Brevet de Technicien Supérieur SNIR

Session 2021

Lycée Polyvalent Touchard-Washington

Projet : Ballon-sonde

Membres de l'équipe

Sabri Sofiane, Pacot Antoine,

Yousfi Ahmed, Bougeot Louis

Dossier technique du Projet

- Partie Personnel -

Sommaire

I. Situation dans le projet

1. Diagrammes du projet

Durant la phase de conception, je me suis beaucoup intéressé à l'envoi des données ainsi que l'enregistrement de celle-ci. Celles-ci nécessitent des modifications de format afin d'être envoyées ou enregistrées.

Voici le diagramme de cas d'utilisation de la partie « Dans le Ballon » dans laquelle je me situe :

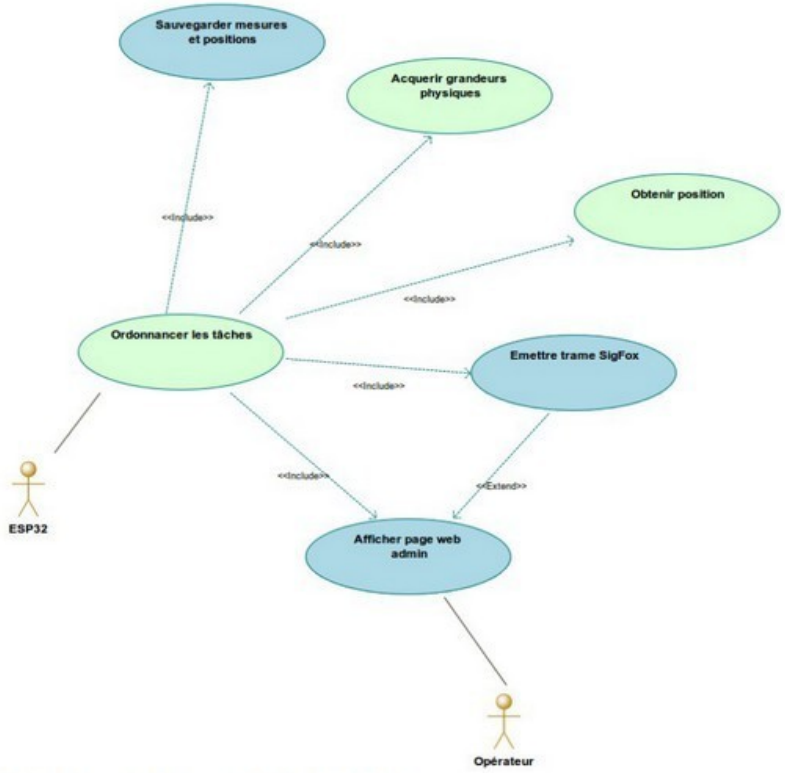


Figure 1: Cas d'utilisation dans le ballon

Comme vous pouvez le voir, ma partie est liée à celle de l'étudiant 3, dont les cas figurent en vert sur le diagramme

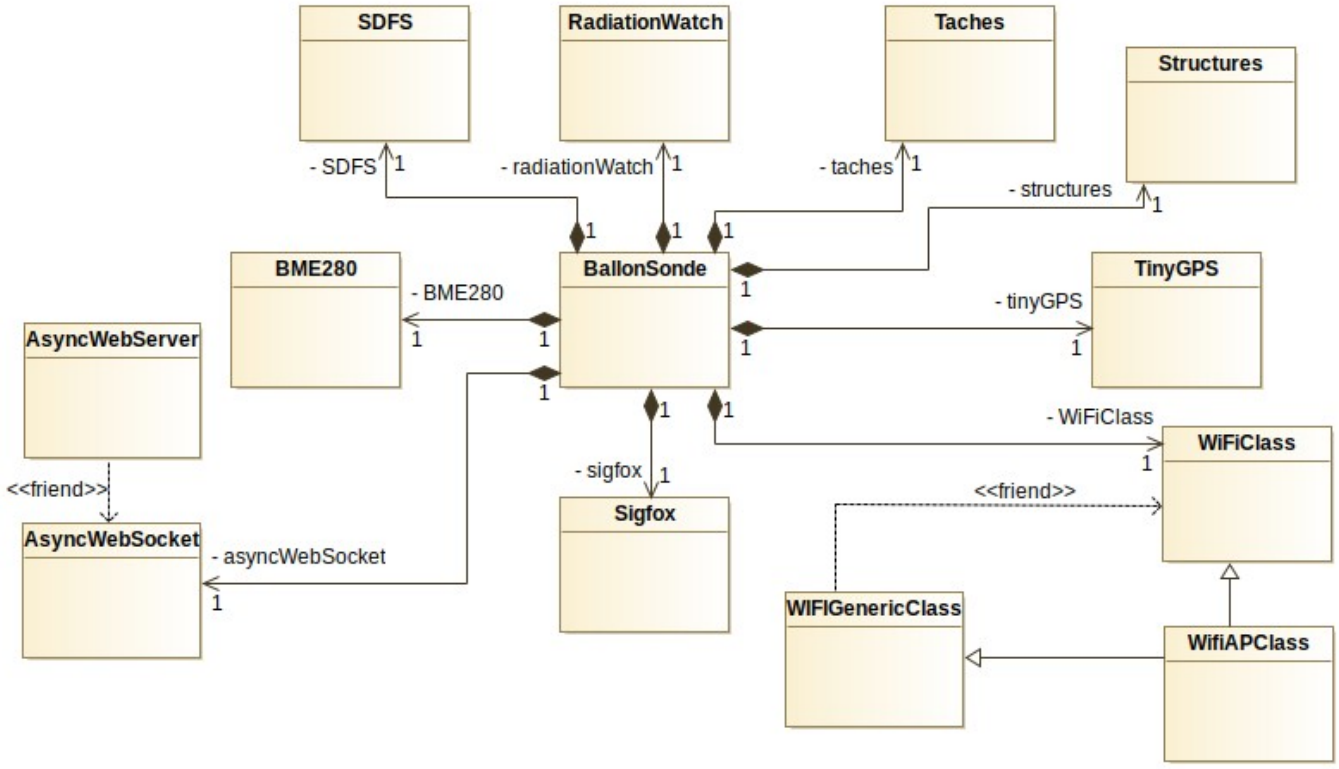
Étant l'étudiant 1, je m'occupe des cas figurant en bleu sur le diagramme, c'est à dire :

- Sauvegarder mesure et position qui sauvegarde les données dans une carte sd, pour qu'elles soient exploitées à la fin du vol. Cet enregistrement permet d'avoir des données plus précises à la fin du vol.

- Émettre trame SigFox qui constitue l'envoi des données pendant le vol. Cet envoi permet de localiser le ballon pendant le vol, et d'avoir les données en temps réel.

- Afficher page web admin qui permet d'accéder à un page web via le réseau WiFi de l'ESP. Cette page web affiche les données en temps réel et possède un bouton pour tester l'émission d'une trame SigFox.

Voici un diagramme de classe complet de la partie « Dans le ballon »



Ce diagramme réunit toutes les classes que j'ai dû utiliser ainsi que celle de l'étudiant 3.

Pour ma part, j'utilise les classes Sigfox, Wifi, structures, taches, et asyncWebServer ainsi que leurs fonctions

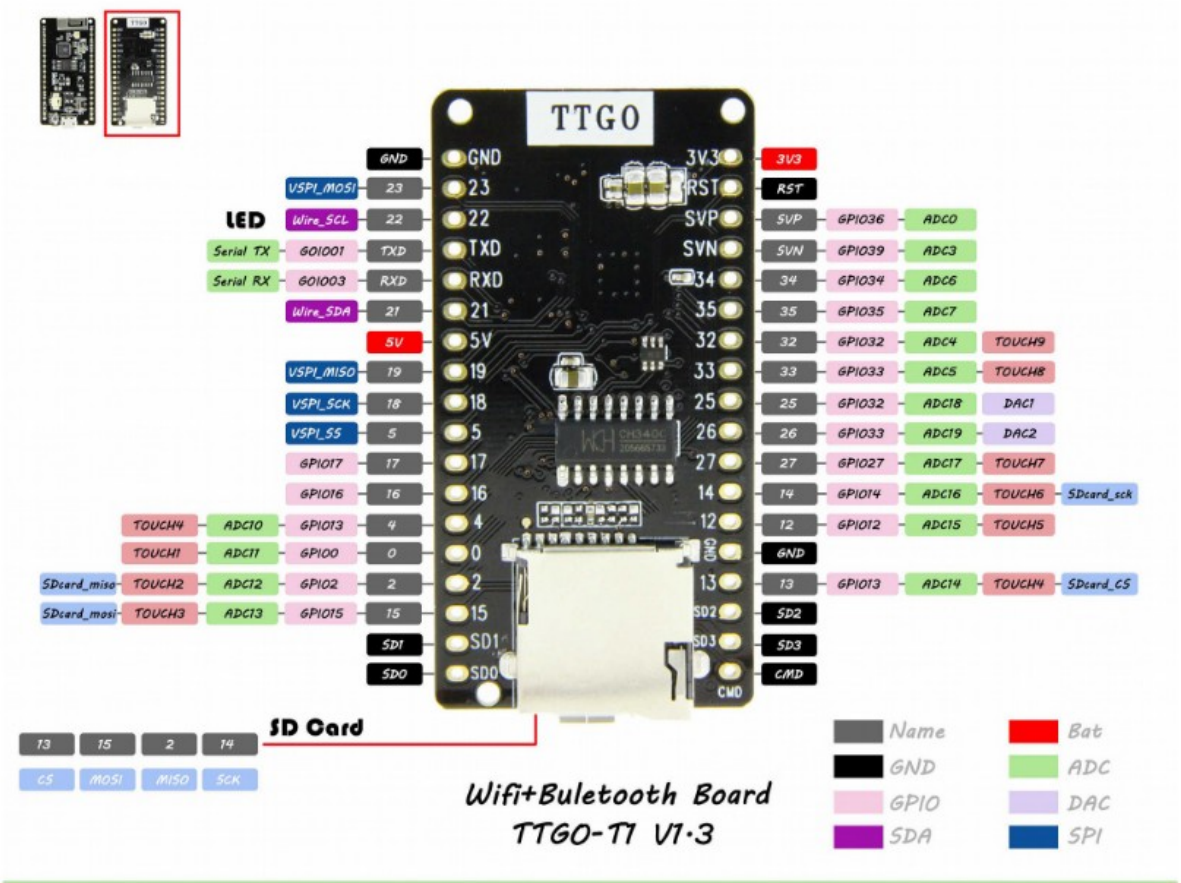
II. Cas d’utilisation « Sauvegarder mesures et positions »

1.1 Description de la tâche

- Ce cas consiste à sauvegarder les données stockées dans une structure partagée, dans une carte SD, dans un fichier CSV.
- Ces données doivent être complète et dans un format exploitable pour l’importation dans une base de données après le vol du Ballon.
- Si il y a une erreur d’enregistrement, la tache ne doit pas être bloquante.

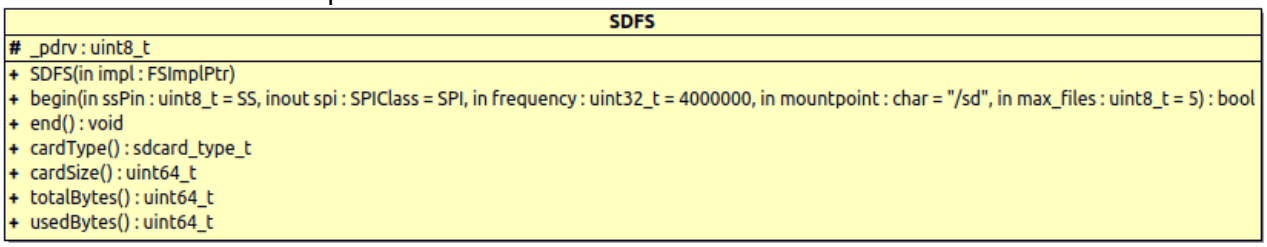
1.2 Conception détaillée

Tout d’abord, voici un schéma montrant le câblage de l’esp32



Câblage SD :
CS → GPIO13 (broche 13)
MOSI → GPIO15 (broche 15)
MISO → GPIO2 (broche 2)
SCK → GPIO14 (broche 14)

Voici la classe SD utilisée pour l'utilisation de la carte SD :



Pour l'utilisation de la carte SD, j'utilise plusieurs fonctions, provenant de classes différentes :

- SPI.begin() : Permet d'initialiser le système de fichier en donnant en paramètres les broches sur lesquels est connectée la carte SD.

- SD.begin() : Permet d'initialiser la connexion à la carte SD

- SD.open() : permet d'ouvrir le fichier dans lequel les données sont enregistrées. Créée le fichier si il n'existe pas.

- fichier.println() : pour écrire dans le fichier en question

Voici un extrait de code pour la tâche tacheCarteSD :

```
xTaskCreate(
        tacheCarteSD, /* Task function. */
        "tacheCarteSD", /* name of task. */
        10000, /* Stack size of task */
        NULL, /* parameter of the task */
        1, /* priority of the task */
        NULL); /* Task handle to keep track of created task */
```

```
void tacheCarteSD(void *pvParameters) // <- une tâche
{

    TickType_t xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();

    //initialisation systeme de fichier
    SPI.begin(SCK_PIN, MISO_PIN, MOSI_PIN, CS_PIN);
    delay(10);

    //initialisation CarteSD
    if (!SD.begin(CS_PIN, SPI, 4000000, "/sd")) {
        Serial.println("Erreur montage Carte SD");
        return;
    }
```

```
fichierCSV = SD.open("/TestCSV.csv", FILE_APPEND); //ouverture du fichier en modification
if (!fichierCSV) {
    Serial.print("Echec ouverture du fichier\n");
}
if (fichierCSV.println(sDonnees)) //écriture de la ligne de données dans la carte SD
{
    Serial.print("Donnee enregistrée\n");
} else {
    Serial.print("pb enregistrement donnee\n");
}
fichierCSV.close();
```

Pour la création des différentes tâches, nous utilisons la fonction xTaskCreate() de FreeRTOS. Une fonction de mutex est aussi disponible avec ce système d'exploitation.

III. Cas d’utilisation « Émettre trame Sigfox »

1. Description de la tâche

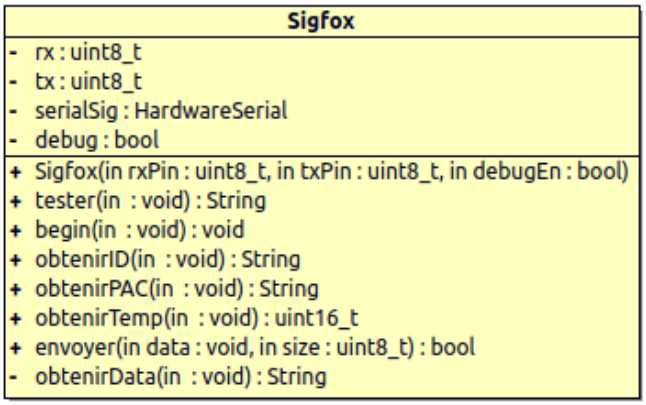
- Cette tâche est la plus importante, car c’est celle qui permet de transmettre les données en temps réel. Et c’est également grâce à cet envoi de données que nous pourrons retrouver la nacelle du ballon.

- Cette trame est envoyée toutes les 10 minutes, car il s’agit d’une contrainte imposée par Sigfox. Nous sommes limités à 6 messages par heure. Donc 1 message toutes les 10 minutes.

- Nous sommes également limités sur la taille des données, la limite est de 96 bits. Le format a donc été travaillé afin de respecter cette contrainte.

2. Conception détaillée

Voici, ci-dessous, le diagramme de la classe utilisée lors de ce cas d’utilisation.



Une méthode a été rajoutée à la classe Sigfox : EncoderTrame().

Cette méthode permet d’encoder la trame pour qu’elle ait le bon format à envoyer.

Afin d’assurer le fonctionnement de la tâche Sigfox, il est nécessaire d’instancier un objet pour cette classe, ici appelé BallonSig.

Les méthodes utilisées sont les suivantes :

- BallonSig.begin() : pour initialiser la connexion au module Sigfox

- BallonSig.coderTrame() : permet de coder la trame au bon format

- BallonSig.envoyer() : pour l’envoi de la trame

Voici un extrait de code qui montre l'utilisation de la classe Sigfox :

```
xTaskCreate(
        tacheCarteSD, /* Task function. */
        "tacheCarteSD", /* name of task. */
        10000, /* Stack size of task */
        NULL, /* parameter of the task */
        1, /* priority of the task */
        NULL); /* Task handle to keep track of created task */
```

```
void tacheSigfox(void *pvParameters) // <- une tâche
{
    delay(30000);

    TickType_t xLastWakeTime;
    xLastWakeTime = xTaskGetTickCount();

    BallonSig.begin();

    for (;;) // <- boucle infinie
    {
        // Verrouillage du mutex
        xSemaphoreTake(mutex, portMAX_DELAY);

        BallonSig.coderTrame(lesDonnees.position, lesDonnees.DonneesCapteurs);
        BallonSig.envoyer(BallonSig.trame, sizeof (BallonSig.trame));

        // Déverrouillage du mutex
        xSemaphoreGive(mutex);

        vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(600000)); // toutes les 600000 ms = 10 minutes
    }
}
```

3. Test unitaire

3.1 Test unitaire de la classe Sigfox

Fiche de test unitaire (matériel)			
Nature :	Fonctionnel	**Référence :**	F1.1
Module :	***Classse Sigfox (cas d'utilisation EmettreTrameSigfox)***		
Objectif :	Vérifier que le matériel est prêt à être utilisé		
Date :	04/05/2021		

Condition du test		
État initial du module		**Environnement du test**
Programme		PC Linux + esp32 connecté en USB avec module sigfox (non suspendu) connecté sur les broches 26 (TX_ESP) et 27(RX_ESP) de l'ESP Accès au backend sigfox + compte utilisateur
Conditions initiales		
L'esp32 est sous tension. Le module SigFox est connecté à l'esp32		

Procédure de test		
Repère	**Opérations**	**Résultats attendus**
1	Vérifier si le module Sigfox et son antenne sont bien connectés à l'esp32	
2	Vérifier que l'esp32 est bien connecté à l'ordi via un câble USB, si ce n'est pas le cas, le faire	La/les led(s) de l'ESP est/sont allumée(s)
3	- Accéder au backend de sigfox (https://backend.sigfox.com/auth/login) - Se connecter avec un compte utilisateur - Naviguer dans Device - Vérifier le statut de communication ainsi que le « token state » du module qui correspond (regarder l'ID) en passant la souris sur chacun d'entre eux	- Le statut de communication affiche « communication allowed » en passant la souris sur le point blanc de la colonne « communication state » sur la ligne qui concerne le module utilisé - le « Token state » affiche « OK » en passant la souris sur l'icône de la colonne « token state » sur la ligne qui concerne le module utilisé

Fiche de test unitaire (programme)			
Nature :	Fonctionnel	**Référence :**	F1.1
Module :	***Classse Sigfox (cas d'utilisation EmettreTrameSigfox)***		
Objectif :	Vérifier le fonctionnement de l'encodage et de l'émission d'une trame Sigfox		
Date :	04/05/2021		

Condition du test		
État initial du module		**Environnement du test**
Programme	**TestSigfox**	- PC Linux avec Netbeans et Putty (les deux logiciels sont configurés pour l'ESP) - esp32 connecté en USB au PC avec module sigfox (non suspendu) connecté sur les broches 26 (RX_ESP) et 27(TX_ESP) de l'ESP - Accès au backend sigfox + compte utilisateur

Conditions initiales
Avoir effectué les opérations de la fiche de test concernant le matériel (esp32 sous tension, connecté à l'ordinateur via un câble USB, et le module SigFox (et son antenne) connecté à l'ESP)

Procédure de test		
Repère	**Opérations**	**Résultats attendus**
1	- Télécharger et ouvrir le programme « BridgeSigfox » via netbeans - Exécuter le programme (le téléverser dans l'ESP) - Ouvrir Putty, charger la configuration « esp32 » et cliquer sur le bouton open - Entrer la commande « AT »	- Le message « OK » doit s'afficher sur le terminal Putty
2	- Entrer la commande « AT$SF=01020304050607080910 1112 » - Se connecter au backend Sigfox (avec un compte utilisateur) : https://backend.sigfox.com/ - Naviguer dans Device, cliquer sur l'ID du module (ici : 2EE048), cliquer sur message (dans la colonne de gauche) et vérifier si le message est bien reçu en observant l'horodatage (légèrement postérieur à l'heure d'envoi de la commande)	Le message « OK » doit s'afficher sur le terminal Putty Le message que vous avez envoyé (après le '=' dans la commande) doit apparaître comme étant le dernier arrivé.

3	- Fermer le programme « BridgeSigFox » et ouvrir le programme « TestSigfox » - Exécuter le programme tel quel (toutes les valeurs sont positives par défaut)	
4	- Se connecter au backend Sigfox (avec un compte utilisateur) : https://backend.sigfox.com/ - Naviguer dans Device, cliquer sur l'ID du module (ici : 2EE048), cliquer sur message (dans la colonne de gauche) et vérifier si le message est bien reçu en observant l'horodatage (légèrement postérieur à l'heure d'exécution du programme)	Le message « 5B8B7AA038136F0103AF6E » doit apparaître comme étant le dernier arrivé et l'horodatage doit correspondre comme étant légèrement postérieur à l'heure d'exécution du programme
5	- Ajouter « /* » à la ligne 51 du programme ainsi que « */ » à la ligne 67 afin de mettre les valeurs positives en commentaire. - Supprimer « /* » de la ligne 70 et « */ » de la ligne 86 afin d'activer la trame comprenant les valeurs négatives - Executer le programme	
6	- Se connecter au backend Sigfox (avec un compte utilisateur) : https://backend.sigfox.com/ - Naviguer dans Device, cliquer sur l'ID du module (ici : 2EE048), cliquer sur message (dans la colonne de gauche) et vérifier si le message est bien reçu en observant l'horodatage (légèrement postérieur à l'heure d'exécution du programme)	Le message « 5C86021544C610A6E603E358 » doit apparaître comme étant le dernier arrivé et l'horodatage doit correspondre comme étant légèrement postérieur à l'heure d'exécution du programme

3.2 Compte rendu du test unitaire

Lors du codage de la trame, il y avait une erreur de données, car la conversion de la latitude était erronée. En effet, la latitude est convertie en entier, et donc multipliée par 1 million. Hors si on multiplie juste par 1 000 000, la valeur se retrouve être arrondie. Il faut donc multiplier par 1 000 000.0 pour garder la bonne valeur.

III. Cas d’utilisation « Afficher Page Web admin »

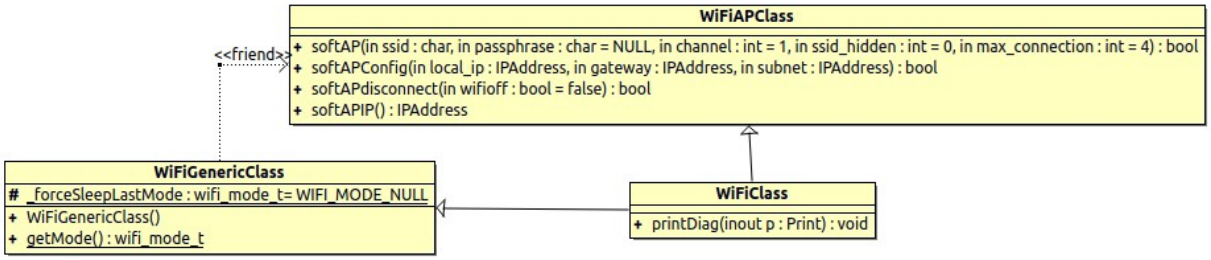
1. Description de la tâche

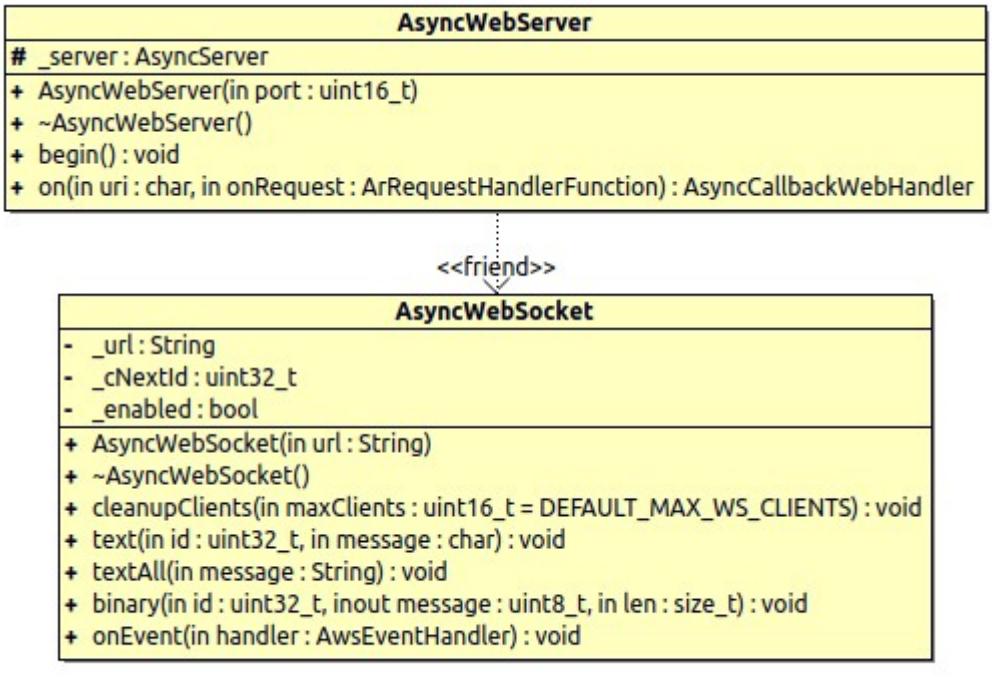
Cette tâche a pour objectif d’afficher une page web lorsqu’on se connecte en Wifi à l’ESP. Cette page permet un affichage en temps réel des données et possède un bouton « Émettre trame sigfox »

Cette page doit servir de test pour voir si les données sont bien reçues et stockées dans la structure partagée, et si l’émission de la trame sigfox fonctionne correctement.

2. Conception détaillée

Voici ci-dessous, les classes utilisées pour cette tâche.





Pour créer le point d'accès WiFi, il faut utiliser les méthodes suivantes :

- Wifi.softAPconfig() : prends en paramètres l'adresse IP locale, l'adresse de passerelle, et le masque afin de configurer le point d'accès.

- Wifi.softAP() : permet de créer le point d'accès avec les paramètres précisés dans la méthode précédente.

Pour utiliser le serveur en web socket, il faut créer un objet de type websocket, ici ws.

Ensuite, il faut utiliser les méthodes suivantes :

- ws.begin() afin de démarrer le serveur

- ws.on() pour rediriger vers la page web

Voici des extraits de code pour utiliser cette classe :

```
#include <Arduino.h>
#include <WiFi.h>
const char *ssid = "ESP32AP";
const char *password = "tototototo";
```

```
WiFi.softAPConfig(IPAddress(192, 168, 5, 1), IPAddress(192, 168, 5, 1),IPAddress(255, 255, 255, 0));
WiFi.softAP(ssid, password); //Initialisation avec WiFi.softAP / ssid et password
void loop()
{
    Serial.print("Adresse IP: ");
    Serial.println(WiFi.softAPIP()); //Affiche l'adresse IP de l'ESP32 avec WiFi.SoftIP
    delay(5000);

}
```

IV. Bilan de la réalisation personnelle

Les points du projet qui sont validés sont les cas « Sauvegarder mesures et position » et « Émettre trame Sigfox ». Il me reste quelques erreurs pour le cas « Afficher page web Admin » qui seront réglées d'ici peu.

Ces erreurs résident dans la page HTML en elle-même et non pas dans la création du serveur.

Pour finir le projet est très intéressant, puisqu'il m'a permit d'apprendre à programmer d'une manière différente, avec des librairies que je ne connaissais pas tels que « Sigfox » ou encore « AsyncWebSocket ».

Ce projet m'a également permis de travailler avec des personnes dont je n'avais pas l'habitude de travailler, et j'en suis ravi. Je remercie donc ces personnes pour l'aide qu'elles m'ont fournie.