



Piscine Bluetooth Low Energy

RAPPORT DE PROJET – LYCEE DU REMPART

Paul MICHELS | BTS Système Numérique | 2018
GitHub : github.com/paulmichels/SNIRPiscineBLE

Sommaire

I - Problématique	4
II - Besoin des personnes clientes	5
III – Cahier des charges	5
1 – Contraintes économiques	5
2 – Contraintes de performances et d'utilisation	6
IV – Concepts clés	7
1 – Le Bluetooth Low Energy	7
<i>L'organisation GATT</i>	7
<i>L'UUID</i>	8
2 – Introduction à Android	8
<i>Android studio</i>	8
<i>Android SDK</i>	8
<i>Java</i>	9
<i>XML</i>	9
3 – Composantes d'une application Android	9
<i>Android Manifest</i>	9
<i>Package</i>	9
<i>Ressources</i>	10
<i>Activité</i>	10
<i>Fragment</i>	10
<i>Service</i>	10
V – Diagrammes	11
4 – Diagramme d'état Android	14
<i>Activité</i>	14
<i>Fragment</i>	14
VI – Déroulement du projet	16
1 – GANTT prévisionnel	16
2 – Phase de développement	17

<i>Première IHM</i>	17
<i>Deuxième IHM avec activités</i>	18
<i>IHM définitive</i>	18
<i>Mise à jour Android</i>	19
<i>Incompatibilité ListView / Fragments</i>	19
<i>Utilisation de handlers</i>	19
4 – Calculs	19
VII – Fonctionnement de l'application.....	20
1 – Itération principale	20
2 – L'activité.....	20
3 – Service Bluetooth Low Energy	22
4 – Les fragments	23
<i>Fragment Bluetooth</i>	23
<i>Fragment Data</i>	24
<i>Fragment SMS</i>	25
<i>Fragment Paramètres</i>	25
<i>Fragment à propos</i>	26
VIII – Cahier des recettes	27
1 – Débogage de l'application	27
2 – Campagne	28
<i>Objectif de la campagne</i>	28
<i>Liste des scenarii</i>	28
<i>Exécution et bilan</i>	28
IX – Conclusion	30
Annexe 1 – Interface Homme-Machine	31
Annexe 2 – Méthodes relatives au fonctionnement du Bluetooth	33
1 - BluetoothFragment.java.....	33
2 – Dans DrawerBase.java.....	34
3 – Dans DataFragment.java.....	37

I - Problématique

D'une part, la diffusion des smartphones en France a connu une ascension fulgurante, en effet, la part des français disposant d'un smartphone est passée de 17% en 2011 à 74 % en 2017. Peu à peu, les smartphones ont supplanté les téléphones portables dits « classiques », et ont été adoptés par toutes les catégories sociales.

D'autre part, l'entretien d'une piscine n'est jamais une chose facile, et demande beaucoup d'attention et d'efforts. Des mesures de pH et de taux de chlore sont à effectuer régulièrement pour s'assurer de l'équilibre chimique dans l'eau.

A partir de ces données, on peut supposer que, chez une personne propriétaire d'une piscine à entretenir, il y a une très forte probabilité pour qu'elle possède un smartphone. On peut alors projeter un travail précis : Comment surveiller l'état de l'eau d'une piscine à l'aide d'un smartphone ?

II - Besoin des personnes clientes

L'utilisateur doit pouvoir être informé de l'état de sa piscine à partir d'une interface sur son téléphone, relié par Bluetooth à un flotteur capable de recueillir des données dans sa piscine. Il peut également alerter le pisciniste responsable en cas de problèmes.

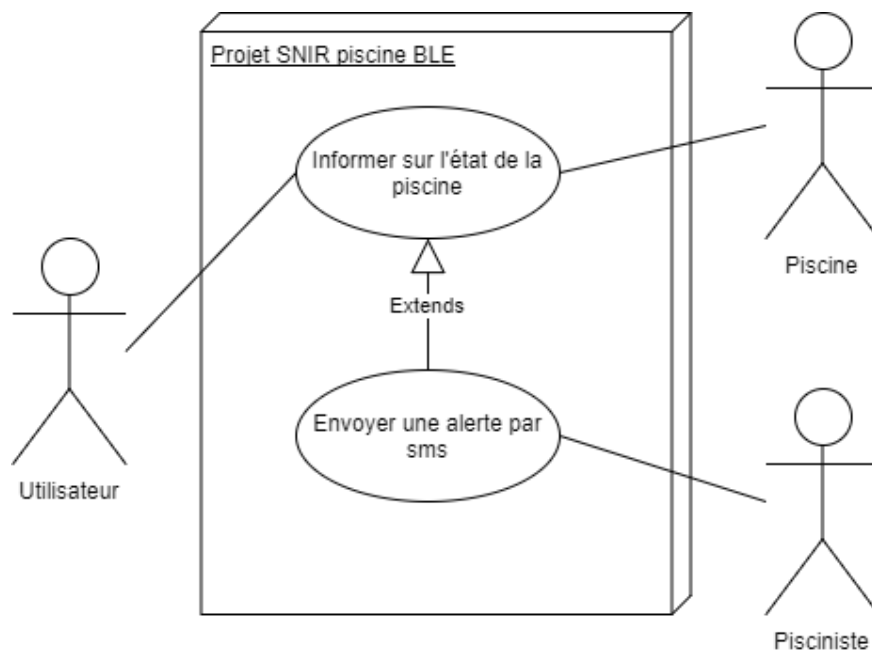


DIAGRAMME DE CAS D'UTILISATION

III – Cahier des charges

1 – CONTRAINTES ECONOMIQUES

Pour le prototype de ce projet, un certain matériel est requis. Nous partirons du principe que l'utilisateur possède déjà un smartphone Android. Il reste alors à réunir un certain nombre de capteurs (pH, température et chlore) et un microprocesseur capable d'émettre des ondes Bluetooth.

Composant	Prix
Sonde pH	30€
Sonde température	16€
Sonde chlore	80€
Carte Genuino 101	38€
Total	164€

2 – CONTRAINTES DE PERFORMANCES ET D'UTILISATION

Certaines contraintes liées aux performances peuvent être rencontrées. Tout d'abord, les contraintes liées au Bluetooth. Cette technologie sans fil consomme beaucoup d'énergie, un problème lié à l'autonomie de la batterie risque donc d'être majeur. C'est pourquoi le choix de l'utilisation du Bluetooth Low Energy est justifié : celui-ci consomme presque 100 fois moins d'énergie que le Bluetooth classique. Sa portée est d'un maximum de 50 mètres dans des conditions optimales ce qui ne devrait pas être une contrainte majeure en soi.

Ensuite, certains smartphones ne sont pas forcément compatibles avec la technologie Bluetooth Low Energy. Les smartphones Android doivent être de version 4.4 (distribution KitKat) minimum pour supporter le BLE. Ci-dessous un tableau présentant la répartition des versions Android en avril 2018 :

Version	Nom de code	Distribution
2.3.3 – 2.3.7	GingerBread	0.3%
4.0.3 – 4.0.4	Ice Cream Sandwich	0.4%
4.1.x, 4.2.x et 4.3	Jelly Bean	4.5%
4.4	Kit Kat	10.5%
5.0 et 5.1	Lollipop	22.9%
6.0	Marshmallow	26%
7.0 et 7.1	Nougat	30.8%
8.0 et 8.1	Oreo	4.6%

La compatibilité entre les smartphone Android en circulation et le Bluetooth Low Energy aujourd'hui est donc de 94.9%.

Enfin, il faut que l'application ne soit pas trop lourde sur ses processus pour que l'expérience de l'utilisateur reste agréable et fluide. Cette contrainte ne sera pas forcément rencontrée lors de la phase de développement : les smartphones sont des appareils relativement puissants et l'application est en théorie légère dans les threads.

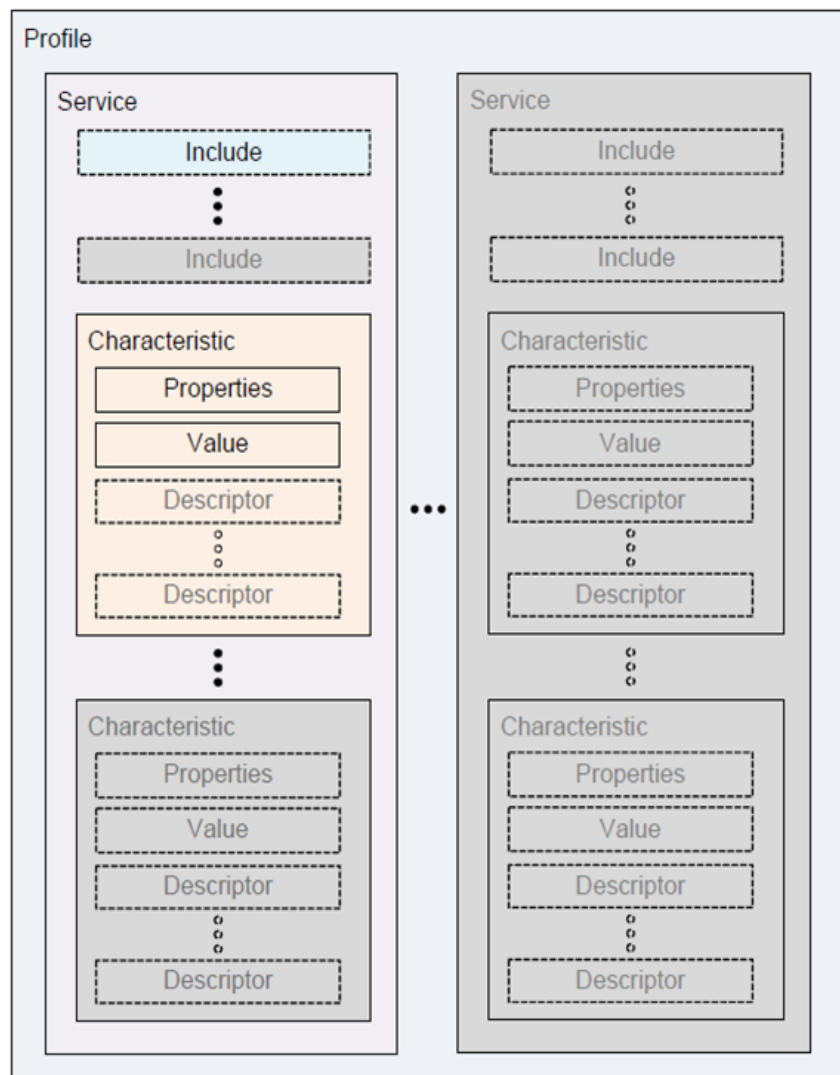
IV – Concepts clés

1 – LE BLUETOOTH LOW ENERGY

L'organisation GATT

Le Bluetooth Low Energy est une technologie sans fil basée sur les spécifications du Bluetooth 4.0. Il est particulièrement adapté aux objets connectés portables qui l'utilisent pour synchroniser les données avec leurs applications respectives.

Le BLE utilise des profils Bluetooth spécifiques appelés Generic Attribute ou GATT. Ils se divisent en trois niveaux : Profile, Service et Characteristic. Le schéma suivant illustre cette organisation :

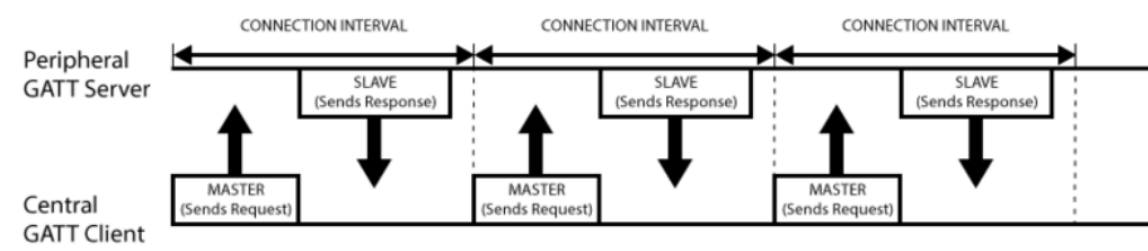


HIERARCHIE DES PROFILES GATT

Un Profile est un ensemble de Services prédéfinis ajouté par le fabricant en fonction de son besoin. Par exemple, on peut trouver dans de nombreux objets le Health Care Profile (un ensemble de services orienté vers le domaine de la santé), le Sport and Fitness Profile (un ensemble de service adapté aux objets dédiés au sport) ou le Alert and Time Profile (services permettant à un objet de recevoir des notifications ou d'autoriser la modification de ses configurations de temps par un serveur distant).

Un Service contient un ensemble de Characteristics qui, permettent d'effectuer certaines actions. Il est possible de fabriquer son propre service en se basant sur ceux déjà existants.

Les Characteristics sont les éléments les plus bas dans l'organisation du GATT. Ils contiennent des valeurs qui peuvent prendre de nombreuses formes : array, string, integer etc... chacune de ces valeurs peut être en lecture seule ou en écriture. En plus de sa valeur, une Characteristic peut se voir attribuer un certain nombre de descriptors dont le but est de permettre la configuration de certaines fonctionnalités comme, par exemple, des notifications.



2 – INTRODUCTION A ANDROID

Android studio

Android Studio est un environnement de développement (IDE) pour développer des applications Android. Il est basé sur IntelliJ IDEA. Android Studio permet principalement d'éditer les fichiers Java et les fichiers de configuration XML d'une application Android.

Android SDK

Le kit de développement (SDK) d'Android est un ensemble complet d'outils de développement. Il inclut un débogueur, des bibliothèques logicielles, un émulateur basé sur QEMU, de la documentation, des exemples de code et des tutoriaux. Les plateformes de développement prises en charge par ce kit sont les distributions sous Noyau Linux, Mac OS X 10.5.8 ou plus, Windows XP ou version ultérieure.

Java

Java est un langage de programmation orienté objet. Les applications Android sont presque essentiellement codées en Java.

XML

XML est le sigle pour Extensible Markup Language. Un document XML est constitué par des éléments ayant chacun une balise de début, un contenu et une balise de fin. Dans le développement Android, le fichier XML affiche l'UI.

3 - COMPOSANTES D'UNE APPLICATION ANDROID

Android Manifest

Ce fichier XML est obligatoire dans tout projet Android, et doit toujours avoir ce même nom. Ce fichier permet au système de reconnaître l'application. Il permet de spécifier différentes options pour des projets, comme le matériel nécessaire pour les faire fonctionner, certains paramètres de sécurité ou encore des informations plus ou moins triviales telles que le nom de l'application ainsi que son icône. Mais ce n'est pas tout, c'est aussi la première étape à maîtriser afin de pouvoir insérer plusieurs activités au sein d'une même application.

Package

Un fichier Android Package (ou APK, pour Android Package Kit) est un format de fichiers pour Android. Un APK (ex. : "fichier.apk") est une collection de fichiers ("package") compressée pour le système d'exploitation Android. L'ensemble constitue un « paquet ».

Ressources

Les ressources sont des fichiers additionnels exploités par le code Java. Il existe plusieurs catégories de ressources, celles utilisées dans le projet sont les suivantes :

- Drawable : contient des fichiers bitmap (.png, .jpg et .gif) ou xml. Ce sont généralement des éléments graphiques.
- Mipmap : contient les icônes de l'application, sous différent format pour rester compatible dans tous les environnements.
- Layout : contient des fichiers xml qui définissent l'UI.
- Menu : contient des fichiers xml qui définissent les menus et sous-menus de l'application.
- Values : des fichiers xml contenant de simples valeurs (Strings, int, des couleurs, des dimensions...).
- Xml : ce sont les fichiers qui vont sauvegarder les paramètres de l'utilisateur (langue, nom, prénom...)

Activité

La classe Activité est un élément crucial d'une application Android. Elle représente l'implémentation et les interactions des interfaces.

Fragment

Les fragments permettent de scinder les activités en composants encapsulés et réutilisables qui possèdent leur propre cycle de vie et leur propre interface graphique. Cela permet de mettre en place des I.H.M. évoluées.

Service

Un service, à la différence d'une activité, ne possède pas de vue mais permet l'exécution d'un algorithme sur un temps indéfini. Il ne s'arrêtera que lorsque la tâche est finie ou que son exécution est arrêtée.

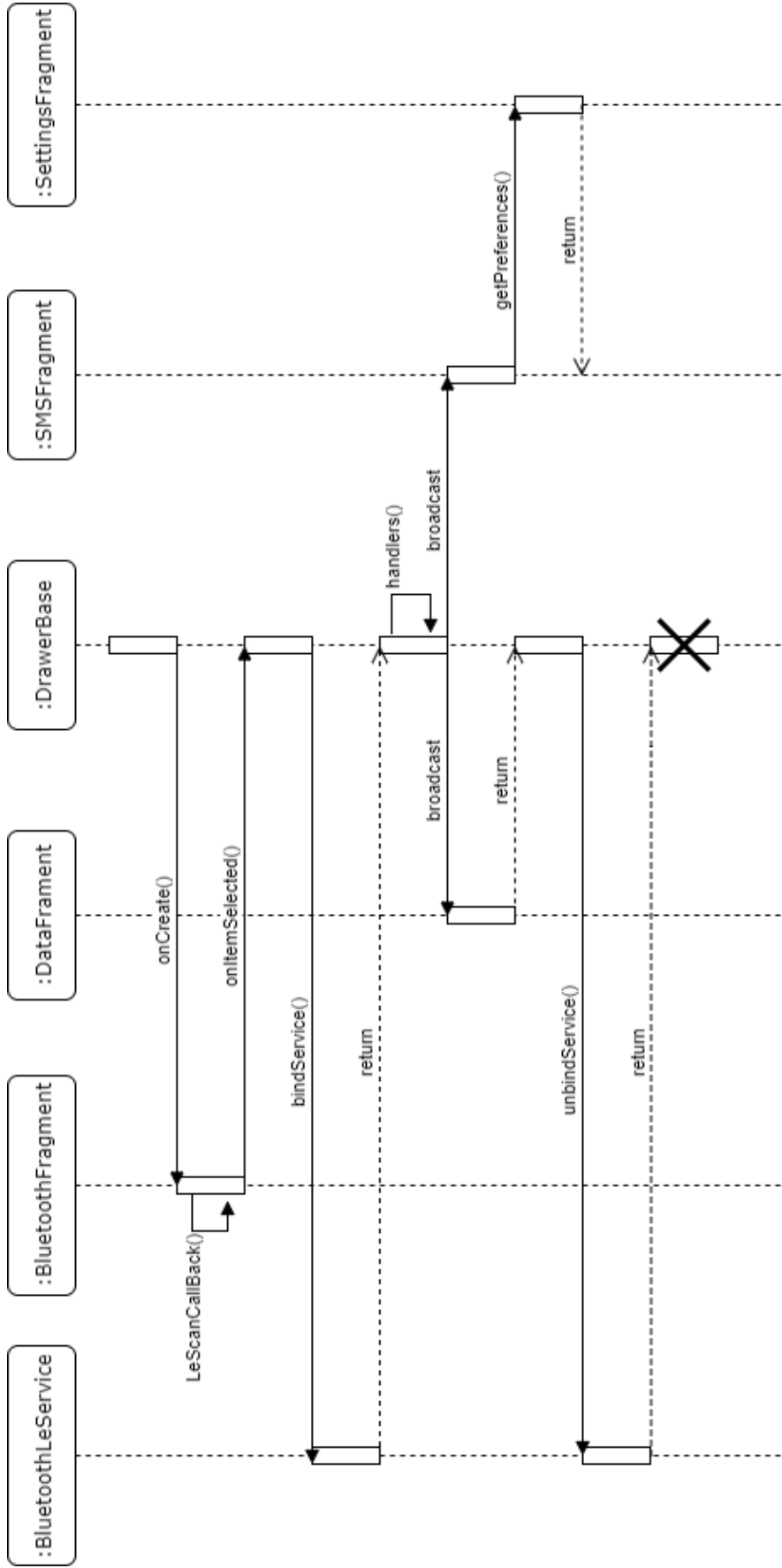


DIAGRAMME DE SEQUENCES

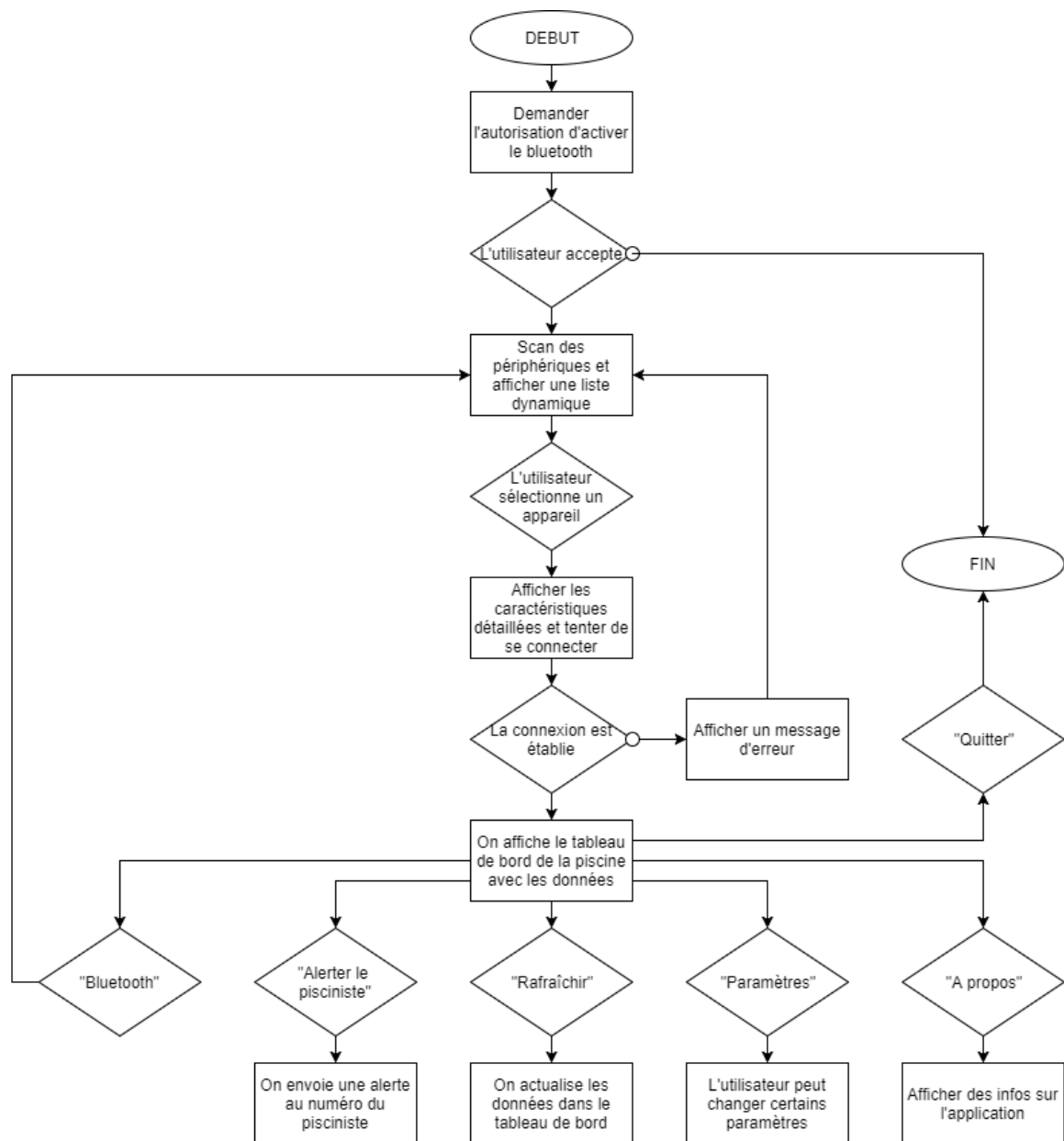


DIAGRAMME D'ACTIVITE

4 - DIAGRAMME D'ETAT ANDROID

Activité

Une activité possède quatre états que sont :

- Active : l'activité est lancée par l'utilisateur, elle s'exécute au premier plan ;
- En Pause : l'activité est lancée par l'utilisateur, elle s'exécute et est visible, mais elle n'est plus au premier plan. Une notification ou une autre activité lui a volé le focus et une partie du premier plan ;
- Stoppée : l'activité a été lancée par l'utilisateur, mais n'est plus au premier plan et est invisible. L'activité ne peut interagir avec l'utilisateur qu'avec une notification ;
- Morte : l'activité n'est pas lancée.

Des méthodes servent à gérer le comportement de l'activité pour chaque état :

- Dans onCreate(), on instancie les objets
- Dans onStart(), on lance les traitements
- Dans onResume(), on s'abonne et on remet le contexte utilisateur
- Dans onPause(), on se désabonne et on enregistre le contexte utilisateur
- Dans onStop(), on arrête les traitements et on désalloue les objets

Fragment

Comme pour les activités, en fonction du cycle de vie du fragment, il faut sauvegarder, restaurer, instancier, détruire les données lors des différentes étapes de ce cycle.

Les principes de l'activité s'appliquent aux fragments de la même manière. Ainsi quand on est dans un fragment :

- Dans onAttach(), on récupère un pointeur vers l'activité contenante
- Dans onCreate(), on instancie les objets non graphiques
- Dans onCreateView(), on instancie la vue et les composants graphiques
- Dans onStart(), on lance les traitements
- Dans onResume(), on s'abonne et on remet le contexte utilisateur
- Dans onPause(), on se désabonne et on enregistre le contexte utilisateur
- Dans onStop(), on arrête les traitements et on désalloue les objets

VI – Déroulement du projet

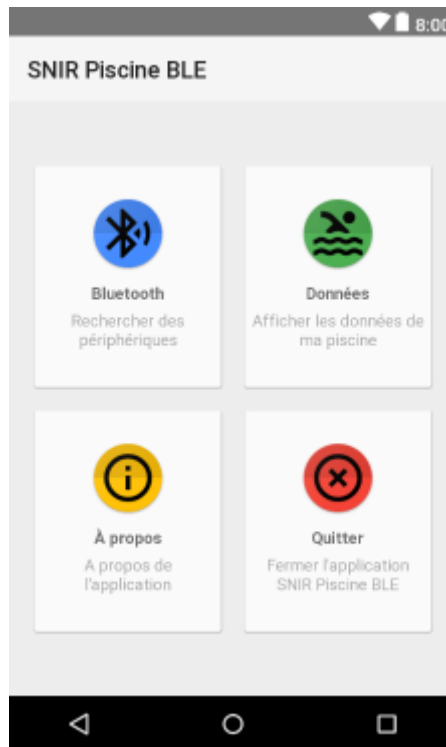
1 – GANTT PREVISIONNEL

Catégorie	Affecté à	Estimé Début	Estimé Fin	Travail estimé (en heures)
Cahier des charges	Mohammed AZZOUZ	03/01/2018	05/01/2018	19
Apprentissage Arduino	Mohammed AZZOUZ	10/01/2018	26/01/2018	57
TP/TEST	Mohammed AZZOUZ	31/01/2018	09/02/2018	38
Organisation Application	Mohammed AZZOUZ	14/02/2018	16/02/2018	19
Arduino Bluetooth	Mohammed AZZOUZ	21/02/2018	02/03/2018	38
Arduino UUID	Mohammed AZZOUZ	07/03/2018	16/03/2018	38
Amélioration Arduino	Mohammed AZZOUZ	21/03/2018	30/03/2018	38
Cahier des charges	Paul MICHELS	03/01/2018	05/01/2018	19
Apprentissage Java/Android	Paul MICHELS	10/01/2018	26/01/2018	57
TP/TEST	Paul MICHELS	31/01/2018	09/02/2018	38
Organisation Application	Paul MICHELS	14/02/2018	16/02/2018	19
Android IHM	Paul MICHELS	21/02/2018	02/03/2018	38
Test Intent/Activité	Paul MICHELS	07/03/2018	16/03/2018	38
Android Bluetooth	Paul MICHELS	21/03/2018	30/03/2018	38
Cahier des charges	Maximilien VERAN	03/01/2018	05/01/2018	19
Apprentissage Java/Android	Maximilien VERAN	10/01/2018	26/01/2018	57
TP/TEST	Maximilien VERAN	31/01/2018	09/02/2018	38
Organisation Application	Maximilien VERAN	14/02/2018	16/02/2018	19
Android IHM	Maximilien VERAN	21/02/2018	02/03/2018	38
Android SMS	Maximilien VERAN	07/03/2018	30/03/2018	76
Android Bluetooth	Professeur Tutorant	01/01/2018	15/06/2018	247

2 – PHASE DE DEVELOPPEMENT

Première IHM

Après avoir assimilé le fonctionnement du code principal, il a fallu créer une IHM. Le premier design de l'application est le suivant :



PREMIERE INTERFACE HOMME-MACHINE

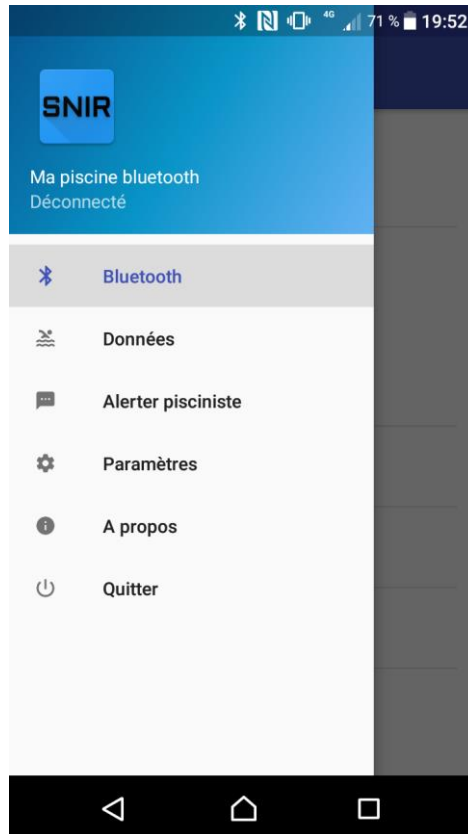
Cette interface contient quatre tuiles :

- La tuile Bluetooth : permet de scanner les appareils BLE et s'y connecter
- La tuile donnée : contient une interface qui affiche les données relatives à l'état de la piscine
- La tuile à propos : contient des informations sur l'application
- La tuile quitter : ferme l'application

Bien que pratique, l'interface a été abandonnée. En effet, l'utilisation d'un menu est plus adaptée à notre application et plus dynamique. De plus, une simple connexion Bluetooth et un tableau de bord ne justifie pas l'utilisation de tuiles, plus intéressantes pour une application possédant de nombreux cas d'utilisation.

Deuxième IHM avec activités

Cette IHM possède un menu déroulant contenant les différentes sections de l'application. Plus discret que notre premier jet, le menu permet à l'utilisateur de naviguer plus aisément entre les différentes fonctionnalités. Le menu est un Navigation Drawer :



DEUXIEME INTERFACE HOMME-MACHINE

L'UI sera celle retenue pour l'application, cependant la programmation a été modifiée. Ici, on a créé une activité définissant le menu, et les autres activités en étaient des dérivées. Ainsi, chaque activité créée construisait le menu en même temps. Cette solution n'était pas fluide, il fallait garder le même menu et changer uniquement le contenu de l'UI.

IHM définitive

Pour palier au problème précédemment évoqué, le menu est devenu la seule et unique activité de notre application, et on utilise des fragments au lieu des activités. Cette pratique est conseillée par Android dans le cas d'un Navigation Drawer. On obtient un résultat plus fluide. Il a fallu cependant adapter notre code à cette pratique : la programmation dans un fragment n'est pas exactement la même que dans une activité.

Mise à jour Android

Après une mise à jour Android obligatoire l'application n'était plus capable d'exécuter la méthode permettant de scanner les appareils à proximité. Il a fallu recoder la méthode en question, en appliquant des méthodes alternatives à nos objets.

Incompatibilité ListView / Fragments

Un fragment ne permet pas l'utilisation des ListView dans l'UI. Nous avons donc remplacé les ListView par des RecyclerView (design introduit par la version Android Lollipop, succédant justement aux ListView).

Cette permutation a non seulement permis d'utiliser des listes dynamiques dans nos fragments, et a en plus élargi la compatibilité de l'application sur une plus large gamme de versions Android. En effet, les ListView ne s'affichent pas sur les nouvelles versions (à partir de la 7.0).

Utilisation de handlers

L'application fournie comme référence permettait d'afficher une seule donnée à la fois, et seulement à la suite d'une interaction de l'utilisateur. Des threads ont donc été créés pour effectuer périodiquement des demandes de données. Chaque thread effectue une demande sur une seule donnée (parmi le pH, la température, le taux de chlore et le bilan) et un dernier thread envoie périodiquement l'ensemble des valeurs par broadcast.

4 – CALCULS

La formule de Nernst permet d'établir le taux de chlore de notre piscine à partir du pH et du potentiel d'oxydo-réduction :

$$C = 10^{-\frac{(715+50(7-pH))+E}{(300+50(7-pH))}}$$

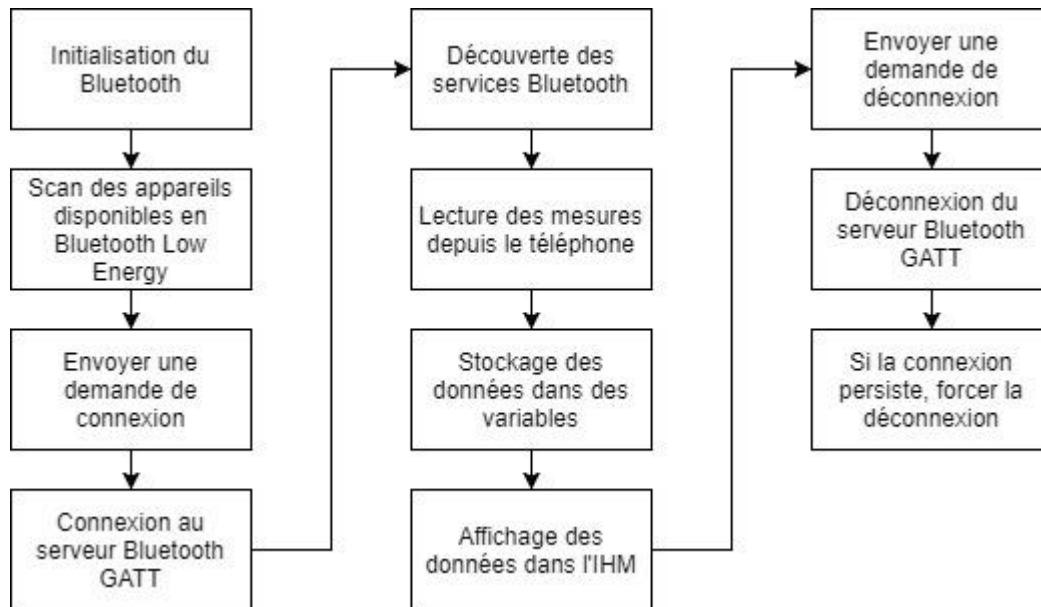
Avec :

- C le taux de chlore en ppm
- E le potentiel d'oxydo-réduction en mV

VII – Fonctionnement de l'application

1 – ITERATION PRINCIPALE

Même si de nombreuses options ont été intégrées au cours du développement, on peut dégager l'itération principale relative au cahier des charges :



Les méthodes utilisées dans cette itération sont détaillées en annexe.

2 – L'ACTIVITE

L'application que nous avons développée ne possède qu'une seule et unique activité. Dans le contexte IHM, son rôle est d'afficher un menu lorsque l'icône en haut à gauche est sélectionnée, et de changer le contenu (les fragments) de l'interface lorsqu'une section est choisie par l'utilisateur (par défaut la section Bluetooth).

Outre la gestion des fragments, l'activité a un rôle essentiel dans la connexion Bluetooth avec le flotteur : c'est ici que l'application se connecte à un serveur GATT, qu'elle reçoit les différents services mais aussi les données envoyées par le serveur (dans notre cas le flotteur) grâce au service BluetoothLEService que nous détaillerons dans la section suivante.

DrawerBase extends AppCompatActivity
<ul style="list-style-type: none"> - mBluetoothAdapter : BluetoothAdapter - mConnectedBluetoothDevice : BluetoothDevice - mBluetoothDevice : BluetoothDevice - mBluetoothLeService BluetoothLeService - mNotifyCharacteristic : BluetoothGattCharacteristic - mServiceConnection : final ServiceConnection - mGattUpdateReceiver : final BroadcastReceiver - mGattCharacteristics : ArrayList<ArrayList<BluetoothGattCharacteristic>> - gattServiceData : ArrayList<HashMap<String, String>> - gattCharacteristicData : ArrayList<ArrayList<HashMap<String, String>>> - gattCharacteristicGroupData : ArrayList<HashMap<String, String>> - charas : ArrayList<BluetoothGattCharacteristic> - handler1 : Handler - handler2 : Handler - handler3 : Handler - handler4 : Handler - runnable1 : Runnable - runnable2 : Runnable - runnable3 : Runnable - runnable4 : Runnable - mDeviceAddress : String - DATA_NOTIFICATION : final String - listName : final String - listUUID : final String - marquer_rang_charac : boolean - pH : double - temperature : double - redox : double - pHCharacteristicProperties : int - temperatureCharacteristicProperties : int - redoxCharacteristicProperties : int - rang_charac : int + REQUEST_ENABLE_BT : final int
<ul style="list-style-type: none"> # onCreate(Bundle savedInstanceState) : void + onResume() : void # onPause() : void # onDestroy() : void + onBackPressed() : void - makeGattUpdateIntentFilter() : final IntentFilter + onNavigationItemSelected(MenuItem item) : boolean + onItemSelected(BluetoothDevice bluetoothDevice) : void - connect(BluetoothDevice bluetoothDevice) : void - displayGattServices(List<BluetoothGattService> gattServices) : void - getPH() : void - getTemperature() : void - getRedox() : void

DIAGRAMME DE CLASSE DE L'ACTIVITE

3 – SERVICE BLUETOOTH LOW ENERGY

Ce service contient une série de méthodes servant à gérer la connexion et la communication des données avec un serveur GATT hébergé par un appareil. Les méthodes permettent de :

- Se connecter
- Se deconnecter
- Lire les caractéristiques
- Obtenir les services
- Envoyer des broadcasts

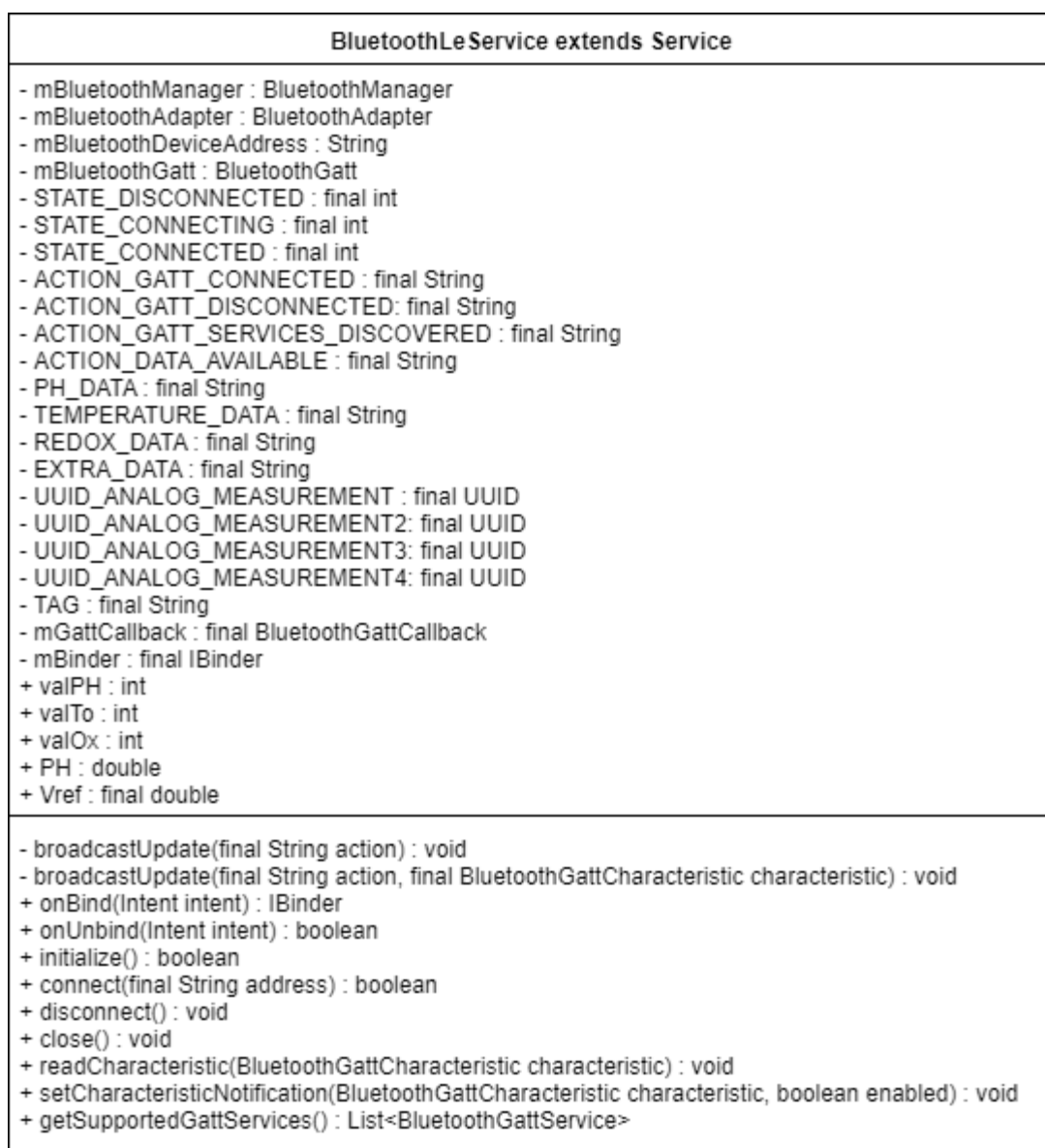


DIAGRAMME DE CLASSE DU SERVICE

4 – LES FRAGMENTS

Fragment Bluetooth

Ce fragment est celui qui sera chargé par défaut à la création de l'activité. Il permet de scanner les serveurs GATT disponibles. Les serveurs sont affichés avec leur nom et leur adresse MAC. On différencie trois sections différentes :

- L'appareil connecté : Si on s'est connecté avec succès à un serveur GATT, la section apparaît dans l'IHM et contient l'appareil.
- Les appareils associés : Ce sont les appareils connus du téléphone. Cette section ne s'affiche que si cette liste existe.
- Les appareils disponibles : Ce sont les appareils qui ont été scannés et qui sont disponibles pour une connexion.

Si l'utilisateur sélectionne un des items dans une liste, le fragment envoie, par l'intermédiaire d'une interface, l'objet BluetoothDevice à l'activité pour qu'elle tente de s'y connecter. L'utilisateur peut balayer l'écran vers le bas pour rafraîchir les listes d'appareils et relancer un scan.

BluetoothFragment extends Fragment
<pre>- mListener : bluetoothFragmentCallBack - mFoundDeviceList : List<BluetoothDevice> - mPairedDeviceList : List<BluetoothDevice> - mConnectedDeviceList : List<BluetoothDevice> - mBluetoothAdapter : BluetoothAdapter - mConnectedDevice : BluetoothDevice - mBluetoothDevice : BluetoothDevice - mScanning : boolean - REQUEST_ENABLE_BT : final int - SCAN_PERIOD : final long - mHandler : Handler - mDeviceAddress : String - mRelativeLayout : RelativeLayout - mPairedDeviceAdapter : DeviceAdapter - mFoundDeviceAdapter : DeviceAdapter - mConnectedDeviceAdapter : DeviceAdapter - mPairedDeviceProgressBar : ProgressBar - mFoundDeviceProgressBar : ProgressBar - mSwipeRefreshLayout : SwipeRefreshLayout - actionBarHeight : int - state : TextView - noDeviceFound : TextView - mLeScanCallback : BluetoothAdapter.LeScanCallback - mGattUpdateReceiver : final BroadcastReceiver + BluetoothFragment() + newInstance(BluetoothDevice bluetoothDevice) : static BluetoothFragment + onCreate(Bundle savedInstanceState) : void + onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) : View + onAttach(Context context) : void + onResume() : void + onDetach() : void + onActivityResult(int requestCode, int resultCode, Intent data) : void + setRecyclerView(RecyclerView recyclerView, DeviceAdapter deviceAdapter, final List<BluetoothDevice> deviceList) : void + setConnectedRecyclerView(RecyclerView recyclerView, DeviceAdapter deviceAdapter, final List<BluetoothDevice> deviceList) : void + getPairedDevice (View view) : void + scanLeDevice(final boolean enable) : void + makeGattUpdateIntentFilter() : static IntentFilter</pre>

DIAGRAMME DE CLASSE DU FRAGMENT

Fragment Data

Ce fragment est celui qui aura pour unique fonction d'afficher les données relatives à l'état de la piscine : le pH, la température, le taux de chlore et le bilan. Ici, aucune interaction avec l'utilisateur n'est intégrée puisque le seul but est d'afficher des informations :

- L'appareil connecté : On reçoit l'objet BluetoothDevice avec le constructeur du fragment depuis l'activité. Si cette donnée est nulle, alors nous ne sommes pas connectés à un flotteur.
- Les mesures du flotteur : Ces données sont récupérées par l'intermédiaire d'un BroadcastReceiver. Les intents sont envoyés depuis l'activité toutes les 5 secondes (par défaut, modifiable dans les options)

DataFragment extends Fragment
<ul style="list-style-type: none">- mBluetoothAdapter : BluetoothAdapter- mDataList : List<Data># mAdapter : Adapter- recyclerView : RecyclerView- mName : String- mDeviceAddress : String- pH : double- temperature : double- redox : double- REQUEST_ENABLE_BT : final int- mBluetoothDevice : BluetoothDevice- mDataUpdateReceiver : final BroadcastReceiver+ DATA_NOTIFICATION : final String
<ul style="list-style-type: none">+ DataFragment()+ newInstance(BluetoothDevice bluetoothDevice) : static DataFragment+ onCreate(Bundle savedInstanceState) : void+ onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) : View+ onAttach(Context context) : void+ onResume() : void+ onDetach() : void- updateRecyclerView() : void

DIAGRAMME DE CLASSE DU FRAGMENT

Fragment SMS

Ce fragment permet d'alerter la personne responsable de la piscine en envoyant un SMS contenant le nom du propriétaire de la piscine, son adresse ainsi que les données récupérées par le flotteur.

Pour que cette fonction puisse être opérationnelle, il est nécessaire d'avoir paramétré le numéro de téléphone ainsi que les coordonnées dans le fragment Paramètres.

SMSFragment extends Fragment
<ul style="list-style-type: none">- pH : double- temperature : double- redox : double- bilan : double- message : EditText- numero : EditText- mPhoneList : List<Phone>- mAdapter : PhoneAdapter- recyclerview : RecyclerView- mDataUpdateReceiver : final BroadcastReceiver+ DATA_NOTIFICATION : final String
<ul style="list-style-type: none">+ onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) : View+ onResume() : void+ onDetach() : void- updateText(EditText editText) : void

DIAGRAMME DE CLASSE DU FRAGMENT

Fragment Paramètres

Ce fragment permet de modifier la fréquence du rafraîchissement des données, le numéro de téléphone du pisciniste ainsi que les coordonnées du propriétaire de la piscine. Ces préférences sont sauvegardées, elles seront donc toujours disponibles même après la fermeture de l'application.

SettingsFragment extends PreferenceFragmentCompat
<ul style="list-style-type: none">- sharedPreferences : SharedPreferences
<ul style="list-style-type: none">+ onCreatePreferences(Bundle bundle, String s) : void+ onResume() : void+ onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) : void+ onPause() : void

DIAGRAMME DE CLASSE DU FRAGMENT

Fragment à propos

Ce fragment ne sert qu'à informer l'utilisateur que cette application est développée dans ce cadre et affiche le nom des élèves ainsi que l'établissement.

AboutFragment extends Fragment
+ onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) : View

DIAGRAMME DE CLASSE DU FRAGMENT

VIII – Cahier des recettes





1 – DEBOGAGE DE L'APPLICATION

Mis à part les problèmes liés aux fonctionnalités de l'application, certaines erreurs de codage conduisent la plupart du temps à un crash de l'application. Il est difficile de savoir exactement à quel endroit, et pourquoi l'application cesse de fonctionner dans le code à partir de l'interface utilisateur de l'appareil. Heureusement, Android Studio propose tous les outils nécessaires pour déboguer notre code, nous nous intéresseront ici au Logcat.

Lorsque l'on lance l'application à partir d'Android Studio, que ce soit sur un émulateur ou sur un appareil réel, le Logcat fournit des informations primordiales lors d'un crash : quelles méthodes ont été appelées, à quelle ligne du code le programme a cessé de fonctionner et quel type d'erreur.

Dans la majorité des crashes, une méthode est appelée sur un objet null, et l'application se ferme. Dans ce cas, le Logcat indiquera par exemple :

```
java.lang.RuntimeException:
Unable to start activity ComponentInfo{com.piscineble.snir.MainActivity}:
java.lang.NullPointerException: Attempt to invoke virtual method 'void
android.widget.Button.setText(java.lang.CharSequence)' on a null object reference
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2325)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2387)
    at android.app.ActivityThread.access$800(ActivityThread.java:151)
    ...
Caused by: java.lang.NullPointerException: Attempt to invoke virtual method 'void
android.widget.Button.setText(java.lang.CharSequence)' on a null object reference
    at com.piscineble.snir.MainActivity.onCreate(MainActivity.java:16)
    at android.app.Activity.performCreate(Activity.java:5990)
    ...
```

-  Le nom de l'erreur
-  La raison de l'erreur
-  La méthode responsable de l'erreur
-  L'endroit de l'erreur

EXTRAIT D'UN LOGCAT

Pendant la phase de développement, les erreurs rencontrées ont été multiples et variées, et cet outil a permis de réparer un code avec des dysfonctionnements de manière efficace, mais aussi d'apprendre à prévoir où un potentiel bug pourrait apparaître, et ainsi, le prévenir.

2 – CAMPAGNE

Objectif de la campagne

On cherche à savoir si les cas d'utilisations fonctionnent tous, pour cela on effectue un certain nombre de tests en suivant des scénarii élaborés à l'avance.

Liste des scénarii

Numéro scénario	Description courte
S ₁	L'utilisateur tente de se connecter au flotteur, est-ce que la connexion s'établit ?
S ₂	L'utilisateur est dans le tableau de bord, est-ce que les données s'affichent ?
S ₃	L'utilisateur décide d'alerter le pisciniste par SMS, est-ce que le message est envoyé et reçu ?

Exécution et bilan

Scénario 1	
Objectif	Vérifier que la communication Bluetooth entre le serveur et le client s'établit correctement.
Prérequis	<ul style="list-style-type: none">- La fonctionnalité Bluetooth est activée sur l'appareil- La carte Genuino 101 est sous tension- Le flotteur n'est pas déjà connecté à un client
Type de test	Fonctionnel
Criticité	La plus haute
Résultat attendu	Le flotteur est détecté par le téléphone, et ce dernier est capable de s'y connecter.
Résultat réel	Résultat attendu

Scénario 2	
Objectif	Vérifier que les données sont collectées et affichées.
Prérequis	<ul style="list-style-type: none"> - La fonctionnalité Bluetooth est activée sur l'appareil - L'appareil est déjà connecté au flotteur
Type de test	Fonctionnel
Criticité	La plus haute
Résultat attendu	Les données s'affichent correctement, dans la bonne section du tableau de bord et ont la bonne valeur.
Résultat réel	Résultat attendu

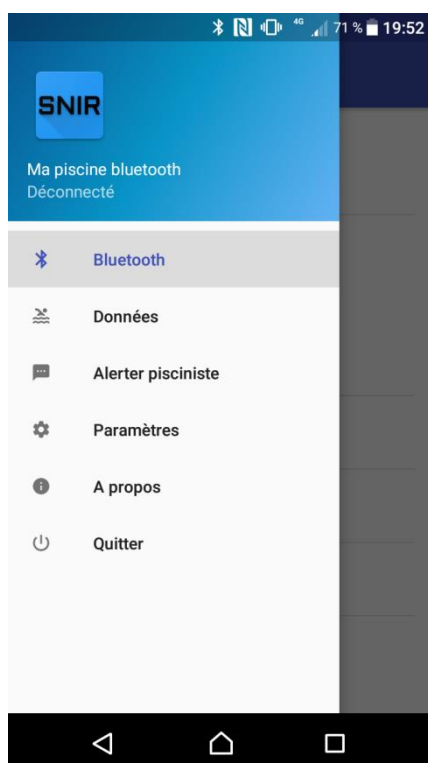
Scénario 3	
Objectif	Vérifier que l'application est capable d'envoyer un SMS.
Prérequis	<ul style="list-style-type: none"> - L'application a la permission d'envoyer des messages - Le numéro de téléphone de destination est paramétré
Type de test	Fonctionnel
Criticité	Moyenne
Résultat attendu	Le message s'envoie correctement, sans message d'erreur, et le téléphone de destination reçoit le même message.
Résultat réel	Résultat attendu

IX – Conclusion

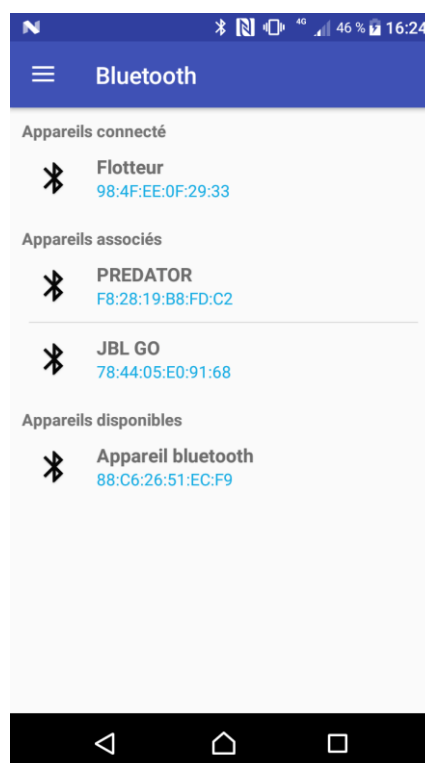
Ce projet a été une excellente expérience où j'ai pu apprendre un nouveau langage et améliorer ma compréhension de la programmation orientée objet, mieux maîtriser le Unified Modeling Language (UML) ainsi que manier et exploiter Android Studio et ses ressources. De plus, j'ai pu développer des capacités de rigueur et d'efficacité dans les processus d'élaboration, de finalisation et de validation d'une production informatique qui me seront précieuses pour la poursuite de mes études, et pour ma carrière professionnelle.

Dans l'ensemble, c'est un projet qui s'est très bien déroulé où beaucoup de travail volontaire en dehors des heures encadrées a été fourni pour que l'application soit opérationnelle et agréable d'utilisation avant la date butoir.

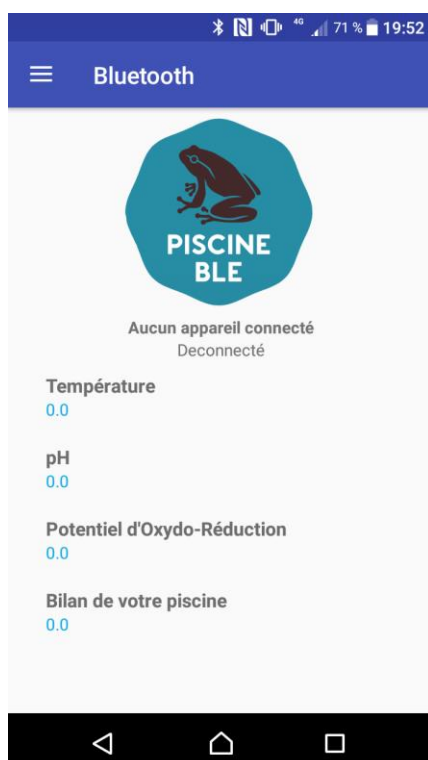
Annexe 1 – Interface Homme-Machine



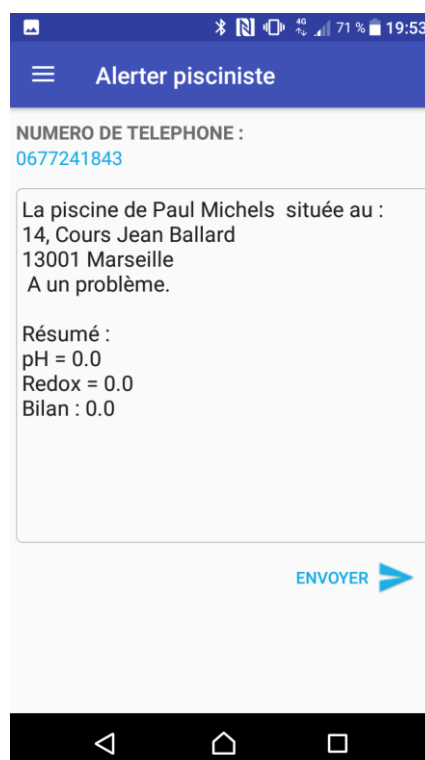
NAVIGATION DRAWER



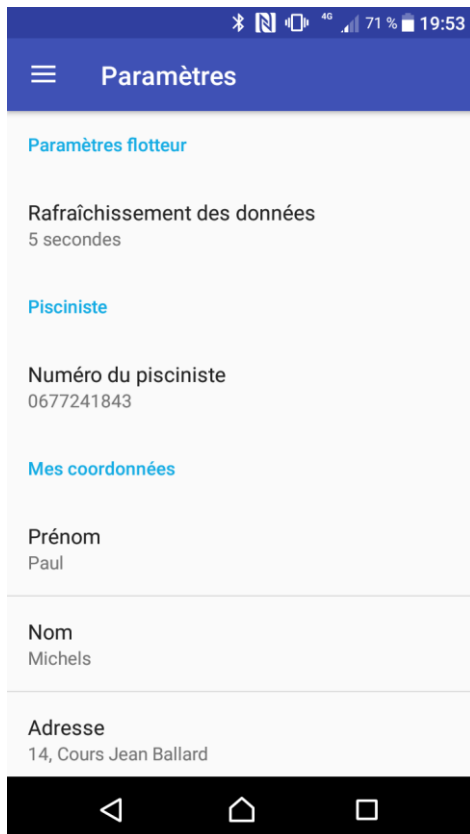
FRAGMENT BLUETOOTH



FRAGMENT DONNEES



FRAGMENT SMS



FRAGMENT PARAMETRES



FRAGMENT A PROPOS

Annexe 2 – Méthodes relatives au fonctionnement du Bluetooth

1 - BLUETOOTHFRAGMENT.JAVA

```
//Trouve les serveurs Bluetooth connus de l'appareil
private void getPairedDevice () {
    Set<BluetoothDevice>pairedDevices=mBluetoothAdapter.getBondedDevices();
    if (pairedDevices.size() > 0) {
        mPairedDeviceList.addAll(pairedDevices);
        mPairedDeviceAdapter.notifyItemInserted(mPairedDeviceList.size());
    }
}

//Scan des serveurs GATT à proximité pendant 6 secondes
private void scanLeDevice(final boolean enable) {
    if (enable) {
        mHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                mBluetoothAdapter.stopLeScan(mLeScanCallback);
            }
        }, SCAN_PERIOD);
        mBluetoothAdapter.startLeScan(mLeScanCallback);
    } else {
        mBluetoothAdapter.stopLeScan(mLeScanCallback);
    }
}

//Rempli la liste d'objet BluetoothDevice au résultat de ScanLeDevice()
private BluetoothAdapter.LeScanCallback mLeScanCallback = new
BluetoothAdapter.LeScanCallback() {

    @Override
    public void onLeScan(final BluetoothDevice device, int rssi, byte[]
scanRecord) {
        if (getActivity() != null) {
            getActivity().runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    if (!mFoundDeviceList.contains(device)) {
                        mFoundDeviceList.add(device);
                        mFoundDeviceAdapter.notifyDataSetChanged();
                    }
                }
            });
        }
    }
};
```

2 – DANS DRAWERBASE.JAVA

```
// On vérifie que le BLE est pris en charge sur l'appareil
if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {
    finish();
}

// Initialisation d'un BluetoothAdapter
final BluetoothManager bluetoothManager =
    (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();

// On vérifie que le bluetooth est pris en charge sur l'appareil
if (mBluetoothAdapter == null) {
    finish();
}

//Etablir une connexion/déconnexion avec le serveur GATT
private final ServiceConnection mServiceConnection = new
ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName componentName, IBinder
service) {
        mBluetoothLeService = ((BluetoothLeService.LocalBinder)
service).getService();
        if (!mBluetoothLeService.initialize()) {
            Log.i(TAG, "Unable to initialize Bluetooth");
        }
        mBluetoothLeService.connect(mDeviceAddress);
    }
    @Override
    public void onServiceDisconnected(ComponentName componentName) {
        mBluetoothLeService = null;
    }
};

// Lorsque l'activité reçoit un BluetoothDevice du BluetoothFragment
private void connect(BluetoothDevice bluetoothDevice) {
    mBluetoothDevice=bluetoothDevice;
    mDeviceAddress = bluetoothDevice.getAddress();
    Intent gattServiceIntent = new Intent(this, BluetoothLeService.class);
    bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);
}
```

```

//Permet d'effectuer des actions en fonctions de l'intent reçu
private final BroadcastReceiver mGattUpdateReceiver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();

        if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {

        } else if
(BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
            mConnectedBluetoothDevice = null;
            if(handler1!=null) {
                handler1.removeCallbacks(runnable1);
            }
            if(handler2!=null) {
                handler2.removeCallbacks(runnable2);
            }
            if(handler3!=null) {
                handler3.removeCallbacks(runnable3);
            }

        } else if
(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
displayGattServices(mBluetoothLeService.getSupportedGattServices());
            handler1 = new Handler();
            handler1.post(runnable1);
            handler2 = new Handler();
            handler2.post(runnable2);
            handler3 = new Handler();
            handler3.post(runnable3);
            handler4 = new Handler();
            handler4.post(runnable4);

        } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action))
        {

            if(intent.getStringExtra(BluetoothLeService.EXTRA_DATA).equals("PH")){

                pH=Double.parseDouble(intent.getStringExtra(BluetoothLeService.PH_DATA));
            } else if
(intent.getStringExtra(BluetoothLeService.EXTRA_DATA).equals("TEMPERATURE"))
            {

                temperature=Double.parseDouble(intent.getStringExtra(BluetoothLeService.TEMPERATURE_DATA));
            } else if
(intent.getStringExtra(BluetoothLeService.EXTRA_DATA).equals("REDOX")){

                redox=Double.parseDouble(intent.getStringExtra(BluetoothLeService.REDOX_DATA));
            }

        }

    }
};

```

```

//Permet d'acqu rir les services GATT disponibles
private void displayGattServices(List<BluetoothGattService> gattServices) {
    if (gattServices == null) return;
    String uuid;
    String unknownServiceString =
        getResources().getString(R.string.unknown_service);
    String unknownCharaString =
        getResources().getString(R.string.unknown_characteristic);

    for (BluetoothGattService gattService : gattServices) {
        HashMap<String, String> currentServiceData = new HashMap<>();
        uuid = gattService.getUuid().toString();

        if
            ((uuid.equals(SampleGattAttributes.PISCINE_SERVICE)) && (!marquer_rang_charac
            )){
                marquer_rang_charac = true;
            }
            currentServiceData.put(
                listName, SampleGattAttributes.lookup(uuid,
                unknownServiceString));
            currentServiceData.put(listUUID, uuid);
            gattServiceData.add(currentServiceData);

            List<BluetoothGattCharacteristic> gattCharacteristics =
                gattService.getCharacteristics();

            for (BluetoothGattCharacteristic gattCharacteristic :
            gattCharacteristics) {
                charas.add(gattCharacteristic);
                if (marquer_rang_charac==true) rang_charac =
                charas.indexOf(gattCharacteristic);
                HashMap<String, String> currentCharaData = new HashMap<>();
                uuid = gattCharacteristic.getUuid().toString();
                currentCharaData.put(
                    listName, SampleGattAttributes.lookup(uuid,
                    unknownCharaString));
                currentCharaData.put(listUUID, uuid);
                gattCharacteristicGroupData.add(currentCharaData);
            }

            mGattCharacteristics.add(charas);
            gattCharacteristicData.add(gattCharacteristicGroupData);
        }
    }
}

//Un des threads cr   pour acqu rir une donn  e (ici le pH)
private Runnable runnable1 = new Runnable() {
    @Override
    public void run() {
        getPH();
        handler1.postDelayed(runnable1, 800);
    }
};

```

```

//Effectue une demande de pH au BluetoothLeService
private void getPH() {
    final BluetoothGattCharacteristic pHCharacteristic =
        charas.get(rang_charac-3);
    pHCharacteristicProperties = pHCharacteristic.getProperties();
    if ((pHCharacteristicProperties |
BluetoothGattCharacteristic.PROPERTY_READ) > 0) {

        if (mNotifyCharacteristic != null) {

mBluetoothLeService.setCharacteristicNotification(mNotifyCharacteristic,
false);
            mNotifyCharacteristic = null;
        }
        mBluetoothLeService.readCharacteristic(pHCharacteristic);
    }
    if ((pHCharacteristicProperties |
BluetoothGattCharacteristic.PROPERTY_NOTIFY) > 0) {
        mNotifyCharacteristic = pHCharacteristic;
        mBluetoothLeService.setCharacteristicNotification(pHCharacteristic,
true);
    }
}

//Filtre pour les intents du broadcast
private static IntentFilter makeGattUpdateIntentFilter() {
    final IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED);
    intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED);
    intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED);
    intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE);
    return intentFilter;
}

```

3 – DANS DATAFRAGMENT.JAVA

```

// Permet de recevoir les intents contenant les données (pH, température...)
private final BroadcastReceiver mDataUpdateReceiver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        pH=Double.parseDouble(intent.getStringExtra("ph"));
        temperature=Double.parseDouble(intent.getStringExtra("temperature"));
        redox=Double.parseDouble(intent.getStringExtra("redox"));
        updateRecyclerView();
    }
};

```