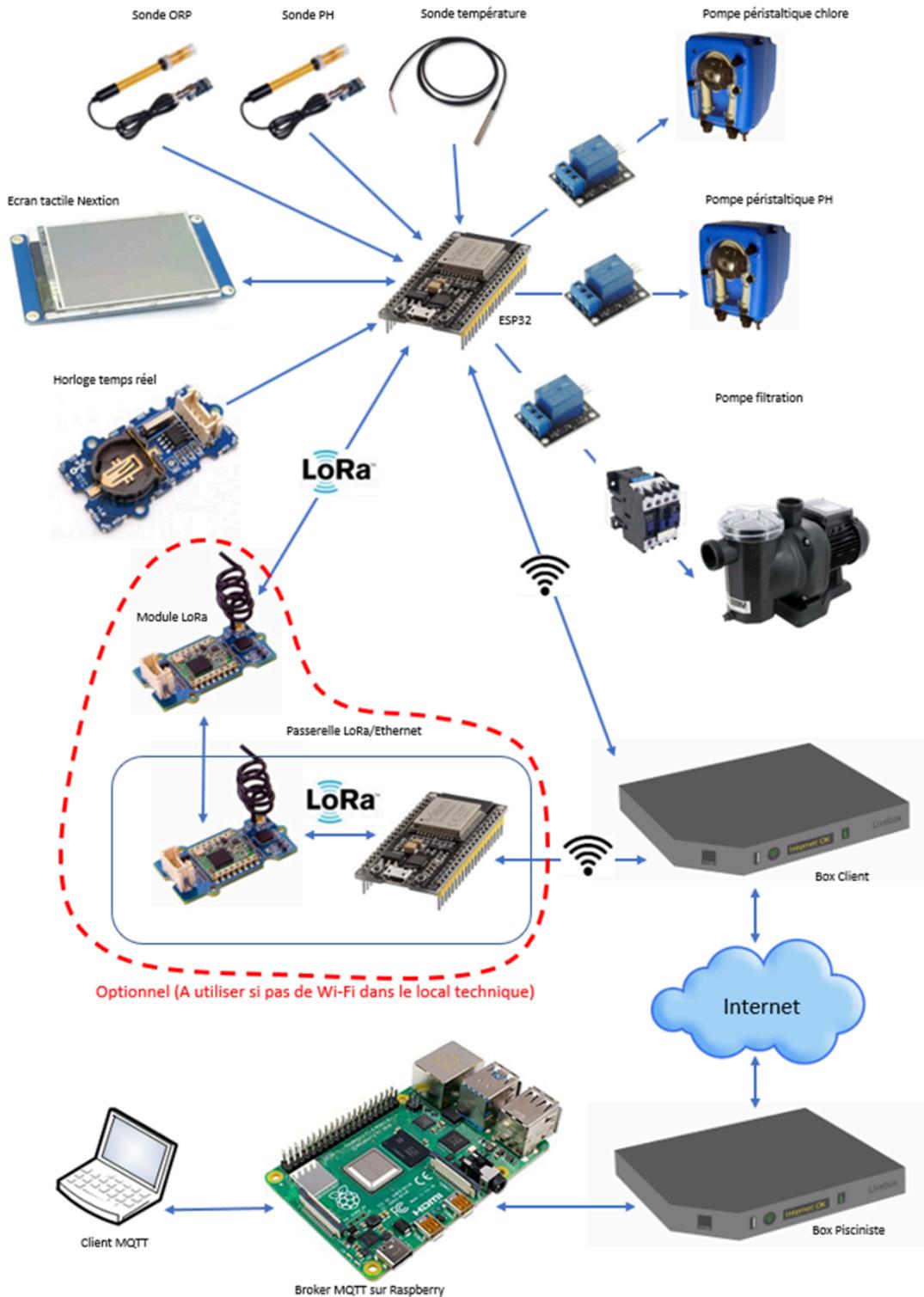
- - être installé chez le particulier.
- - être le serveur de l'entreprise.



2. Diagramme des cas d'utilisations



3. Solutions et technologies utilisées



Grâce à une interface connectée et à des capteurs de température, de chlore et de pH, les informations recueillies par ces capteurs seront directement accessibles via l'interface. Pour la communication entre ces composants, deux options sont possibles : une connexion via un broker MQTT ou via le Wi-Fi. Une fois les données des capteurs récupérées, l'utilisateur pourra déterminer les actions nécessaires pour maintenir l'eau de la piscine conforme aux normes, garantissant ainsi une eau saine.

Voici le matériel et logiciel que nous allons utiliser

- Un Ecran Nextion
- Une ESP 32
- Un Module LoRa
- Un broker MQTT
- Des capteurs (température, chlore et Ph)
- Une Pompe péristaltique

Ecran Nextion



l'écran nextion est un IHM (Human Machine Interface) qui nous permettra d'afficher une interface et grâce à celui si il nous permettra de lire les données que l'on aura récupéré en amont

ESP 32:

Un ESP32 est un microcontrôleur qui intègre Wifi et bluetooth. Il nous permettra de connecter nos capteurs dessus, la pompe,...



Le module LoRa :

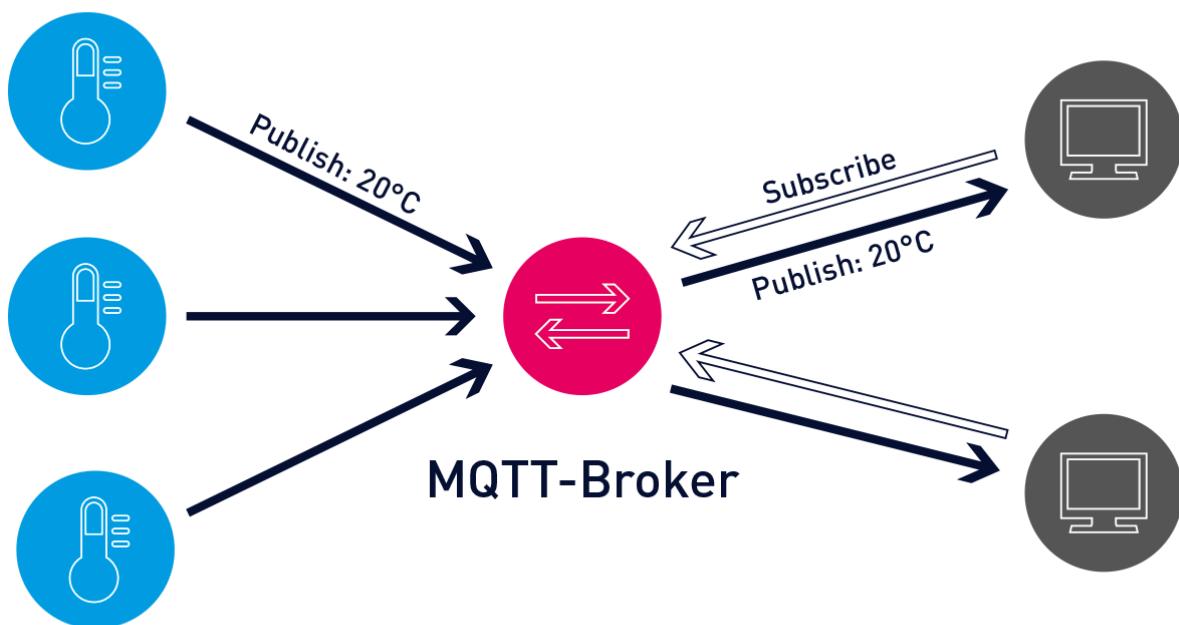
LoRaWAN est un protocole de communication radio fondé sur la technologie LoRa. Dans le cadre de l'Internet des objets, il permet de structurer un réseau étendu à basse consommation, intégrant des équipements terminaux à faible consommation électrique par l'intermédiaire de passerelles.



MQTT:

Nous utilisons également MQTT, un protocole de messagerie basé sur le modèle publish-subscribe et le protocole TCP/IP. Dans l'architecture MQTT, il y a deux types de systèmes : les clients et les brokers (courtiers). Le broker, qui est le serveur central, reçoit les messages des clients et les retransmet à d'autres clients. Les clients ne communiquent donc jamais directement entre eux, mais toujours via le broker. Chaque client peut être éditeur, abonné, ou les deux.

MQTT est un protocole orienté événements. Pour minimiser le nombre de transmissions, les données ne sont pas envoyées à des intervalles définis ni en continu. Un client publie uniquement lorsqu'il a des informations à transmettre, et le broker transmet ces informations aux abonnés uniquement lorsqu'il reçoit de nouvelles données.



Les Capteurs :

Nos différents capteurs sont le capteur de température, de Chlore et du PH. Leurs rôles sont de pouvoir prendre les mesures de ces différents paramètres afin de pouvoir les communiquer à l'interface.



La Pompe péristaltique :

Une pompe péristaltique, aussi appelée pompe à galets, est utilisée pour le pompage de liquides et de gaz. Le fluide, qu'il soit liquide ou gazeux, est contenu dans un tube flexible et est entraîné par un mécanisme qui presse le tube à l'intérieur de la pompe. La pompe péristaltique se compose d'une tête circulaire contenant un tube flexible par lequel le fluide circule. Le tube est déformé par un rotor équipé de rouleaux ou galets qui le compressent contre la tête circulaire. Les galets, en comprimant successivement des portions du tuyau lors de leur rotation, déplacent le fluide dans une direction constante. L'élasticité du tuyau permet l'aspiration du fluide à l'entrée de la pompe.

Le fluide n'entre jamais en contact avec le rotor, étant seulement en contact avec l'intérieur du tube. Cela élimine tout risque de contamination, corrosion, ou abrasion du rotor par des fluides agressifs ou chargés. Le déplacement du fluide est marqué par des pulsations causées par le passage des galets.



4. Diagramme de déploiement

Diagramme de déploiement avec wifi dans le local technique:

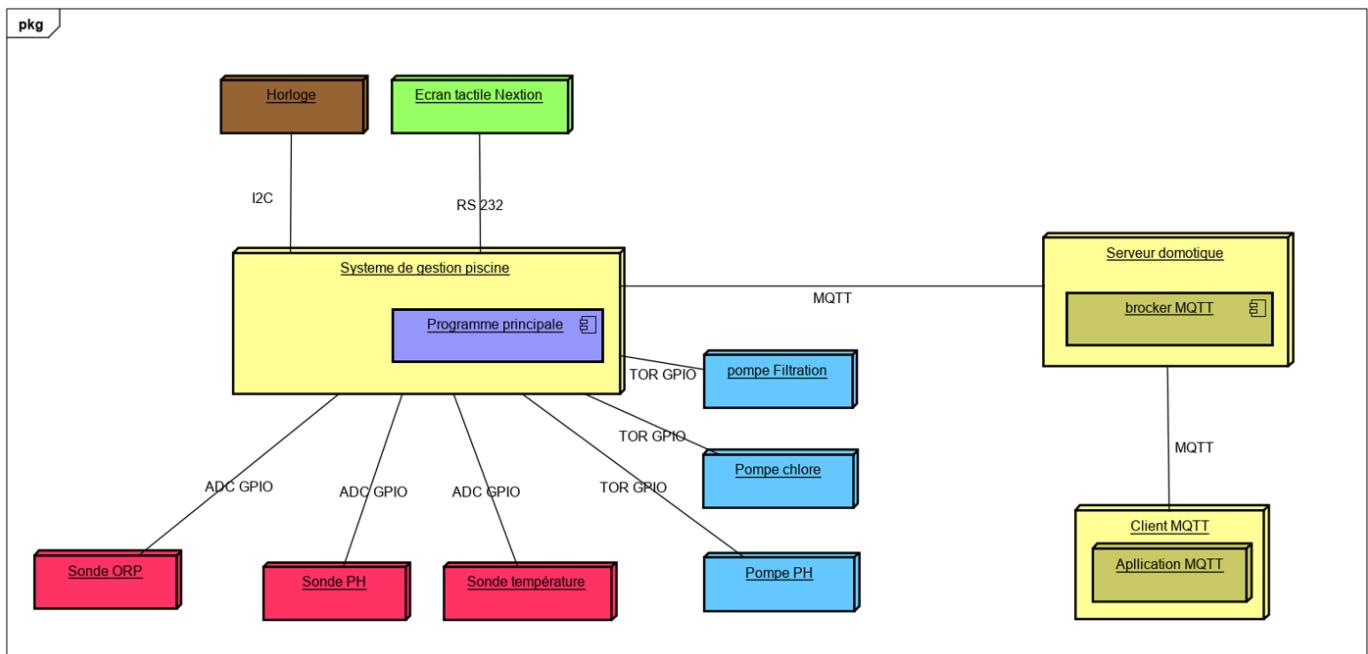
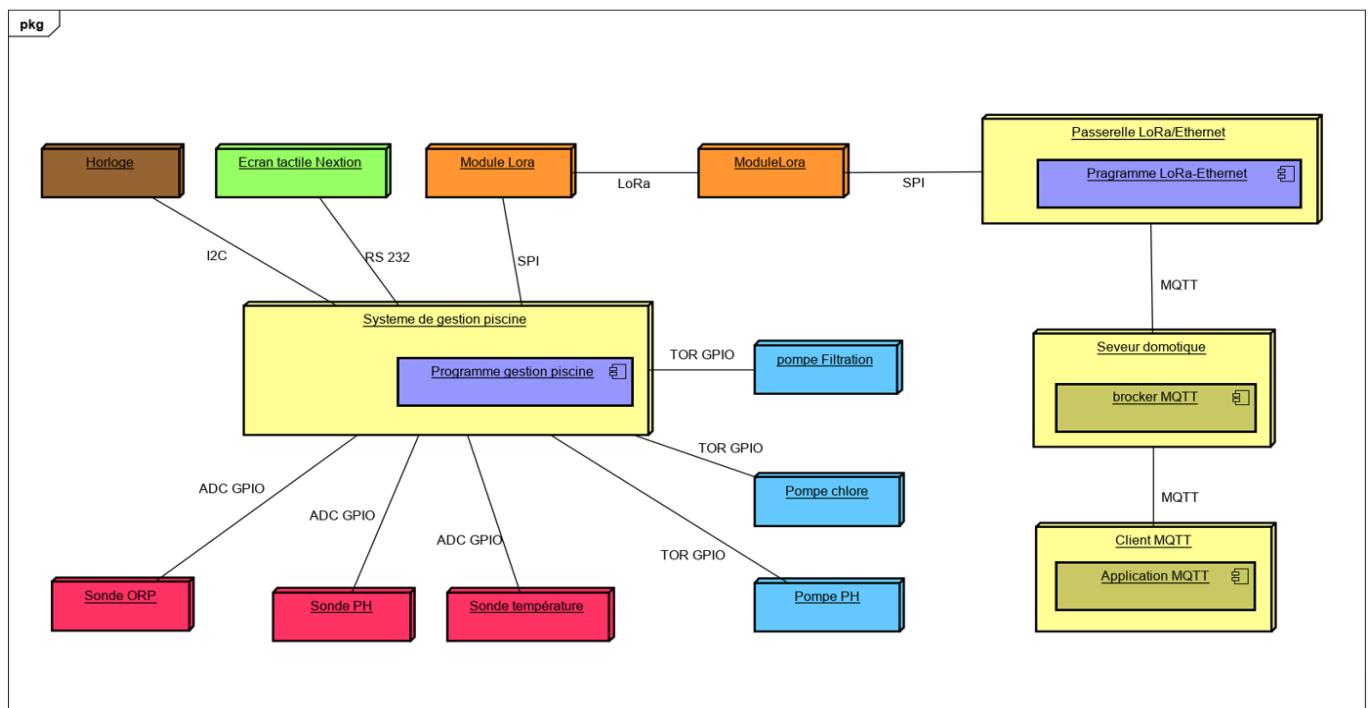
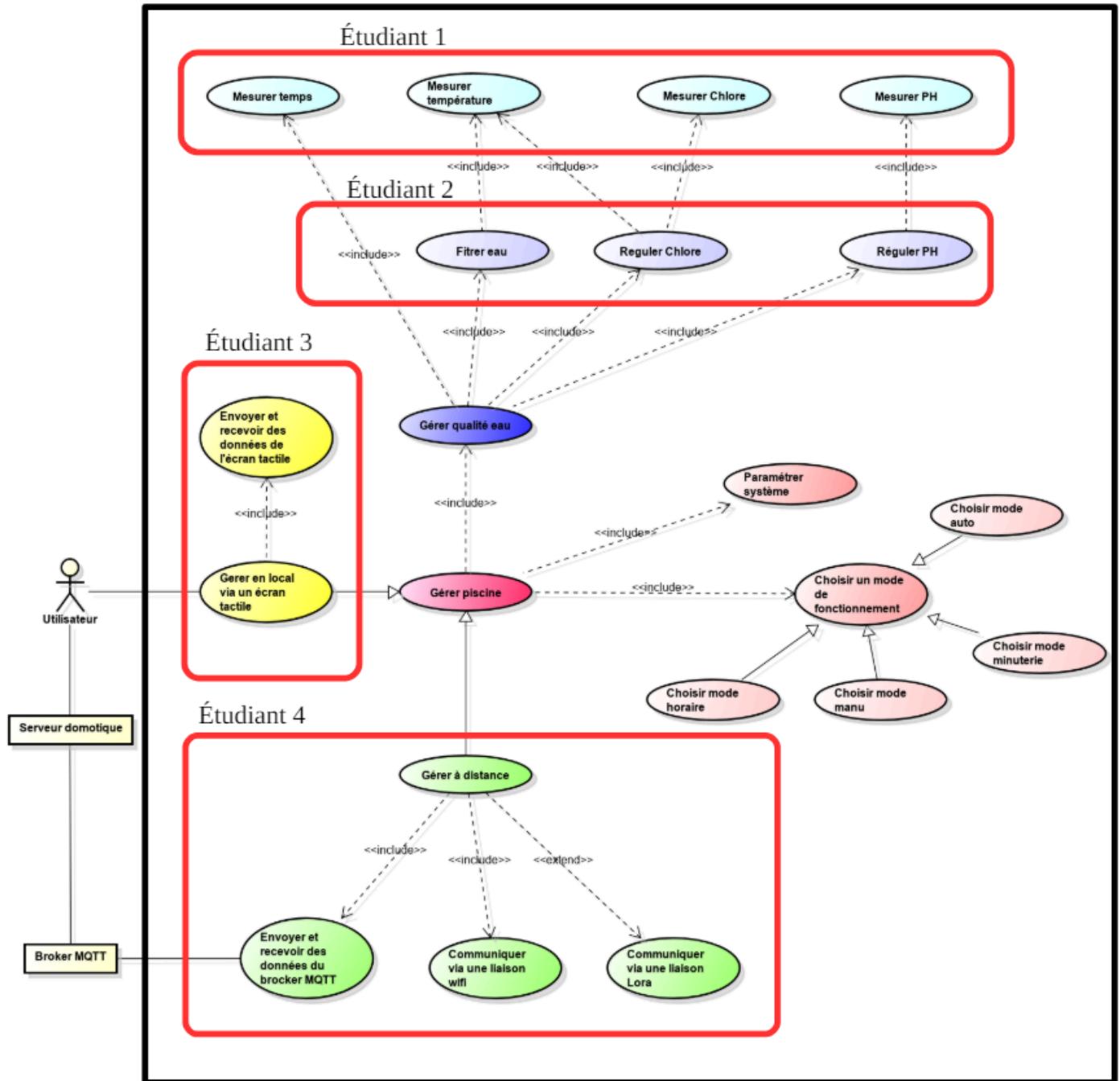


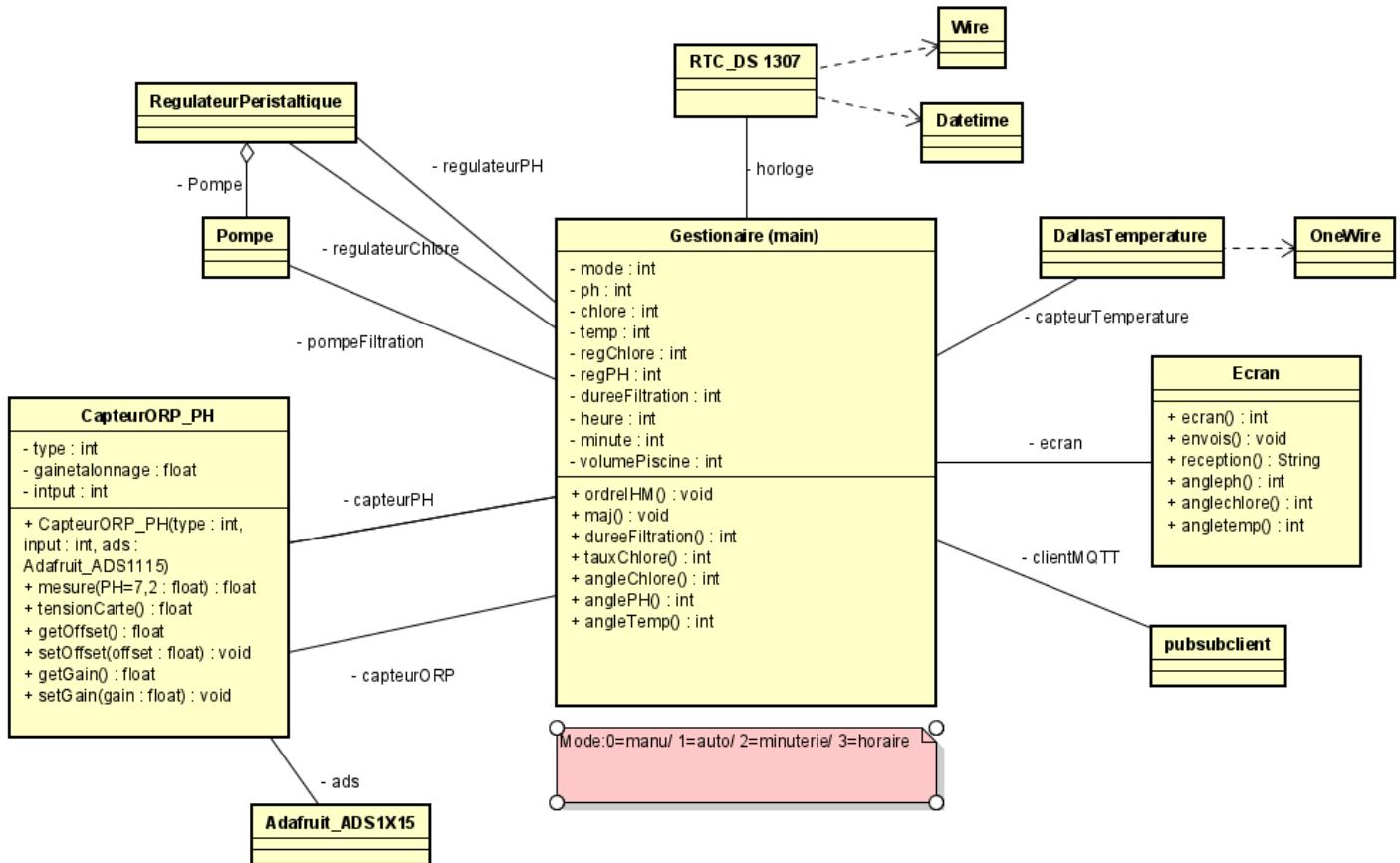
Diagramme de déploiement sans wifi dans le local technique:



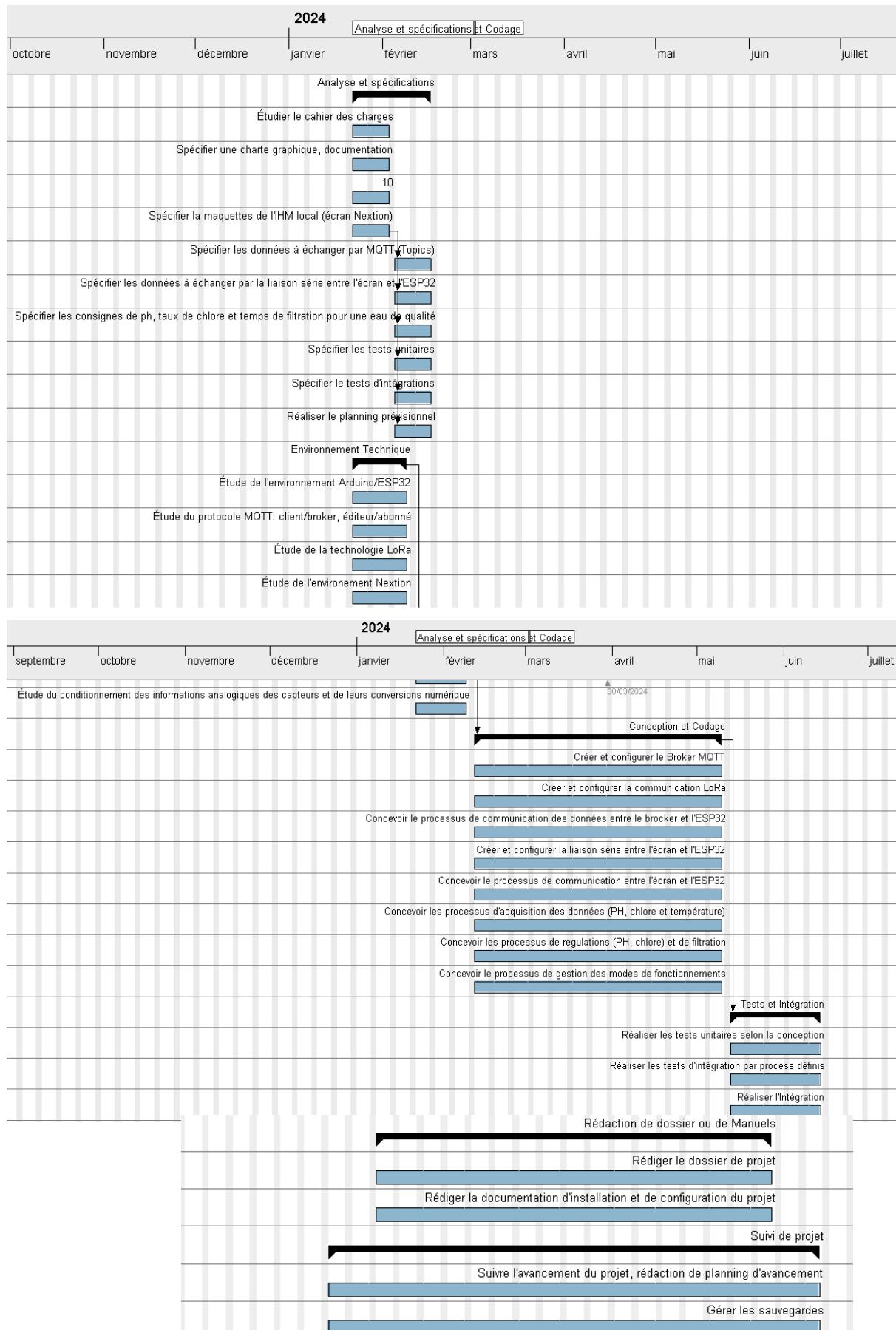
5. Répartition des tâches



6. Diagramme des cas d'utilisations



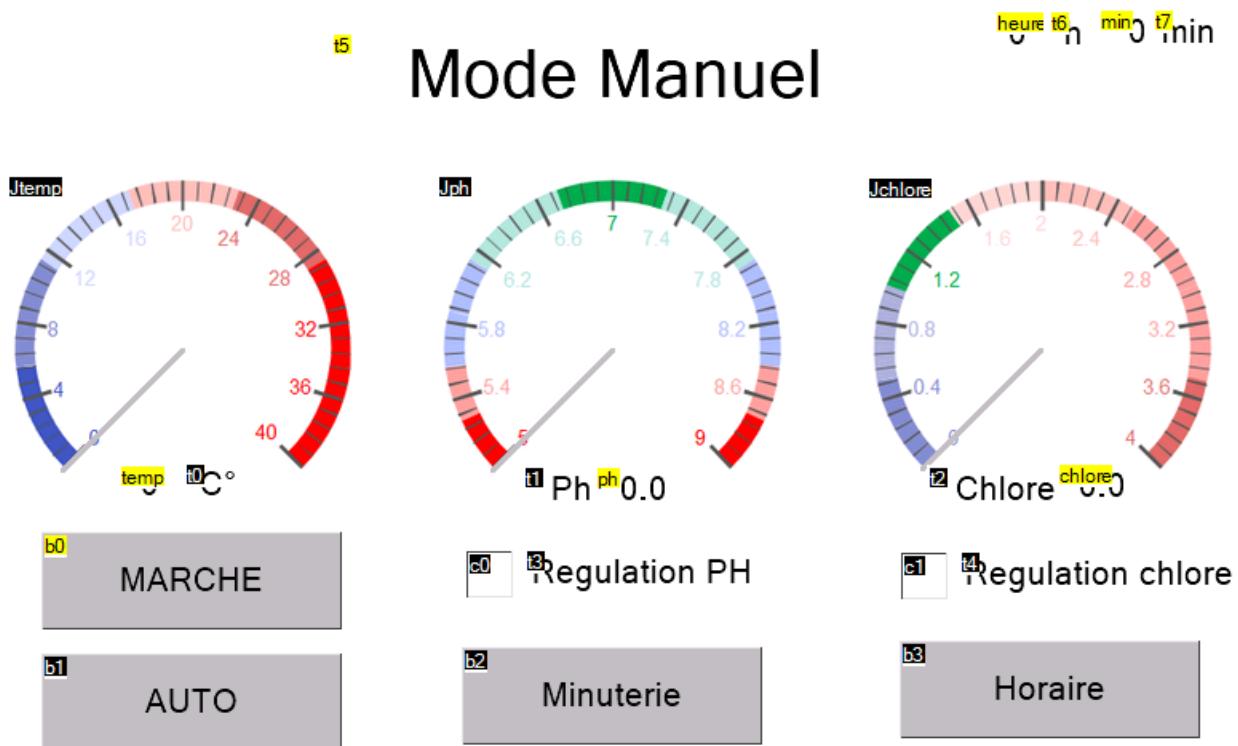
7. Planning prévisionnel



8. Recette partielle

L'objectif final de ce projet est de pouvoir contrôler la qualité de l'eau à travers une interface à distance ou en local à l'aide d'un écran tactile.

Actuellement, nous pouvons contrôler la piscine uniquement avec le mode manuel. À ce point si, on peut utiliser les pompes ainsi que recevoir les données des différents capteurs. L'interface est terminée et elle est simple d'utilisation. Les différents capteurs envoient les informations directement sur l'interface et l'utilisateur peut changer de mode comme il le souhaite depuis l'IHM.



2. Partie individuelle (DUCAROUGE Valentin)

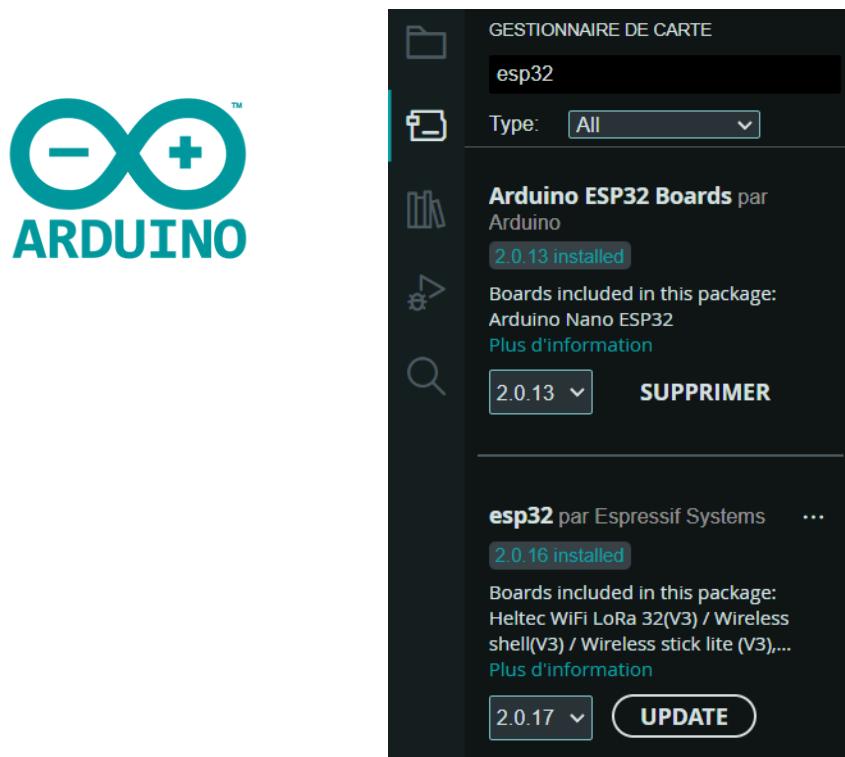
1. Introduction

Dans ma partie, j'ai dû gérer la collecte d'informations telle que la température, le pH, le chlore ainsi que l'heure à travers différents capteurs. Pour cela, j'ai développé un programme en C++ sur le logiciel Arduino avec différentes classes pour mener à bien l'échange d'informations, cela me permettant de gérer la communication entre les différents capteurs et la carte électronique ESP32.

2. Familiarisation avec le logiciel Arduino

Afin de me familiariser avec ce logiciel, j'ai tout d'abord commencé par apprendre à allumer et éteindre une LED automatiquement puis avec l'aide d'un bouton.

Pour effectuer ce programme, en amont j'ai dû configurer la carte en sélectionnant une carte dans l'IDE Arduino pour lui faire comprendre sur quelle carte on transférera le programme plus tard. Pour cela, je suis allé dans le gestionnaire de carte et j'ai installé plusieurs lots de cartes pour l'esp32.



Ensuite j'ai testé plusieurs cartes différentes afin de trouver la bonne pour notre ESP32. La bonne carte pour notre ESP32 était "ESP32 Dev Module".

3. Protocole de communication

La liaison série définie dans notre cas la relation entre le port série du PC et la carte ESP32 via une vitesse de transmission fixé à 9600 bauds. Cette liaison série nous permettra durant nos tests, de pouvoir faire afficher dans le moniteur série différentes valeurs récupérées par nos capteurs.

4. Capteur de température

Pour s'assurer du bon fonctionnement du capteur de température, on doit inclure dans le code plusieurs librairies :

```
#include <OneWire.h>
#include <DallasTemperature.h>
```

Une fois ces librairies déclarées, on doit définir sur quelle broche notre capteur de température est branché :

```
#define ONE_WIRE_BUS 4
```

Dans mon cas, il est branché sur la broche numéro 4. Ensuite, je vais créer un objet sensors qui me permettra d'utiliser les commandes contenues dans la librairie DallasTemperature.

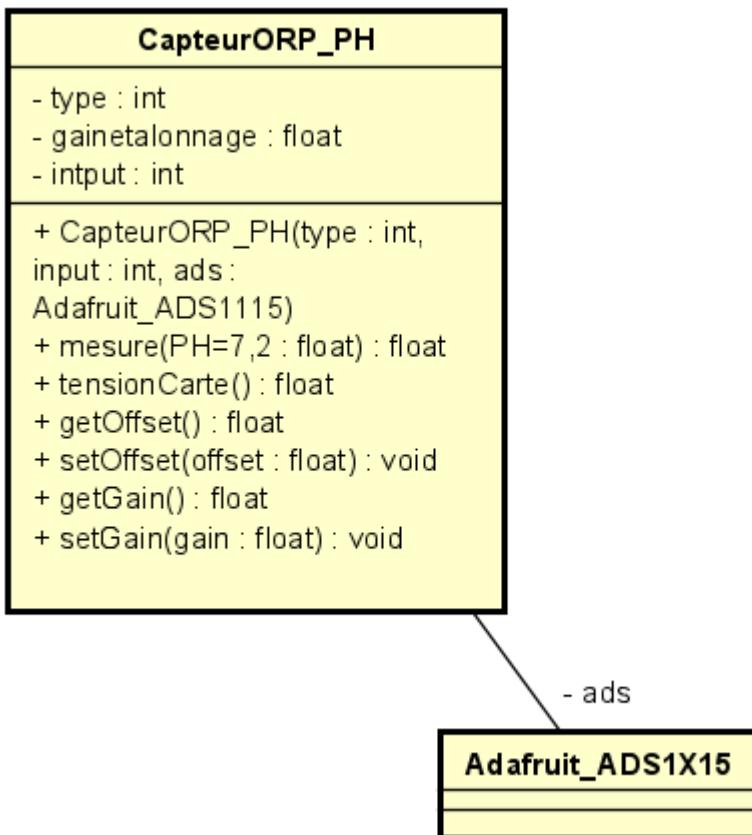
```
DallasTemperature sensors(&oneWire);
```

Ensuite pour demander la température et la faire afficher on procède de cette manière :

```
void loop(void){
    // Envoie une requête pour récupérer la température
    sensors.requestTemperatures();
    Serial.print("Celsius temperature: ");
    // Affiche la température en degrés Celsius
    Serial.print(sensors.getTempCByIndex(0));
    delay(1000);
}
```

En mettant cette demande dans le void loop, on indique que l'on veut que ce programme soit exécuté en permanence. Donc pour une meilleure lisibilité, on met une pause d'une seconde entre chaque demande.

5. Analyse de la classe CapteurORP.PH



La classe **CapteurORP_PH** nous permet de récupérer les données des capteurs de pH et Chlore, sachant que ces capteurs ne sont pas numériques, mais analogiques. On doit donc effectuer une conversion Analogique/Numérique. On y retrouve le constructeur **CapteurORP_PH** ainsi que les méthodes **tensioncarte**, **mesure** ainsi que tous ceux qui possèdent un plus avant le nom.

type: permet de choisir quelle valeur on souhaite obtenir entre le pH et le chlore.

input: sert à déterminer quelles entrées du Grove - ADS1115 on va utiliser, pour cela on le met à 0 si on veut calculer le pH et 1 si on veut le chlore.

gainetalonnage : permet de garantir une mesure des valeurs précises et fiables.

mesure : permet de mesurer et de récupérer les différentes valeurs émises par les capteurs pour la température, le pH et le chlore.

tensionCarte : permet de calculer une tension différente suivant si input est à 0 ou à 1.

setoffset: ajuste la correction des erreurs de base des capteurs.

getoffset: récupère la valeur actuelle de la correction pour des mesures précises de température, pH et chlore.

setgain: ajuste la sensibilité des capteurs

getgain: récupère la valeur actuelle du gain pour des mesures précises de température, pH et chlore.

6. Mise en œuvre de la classe CapteurORP.PH

Classe CapteurORP_PH :

Méthode mesure :

```

float CapteurORPPH::mesure(float pH) {
    if(input==0) // 0 = fonction ph, 1 = fonction ppm
    {
        float voltage = tensionCarte();
        // Interpolation linéaire pour convertir la tension en pH
        float slope = 14.0 / 0.828; // 14 / (0.414 * 2)
        float intercept = 7.0;
        float pH = slope * voltage + intercept;

        // Vérification de la plage de pH
        if (pH < 0.0) {
            pH = 0.0;
        } else if (pH > 14.0) {
            pH = 14.0;
        }
        return pH;
    }
    else
    {
        float voltage = tensionCarte(); // récupération de la tension
        float ppm = 0; // Initialisez ppm à 0
        voltage=voltage*1000; // conversion de la tension en V
        if (pH < 6.45 || pH > 14) { // si le pH est en dehors de cette
plage, renvoie "-1" pour dire qu'il n'y a pas de ppm
            return -1;
        }
        else if (pH >= 6.45 && pH <= 6.55) { // calcul du ppm en fonction
de la plage du pH calculé précédemment
            return 0.0053 * exp(0.0071 * voltage);
        }
        else if (pH >= 6.55 && pH <= 6.65) {
            return 0.0051 * exp(0.0072 * voltage);
        }
        else if (pH >= 6.65 && pH <= 6.75) {
            return 0.0048 * exp(0.0073 * voltage);
        }
        else if (pH >= 6.75 && pH <= 6.85) {
            return 0.0046 * exp(0.0074 * voltage);
        }
        else if (pH >= 6.85 && pH <= 6.95) {
            return 0.0044 * exp(0.0076 * voltage);
        }
    }
}

```

```

    }

    else if (pH >= 6.95 && pH <= 7.05) {
        return 0.0041 * exp(0.0077 * voltage);
    }

    else if (pH >= 7.05 && pH <= 7.15) {
        return 0.0039 * exp(0.0078 * voltage);
    }

    else if (pH >= 7.15 && pH <= 7.25) {
        return 0.0037 * exp(0.0079 * voltage);
    }

    else if (pH >= 7.25 && pH <= 7.35) {
        return 0.0035 * exp(0.0081 * voltage);
    }

    else if (pH >= 7.35 && pH <= 7.45) {
        return 0.0033 * exp(0.0082 * voltage);
    }

    else if (pH >= 7.45 && pH <= 7.55) {
        return 0.0031 * exp(0.0084 * voltage);
    }

    else if (pH >= 7.55 && pH <= 7.65) {
        return 0.0029 * exp(0.0085 * voltage);
    }

    else if (pH >= 7.65 && pH <= 7.75) {
        return 0.0027 * exp(0.0087 * voltage);
    }

    else if (pH >= 7.75 && pH <= 7.85) {
        return 0.0025 * exp(0.0089 * voltage);
    }

    else if (pH >= 7.85 && pH <= 7.95) {
        return 0.0024 * exp(0.009 * voltage);
    }

    else if (pH >= 7.95 && pH <= 8.05) {
        return 0.0022 * exp(0.0092 * voltage);
    }

    else{
        return 0.0;
    } }
}

```

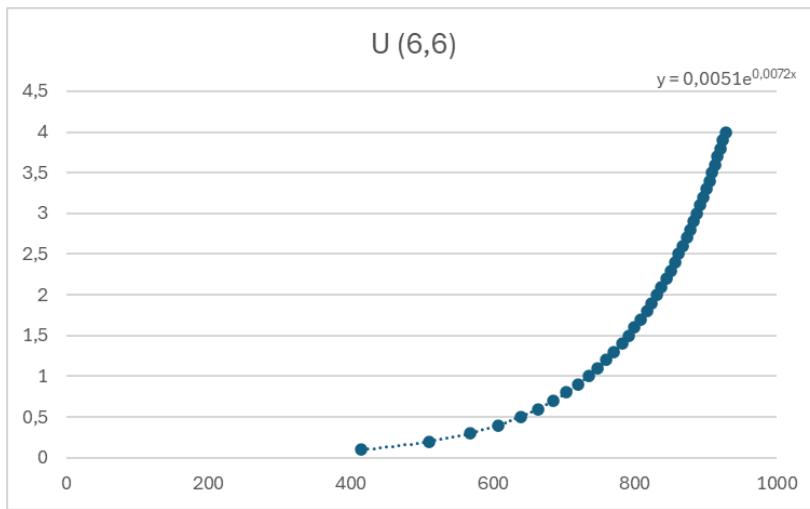
La méthode “mesure” est longue car elle effectue deux tâches distinctes : le calcul du pH et le calcul du ppm. Le calcul du ppm est particulièrement complexe car il dépend de la plage du pH mesuré, nécessitant des conditions multiples et des formules spécifiques pour chaque

intervalle de pH. Cette variabilité et la multitude de cas à traiter rendent le processus plus détaillé et étendu.

Voici le tableau ainsi qu'un graphique qui m'a permis de trouver toute les valeurs en ppm suivant le pH :

PPM	U (6,5)	U (6,6)	U (6,7)	U (6,8)	U (6,9)	U (7)	U (7,1)	U (7,2)	U (7,3)	U (7,4)	U (7,5)	U (7,6)	U (7,7)	U (7,8)	U (7,9)	U (8)
0,1	415	415	415	415	415	415	415	415	415	415	415	415	415	415	415	415
0,2	513	511	510	508	507	505	504	502	501	499	498	496	495	493	492	490
0,3	570	568	565	563	561	558	556	553	551	549	546	544	541	539	537	534
0,4	611	608	605	602	599	596	593	590	587	584	581	578	575	572	569	566
0,5	642	639	635	632	628	625	621	618	614	611	607	604	600	597	593	590
0,6	668	664	660	656	652	648	645	641	637	633	629	625	621	617	613	610
0,7	690	685	681	677	673	669	664	660	656	652	647	643	639	635	631	626
0,8	709	704	699	695	690	686	681	677	672	668	663	659	654	650	645	641
0,9	725	720	716	711	706	701	697	692	687	682	677	673	668	663	658	654
1	740	735	730	725	720	715	710	705	700	695	690	685	680	675	670	665
1,1	753	748	743	738	733	727	722	717	712	707	701	696	691	686	681	675
1,2	766	760	755	750	744	739	733	728	723	717	712	706	701	696	690	685
1,3	777	771	766	760	755	749	744	738	732	727	721	716	710	705	699	693
1,4	787	782	776	770	765	759	753	747	742	736	730	724	719	713	707	702
1,5	797	791	785	780	774	768	762	756	750	744	738	733	727	721	715	709
1,6	806	800	794	788	782	776	770	764	758	751	746	740	734	728	722	715
1,7	815	809	803	796	790	784	778	772	766	760	753	747	741	735	729	723
1,8	823	817	810	804	798	792	785	779	773	766	760	754	748	741	735	729
1,9	831	824	818	811	805	799	792	786	779	773	767	760	754	747	741	735
2	838	831	825	818	812	805	799	792	786	779	773	766	760	753	747	740
2,1	845	838	831	825	818	812	805	798	792	785	779	772	765	759	752	746
2,2	851	845	838	831	824	818	811	804	798	791	784	777	771	764	757	751
2,3	858	851	844	837	830	824	817	810	803	796	789	783	776	769	762	755
2,4	864	857	850	843	836	829	822	815	808	801	795	788	781	774	767	760
2,5	868	862	855	848	841	834	827	820	813	806	799	792	785	778	771	764
2,6	875	868	861	854	847	839	832	825	818	811	804	797	790	783	776	769
2,7	880	873	866	859	852	844	837	830	823	816	809	801	794	787	780	773
2,8	885	878	871	864	856	849	842	835	827	820	813	806	798	791	784	777
2,9	890	883	876	868	861	854	846	839	832	824	817	810	803	795	788	781
3	895	888	880	873	866	858	851	843	836	829	821	814	806	799	792	784
3,1	900	892	885	877	870	862	855	847	840	833	825	818	810	803	795	788
3,2	904	897	889	882	874	867	859	851	844	836	829	821	814	806	799	791
3,3	909	901	893	886	878	871	863	855	848	840	833	825	817	810	802	795
3,4	913	905	897	890	882	874	867	859	851	844	836	828	821	813	806	798
3,5	917	909	901	894	886	878	871	863	855	847	840	832	824	816	809	801
3,6	921	913	905	897	890	882	874	866	859	851	843	835	827	820	812	804
3,7	925	917	909	901	893	885	878	870	862	854	846	838	831	823	815	807
3,8	928	921	913	905	897	889	881	873	865	857	849	842	834	826	818	810
3,9	932	924	916	908	900	892	884	876	868	860	853	845	837	829	821	813
4	936	928	920	912	904	896	888	880	872	864	856	848	840	832	824	816

U correspond au pH pris en compte. Voici un graphique pour obtenir le ppm pour un pH compris entre 6,55 et 6,65.



Méthode tensionCarte:

```
float CapteurORPPH::tensionCarte() {
    // Implémentation de la fonction tensionCarte
}
```

```

if(input==1)
    return ads->computeVolts(ads->readADC_Differential_2_3()); // récupère la tension pour le pH
else
    return ads->computeVolts(ads->readADC_Differential_0_1()); // récupère la tension pour le ppm
}

float CapteurORPPH::getOffset() {
    return offset;
}

```

Méthode offset:

```

float CapteurORPPH::getOffset() {
    return offset;
}

void CapteurORPPH::setOffset(float offset) {
    this->offset=offset;
}

```

Méthode gain:

```

float CapteurORPPH::getGain() {
    // Implémentation de la fonction getGain
    return 0;
}

void CapteurORPPH::setGain(float gain) {
    // Implémentation de la fonction setGain
}

```

7. Difficultés rencontrées

Pour moi, la plus grosse difficulté était de calculer le taux de chlore en ppm, car cela nécessite de gérer de nombreuses conditions et de choisir la formule appropriée en fonction de la plage de pH mesurée, ce qui rend le code complexe et sujet à erreurs.

8. Conclusion personnelle

Le projet "Piscine tranquille - CQEP" pour l'entreprise CORTES PISCINES & SPAS représente un défi technique et intellectuel significatif. Ce projet m'a permis de développer et d'appliquer mes compétences en électronique, programmation et intégration de systèmes embarqués. L'une des plus grandes difficultés que j'ai rencontrées a été de calculer le taux de chlore en ppm. Ce calcul est complexe car il varie en fonction de la plage de pH mesurée, nécessitant la gestion de nombreuses conditions et l'application de formules spécifiques. Cela a rendu le code long et détaillé, mais également précis et adapté aux besoins spécifiques du projet.

En conclusion, ce projet a été une expérience enrichissante et formatrice. Il m'a permis de renforcer mes compétences techniques et de développer une approche méthodique et rigoureuse pour la résolution de problèmes complexes.

3. Partie individuelle (OTT Mathieu)

1. Introduction

Durant ma partie j'ai eu pour tâche la gestion de la filtration, ainsi que la régulation du chlore et du pH de la piscine. Cela implique la mise en place d'algorithmes et de mécanismes de contrôle permettant de maintenir ces paramètres à des niveaux optimaux pour assurer la qualité de l'eau pour le client.

Pour la gestion de la filtration, j'ai dû élaborer un algorithme qui prend en compte les plages de fonctionnement définies en matière de température de l'eau. Ces algorithmes permettent de déterminer la durée de filtration nécessaire pour chaque plage de températures, en assurant une filtration efficace et économique en énergie.

Plage	Température de l'eau	Temps de filtration
1	<10°C	1h
2	Entre 10°C et 14°C	2h
3	Entre 15°C et 18°C	3h
4	Entre 18°C et 20°C	5h
5	Entre 20 et 24°C	7h
6	Entre 25 et 28°C	10h
7	>28°C	12h

En ce qui concerne la régulation du chlore et du pH, j'ai mis en place des mécanismes pour surveiller en continu les niveaux de ces produits chimiques dans l'eau de la piscine. En fonction des mesures fournies par les capteurs, notre système pourra ajuster automatiquement l'injection de produits chimiques pour maintenir les niveaux optimaux.

Plage	Température	Temps de filtration	Chlore	pH
1	<10°C	1h	0ppm	7.2
2	Entre 10°C et 14°C	2h	0.5ppm	7.2
3	Entre 15°C et 18°C	3h	1ppm	7.2
4	Entre 18°C et 20°C	5h	1.3ppm	7.2
5	Entre 20 et 24°C	7h	1.5ppm	7.2
6	Entre 25 et 28°C	10h	1.5ppm	7.2
7	>28°C	12h	1.5ppm	7.2

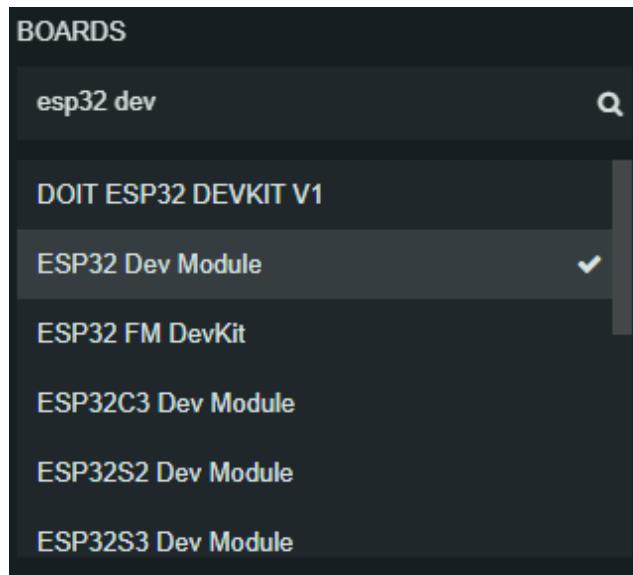
Par exemple, lorsque le niveau de chlore est trop bas, notre système peut déclencher l'ajout de chlore pour désinfecter l'eau, tandis que des ajustements similaires peuvent être effectués pour maintenir le pH dans la plage souhaitée.

2. Développement et transfert de programme pour ESP32

Pour faire cela, j'ai programmé en C++ à l'aide du logiciel arduino plusieurs classes pour me permettre de contrôler les pompes ainsi que leur régulation. J'ai ensuite transféré le programme sur une carte ESP 32.

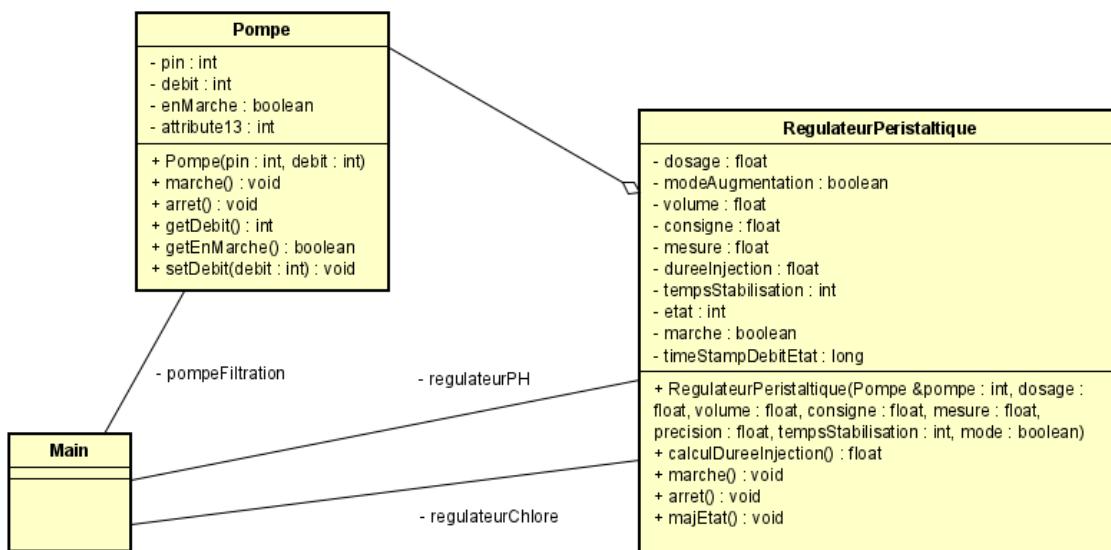
Tout d'abord, n'ayant jamais utilisé le logiciel Arduino, j'ai dû apprendre à l'utiliser en allumant une LED avec une carte ESP32, une plaque d'essai et des fils de connexion.

Pour transférer mon programme sur l'ESP32, j'ai dû configurer la carte ESP32 en sélectionnant la carte "ESP32 Dev Module" et en choisissant le port USB qui est relié à l'ESP32.



3. Analyse des classes Pompe et RegulateurPeristaltique

Diagramme de classe :



Classe Pompe :

La classe Pompe est utile pour gérer le fonctionnement de la pompe. Elle utilise plusieurs méthodes :

Méthode **marche** : elle permet de mettre en marche la pompe.

Méthode **arrêt** : elle permet d'arrêter la pompe.

Méthode **setDebit** : elle permet de définir la valeur du débit

Méthode **getDebit** : elle permet de renvoyer la valeur du débit

Méthode **getEnMarche** : elle permet de renvoyer 0 ou 1 en fonction de l'état de fonctionnement de la pompe

Classe regulateurPeristaltique :

Méthode **marche** : elle permet de mettre en marche le régulateur péristaltique.

Méthode **arrêt** : elle permet d'arrêter le régulateur péristaltique.

Méthode **calculDureeInjection** : Renvoie la durée d'injection de la pompe en seconde pour injecter la bonne quantité de produit.

Méthode **majEtat** : elle permet de passer d'un état à un autre

4. Fonctionnement des pompes péristaltiques

Le régulateur péristaltique fonctionne à travers 5 états exprimés ci-dessous :

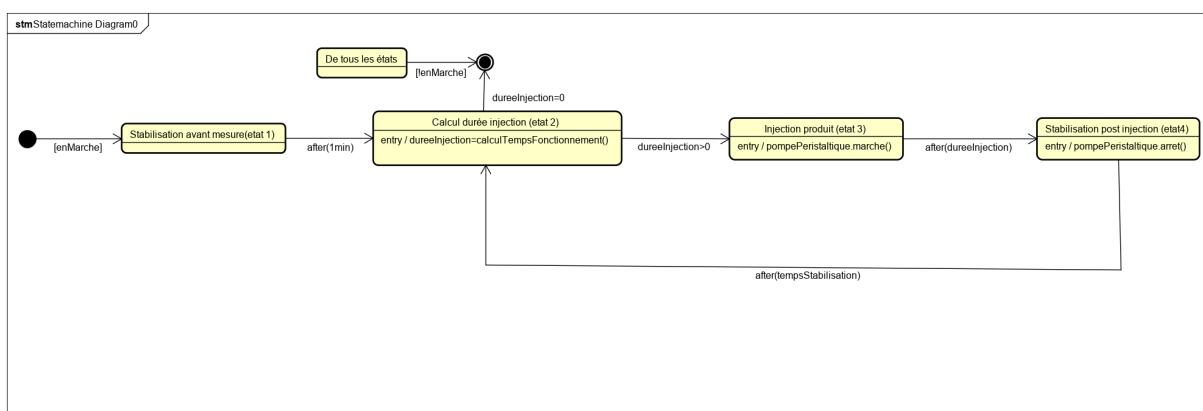
Etat 0 : Pompe à l'arrêt.

Etat 1 : Stabilisation avant mesure

Etat 2 : Calcul de la durée d'injection (du temps de fonctionnement de la pompe) -> pompe péristaltique à l'arrêt.

Etat 3 : Injection du produit -> pompe péristaltique en marche.

Etat 4 : Stabilisation post-injection



Ce diagramme montre que dans chaque état, lorsque le régulateur péristaltique est à l'arrêt, le programme s'arrête. Lorsque celui-ci se met en marche, il passe dans l'état 1 (état de stabilisation avant la mesure). Après 1 minute de stabilisation, le régulateur péristaltique passe dans l'état 2 où le calcul de la durée d'injection se lance. Ensuite, si la durée d'injection est égale à 0, le programme s'arrête : la qualité de l'eau est bonne. Mais si la

durée d'injection est supérieure à 0, le régulateur péristaltique passe dans l'état 3 et injecte donc le produit pendant une certaine durée calculée dans l'état 2 : la pompe se met donc en marche. Enfin, une fois la durée d'injection terminée, le régulateur péristaltique passe dans l'état 4 qui est un état de stabilisation de post-injection. Pour finir, après un certain temps de stabilisation, le régulateur péristaltique repasse dans l'état 2 et refait donc un calcul de durée d'injection pour déterminer s'il faut réinjecter du produit ou non.

5. Mise en oeuvre des classes Pompe et regulateurPeristaltique

Classe Pompe :

Méthode **marche** :

```
void Pompe::marche() {
    digitalWrite(pin, LOW);
    enMarche = true;
}
```

Méthode **arrêt** :

```
void Pompe::arret() {
    digitalWrite(pin, HIGH);
    enMarche = false;
}
```

Méthode **setDebit** :

```
void Pompe::setDebit(int nouveauDebit) {
    debit = nouveauDebit;
}
```

Méthode **getDebit** :

```
int Pompe::getDebit() const {
    return debit;
}
```

Méthode **getEnMarche** :

```
bool Pompe::getEnMarche() const {
```

```

    return enMarche;
}

```

Classe regulateurPeristaltique :

Méthode marche :

```

void RegulateurPeristaltique::marche()
{
    pompe->marche();
    enMarche = true;
}

```

Méthode arrêt :

```

void RegulateurPeristaltique::arret()
{
    pompe->arret();
    enMarche = false;
}

```

Méthode calculDureeInjection :

```

float RegulateurPeristaltique::calculDureeInjection()
{
    int ecart = consigne - mesure;
    float dureeInjection = (3.6 * ecart * volume *
dosage) / pompe->getDebit();
    return dureeInjection;
}

```

Méthode majEtat :

```

void RegulateurPeristaltique::majEtat()
{
    if (etat == 0 && enMarche)
    {
        etat = 1;
        timeStamp = millis();
    }
    if ((etat == 1 && millis() - timeStamp > 60000) || (etat == 4 &&
millis() - timeStamp > tempsStabilisation*1000))
    {
        etat = 2;
        timeStamp = millis();
        dureeInjection = calculDureeInjection();
    }
}

```

```
if (etat == 2 && dureeInjection > 0)
{
    etat = 3;
    timeStamp = millis();
    pompe->marche();
}
if (etat == 3 && millis() - timeStamp > dureeInjection*1000)
{
    etat = 4;
    timeStamp = millis();
    pompe->arret();
}
if (etat == 2 && dureeInjection == 0 || !enMarche)
{
    etat = 0;
    pompe->arret();
}
}
```

6. Difficultées rencontrées

Au début j'ai eu un peu de mal à prendre en main le logiciel Arduino, j'avais du mal à comprendre comment transmettre le code à la carte ESP32. Ensuite j'ai aussi rencontré des difficultés pour trouver la formule pour calculer la durée d'injection nécessaire pour rétablir un niveau de chlore, de filtration ou de pH optimal.

7. Conclusion personnelle

Je suis personnellement satisfait de l'avancement du projet jusqu'à présent et confiant dans notre capacité à livrer une solution efficace et fiable qui répondra aux attentes du client. Travailler sur ce projet m'a permis d'approfondir mes connaissances en conception d'algorithmes et de systèmes de contrôle, tout en me donnant l'opportunité de collaborer avec une équipe. J'ai hâte de voir le résultat final de notre travail et de contribuer au succès de ce projet innovant.

4. Partie individuelle (CELLIER Nathan)

1. Introduction

Afin de pouvoir réaliser ma partie, il m'a fallu apprendre à utiliser l'application Nextion Editor ainsi que ses différents composants pour ensuite être en capacité de créer une interface homme-machine (IHM) fonctionnelle et la mettre sur l'écran Nextion.

Cette IHM servira par la suite d'intermédiaire entre l'utilisateur et la piscine qui y sera connecté.

2. Étude de Solution

L'outil Nextion Editor permet de créer des interfaces homme-machine (IHM) que l'on peut ensuite importer directement sur l'écran, dans ce cas, un écran Nextion. Cette application offre de nombreux composants pour la création de l'interface, tels que des boutons, des textes, des barres de progression, des images, etc.

Une fois les objets nécessaires pour le projet sélectionné, il suffit de les placer selon les besoins. L'avantage de cet outil réside dans la grande liberté qu'il offre pour le choix et la disposition des éléments, permettant de nombreuses possibilités tout en étant facile à utiliser. De plus, la gestion d'événements facilite le codage ultérieur, le rendant plus rapide et moins laborieux.

Le choix de cette solution s'est imposé naturellement, car nous avions peu d'options disponibles. Il nous fallait une application compatible avec l'écran Nextion dont nous disposions pour le projet.

3. La liaison série

Pour communiquer entre nextion editor et l'ide arduino nous passerons par la liaison série et pour pouvoir faire fonctionner les deux il faut suivre une trame.

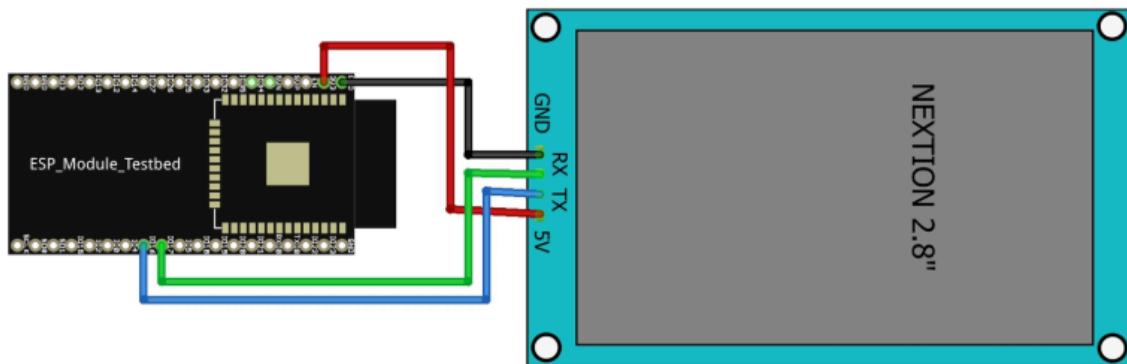
La liaison série entre Nextion Editor et Arduino permet de communiquer efficacement pour contrôler l'affichage sur l'écran Nextion et réagir aux interactions utilisateur.

La trame est constituée de trois 0xFF qui terminent chaque commande envoyée à l'écran.

4. Nextion Editor

L'interface créée grâce à Nextion se repose essentiellement sur de la programmation objet. De ce fait, on peut aussi gérer les évènements des différents objets. Une fois l'interface créée avec son code sur Arduino effectuée, il faut pouvoir ensuite communiquer avec le système. Pour ce faire, je mets en place une liaison entre l'écran Nextion et l'USP32.

schéma du montage :



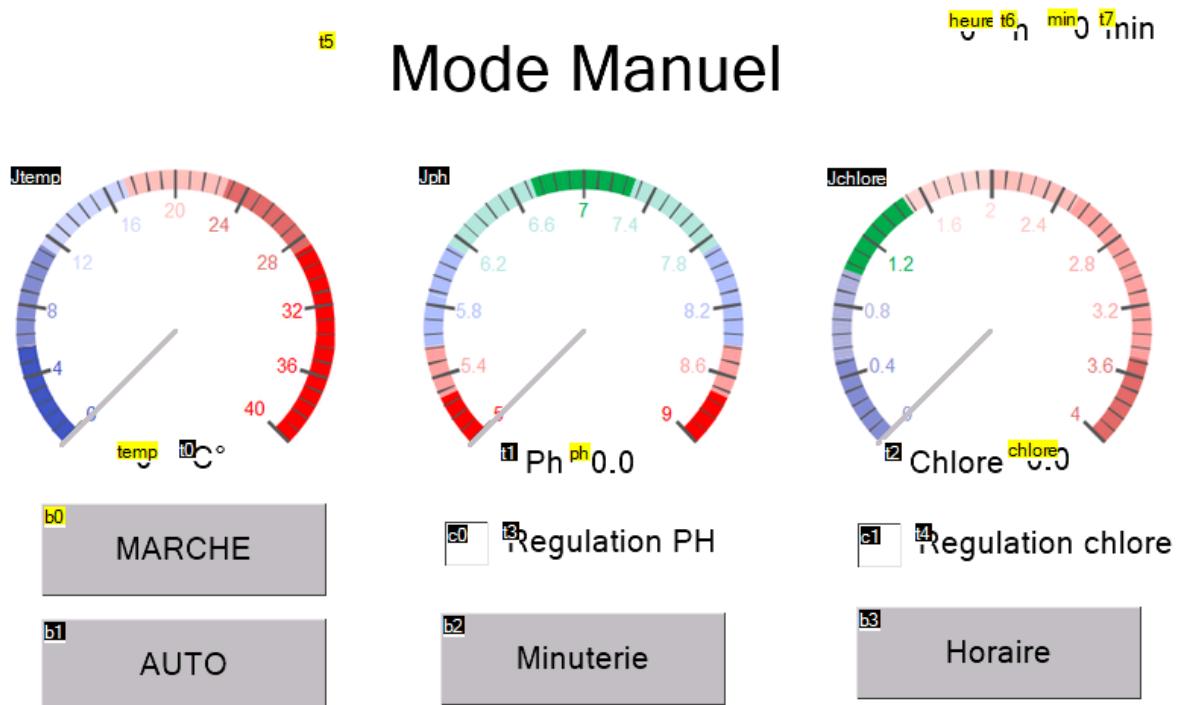
5. Possibiliter sur nextion

Sur Nextion nous avons different possibilité de personnalisation pour créer son interface, puis nous pouvons aussi modifier chaque action

Attribute	Value
b1(Button)	
type	98
id	8
objname	b1
vscope	global
sta	solid color
style	3D_Auto
font	0
bco	50712
bco2	1024
pco	0
pco2	65535
xcen	Center
ycen	Center
txt	AUTO
txt_maxl	20
isbr	False
spax	0
spay	0
x	41
y	408
w	188
h	61

Pour rendre l'interface utilisable et pratique, j'ai commencé par modifier les attributs de chaque objet présent. J'ai attribué des noms significatifs et logiques à chacun d'eux afin de simplifier la programmation sur Arduino. Après avoir effectué ces modifications, j'ai programmé des événements pour les boutons présents.

interface du projet :



6. Fonction pour calculer les angles

Pour que les aiguilles des jauge de l'application nextion puissent se placer au bon endroit il nous a fallu calculer son angle car l'angle 0° se situe à l'horizontale ce qui crée un problème car la valeur la plus basse ne pourra pas aller plus bas que ce niveau, pour remédier à cela il faut faire un calcul. Exemple avec le ph, il nous faut connaître la valeur entre les valeurs maximales et minimales, dans ce cas elle est de 4 (max 9 : min 5) pour un angle maximal de 268° et la valeur minimale se situe à 44° , la valeur qui correspond au $0^\circ = 5.656$ ce qui nous permet de savoir que $1^\circ = 1/62$.

la fonction ph ressemble donc à ça:

```
float anglePH(float PH)
{
    if (PH<5)
        PH=5;
```

```

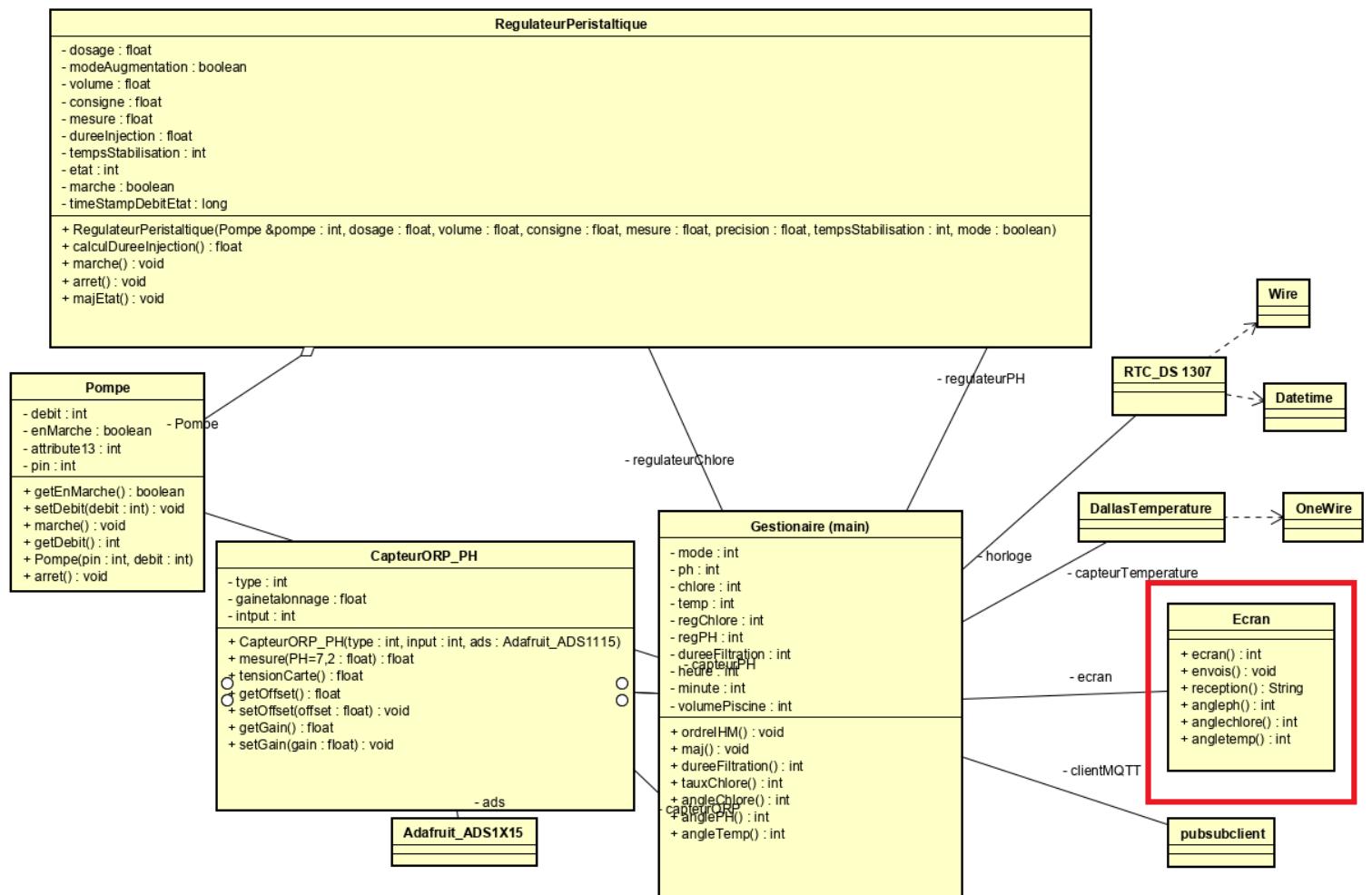
else (PH>9)
    PH=9;

if (PH>5.656)
    return angle=(PH-5.656)*67;

else
    return angle=180-(5.656-PH)*67;
}

```

7. Diagramme de classe



The screenshot shows a software interface with a yellow background. At the top, the word "Ecran" is written in bold black capital letters. Below it, there is a horizontal line with two colored circles: a yellow one on the right and a green one on the far left. To the left of the yellow circle, there is a small icon of a computer monitor. A list of methods is displayed below the line:

- + ecran() : int
- + envois() : void
- + reception() : String
- + angleph() : int
- + anglechlore() : int
- + angletemp() : int

8. Difficultés rencontrées

J'ai eu de nombreux problèmes avec ma partie personnelle, surtout au niveau de la compréhension de Nextion car c'était un nouvel outil de travail. La création d'interfaces était aussi quelque chose de nouveau pour moi, donc il m'a fallu du temps. Pour remédier à cela, des vidéos YouTube m'ont été d'une grande aide.

9. Conclusion

Grâce à ce projet, j'ai pu avoir l'opportunité d'apprendre de nombreuses choses intéressantes comme notamment la création et l'utilisation de Nextion Editor et coder avec Arduino. Pouvoir travailler en équipe et rassembler nos différentes parties et les faire fonctionner ensemble était vraiment intéressant comme expérience.

5. Partie individuelle (JOB Alexis)

1. Introduction

Pour la réalisation de ma partie personnelle, j'ai dû apprendre à gérer principalement un protocole MQTT. Il m'a fallu savoir comment recevoir et envoyer des données. J'ai dû d'abord installer MQTT sur une raspberry.

2. C'est quoi MQTT

Le protocole MQTT a pour but principal de transmettre des chaînes de caractères simples d'un serveur (appelé broker) vers ses abonnés. Pour cela, nous devons créer des topics (sujets) qui permettent l'échange et la communication entre un abonné et le topic auquel il est souscrit. Un topic est une chaîne de caractères au format UTF-8 utilisée par le broker pour identifier et filtrer les messages de chaque client connecté. Il peut être composé de plusieurs niveaux, chaque niveau étant séparé par un slash ('/').

3. Mise en place

J'ai tout d'abord installé un raspberry (qui nous fait office de serveur), pour pouvoir installer mon serveur MQTT.

J'ai d'abord installé l'os grâce au logiciel Raspberry PI Manager :



J'ai ensuite installé le serveur MQTT grâce à mosquitto



1. Mettre à jour le système

`sudo apt update`

`sudo apt upgrade`

2. Installer Mosquitto et ses clients

`sudo apt install -y mosquitto mosquitto-clients`

3. Activer et démarrer Mosquitto

`sudo systemctl enable mosquitto`

`sudo systemctl start mosquitto`

4. Vérifier l'installation

Pour vérifier que Mosquitto fonctionne correctement, utilisez les commandes suivantes pour publier et vous abonner à un message test.

Dans une première fenêtre de terminal, abonnez-vous au topic `test/topic` :

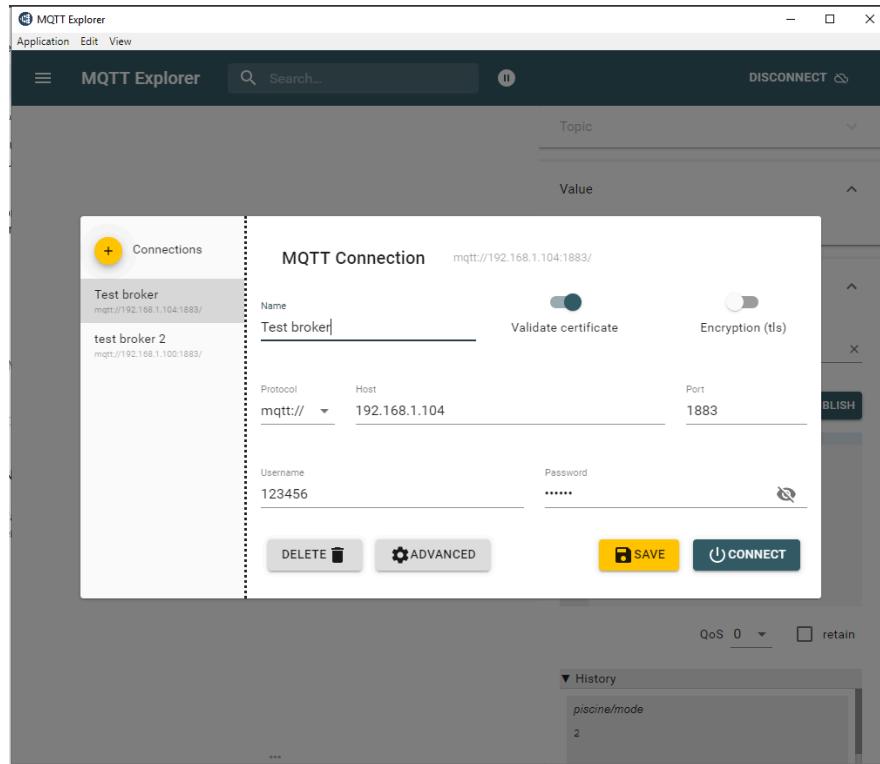
`mosquitto_sub -t test/topic -v`

Dans une deuxième fenêtre de terminal, publiez un message sur le même topic :

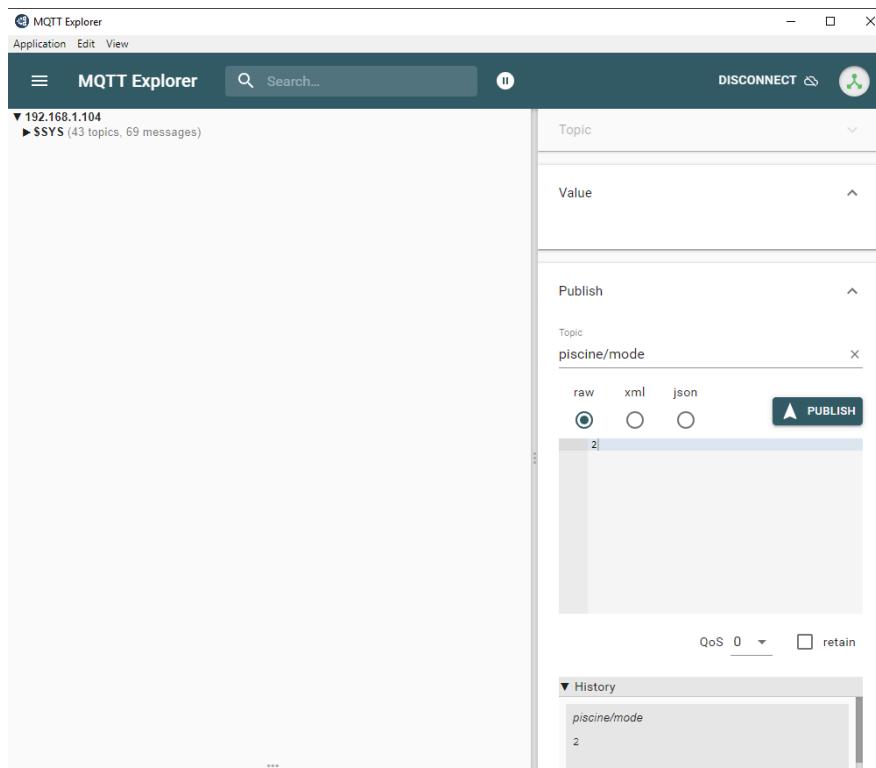
`mosquitto_pub -t test/topic -m "Hello MQTT"`

Si l'installation est correcte, vous devriez voir le message "Hello MQTT" dans la première fenêtre du terminal.

Par la suite, j'ai installé MQTT Explorer sur un autre poste sur le même réseau pour pouvoir me connecter au serveur MQTT.



Une fois connecté nous avons cette interface:



Grâce à ce logiciel nous pouvons envoyer ou recevoir des informations.
Suite à ceci je me suis familiarisé avec ou je me suis d'abord envoyé plusieurs informations.

```
#include <WiFi.h>
#include <PubSubClient.h>

// Informations sur le réseau WiFi
const char* ssid = "Projet";
const char* password = "12345678";

// Informations sur le broker MQTT
const char* mqtt_server = "192.168.1.104";
const int mqtt_port = 1883;
const char* mqtt_user = "123456";
const char* mqtt_password = "123456";
const char* mqtt_topic = "piscine";

// Broche à laquelle est connecté le potentiomètre
const int pinPot = 33;

WiFiClient espClient;
PubSubClient clientMQTT(espClient);

void setup_wifi() {

    delay(10);
    Serial.println();
    Serial.print("Connexion au réseau WiFi");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connecté");
    Serial.println("Adresse IP: ");
    Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message reçu [");
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
    }
    Serial.println("]");
}
```

```
Serial.print(topic);
Serial.print("] ");
for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
}
Serial.println();
}

void reconnect() {
    while (!clientMQTT.connected()) {
        Serial.print("Tentative de connexion MQTT...");
        if (clientMQTT.connect("ESP32Client", mqtt_user, mqtt_password)) {
            Serial.println("Connecté au broker MQTT");
            clientMQTT.subscribe(mqtt_topic);
        } else {
            Serial.print("Échec, rc=");
            Serial.print(clientMQTT.state());
            Serial.println(" Nouvelle tentative dans 5 secondes");
            delay(5000);
        }
    }
}

void setup() {
    analogReadResolution(12);
    Serial.begin(115200);
    setup_wifi();
    clientMQTT.setServer(mqtt_server, mqtt_port);
    clientMQTT.setCallback(callback);
}

void loop() {
    if (!clientMQTT.connected()) {
        reconnect();
    }
    clientMQTT.loop();

    // Lecture de la valeur analogique du potentiomètre
    int analogValue = analogRead(33);
    Serial.printf("ADC analog value = %d\n",analogValue);

    // Envoi de la valeur du potentiomètre au broker MQTT
}
```

```

clientMQTT.publish(mqtt_topic, String(analogValue).c_str());
```

// Attente de quelques millisecondes avant de lire à nouveau la valeur du potentiomètre

```

delay(100);
}
```

Ce code me permet de lire une valeur d'un potentiomètre et de me la donner en direct sur MQTT Explorer

Par la suite j'ai essayé de faire allumer plusieurs leds directement via MQTT explorer qui envoie l'info directement à l'esp 32

Voici le code pour allumer les 3 leds:

```

#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "Projet";
const char* password = "12345678";
const char* mqtt_server = "192.168.1.100";
const int mqtt_port = 1883;
const char* mqtt_user = "123456";
const char* mqtt_password = "123456";
const char* mqtt_topic1 = "led1"; // Sujet MQTT pour contrôler la LED 1
const char* mqtt_topic2 = "led2"; // Sujet MQTT pour contrôler la LED 2
const char* mqtt_topic3 = "led3"; // Sujet MQTT pour contrôler la LED 3

const int ledPin1 = 23; // Broche de la LED 1
const int ledPin2 = 19; // Broche de la LED 2
const int ledPin3 = 4; // Broche de la LED 3

WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
    delay(10);
    Serial.println();
    Serial.print("Connexion au réseau WiFi");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
```

```
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connecté");
Serial.println("Adresse IP: ");
Serial.println(WiFi.localIP());
}

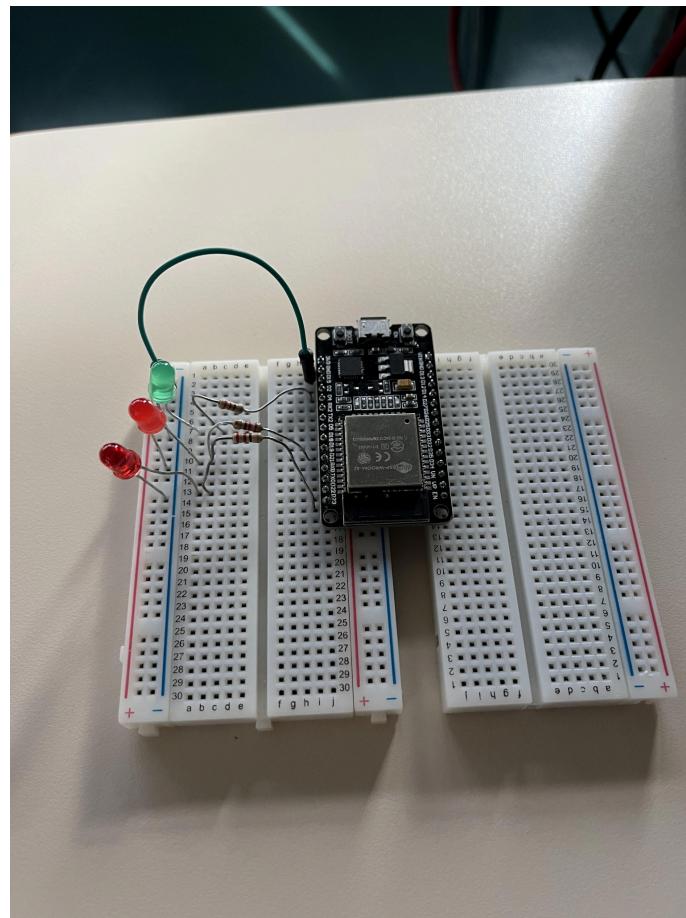
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message reçu [");
    Serial.print(topic);
    Serial.print("] ");
    String message = "";
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
        message += (char)payload[i];
    }
    Serial.println();
}

// Allumer ou éteindre la LED en fonction du message reçu
if (strcmp(topic, mqtt_topic1) == 0) {
    if (message == "on") {
        digitalWrite(ledPin1, HIGH);
    } else if (message == "off") {
        digitalWrite(ledPin1, LOW);
    }
} else if (strcmp(topic, mqtt_topic2) == 0) {
    if (message == "on") {
        digitalWrite(ledPin2, HIGH);
    } else if (message == "off") {
        digitalWrite(ledPin2, LOW);
    }
} else if (strcmp(topic, mqtt_topic3) == 0) {
    if (message == "on") {
        digitalWrite(ledPin3, HIGH);
    } else if (message == "off") {
        digitalWrite(ledPin3, LOW);
    }
}

void reconnect() {
    while (!client.connected()) {
```

```
Serial.print("Tentative de connexion MQTT...");  
if (client.connect("ESP32Client", mqtt_user, mqtt_password)) {  
    Serial.println("Connecté au broker MQTT");  
    client.subscribe(mqtt_topic1);  
    client.subscribe(mqtt_topic2);  
    client.subscribe(mqtt_topic3);  
} else {  
    Serial.print("Échec, rc=");  
    Serial.print(client.state());  
    Serial.println(" Nouvelle tentative dans 5 secondes");  
    delay(5000);  
}  
}  
  
void setup() {  
    pinMode(ledPin1, OUTPUT);  
    pinMode(ledPin2, OUTPUT);  
    pinMode(ledPin3, OUTPUT);  
    Serial.begin(115200);  
    setup_wifi();  
    client.setServer(mqtt_server, mqtt_port);  
    client.setCallback(callback);  
}  
  
void loop() {  
    if (!client.connected()) {  
        reconnect();  
    }  
    client.loop();  
}
```

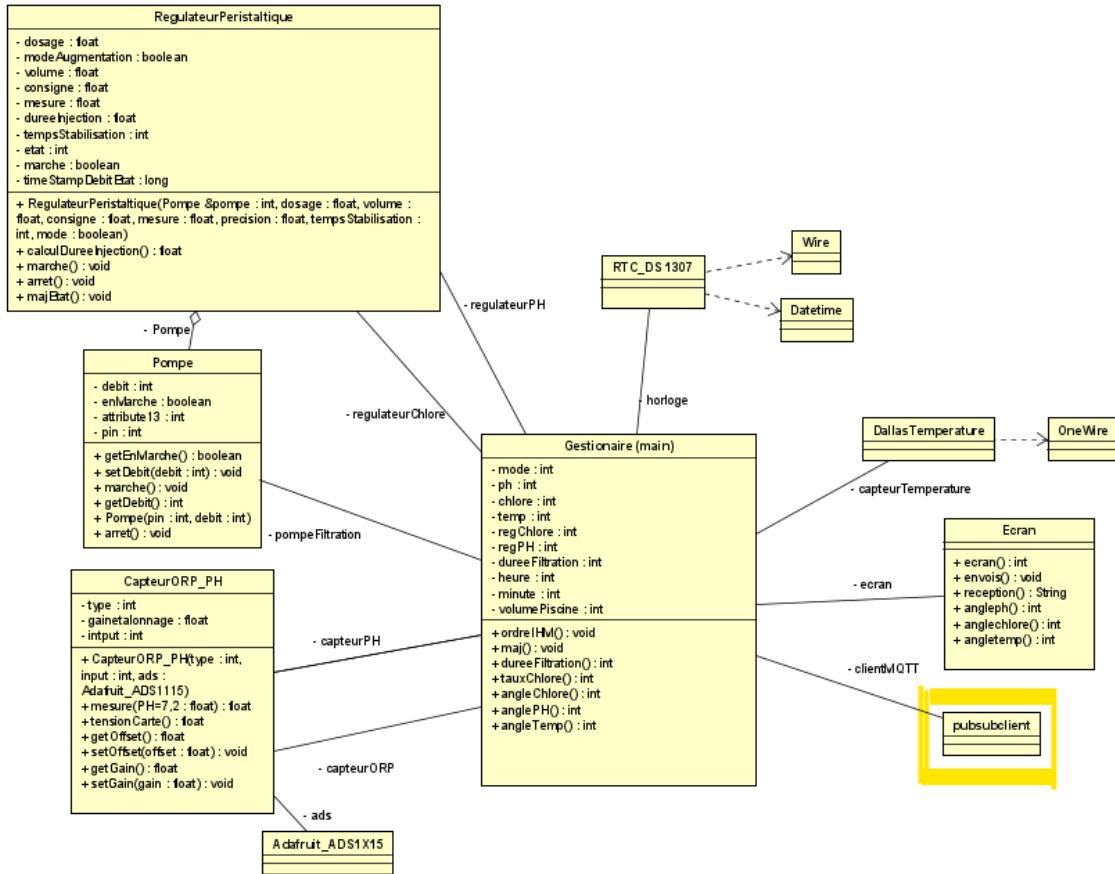
Voici le montage de l'esp 32 est des 3 leds



Après plusieurs tests effectués et après avoir pris le fonctionnement de mosquito en main. Nous sommes partis sur la mise en commun du groupe (voir annexes)

Diagramme de classe:

Voici ma partie dedans PubSubClient est la bibliothèque que j'ai utilisé dans arduino



4. Difficultés rencontrées

J'ai d'abord rencontré des problèmes, pour installer mosquito sur la raspberry plusieurs problèmes de configuration. La raspberry a planté une fois qui m'a fait recommencer toute la configuration de celle-ci. Pour allumer les 3 LEDS j'ai mis du temps à trouver la fonction pour l'envoyer sur l'esp32. Tout était nouveau pour moi, j'ai appris tout le long et j'apprends encore

5. Conclusion

Ma partie personnelle est assez complète, cela m'a permis d'apprendre plusieurs protocoles et des techniques dont je ne m'étais jamais servis auparavant et cela apporte de nouvelles expériences et de nouvelles compétences en plus.

6. Conclusion globale

Actuellement, le projet "Piscine tranquille - CQEP" est toujours en cours de réalisation. Nous visons à le terminer d'ici la toute dernière soutenance orale. Néanmoins, la réalisation de ce projet a été très enrichissante pour nous. Elle nous a permis de mettre en pratique le savoir acquis en cours, tout en découvrant de nouvelles technologies et de nouveaux composants.

Travailler sur ce projet nous a confrontés à des défis techniques complexes, notamment le calcul du taux de chlore en ppm et la gestion des différents modes de fonctionnement des systèmes de piscine. Ces tâches ont requis une compréhension approfondie des capteurs, de l'électronique et de la programmation embarquée. Par ailleurs, nous avons exploré des solutions de communication avancées, telles que le Wi-Fi et LoRa, pour permettre une gestion locale et à distance des systèmes de piscine.

Cette expérience nous a permis de renforcer nos compétences en développement de logiciels et en intégration de systèmes. Elle nous a également appris à adopter une approche méthodique pour la résolution de problèmes complexes et à travailler en équipe pour atteindre un objectif commun.

En conclusion, bien que le projet ne soit pas encore achevé, nous sommes confiants dans notre capacité à le finaliser avec succès. Nous sommes fiers du chemin parcouru et des connaissances acquises tout au long de cette aventure. Nous sommes impatients de voir l'impact positif de notre travail sur les services proposés par CORTES PISCINES & SPAS et sur la satisfaction de leurs clients.

7. Annexes

1. Main GestionnairePiscine

```
#include "Arduino.h"
#include "pompe.h"
#include "CapteurORPPH.h"
#include "ecran.h"
#include "RegulateurPeristaltique.h"
#include <OneWire.h>
#include <RTClib.h>
#include <DallasTemperature.h>
#include <PubSubClient.h>
#include <WiFi.h>

// Déclarations des pins et constantes
#define PIN_POMPE_PH 14
#define PIN_POMPE_CHLORE 27
#define PIN_POMPE_Filtration 26
#define DEBIT_POMPE_PH 2.22
#define DEBIT_POMPE_CHLORE 2.22
#define DEBIT_POMPE_Filtration 20
#define VOLUME_PISCINE 60
#define DOSAGE_CHLORE 8
#define DOSAGE_PH 10
#define CONSIGNE_PH 72
#define CONSIGNE_CHLORE 10
#define PRECISION_PH 1
#define PRECISION_CHLORE 1
#define STABILISATION 60

// WiFi et MQTT
const char* ssid = "Projet";
const char* password = "12345678";
const char* mqtt_server = "192.168.1.104";
const int mqtt_port = 1883;
const char* mqtt_user = "123456";
const char* mqtt_password = "123456";
const char* mqtt_topic_temp = "piscine/temperature";
const char* mqtt_topic_ph = "piscine/ph";
const char* mqtt_topic_chlore = "piscine/chlore";
```

```

const char* mqtt_topic_pompe_ph = "piscine/pompe/ph";
const char* mqtt_topic_pompe_chlore = "piscine/pompe/chlore";
const char* mqtt_topic_pompe_filtration = "piscine/pompe/filtration";
const char* mqtt_topic_mode= "piscine/mode";

WiFiClient espClient;
PubSubClient client(espClient);

// Déclarations des objets
Adafruit_ADS1115 ads;
CapteurORPPH capteurORP(1, 1, ads);
CapteurORPPH capteurPH(0, 0, ads);
Pompe pompeFiltration(PIN_POMPE_Filtration, DEBIT_POMPE_Filtration);
Pompe pompeChlore(PIN_POMPE_CHLORE, DEBIT_POMPE_CHLORE);
Pompe pompePH(PIN_POMPE_PH, DEBIT_POMPE_PH);
RegulateurPeristaltique regulateurChlore(pompeChlore, DOSAGE_CHLORE,
VOLUME_PISCINE, CONSIGNE_CHLORE, 0, PRECISION_CHLORE, STABILISATION,
1);
RegulateurPeristaltique regulateurPH(pompePH, DOSAGE_PH,
VOLUME_PISCINE, CONSIGNE_PH, 0, PRECISION_PH, STABILISATION, 1);
RTC_DS1307 horloge;
OneWire oneWire(4);
DallasTemperature capteurTemperature(&oneWire);
Ecran ecran(9600);

int mode = 0; //0->Manu 1->Auto 2->Minuterie
3->Horaire
int volPisine = VOLUME_PISCINE;
int temp = 0;
int ph = 0;
int chlore = 0;
int consigneChlore = CONSIGNE_CHLORE;
int regPH = 0;
int regChlore = 0;
int dureeFiltre = 0;
int heure = 0;
int minute = 0;

void setup_wifi() {
    delay(10);
    Serial.println();

    Serial.print("Connexion au réseau WiFi");
}

```

```

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connecté");
Serial.println("Adresse IP: ");
Serial.println(WiFi.localIP());
}

void reconnect() {
    while (!client.connected()) {
        Serial.print("Tentative de connexion MQTT...");
        if (client.connect("ESP32Client", mqtt_user, mqtt_password)) {
            Serial.println("Connecté au broker MQTT");
            // S'abonner aux topics pour le contrôle des pompes
            client.subscribe(mqtt_topic_pompe_ph);
            client.subscribe(mqtt_topic_pompe_chlore);
            client.subscribe(mqtt_topic_pompefiltration);
            client.subscribe(mqtt_topic_mode);
        } else {
            Serial.print("Échec, rc=");
            Serial.print(client.state());
            Serial.println(" Nouvelle tentative dans 5 secondes");
            delay(5000);
        }
    }
}

void callback(char* topic, byte* payload, unsigned int length) {
    String message;
    for (unsigned int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    message.trim();

    Serial.print("Message reçu [");
    Serial.print(topic);
    Serial.print("]: ");
    Serial.println(message);

    if (String(topic) == mqtt_topic_mode)
}

```

```
{  
    mode=message.toInt();  
    if(mode==0)  
        ecran.envoie("page 1");  
    else if(mode==1)  
        ecran.envoie("page 2");  
    else if(mode==2)  
        ecran.envoie("page 3");  
    else if(mode==3)  
        ecran.envoie("page 4");  
}  
  
/*  
if (String(topic) == mqtt_topic_pompe_ph) {  
    if (message == "1") {  
        pompePH.marche();  
    } else if (message == "0") {  
        pompePH.arret();  
    }  
} else if (String(topic) == mqtt_topic_pompe_chlore) {  
    if (message == "1") {  
        pompeChlore.marche();  
    } else if (message == "0") {  
        pompeChlore.arret();  
    }  
} else if (String(topic) == mqtt_topic_pompe_filtration) {  
    if (message == "1") {  
        pompeFiltration.marche();  
    } else if (message == "0") {  
        pompeFiltration.arret();  
    }  
}  
*/  
}  
  
void setup() {  
    capteurTemperature.begin();  
    Serial.begin(9600);  
    if (!horloge.begin()) {  
        Serial.println("Couldn't find RTC");  
    }
```

```
    Serial.flush();
    while (1);
}

if (!ads.begin()) {
    Serial.println("Failed to initialize ADS.");
    while (1);
}

ads.setGain(GAIN_FOUR);

setup_wifi();
client.setServer(mqtt_server, mqtt_port);
client.setCallback(callback);
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    maj();
    ordreIHM();

/*Serial.print("ph=");
Serial.println(ph);
Serial.print("chlore=");
Serial.println(chlore);
Serial.print("temp=");
Serial.println(temp);*/

// Envoi des données à MQTT
/*client.publish(mqtt_topic_temp, String(temp).c_str());
client.publish(mqtt_topic_ph, String(ph).c_str());
client.publish(mqtt_topic_chlore, String(chlore).c_str());*/
}

delay(500); // Envoi des données toutes les 0.5 secondes
}

int dureeFiltration() {
    if (temp < 10)
        return 1;
```

```
else if (temp < 14)
    return 2;
else if (temp < 18)
    return 3;
else if (temp < 20)
    return 5;
else if (temp < 24)
    return 7;
else if (temp < 28)
    return 10;
else
    return 12;
}

int tauxChlore() {
    if (temp < 10)
        return 0;
    else if (temp < 14)
        return 5;
    else if (temp < 18)
        return 10;
    else if (temp < 20)
        return 13;
    else
        return 15;
}

void maj() {
    capteurTemperature.requestTemperatures();
    temp = (int)capteurTemperature.getTempCByIndex(0);

    int mesurePH = (int)(capteurPH.mesure()*10);
    if(mesurePH!=ph)
    {
        ph=mesurePH;
        String msg="var.ph.val='"+(String)ph;
        ecran.envoie(msg);
        int angle=(int)anglePH(ph/10.0);
        Serial.print("anglePH=");
        Serial.println(angle);
        msg="var.anglePH.val='"+(String)angle;
        ecran.envoie(msg);
        client.publish(mqtt_topic_ph, String(ph).c_str());
    }
}
```

```
    Serial.print("ph=");
    Serial.println(ph);
}

chlore = capteurORP.mesure(ph/10);

DateTime now = horloge.now();
int mesureHeure = now.hour();
int mesureMinute = now.minute();
if(mesureMinute!=minute)
{
    minute=mesureMinute;
    String msg="var.min.val='"+(String)mesureMinute;
    ecran.envoie(msg);
    Serial.print("minute=");
    Serial.println(minute);
}

dureeFiltre = dureeFiltration();
consigneChlore = tauxChlore();
}

float anglePH(float PH)
{
    if(PH<5)
        PH=5;

    else if(PH>9)
        PH=9;

    if(PH>5.656)
        return (PH-5.656)*67;
    else
        return 360-((5.656-PH)*67);

}

void ordreIHM()
{
    String msg=ecran.reception();
    if(msg.length()>0)
    {
        Serial.print("msg=");
        Serial.println(msg);
```

```

int index=msg.indexOf("mode=");
if(index!=-1)
{
    Serial.println("Mesg mode recu");
    index+=5;
    String valeur=msg.substring(index,msg.indexOf(";",index));
    mode=valeur.toInt();
    Serial.print("valeur=");
    Serial.println(valeur);
    client.publish(mqtt_topic_mode, String(mode).c_str());
}

}
}

```

2. CapteurORPPH.h

```

#ifndef CapteurORPPH_H
#define CapteurORPPH_H

#include <Adafruit_ADS1X15.h>

using namespace std;

class CapteurORPPH {
private:
    int type;
    float gainEtalonnage;
    float offset;
    int input;
    Adafruit_ADS1115 *ads;
public:
    CapteurORPPH(int type, int input, Adafruit_ADS1115 &ads);
    float mesure(float pH = 7.2);
    float tensionCarte();
    float getOffset();
    void setOffset(float offset);
    float getGain();
    void setGain(float gain);
};

#endif // CapteurORPPH_H

```

3. CapteurORPPH.cpp

```
#include "CapteurORPPH.h"

CapteurORPPH::CapteurORPPH(int type, int input, Adafruit_ADS1115 &ads)
{
    this->ads=&ads;
    this->setGain(GAIN_FOUR);
    this->type=type;
    this->input=input;
    gainEtalonnage=1;
    offset=0;
}

float CapteurORPPH::mesure(float pH) {
    if(input==0) // 0 = fonction ph, 1 = fonction ppm
    {
        float voltage = tensionCarte();
        // Interpolation linéaire pour convertir la tension en pH
        float slope = 14.0 / 0.828; // 14 / (0.414 * 2)
        float intercept = 7.0;
        float pH = slope * voltage + intercept;

        // Vérification de la plage de pH
        if (pH < 0.0) {
            pH = 0.0;
        } else if (pH > 14.0) {
            pH = 14.0;
        }
        return pH;
    }
    else
    {
        float voltage = tensionCarte(); // récupération de la tension
        float ppm = 0; // Initialisez ppm à 0
        voltage=voltage*1000; // conversion de la tension en V
        if (pH < 6.45 || pH > 14) { // si le pH est en dehors de cette
plage, renvoie "-1" pour dire qu'il n'y a pas de ppm
            return -1;
        }
        else if (pH >= 6.45 && pH <= 6.55) { // calcul du ppm en fonction
de la plage du pH calculé précédemment
            return 0.0053 * exp(0.0071 * voltage);
        }
    }
}
```

```
    }
    else if (pH >= 6.55 && pH <= 6.65) {
        return 0.0051 * exp(0.0072 * voltage);
    }
    else if (pH >= 6.65 && pH <= 6.75) {
        return 0.0048 * exp(0.0073 * voltage);
    }
    else if (pH >= 6.75 && pH <= 6.85) {
        return 0.0046 * exp(0.0074 * voltage);
    }
    else if (pH >= 6.85 && pH <= 6.95) {
        return 0.0044 * exp(0.0076 * voltage);
    }
    else if (pH >= 6.95 && pH <= 7.05) {
        return 0.0041 * exp(0.0077 * voltage);
    }
    else if (pH >= 7.05 && pH <= 7.15) {
        return 0.0039 * exp(0.0078 * voltage);
    }
    else if (pH >= 7.15 && pH <= 7.25) {
        return 0.0037 * exp(0.0079 * voltage);
    }
    else if (pH >= 7.25 && pH <= 7.35) {
        return 0.0035 * exp(0.0081 * voltage);
    }
    else if (pH >= 7.35 && pH <= 7.45) {
        return 0.0033 * exp(0.0082 * voltage);
    }
    else if (pH >= 7.45 && pH <= 7.55) {
        return 0.0031 * exp(0.0084 * voltage);
    }
    else if (pH >= 7.55 && pH <= 7.65) {
        return 0.0029 * exp(0.0085 * voltage);
    }
    else if (pH >= 7.65 && pH <= 7.75) {
        return 0.0027 * exp(0.0087 * voltage);
    }
    else if (pH >= 7.75 && pH <= 7.85) {
        return 0.0025 * exp(0.0089 * voltage);
    }
    else if (pH >= 7.85 && pH <= 7.95) {
        return 0.0024 * exp(0.009 * voltage);
    }
}
```

```

        else if (pH >= 7.95 && pH <= 8.05) {
            return 0.0022 * exp(0.0092* voltage);
        }
        else{
            return 0.0;
        } }

}

float CapteurORPPH::tensionCarte() {
    // Implémentation de la fonction tensionCarte
    if(input==1)
        return ads->computeVolts(ads->readADC_Differential_2_3()); // récupère la tension pour le pH
    else
        return ads->computeVolts(ads->readADC_Differential_0_1()); // récupère la tension pour le ppm
}

float CapteurORPPH::getOffset() {
    return offset;
}

void CapteurORPPH::setOffset(float offset) {
    this->offset=offset;
}

float CapteurORPPH::getGain() {
    // Implémentation de la fonction getGain
    return 0;
}

void CapteurORPPH::setGain(float gain) {
    // Implémentation de la fonction setGain
}

```

4. RegulateurPeristaltique.h

```

#ifndef REGULATEUR_PERISTALTIQUE_H
#define REGULATEUR_PERISTALTIQUE_H

#include "pompe.h"

```

```

class RegulateurPeristaltique {
private:
    Pompe *pompe;
    float dosage;
    bool modeAugmentation;
    float volume;
    float consigne;
    float mesure;
    float precision;
    float dureeInjection;
    int tempsStabilisation;
    int etat;
    bool enMarche;
    unsigned long timeStamp;

public:
    RegulateurPeristaltique(Pompe& pompe, float dosage, float volume,
                           float consigne, float mesure, float precision, int tempsStabilisation,
                           bool mode);
    float calculDureeInjection();
    void marche();
    void arret();
    void majEtat();
};

#endif

```

5. RegulateurPeristaltique.cpp

```

#include "soc/io_mux_reg.h"
#include "RegulateurPeristaltique.h"
#include <Arduino.h>

RegulateurPeristaltique::RegulateurPeristaltique(Pompe& pompe, float dosage, float volume, float consigne, float mesure, float precision, int tempsStabilisation, bool mode)
    : pompe(&pompe), dosage(dosage), volume(volume), consigne(consigne), mesure(mesure), precision(precision), tempsStabilisation(tempsStabilisation), modeAugmentation(mode), etat(0), enMarche(false), timeStamp(0) {}

```

```
float RegulateurPeristaltique::calculDureeInjection()
{
    int ecart = consigne - mesure;
    float dureeInjection = (3.6 * ecart * volume *
dosage)/pompe->getDebit();
    return dureeInjection;
}

void RegulateurPeristaltique::marche()
{
    pompe->marche();
    enMarche = true;
}

void RegulateurPeristaltique::arret()
{
    pompe->arret();
    enMarche = false;
}

void RegulateurPeristaltique::majEtat()
{
    if (etat == 0 && enMarche)
    {
        etat = 1;
        timeStamp = millis();
    }
    if ((etat == 1 && millis() - timeStamp > 60000) || (etat == 4 &&
millis() - timeStamp > tempsStabilisation*1000))
    {
        etat = 2;
        timeStamp = millis();
        dureeInjection = calculDureeInjection();
    }
    if (etat == 2 && dureeInjection > 0)
    {
        etat = 3;
        timeStamp = millis();
        pompe->marche();
    }
    if (etat == 3 && millis() - timeStamp > dureeInjection*1000)
    {
        etat = 4;
    }
}
```

```

        timeStamp = millis();
        pompe->arret();
    }

    if (etat == 2 && dureeInjection == 0 || !enMarche)
    {
        etat = 0;
        pompe->arret();
    }
}

```

6. pompe.h

```

#ifndef POMPE_H
#define POMPE_H

class Pompe {
private:
    int pin;
    int debit;
    bool enMarche;

public:
    Pompe(int pin, int debit);
    void marche();
    void arret();
    int getDebit() const;
    bool getEnMarche() const;
    void setDebit(int nouveauDebit);
};

#endif

```

7. pompe.cpp

```

#include "pompe.h"
#include <Arduino.h>

Pompe::Pompe(int pin, int debit) : pin(pin), debit(debit),
enMarche(false) {
    pinMode(pin, OUTPUT);
    arret();
}

void Pompe::marche() {

```

```

        digitalWrite(pin,LOW);
        enMarche = true;
    }

void Pompe::arret() {
    digitalWrite(pin,HIGH);
    enMarche = false;
}

int Pompe::getDebit() const {
    return debit;
}

bool Pompe::getEnMarche() const {
    return enMarche;
}

void Pompe::setDebit(int nouveauDebit) {
    debit = nouveauDebit;
}

```

8. ecran.h

```

#ifndef ecran_h
#define ecran_h
#include "Arduino.h"

class Ecran
{
public:
    Ecran(int rate);
    void envoie(String message);
    String reception();
};

#endif

```

9. ecran.cpp

```

#include "ecran.h"

Ecran::Ecran(int rate)
{
    Serial2.begin(rate);
}

```

```
}
```

```
void Ecran::envoie(String message)
{
    Serial2.print(message);
    Serial2.write(0xFF);
    Serial2.write(0xFF);
    Serial2.write(0xFF);
}

String Ecran::reception()
{
    String message="";
    while(Serial2.available())
    {
        message=Serial2.readString();
    }
    return message;
}
```