Partie Personnel

Etudiant N° 1

Ayoub Fathallah

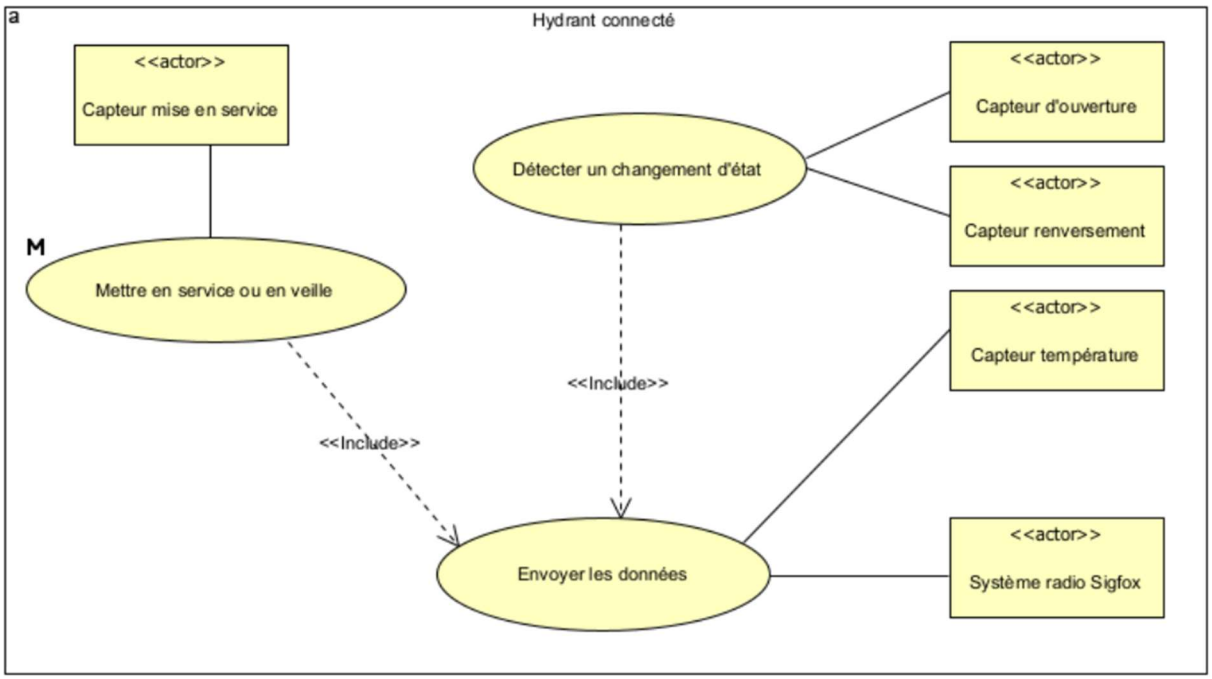
Table des matières

Introduction personnelle

Mon Rôle dans le Projet

Dans ce projet, mon rôle consiste à rendre l'hydrant connecté et intelligent, en lui permettant de détecter les anomalies, de les transmettre pour traitement, et de les enregistrer dans une base de données adaptée. L'objectif est de transformer l'hydrant en un dispositif capable de surveiller son propre état et les conditions environnantes, de détecter des événements tels que l'ouverture ou la fermeture de la vanne, l'inclinaison, ou des changements de température. Les informations recueillies sont ensuite envoyées, traitées, et stockées de manière sécurisée, avec la possibilité de déclencher des alertes pour informer rapidement les autorités municipales en cas d'anomalie. Cette transformation permet non seulement une surveillance continue et proactive, mais facilite également les interventions et la gestion à distance, assurant ainsi une meilleure maintenance et sécurité des infrastructures.

Fonctionnalités à Réaliser



1. Détection de l'ouverture et de la fermeture de la vanne

Je suis chargé de mettre en place un système capable de détecter l'ouverture et la fermeture de la vanne de l'hydrant. Cette fonctionnalité permet de surveiller l'utilisation de l'hydrant en temps réel, garantissant ainsi une réponse rapide en cas d'utilisation non autorisée ou de fuite.

2. Détection de l'inclinaison ou du renversement

Une autre tâche importante est d'intégrer un mécanisme pour détecter si l'hydrant est incliné ou renversé. Cette fonctionnalité est essentielle pour prévenir les actes de vandalisme et pour détecter les incidents susceptibles de compromettre l'intégrité de l'hydrant.

3. Transmission des données

Les données collectées par l'hydrant doivent être envoyées de manière fiable à un système cloud. Chaque événement détecté est transmis rapidement pour permettre une surveillance à distance efficace. Les données incluent l'état de la vanne, l'inclinaison de l'hydrant, et les conditions environnementales telles que la température.

4. Intégration avec la Base de Données

Je conçois également une base de données pour stocker toutes les informations transmises par les hydrants connectés. Cette base de données est structurée de manière à :

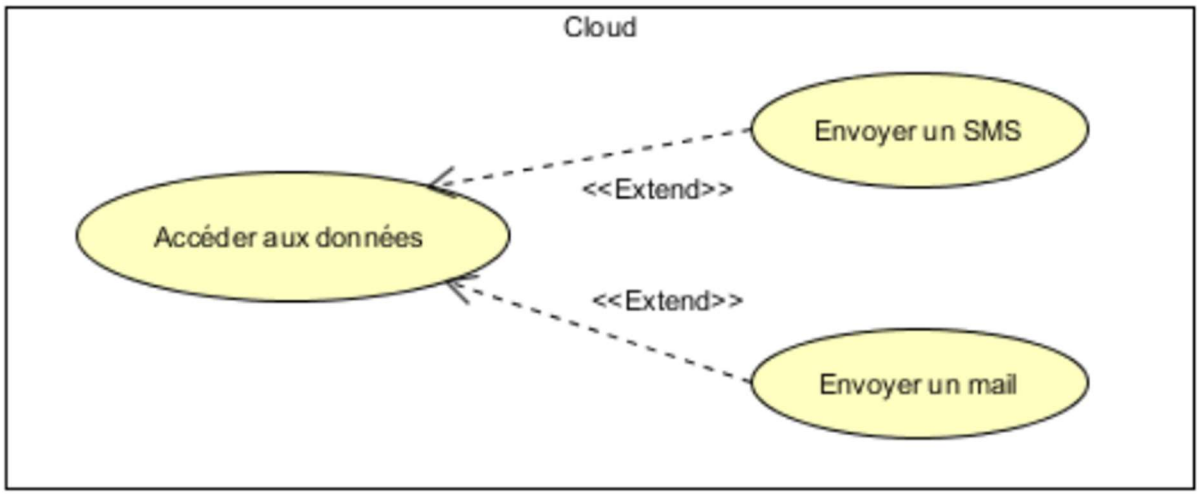
- Enregistrer les événements détectés par l'hydrant (ouverture, fermeture, inclinaison, etc.)

- Stocker les données précédentes

- Faciliter l'accès rapide et l'analyse des données pour générer des rapports et des alertes

- Contenir une liste des hydrants et les communes (mairies) sous la responsabilité desquelles ils se trouvent, permettant d'informer la bonne commune.

5. Développement du Service REST



Pour assurer une gestion efficace des informations et des alertes, je vais développer un service REST. Ce service permet :

- La réception des données envoyées par les hydrants

- Le traitement et la vérification des données reçues

- L'insertion des informations validées dans la base de données

- La détection des conditions nécessitant une alerte (par exemple, une ouverture prolongée ou un renversement de l'hydrant)

- L'envoi d'alertes via SMS ou email aux responsables concernés

Contraintes

Pour la base de données :

- Une base de données MySQL doit être conçue pour stocker toutes les informations transmises par les hydrants connectés, garantissant ainsi une gestion efficace et sécurisée des données.

Pour les hydrants connectés :

- Le système embarqué des hydrants doit être conçu pour garantir un fonctionnement autonome et efficace.

- Le langage de programmation utilisé pour développer le logiciel embarqué des hydrants doit être le C/C++ pour assurer une performance optimale.

- Les hydrants connectés doivent avoir une très faible consommation d'énergie afin de prolonger la durée de vie de leurs piles.

- Les composants du système numérique des hydrants ainsi que les capteurs doivent être totalement étanches à l'eau pour garantir leur fonctionnement dans toutes les conditions météorologiques.

- Les capteurs utilisés dans les hydrants connectés doivent être activés sans contact pour éviter toute altération du fonctionnement de l'hydrant.

- Le réseau Sigfox doit être utilisé pour la transmission des données des hydrants connectés, assurant ainsi une communication fiable sur de longues distances.

Pour le service REST des alertes :

- Le développement du service REST chargé de gérer les alertes doit être effectué en utilisant le langage de programmation PHP.

- Le service REST doit être capable d'envoyer des alertes via SMS ou email aux responsables concernés en cas de détection d'anomalies ou d'événements critiques.

Choix des composants

1. Système embarqué - ESP32-DevKitC V2 :

Le choix de l'ESP32-DevKitC V2 comme système embarqué s'explique par sa capacité à répondre aux exigences techniques du projet, telles que le langage de programmation C++, sa faible consommation d'énergie et sa capacité à entrer en mode DeepSleep. De plus, ayant déjà travaillé avec cette plateforme lors de mes années en BTS.

2. ADXL345 - Capteur de renversement :

Pour détecter les renversements des hydrants, nous avons opté pour l'ADXL345. Ce capteur permet de calculer l'accélération sur trois axes, ce qui facilite la mesure précise de l'inclinaison des hydrants. Sa précision et sa réactivité en font un choix idéal pour assurer une détection fiable des angles d'inclinaison, contribuant ainsi à la sécurité et à la fiabilité des hydrants connectés.

3. Interrupteur Lame Souple (ILS) - Capteur d'ouverture et de fermeture :

L'ILS, ou Interrupteur Lame Souple, est un interrupteur mécanique à très basse consommation. Son fonctionnement repose sur la déformation d'une lame souple sous l'effet d'un champ magnétique, ce qui permet de détecter l'ouverture ou la fermeture d'un circuit électrique. Cette solution simple et efficace offre une détection fiable de l'état d'ouverture des hydrants, tout en minimisant la consommation d'énergie.

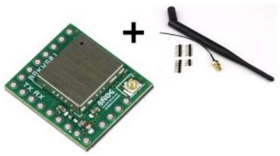
4. ILS de mise en service:

De même que pour l'ouverture et la fermeture, nous utilisons un ILS pour la mise en service des hydrants. Son fonctionnement mécanique et sa très basse consommation en font un choix optimal pour activer ou mettre en veille prolongée les hydrants de manière fiable et économe en énergie.

5. BMP280 - Capteur de température :

Le BMP280 est un capteur de température qui répond aux exigences du projet en termes de faible consommation d'énergie et de précision. Ayant déjà travaillé avec ce composant lors de mes projets précédents, je sais qu'il offre des performances fiables et une intégration aisée dans notre système. Son coût abordable en fait également un choix économique pour notre projet.

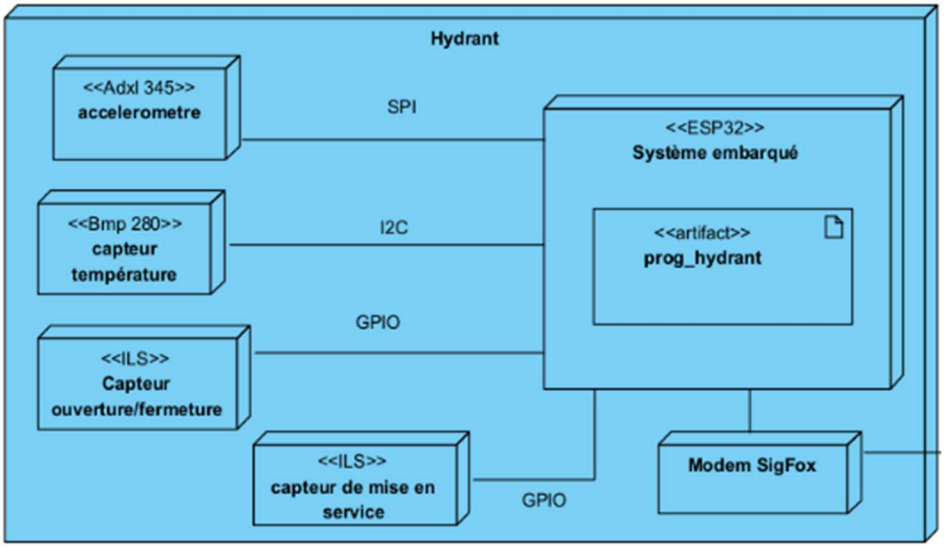
6. Modem Sigfox - BRKWS01:



Pour assurer une transmission fiable des données sur de longues distances, nous avons choisi le modem Sigfox BRKWS01. Sa portée de transmission étendue et sa faible consommation d'énergie en font un choix optimal pour les applications IoT telles que les hydrants connectés. En utilisant le réseau Sigfox, nous garantissons une connectivité robuste et une communication efficace entre les hydrants et le système de surveillance centralisé.

Diagrammes de Déploiement

Voici comment le prototype de borne d'incendie connectée est organisé, avec une illustration des différents composants et leurs connexions :



Développement et implémentation

Le DeepSleep

Le mode Deep Sleep de l'ESP32 permet à l'appareil de réduire sa consommation d'énergie en désactivant les parties non essentielles du microcontrôleur. L'ESP32 peut se réveiller de cet état de faible consommation en réponse à plusieurs types d'événements, notamment des signaux externes sur des pins spécifiques (EXT0, EXT1) et des timers. Ces réveils permettent de réactiver l'appareil uniquement lorsqu'une action est requise, prolongeant ainsi la durée de vie de la batterie.

La fonction suivante, `void way_by_wakeup_reason()`, illustre comment notre système gère les différents types de réveils :

```
void way_by_wakeup_reason()
{
  esp_sleep_wakeup_cause_t source_reveil;

  source_reveil = esp_sleep_get_wakeup_cause();

  switch (source_reveil)
  {
  case ESP_SLEEP_WAKEUP_EXT0:
    Serial.println("Réveil causé par un signal externe avec RTC_IO (ext0)");
    activation(); // fonction si manipulation de l'ILS Technitien
    break;
  case ESP_SLEEP_WAKEUP_EXT1:
    Serial.println("Réveil causé par un signal externe avec RTC_CNTL (ext1)");
    detecteAnomalie();
    break;
  case ESP_SLEEP_WAKEUP_TIMER:
    Serial.println("Réveil causé par un timer");
    break;
  case ESP_SLEEP_WAKEUP_TOUCHPAD:
    Serial.println("Réveil causé par un touchpad");
    break;
  default:
    Serial.printf("Réveil pas causé par le Deep Sleep: %d\n", source_reveil);
    Serial.println("\r");
    desactivation();
    break;
  }
}
```

Explication de la Fonction

La fonction `void way_by_wakeup_reason()` a pour but de déterminer la cause du réveil de l'ESP32 et d'exécuter des actions spécifiques en fonction de cette cause. Voici une explication détaillée de son fonctionnement :

- La variable `source_reveil` est déclarée pour stocker la cause du réveil.
- La fonction `esp_sleep_get_wakeup_cause()` est appelée pour déterminer cette cause.

Analyse de la cause du réveil :

Le code utilise une structure `switch` pour déterminer l'action à exécuter en fonction de la valeur de `source_reveil`.

Cas ESP_SLEEP_WAKEUP_EXT0 :

La cause du réveil vient d'un signal externe via RTC_IO (ext0). Dans mon projet, ce cas correspond à la mise en service du dispositif par un signal externe sur un pin spécifique.

Cas ESP_SLEEP_WAKEUP_EXT1 :

La cause du réveil vient d'un signal externe via RTC_CNTL (ext1). Ce cas correspond à un réveil par un changement d'état sur plusieurs pins, utilisé pour la détection d'anomalies.

Cas ESP_SLEEP_WAKEUP_TIMER :

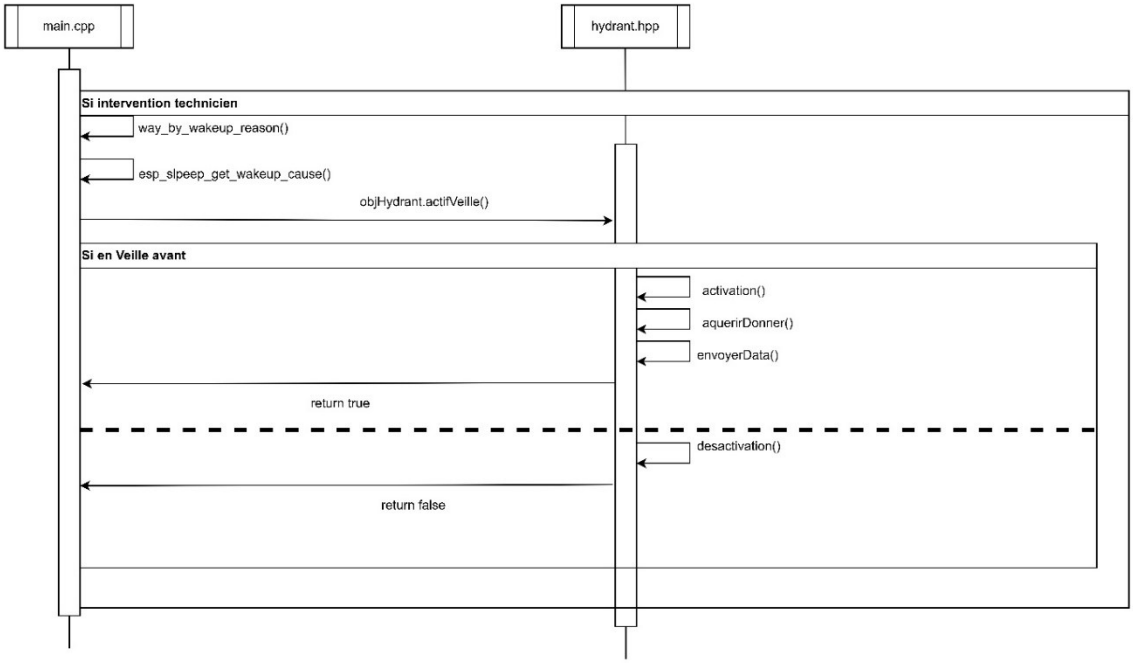
La cause du réveil est dû à un timer. Comme son nom l'indique, ce réveil se produit à des intervalles de temps réguliers définis par un timer.

Mettre en service ou en veille

La mise en service et faite lorsqu'un aimant et rapprocher sur l'ILS de mise en service. Ce qui va donner lieu à un réveil et exécuter. Sur la RTC_IO 25 qui est la broche que j'ai choisi pour la mise en service.

Pour programmer ce dernier en tant que source de réveil il suffit de configurer une broche RTC_IO pour le réveil EXT0 en utilisant l'API d'ESP-IDF ou d'Arduino.

```
// Configure le GPIO25 comme source de réveil quand la tension vaut 1V
esp_sleep_enable_ext0_wakeup(GPIO_NUM_25, HIGH);
```



Concernant notre système, une fois la manipulation du technicien effectuée, le code se réveillera en utilisant EXT0. Et en suite appeler la fonction actifVeille de la classe hydrant.

```
bool Hydrant::actifVeille()
{
    if (etatHydrant == VEILLE) // si en veille avant mise en marche
    {
        // activation
        activation();

        etatHydrant = FERME; // ouvert 0, fermé 1, renverse 2, veille 3.

        aquerirDonner();

        envoyerData();

        Serial.println("envoie SIGFOX");
        return true;
    }
    else // sinon mise en veille
    {
        desactivation();
        return false;
        // desactivation technitien pourquoi ??
    }
}
```

Cette fonction commence par vérifier si l'état de l'hydrant est en veille. Si l'hydrant est effectivement en veille, cela signifie qu'il est prêt à être activé. Et l'on procède alors à son activation, ce qui va permettre de programmer d'autres sources de réveil utilisé pour détecter un changement d'état. Après cette activation, ce dernier envoie une trame d'activation.

En revanche, si l'hydrant n'est pas en veille, cela signifie qu'il n'a pas besoin d'être activé. Dans ce cas, on s'occupe de le désactiver l'hydrant. Lorsque cette dernier est utilisé elle va désactiver toutes sources de réveil et ne programmer que le réveil via l'ILS du technicien.

```
esp_sleep_disable_wakeup_source(ESP_SLEEP_WAKEUP_ALL); // disable all wakeup source

// Configure le GPIO4 comme source de réveil quand la tension vaut 1V
esp_sleep_enable_ext0_wakeup(GPIO_NUM_25, HIGH);
```

La désactivation peut être nécessaire pour des raisons techniques non précisées.

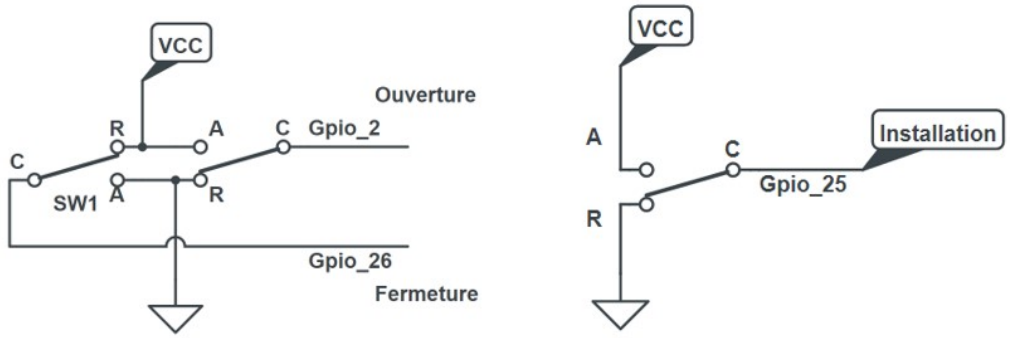
Détecter un changement d'état

Dans mon projet il a deux changements d'état qui provoque un réveil. Il y a soit d'une part l'ouverture/fermeture de la vanne ou alors le renversement suite a un accident par exemple.

Les Sources de réveils

Pour cela il suffit de programmer une source de réveil sur plusieurs broches. (voir doc contacts REED hydrants)

Capteur d'ouverture (présence d'eau) et capteur fermeture (absence d'eau) (schéma de gauche).



Légende :

- C : Commun
- R : Position au repos
- A : Position activé

Pour configurer un réveil sur plusieurs broches avec EXT1, il faut d'abord définir un masque binaire où chaque bit correspond à une broche spécifique.

```
#define BUTTON_PIN_BITMASK_OUVERT 0x100000004 // GPIO32 et GPIO02 => 2^32 + 2^2 = 4294967300 = 0x100000004
#define BUTTON_PIN_BITMASK_FERME 0x104000000  // GPIO32 et GPIO026 => 2^32 + 2^26 = 4362076160 = 0x104000000
```

Dans ce cas, si mon cas je dois pouvoir soit détecter une inclinaison et une ouverture ou une inclinaison et une fermeture (suite a une ouverture).

Détection Renversement et Fermeture :

```
// Configure le GPIO26 et GPIO32 comme source de réveil quand la tension vaut 1V
esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK_FERME, ESP_EXT1_WAKEUP_ANY_HIGH);
```

Détection Renversement et Ouverture :

```
// Configure le GPIO2 et GPIO32 comme source de réveil quand la tension vaut 1V
esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK_OUVERT, ESP_EXT1_WAKEUP_ANY_HIGH);
```

Adaptation des Sources de Réveil

Au démarrage de notre système, l'hydrant configure ses sources de réveil pour détecter les changements d'inclinaison et l'état d'ouverture. Cela permet de surveiller les changements initiaux de l'état de l'hydrant. Après une détection d'ouverture, le système adapte ses sources de réveil pour se concentrer spécifiquement sur les inclinaisons et la fermeture. Cette adaptation assure que le système reste vigilant sur les événements critiques suivants l'ouverture de l'hydrant.

En ajustant les sources de réveil en fonction de l'état actuel de l'hydrant, notre système optimise la surveillance et la gestion des événements, garantissant ainsi une réponse rapide et appropriée à chaque situation.

Gestion du Signal de Réveil de l'ADXL345

L'ADXL345 doit lui aussi envoyer un signal sur la broche 32 qui correspond à la broche de réveil de l'adxl345. Il faut maintenant programmer notre accélérateur pour qu'il envoie du courant lors qu'il détecte, un changement brutal de l'accélération.

```
ptrObjC_ADXL345->activeReveil(); // parametrer le reveil de l'ADXL
```

Ce qui va permettre a l'ADXL345 afin qu'il envoie un signal continu sur la broche 32 pour le réveil et détecte les changements brutaux d'accélération.

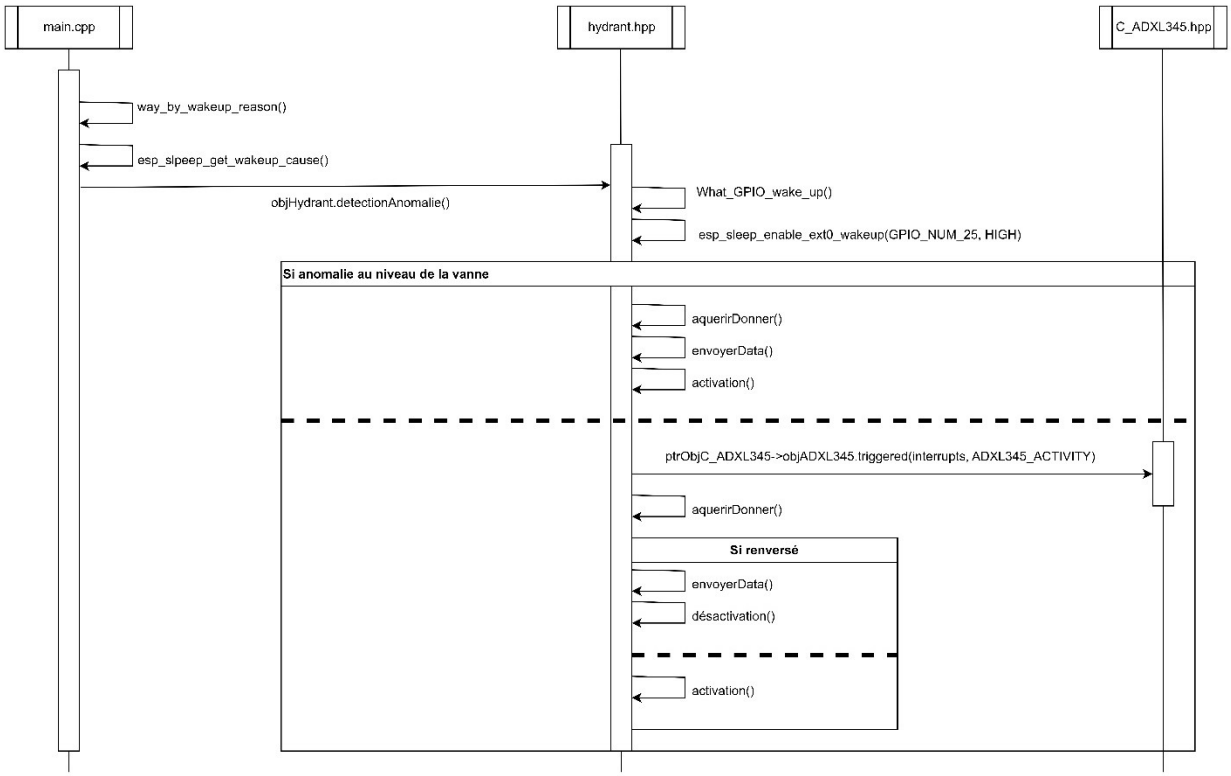
```
// Activer la détection de mouvement sur les axes spécifiés (X, Y, Z)
objADXL345.setActivityXYZ(1, 1, 0); // "adxl.setActivityXYZ(X, Y, Z);" (1 == ON, 0 == OFF)

// Définir le seuil de détection de mouvement (8 correspond à 62.5mg par incrément)
objADXL345.setActivityThreshold(8); // Inactivity thresholds (0-255) 3 detection lente - 8 detection correct - 18 detection brutal

// Configurer les interruptions pour qu'elles surviennent sur la broche INT1 ou INT2.
objADXL345.setImportantInterruptMapping(0, 0, 0, 1, 0);

// Activer les interruptions pour le mode d'activité (1 == ON, 0 == OFF)
objADXL345.ActivityINT(1);
```

Après le réveil il faut indiquer à l'ADXL345 que nous avons bien pris en compte son signal.

```
ptrObjC_ADXL345->objADXL345.triggered(interrupts, ADXL345_ACTIVITY);
```

Cela informe l'ADXL345 que son signal de réveil a été reconnu par le microcontrôleur, ce qui permet à l'ADXL345 de désactiver le signal continu.

Après quoi, l’ESP est en mesure de détecter un changement d'état sur les broches RTC_IO associées aux sources de réveil, pour exécuter un réveil EXT1.



Lorsque ce dernier ce réveil en EXT1 il exécute la fonction detectionAnomalie. Cette fonction et sous la forme d’un Switch avec comme expression de contrôle la broche de réveil que l’on obtient grâce a la fonction What_GPIO_wake_up.

Dans notre système de surveillance d'hydrant, la fonction `detecteAnomalie` est responsable de détecter toute anomalie ou événement inhabituel.

Cette fonction utilise un mécanisme de réveil basé sur trois sources distinctes pour détecter les différents états de l'hydrant : réveil suite à une ouverture (sur la broche 2), réveil suite à une fermeture (sur la broche 26), réveil suite a une (sur la broche 32).

Récolter des données

Récupérer l'angle

Lecture des valeurs d'accélération :

Nous commençons par lire les informations d'accélération sur les trois.

```
objADXL345.readAccel(&x, &y, &z);
```

Cela nous donne les valeurs numériques brutes des accélérations sur les axes x, y, et z.

Calcul des angles :

Étant donné que notre hydrant n'est généralement soumis à aucun mouvement autre que les forces statiques, la seule force exercée sur lui est celle de la gravité. La gravité, avec une accélération de 1g (où 1g est la référence de l'unité d'accélération due à la gravité terrestre), est la seule accélération que notre système subit.

Selon la documentation technique de l'ADXL345, la précision du capteur est de 4 mg/LSB (Least Significant Bit), ce qui signifie que chaque unité numérique dans les valeurs lues correspond à une accélération de 4 mg. Par exemple, si la valeur numérique sur un axe est 250 (et zéro sur les autres axes), cela signifie que cet axe subit une accélération de 250 * 4 mg = 1000 mg = 1g, représentant ainsi toute l'accélération due à la gravité.

Nous multiplions par 90 pour convertir l'accélération en un angle.

```
angleX = 90 * PRECISION * abs(x); // calcule angle X
angleY = 90 * PRECISION * abs(y); // calcule angle Y
angleZ = 90 * PRECISION * abs(y); // calcule angle Z
```

Récupération de la Température

On récupère la température grâce au capteur de température BMP280, exprimée en degrés Celsius.

```
ptrObjC_BMP280->setTempC();
```

Envoyer les données

Si un envoie de donnée et nécessaire alors ce dernier sera envoyé a notre modem Sigfox pour envoyer nos données au cloud Sigfox. Or Sigfox ne permet d'envoyer que 12octet de donnée par message. C'est pour quoi nous devons faire entrer les donnes que l'on veut envoyer dans ces 12 octets.

b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11

Pour faire rentrer nos donnée (état de l'hydrant, angle, température) a envoyer dans cette trame, nous avons fait correspondre un état à un chiffre et multiplier par 10 et 100 l'angle et la température pour ne pas avoir de chiffre a virgule.

Composition de la trame

État de l'hydrant : 1 caractère hexadécimal. Cela suffit car la valeur de l'état ne dépasse jamais 3 en décimal.

Angle : 3 caractères hexadécimaux. L'angle, multiplié par 10 pour conserver une décimale, ne dépasse jamais 900 en décimal, ce qui peut être représenté par 3 caractères hexadécimaux.

Température : 4 caractères hexadécimaux. La température, multipliée par 100 pour conserver deux décimales, ne dépasse jamais 8500 en décimal, ce qui nécessite 4 caractères hexadécimaux.

Ce qui nous fait utiliser 4 octets seulement

N0	N1	N2	N3	N4	N5	N6	N7

Exemple :

Etat : Ouvert ; Angle : 11.4 ; Température : 26.76

Etat : 0 ; Angle : 114 ; Température : 2676

Etat : 0 ; Angle : 72 ; Température : A74

Construction de la Trame Sigfox

Conversion des Données

Nous convertissons l'état de l'hydrant, l'angle et la température en une chaîne de caractères hexadécimaux compacte. Les conversions nécessaires sont effectuées pour assurer une précision adéquate tout en supprimant les virgules des valeurs flottantes.

```
int prmAngle10 = prmAngle * 10;              // angle * 10 pour avoir de la precision et supprimer la virgule
int prmTemperature100 = prmTemperature * 100; // temperature * 100 pour avoir de la precision et supprimer la virgule

// Vérifier que les valeurs sont dans les plages attendues
if (prmEtat > 15)
{
    prmEtat = 15; // 15 est le maximum pour un demi-octet
}
if (prmAngle10 > 4095)
{
    prmAngle10 = 4095; // 4095 est le maximum pour 3 caractères hexadécimaux
}
if (prmTemperature100 > 65535)
{
    prmTemperature100 = 65535; // 65535 est le maximum pour 4 caractères hexadécimaux
}
```

Formatage de la trame

```
char frame[9];// S'assurer qu'il y a suffisamment d'espace
```

La fonction “ sprintf ’’ est utilisée pour formater les données en une chaîne de caractères hexadécimaux.

```
// Construire la trame
sprintf(frame, "%1X%03X%04X", prmEtat, prmAngle10, prmTemperature100);
// ou
// >> 8 et & 0x00FF
```

% : Indique le début d'un spécificateur de format.

0 : Remplit le champ avec des zéros si la valeur est plus courte que la largeur spécifiée.

1,3,4 : Spécifie que la largeur minimale du champ est de 1,3 ou 4 caractères.

X : Indique que la valeur doit être formatée en hexadécimal.

Envoie message au Modem

Pour dialoguer avec le modem et envoyer un message au cloud Sigfox, nous utilisons des commandes AT. Pour envoyer notre message, nous utilisons la commande AT$SF= suivie de notre trame de données en format hexadécimal. Le caractère \r ajouté à la fin de la commande indique le retour à la ligne.

```
String data = buildFrame(prmEtat, prmAngle, prmTemperature);

Trame = "AT$SF=" + data;
objSwSer1.print(Trame + '\r');
```

Service REST

Réception donnée

Pour recevoir les données envoyées par Sigfox, nous utilisons une adresse de callback, qui est l'URL de notre API. Cette URL est configurée dans le backend Sigfox pour recevoir les messages. Lorsqu'un message est envoyé, il est encapsulé dans une structure spécifique. Cette structure garantit que seules les données conformes à notre format peuvent être enregistrées.

```
{
  "this": "succeeded",
  "by": "postting",
  "the": "SigFox",
  "with": [
    {
      "thing": "Hydrant2024",
      "created": "{time}",
      "content": {
        "device": "{device}",
        "data": "{data}"
      }
    }
  ]
}
```

Ce format inclut des champs spécifiques tels que l'identifiant du dispositif Sigfox (device) et les données envoyées (data). Même si quelqu'un découvre notre adresse de service REST, ils ne pourront pas envoyer des messages valides sans connaître cette structure précise.

Intégration avec la base de données

Une fois les données reçues, nous les intégrons dans notre base de données. Voici le processus détaillé :

Extraction des Données :

Nous extrayons la trame de données encapsulée dans le message JSON reçu.

```
$prmDonnee = $data['with'][0]['content']['data'];
```

Décodage des Données :

Les données envoyées par Sigfox ont été préalablement compactées et formatées en une trame hexadécimale. Maintenant, nous devons décompacter et déformatter cette trame pour récupérer les valeurs exactes de l'état de l'hydrant, de l'angle et de la température.

- État de l'hydrant : représenté par le premier caractère de la trame hexadécimale. Converti en décimal, il nous donne l'état exact de l'hydrant.
- Angle : représenté par les trois caractères suivants de la trame. Avant l'envoi, l'angle avait été multiplié par 10 pour garantir la précision, puis converti en hexadécimal. Nous devons le reconvertir et le diviser par 10 pour obtenir l'angle original.

- Température : représentée par les quatre derniers caractères de la trame. Avant l'envoi, la température avait été multipliée par 100 pour garantir la précision, puis convertie en hexadécimal. Nous devons la reconvertir et la diviser par 100 pour obtenir la température originale.

```
$etatHex = substr($prmDonnee, 0, 1);
$angleHex = substr($prmDonnee, 1, 3);
$tempHex = substr($prmDonnee, 4, 4);

$etat = hexdec($etatHex);
$angle = hexdec($angleHex) / 10;
$temp = hexdec($tempHex) / 100;
```

Identification de l'Hydrant :

Pour associer les données reçues à un hydrant spécifique, nous utilisons l'identifiant Sigfox du dispositif. Cet identifiant nous permet de retrouver l'identifiant interne de l'hydrant dans notre base de données.

```
$prmDevice = $data['with'][0]['content']['device'];

$r = $this->model->getIDbyCodeSigFox($prmDevice);
$idHydrant = $r[0]['idHydrant'];
```

Création d'une Nouvelle Entrée :

Avec les informations décodées et l'identifiant de l'hydrant, nous créons une nouvelle entrée dans la base de données. Cette nouvelle entrée inclut l'état de l'hydrant, l'angle et la température.

```
$result = $mdonneeModel->createDonnee($idHydrant, $etatText, $angle, $temp);
```

Gestion des alertes

Après l'enregistrement des données des hydrants dans la base de données, notre système vérifie si l'état reçu du cloud Sigfox doit être traité comme une alerte. Trois situations déclenchent une alerte : l'ouverture, la fermeture (suite à une ouverture), et l'inclinaison anormale de l'hydrant. Pour chaque type d'alerte, un message personnalisé est envoyé aux destinataires concernés.

À chaque requête du service REST, nous vérifions si l'état de l'hydrant correspond à l'une des situations d'alerte. Si tel est le cas, une alerte est automatiquement déclenchée.

Cette alerte est envoyée sous forme de mail qui contient un message personnalisé en fonction de la situation détectée, ainsi que des informations précises sur l'hydrant concerné.

```
switch ($etat) {
    case 0:
        $etatText = "OUVERT";
        $message = "🚰 Alerte: L'hydrant " . $idHydrant . " est actuellement OUVERT. Veuillez vérifier immédiatement. Plus d'infos: ";
        break;
    case 1:
        $etatText = "FERME";
        $message = "✅ L'hydrant " . $idHydrant . " est actuellement FERMÉ. Tout est en ordre. Plus d'infos: ";

        break;
    case 2:
        $etatText = "RENVERSE";
        $message = "🚨 Alerte: L'hydrant " . $idHydrant . " est RENVERSé. Intervention nécessaire. Plus d'infos: ";
        break;
```

De plus, un lien est inclus dans l'email pour permettre aux destinataires d'accéder rapidement aux détails de l'hydrant associé.

```
$link = "https://surveillancehydrant.alwaysdata.net/Ctableau/detail/" . $idHydrant;
$emailMessage = $message . $link;

$email = \Config\Services::email();
$email->setTo('louisBodin.hydrant@gmail.com');
$email->setFrom('projet.hydrant@gmail.com', 'Projet Hydrant');
$email->setSubject('Alerte Hydrant ' . $idHydrant);
$email->setMessage($emailMessage);
```

Conclusion personnelle