BTS SNIR
Session 2020
Lycée Polyvalent Touchard-Washington

PROJET : CONTRÔLEUR DE RONDE

Dossier technique du projet
Partie Individuelle

Mathis HUREAU

Table des matières

1 - CONCEPTION DES APPLICATIONS SUR ORDINATEUR ET SMARTPHONE....2
1.1 - Synoptique des applications....2
1.2 - Liste des tâches sur le smartphone et l'ordinateur....2
2 - PRÉREQUIS DE DÉVELOPPEMENT....3
2.1 - Environnement de développement....3
2.2 - Installation de l'outil « adb »....3
2.2.1 - Installation sous Linux....3
2.2.2 - Installation sous Windows....4
2.3 - Activation du mode Débogage (Android)....5
3 - DÉVELOPPEMENT DE L'IDENTIFICATION D'UN AGENT DE SÉCURITÉ....7
3.1 - Prérequis....7
3.2 - Problème de développement....7
4 MISE EN PLACE DES BASES DE DONNÉES MYSQL ET SQLITE....8
4.1 - Synoptique des deux bases de données....8
4.1.1 - *Structure de la base de données de l'ordinateur*....8
4.1.2 - *Structure de la base de données du smartphone*....9
5 - DÉVELOPPEMENT DE LA SYNCHRONISATION DU SMARTPHONE D'UN AGENT....10
5.1 - Prérequis....10
5.2 - Structure de la partie Synchronisation....11
5.3 - Listage des appareils connectés....12
5.4 - Développement de l'envoi vers le smartphone....13
5.4.1 - Principe du transfert entre les deux bases de données....14
5.5 - Développement de la réception sur l'ordinateur....15

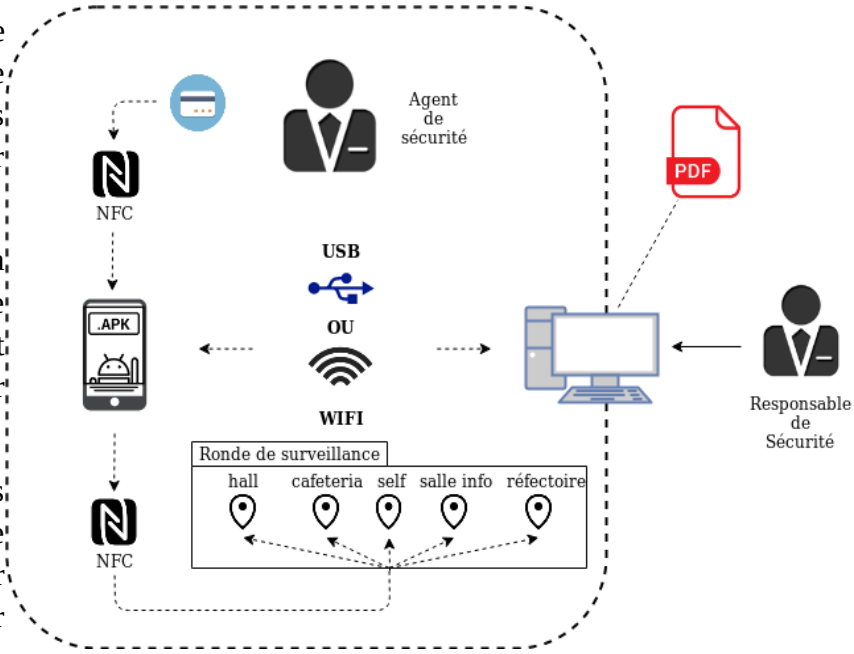
1 - CONCEPTION DES APPLICATIONS SUR ORDINATEUR ET SMARTPHONE

1.1 - Synoptique des applications

Pour que l'agent de sécurité reçoive les rondes dont il doit s'occuper, il faut que le responsable de sécurité ait transféré les rondes sur le smartphone, après les avoir préparées.

L'agent de sécurité effectue ses rondes à l'aide de la base de données récupérée de l'ordinateur, il constate les anomalies et retourne au poste de supervision pour terminer sa ronde.

Le rapport de ronde est édité à l'aide des informations obtenues lors de la ronde de l'agent, une fois que la base de données sur le smartphone a été récupérée sur l'ordinateur.



1.2 - Liste des tâches sur le smartphone et l'ordinateur

Pour mener le projet à son terme, je suis chargé de réaliser les parties suivantes sur l'application du Smartphone et de l'ordinateur :

Application Android sur le smartphone :

- Identifier un agent de sécurité
- Téléchargez des rondes de surveillance
- Envoyer une ronde de surveillance

Application de supervision sur l'ordinateur :

- Synchroniser le smartphone d'un agent

Les tâches « Envoyer une ronde de surveillance » et « Téléchargez des rondes de surveillance » composent toutes les deux la partie « Synchroniser le smartphone d'un agent ». C'est pourquoi je ne compte pas ces dernières comme des tâches à part entière car l'outil adb que j'utilise pour la synchronisation permet la réalisation de ces deux parties.

2 - PRÉREQUIS DE DÉVELOPPEMENT

2.1 - Environnement de développement

Les outils nécessaires au fonctionnement des deux programmes sont présents dans le kit de développement SDK, nécessaire à leur fonctionnement. J'ai tout d'abord besoin de communiquer avec le smartphone à partir de l'ordinateur. Pour ceci, j'utilise l'outil adb qui permet d'interagir par le biais de lignes de commandes avec un smartphone connecté en usb, que l'on peut installer et utiliser facilement sur Linux et Windows. De plus, il faut installer les drivers usb nécessaires pour la pleine utilisation de cet outil.

Pour une partie de la synchronisation, l'application que nous réutilisons des années précédentes utilisait un smartphone rooté. C'est à dire qu'après une manipulation peu recommandé sur les smartphones, car elle fait notamment perdre la garantie et fais dysfonctionner certaines applications, il est possible d'obtenir des droits d'écriture, de lecture et de manipulation supplémentaire. Partie détaillée dans les problèmes rencontrés.

2.2 - Installation de l'outil « adb »

2.2.1 - Installation sous Linux

Sous Linux, il faut avoir l'accès root et d'exécuter la commande :

apt-get install Android-tools-adb Android-tools-adb

Pour vérifier le bon fonctionnement de l'outil on tape la commande :

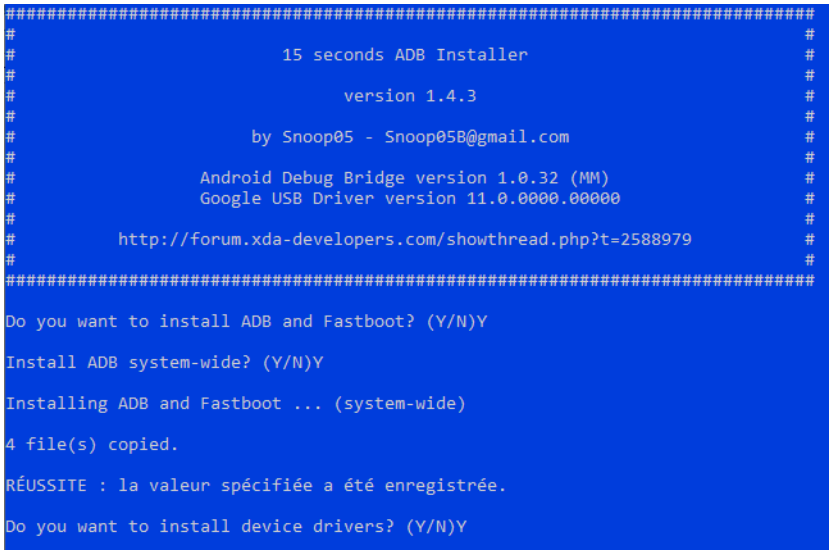
adb

On obtient alors la version installé de l'outil ainsi que quelques aides comme le montre la capture suivante :

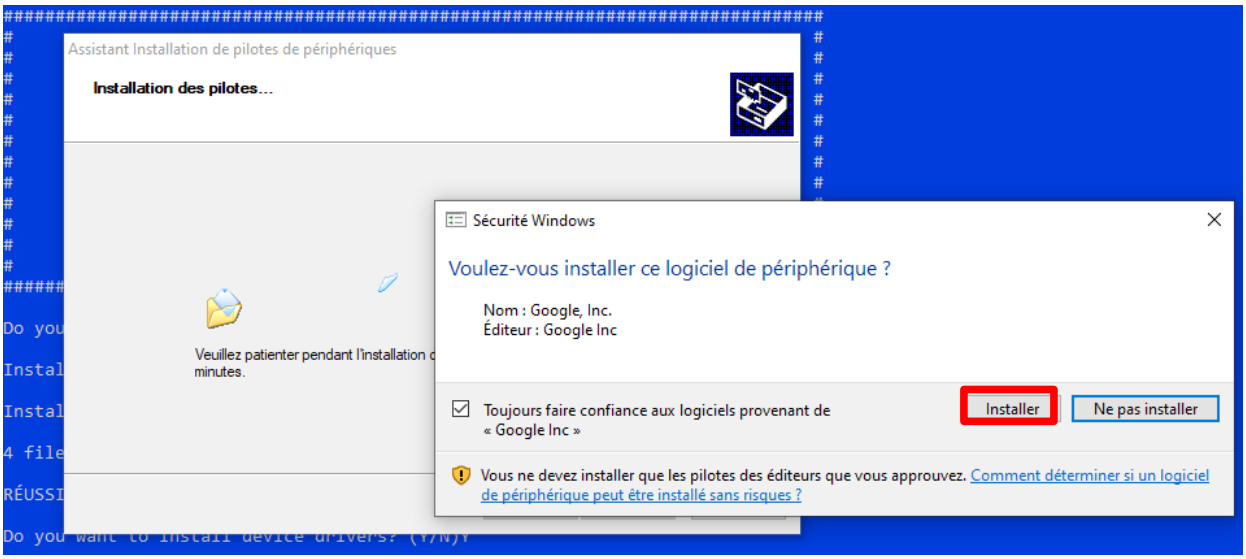
2.2.2 - Installation sous Windows

On trouve l'installateur de l'outil adb à l'adresse androidfilehost.com , il faut lancer l'exécutable *setup-1.4.3.exe* et suivre les instructions suivantes :

- Taper sur la touche « Y » puis confirmer l'installation de adb et fastboot en appuyant sur la touche « Entrée ». Répétez deux fois l'opération pour adb system-wide et les drivers usb comme sur la capture suivante.



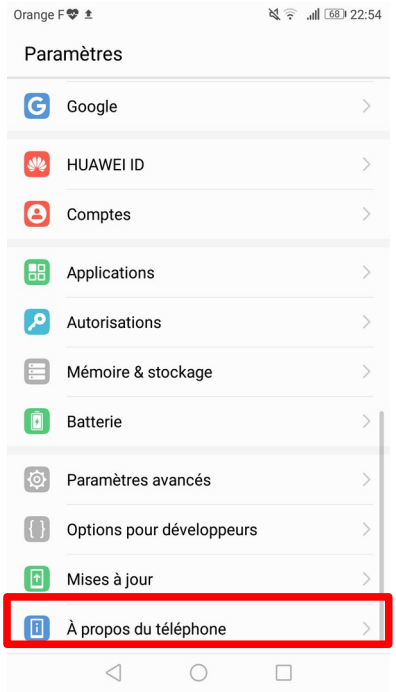
- Le script va détecter puis lancer le gestionnaire d'installation qui correspond au type de Windows. Cliquez sur « Suivant » pour commencer l'installation des drivers USB Google, puis confirmer l'installation en appuyant sur « Terminer ».

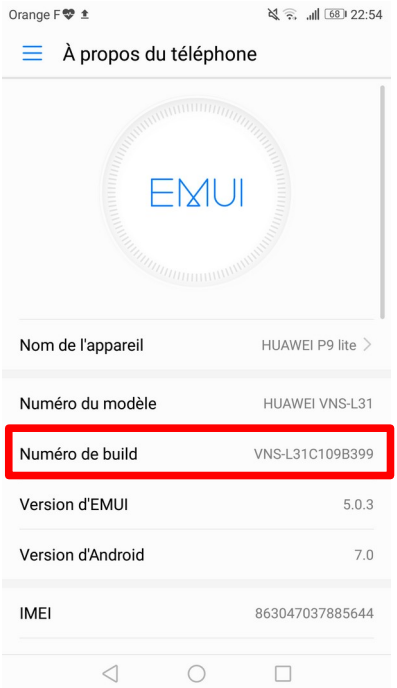


- Un dossier « adb » est apparu à la racine du disque « C:\ », il faut donc lancer l'exécutable du même nom pour démarrer le service. Pour vérifier le bon fonctionnement on peut utiliser la commande « adb » comme Linux.

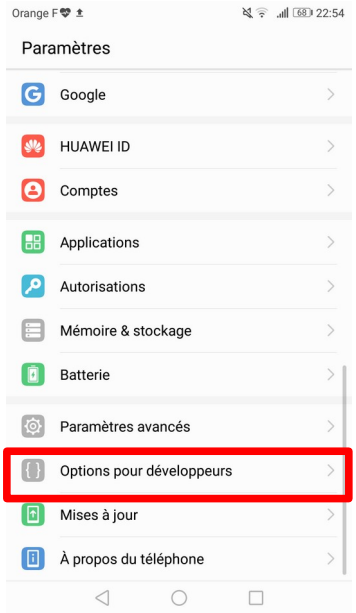
2.3 - Activation du mode Débogage (Android)

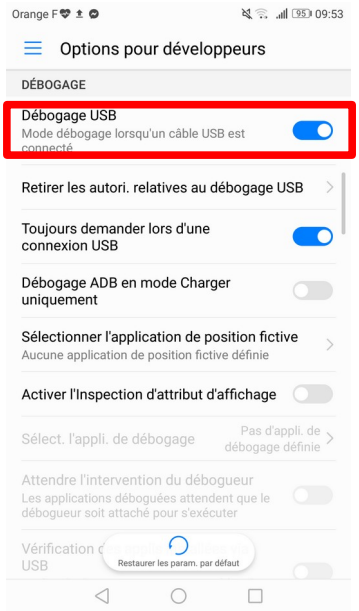
Afin de développer sur nos smartphone, il a fallu activer le mode débogage, premièrement pour installer les applications développé sur l'ordinateur avec Qt, puis pour communiquer entre le Smartphone et le PC. Pour activer le mode de débogage, il faut se rendre dans les réglages du smartphone, tout en bas dans l'onglet «À propos du téléphone» et taper 7 fois sur le «Numéro de build» comme ci-dessous pour débloquer le mode développeur :

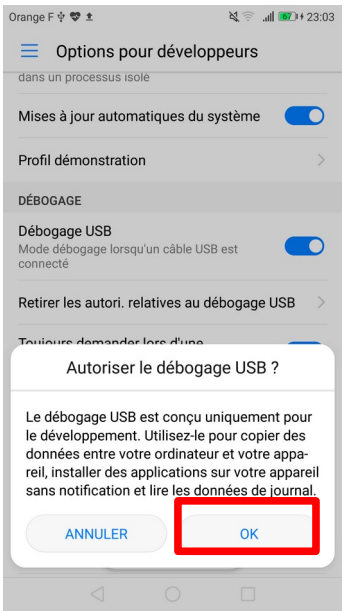




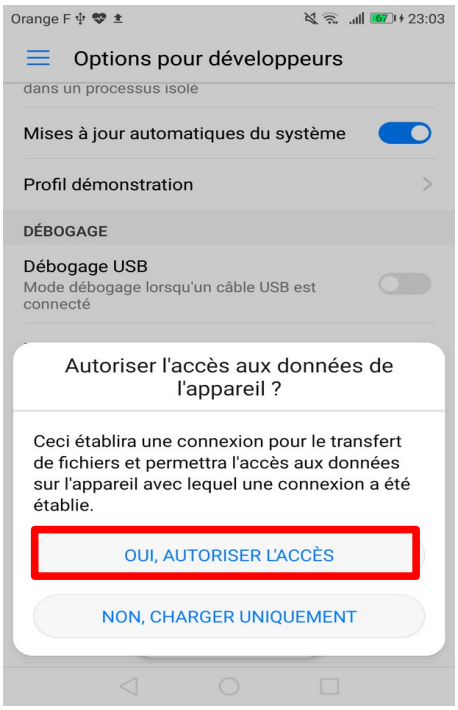
Il faut ensuite se rendre dans l'onglet «Options pour les développeurs» de l'application paramètre et descendre jusqu'à trouver l'option «Débogage USB» puis l'activer. Un message de confirmation va apparaître, il faut le confirmer.

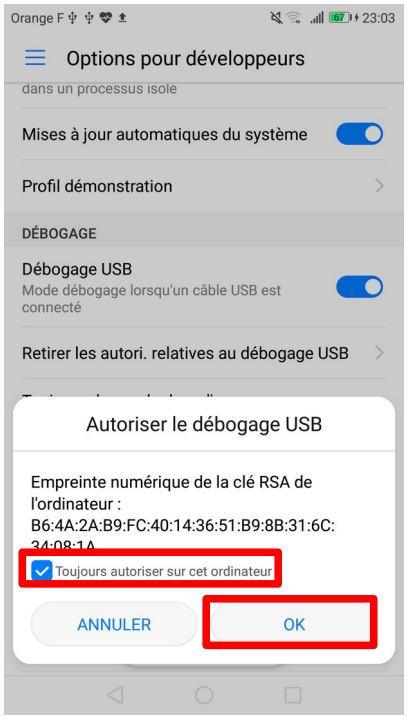






De plus pour activer l'accès au téléphone lorsqu'on le connecte à l'ordinateur, il faut activer l'accès au téléphone puis reconfirmer l'autorisation du débogage sur l'ordinateur utilisé. Pour éviter d'autoriser le débogage à chaque fois que l'on connecte le smartphone, on coche la case « Toujours autoriser sur cet ordinateur » comme ci-dessous :





Maintenant il est possible d'utiliser les commandes adb pour tester le bon fonctionnement de l'outil et la bonne connectivité entre le smartphone et l'ordinateur. Par exemple, si je veux obtenir la liste des smartphones connectés en usb à l'ordinateur il faut que je tape la commande :

adb devices

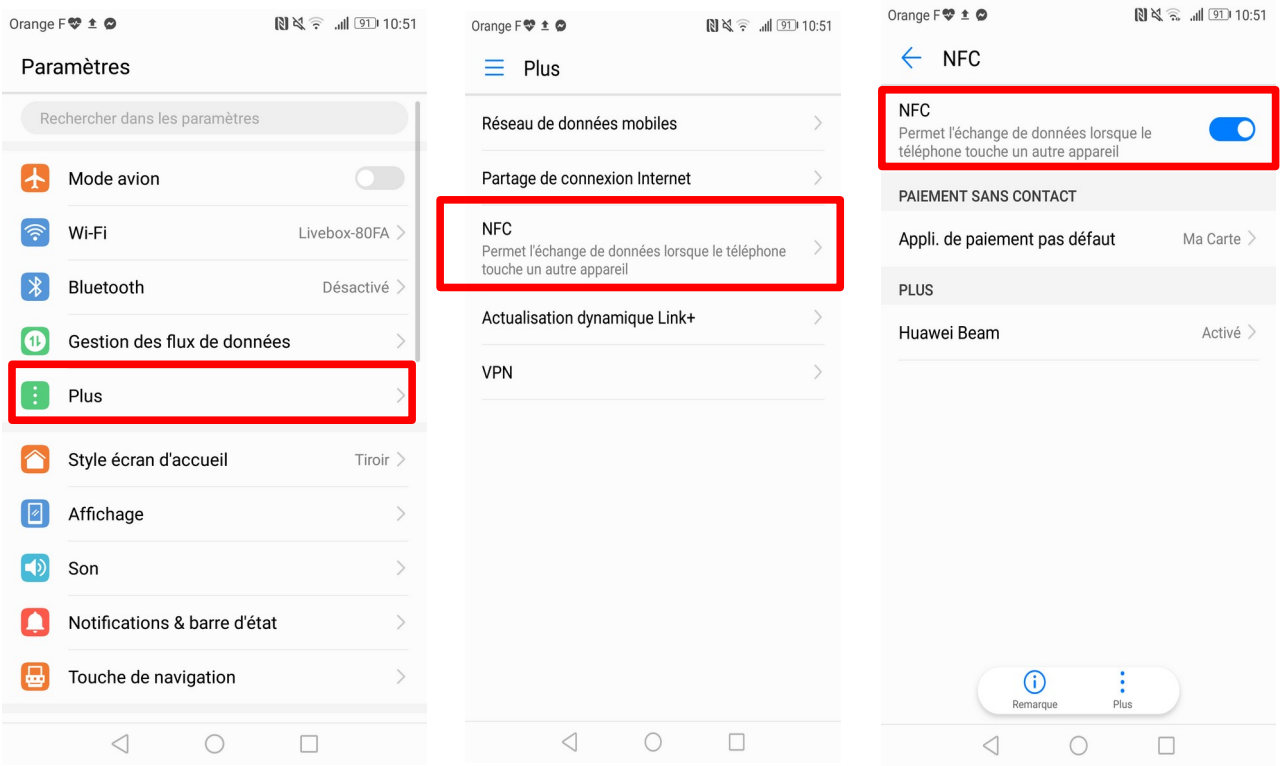
```
C:\Users\hurea>adb devices
List of devices attached
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
4TE7N16B17000721        device
```

On observe qu'il y a un téléphone détecté, il est identifiable grâce au numéro de série que l'on retrouve dans les réglages du smartphone dans l'onglet « A propos du téléphone » puis dans l'onglet « État ». En scrollant un petit peu, on retrouve le même numéro, ici « 4TE7N16B17000721 ».

3 - DÉVELOPPEMENT DE L'IDENTIFICATION D'UN AGENT DE SÉCURITÉ

3.1 - Prérequis

L'identification de l'agent de sécurité requiert que la technologie « NFC » (Near Field Communication) soit présente sur le smartphone. Pour l'activer il faut se rendre dans les réglages du smartphone, dans l'onglet « Plus » puis « NFC » comme les captures ci-dessous :



3.2 - Problème de développement

Suite au confinement déclenché au mois de Mars, j'ai dû installer tous les outils, le logiciel de développement et drivers nécessaire au fonctionnement de l'application sur mon ordinateur personnel. J'ai essayé de compiler sur mon smartphone des applications sans succès, sur Windows et Linux à cause de soucis de drivers sqlite. Je me suis retrouvé bloqué pour développer cette partie car lorsque Brice Mezerette, développeur de l'application du Smartphone, et moi avons essayé de nous partager sa machine virtuelle pour que je puisse développer sous Linux comme il le fait, nous nous sommes rendu compte que partager 51Go par internet est très compliqué.

C'est pourquoi après avoir discuté avec nos professeurs et entre nous, nous nous sommes mis d'accord pour que cette tâche soit réalisé par Brice Mezerette, avec mon soutien à distance, car cette tâche nécessite d'une part l'utilisation du NFC qu'il maîtrise et d'autre part l'identification à l'aide d'une base de données est déjà utilisé sur l'application de supervision. De plus, il s'agit d'une tâche qui ne bloque pas réellement le fonctionnement du système, elle sera donc étudiée et réalisée plus tard.

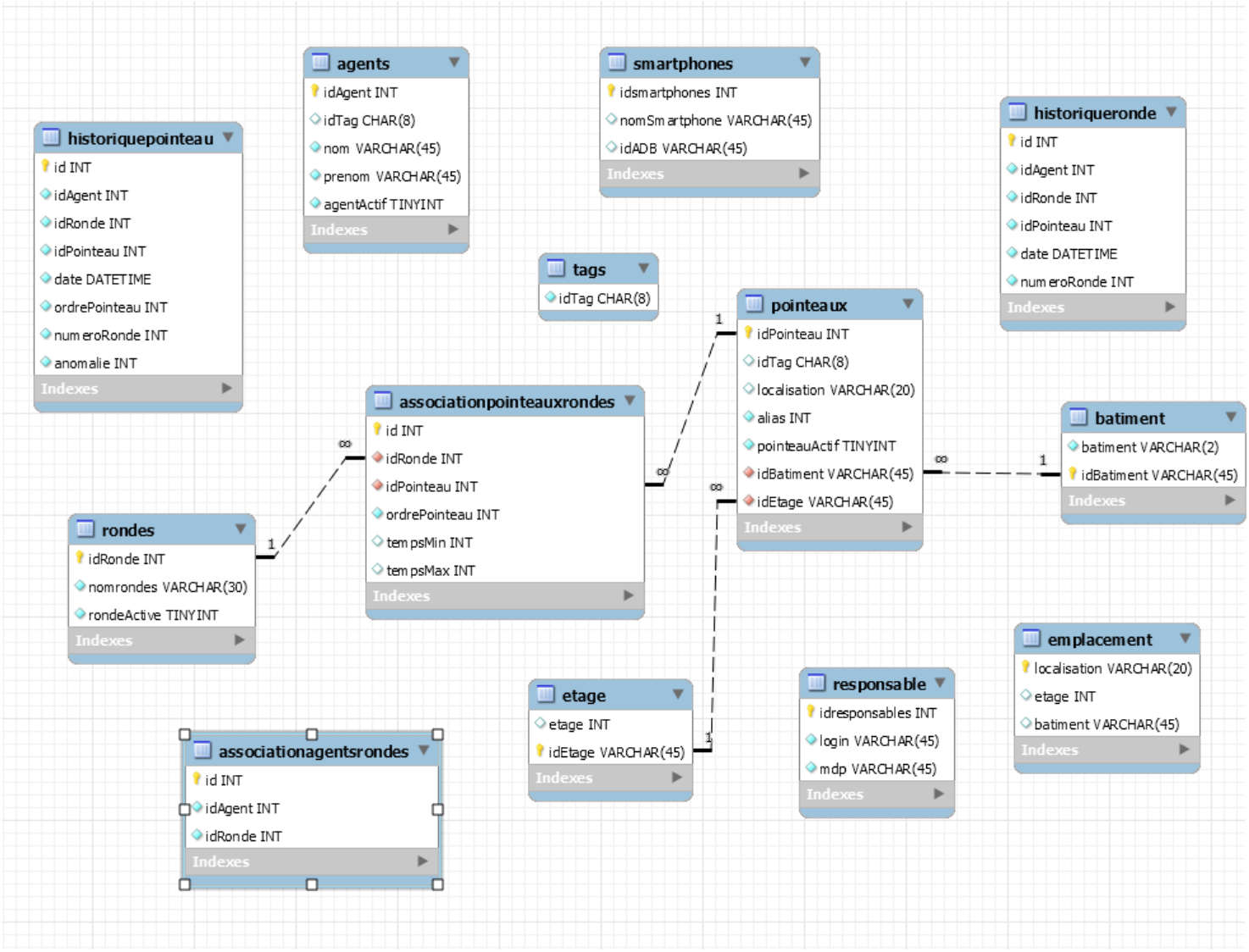
4 MISE EN PLACE DES BASES DE DONNÉES MYSQL ET SQLITE

4.1 - Synoptique des deux bases de données

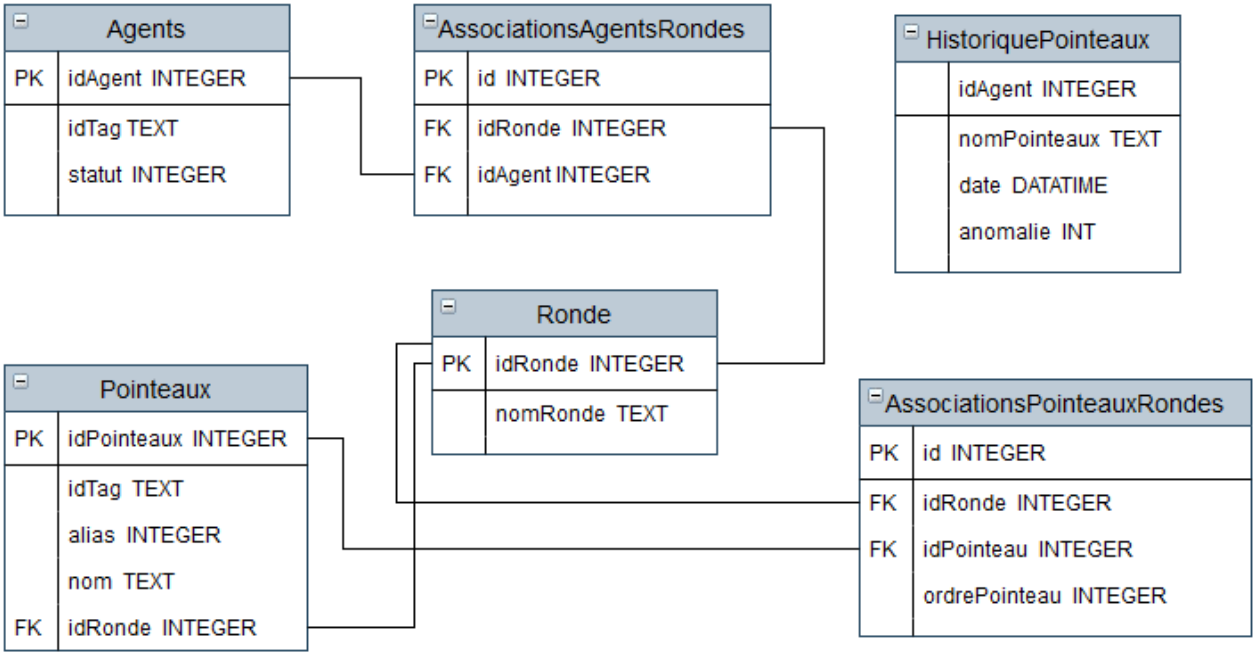
Pour sauvegarder les rondes, les pointeaux, les agents et pour réutiliser ces informations nous utilisons deux types de base de données, Mysql et Sqlite. La première est utilisée sur l'ordinateur pour la supervision des rondes et la seconde est utilisée sur le smartphone.

Voici ci-dessous la structure des deux bases de données :

4.1.1 - *Structure de la base de données de l'ordinateur*



4.1.2 - *Structure de la base de données du smartphone*



La base de données du smartphone est complétée à partir des informations présentes dans celles de l'ordinateur. Lors de la synchronisation, le programme incrémente les valeurs que je souhaite transmettre dans un nouveau fichier nommé « ControleurDeRonde.db », qui est une copie d'une base de donnée que j'appelle « ControleurDeRondeModele.db » qui contient la structure de la base de donnée à transmettre au smartphone.

Voici un exemple de création de la table Pointeaux de la base de données du smartphone :

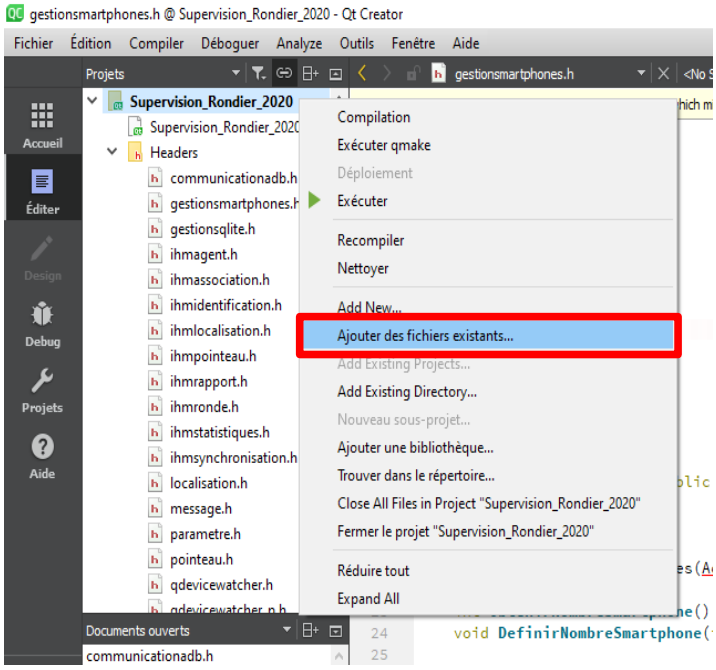
```
CREATE TABLE Pointeaux (
   idPointeaux INTEGER PRIMARY KEY,
   idTag TEXT,
   alias  INTEGER,
   nom  TEXT,
   idRonde  INTEGER,
   FOREIGN KEY (idRonde) REFERENCES Ronde (idRonde)
);
```

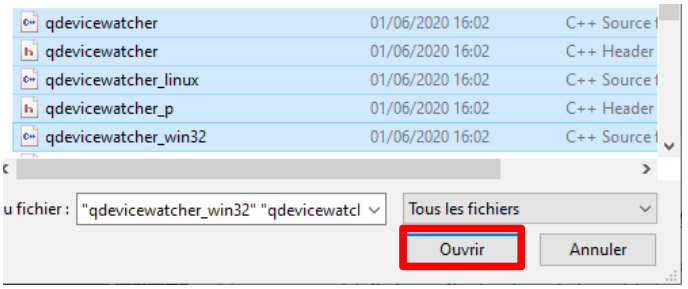
5 - DÉVELOPPEMENT DE LA SYNCHRONISATION DU SMARTPHONE D'UN AGENT

5.1 - Prérequis

Comme précédemment annoncé, l'application de supervision de l'ordinateur utilise l'outil « adb » pour toute les communications avec le smartphone et est programmée sous la version 5.12.3 de Qt Creator.

J'utilise la librairie nommée QdeviceWatcher, cette librairie utilisée par le précédent groupe permet de détecter la connexion d'appareils. Elle est en libre accès via github (https://github.com/wang-bin/qdevicewatcher). Il est très facile d'installer et d'utiliser cette librairie, il suffit d'ajouter les fichiers dans le projet Qt :





Puis d'inclure la librairie dans le programme de Qt:

```
#include "qdevicewatcher.h"
```

Il est assez simple de comprendre le fonctionnement de cette librairie. Des signaux sont émis à chaque fois qu'un périphérique est connecté ou déconnecté de l'ordinateur. On utilise ces signaux pour qu'à chaque détection d'un appareil, on appelle une méthode pour exécuter une action précise.

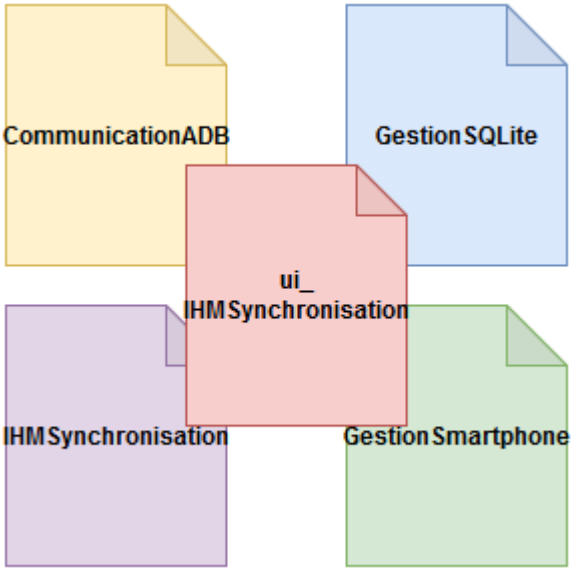
Voici un exemple de signal connecté à un slot qui exécute la détection des appareils connectés :

```
connect(detectionUSB, SIGNAL(deviceAdded(QString)), this, SLOT(onSmartphoneConnecte()))
```

- detectionUSB est le nom de l'attribut de la classe QDeviceWatcher se chargeant de la détection des appareils.
- Le signal deviceAdded() est le signal émit lors de la détection d'un appareil.
- Le slot onSmartphoneConnecte() est la méthode appelée lorsque le signal est reçu.

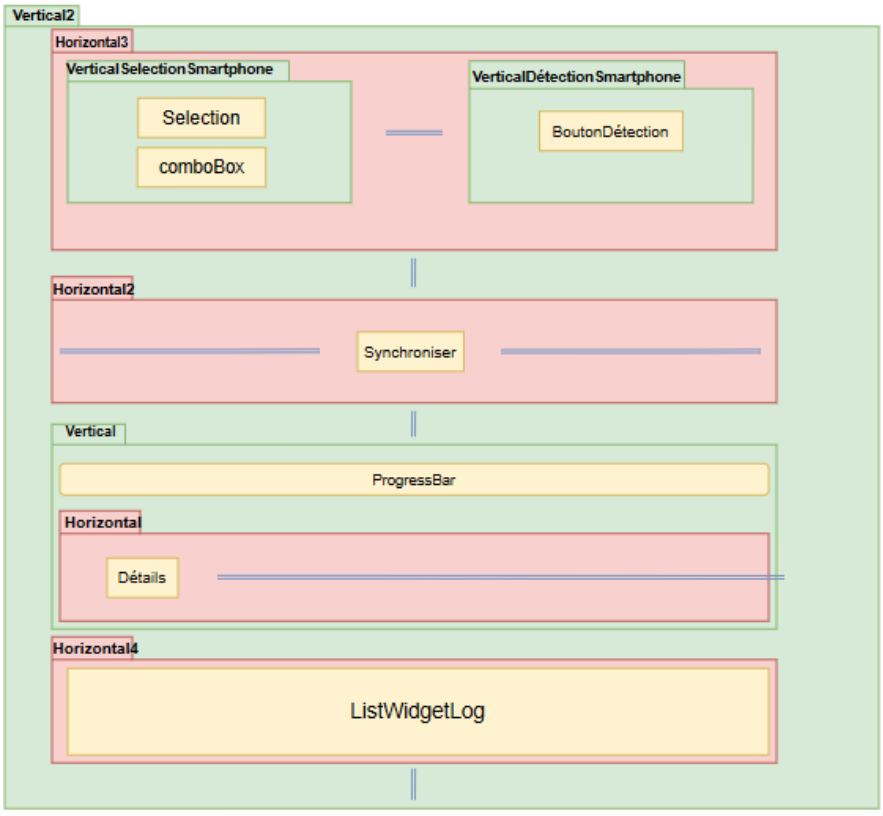
5.2 - Structure de la partie Synchronisation

La partie synchronisation utilise au total 5 classes :

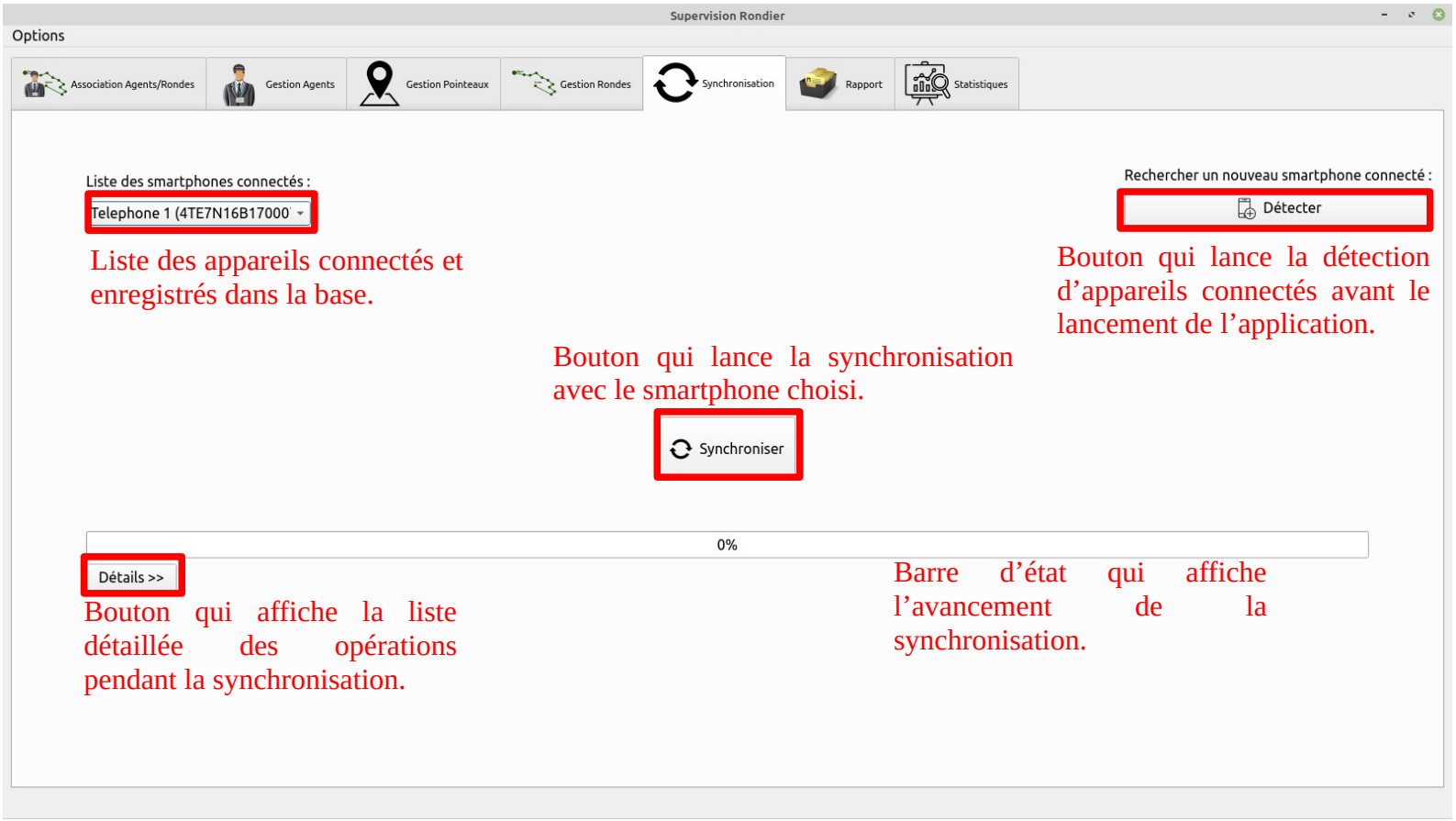


- ui_IHMSynchronisation est la classe qui s'occupe de créer l'interface en dynamique de l'onglet rapport.
- CommunicationADB s'occupe de transférer des fichiers entre le smartphone et le poste de supervision.
- GestionSQLite s'occupe de la connectivité à la base de données sqlite et du transfert entre les deux bases.
- IHMSynchronisation s'occupe des actions à effectuer lorsqu'il y a des interaction sur l'interface .
- GestionSmartphone gère les commandes adb.

La structure de l'Interface Homme Machine correspond au schéma que j'ai créé après avoir étudié la classe ui_IHMSynchronisation des précédentes années :



On obtient alors cette interface :



5.3 - Listage des appareils connectés

Pour pouvoir synchroniser le smartphone il faut tout d'abord le choisir dans la liste de l'interface celui que l'on souhaite utiliser. Ces smartphones sont récupérés grâce à une liste de smartphones enregistré dans la table « smartphones » de la base de données.

- Si un nouveau téléphone est connecté au lancement de l'application, après l'authentification du superviseur :

La méthode **DemanderValeur()** de la classe **Message** demande d'attribuer un nom au numéro que l'on vient de récupérer avec la méthode **mettreAJourListeSmartphone()** de la classe C**ommunicationADB**. Les informations sont enregistrées dans la base de données de l'ordinateur avec la méthode **AjouterSmartphone()** de la classe **AccesMysql**.

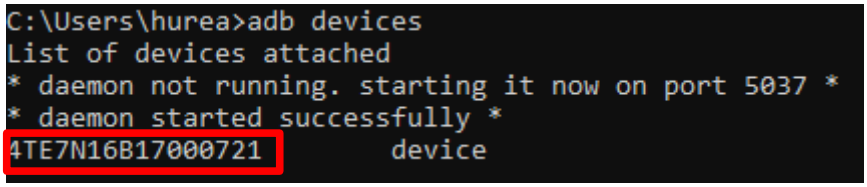
- Après avoir enregistré le nouveau smartphone ou si le smartphone est déjà connu :

La méthode **ActualiserListeSmartphone()** de la classe **GestionSmartphone** incrémente ces nom et numéro de téléphone dans la **listeSmartphone** récupérés de la base de donnée. La classe **IHMSynchronisation** possède une méthode **onListeSmartphoneChange** que l'on appelle lorsque l'on reçoit le signal **listeSmartphoneChange** émit par la classe **GestionSmartphone**. On met à jour le comboBox avec **miseAJourComboBox()** dans cette méthode pour y afficher la liste des smartphones enregistrés dans la base de données.

Pour récupérer uniquement l'id du smartphone lorsqu'on utilise la commande adb, j'applique une expression avec la méthode **onQProcessListeAppareilsReadyRead()** de la classe **CommunicationADB** qui est **\\n(.*?)\\t** qui indique que :

- «(.*?)» → Je capture tous les caractères sur toutes les lignes.
- «\n» → Seulement les caractères se trouvant après un retour à la ligne.
- «\t» → Et seulement les caractères avant une tabulation.

On retrouve alors uniquement l'id du smartphone entouré en rouge ci-dessous :



5.4 - Développement de l'envoi vers le smartphone

Pour le déroulement d'une ronde l'agent a besoin d'informations telles que les rondes, les pointeaux, les agents… Pour envoyer ces dernières je converti les informations que je désire dans une base de donnée en sqlite, utilisable sur smartphone, que je peux facilement transférer sur le smartphone grâce à adb.

Pour cela je créer un fichier nommé « ControleurDeRondeModele.db » qui est la structure, sans valeurs, de la base de donnée à envoyer sur le téléphone.

```
cheminModeleSQLite = QCoreApplication::applicationDirPath() + "/ControleurDeRondeModele.db";
cheminSQLiteTransfere = QCoreApplication::applicationDirPath() + "/ControleurDeRonde.db";
cheminSQLiteCopie = QCoreApplication::applicationDirPath() + "/ControleurDeRondeCopie.db";
```

Lors de la synchronisation, une copie de ce fichier est effectué, correspondant au fichier « ControleurDeRonde.db », dans le build de l'application. Lorsque la copie est effectué, le programme exécute la méthode **synchroniserDoneesPC()** de la classe **GestionSQLite** qui appelle les méthodes :

- **synchroniserAgents()**
- **synchroniserPointeaux()**
- **synchroniserRondes()**
- **synchroniserAssociationsAgentsRondes()**

5.4.1 - Principe du transfert entre les deux bases de données

Je vais prendre l'exemple de la méthode **synchroniserPointeaux()** de la classe **GestionSQLite** pour expliquer comment fonctionne le transfert entre les deux bases de données.

Tout d'abord je récupère les pointeaux enregistrés dans la bdd avec la méthode **ObtenirPointeaux()** de la classe **AccesMysql.**

```
SELECT * FROM pointeaux ORDER BY alias
```

Pour chaque pointeau, **unPointeau** est créé et permet de conserver toutes les informations d'un pointeaux.

```
unPointeau = new Pointeau(requete.value("alias").toInt(),
requete.value("idTag").toString(), requete.value("localisation").toString(), requete.value("idPointeau").toInt());
```

Ensuite, les pointeaux obtenus sont ajoutés dans une liste appelé **listePointeaux**.

Pour insérer les valeurs dans la base de données qui sera envoyé sur le téléphone, j'utilise la requête :

```
INSERT INTO Pointeaux(idPointeaux, idTag, alias, nom) values(:idPointeau, :idTag, :alias, :localisation)
```

Chaque valeur comme par exemple « :idPointeau » est la valeur récupéré dans **listePointeaux** que l'on a associé précédemment et récupéré comme ci-dessous :

```
requeteSQLite->bindValue(":idPointeau", listePointeaux.at(i)->ObtenirIdPointeau())
```

La base de données se rempli donc au fur et à mesure. Pour détaillé le remplissage, le programme utilise des signaux pour connaître les différentes étapes. Ce sont les messages de ces signaux qui seront affichés dans la liste détaillés sur l'IHM.

J'ai connecté le signal **nouveauLog()** dans la méthode **on_pushButtonSynchroniser_clicked()** de la classe **IHMSynchronisation**, qui renvoie à la méthode **onNouveauLog()** de cette même classe. Donc lorsque j'émets un signal dans une méthode de la classe **GestionSQLite** comme ceci :

```
emit nouveauLog("Copie pointeaux en cours...", 1);
```

Je récupère donc le message « Copie pointeaux en cours » pour l'insérer dans la liste détaillée que l'on affiche en appuyant sur le bouton « Détails ».

J'applique le même principe pour les autres méthodes de synchronisation.

Pour envoyer le fichier obtenu après que les données soit transférées, j'utilise la méthode **envoyerFichierSQLite()** de la classe **CommunicationADB** qui exécute la commande :

```
adb -s " + idSmartphone + " push \"" + cheminSQLitePC + "\" \"" + cheminSQLiteAndroid + "\""
```

Avec comme paramètre cheminSQLitePC et cheminSQLiteAndroid défini plus tôt dans la partie « Développement de l'envoi vers le smartphone ».

5.5 - Développement de la réception sur l'ordinateur

5.5.1 - Principe de fonctionnement

La récupération des données du téléphone vers le smartphone s'effectue après que les données de l'ordinateur ont été transférée sur ce dernier. Pour cela, le slot **onQProcessLogsReadyRead()** de la classe **IHMSynchronisation** est appelé lors de la réception du signal **ReadyRead()** émis lors de l'exécution de la commande suivante :

```
"adb -s" + ui->comboBoxSelectionSmartphone->itemData(ui->comboBoxSelectionSmartphone->currentIndex()).toString() + " logcat -c"
```

Cette commande permet d'accéder au fichier log du smartphone et lit ce dernier pour récupérer le message « BDDSynchroFin ». Lors de la lecture de ce message, on lance la récupération des informations à l'inverse, en copiant les informations du smartphone vers l'ordinateur en utilisant la commande :

```
"adb -s " + idSmartphone + " pull \"" + cheminSQLiteAndroid + "\" \"" + cheminSQLiteCopie + "\""
```

5.5.2 - Problème rencontré lors de la récupération

Comme précisé au début de ce dossier, l'application utilisait un téléphone « rooté » afin d'accéder à tous les fichiers du téléphone. Lors de mes différentes manipulations du programme pour la partie récupération des données du smartphone sur mon téléphone non rooté j'ai remarqué qu'un souci de permissions m'empêche d'accéder aux logs du smartphone, donc il m'est impossible de lancer la partie de récupération des données comme le montre la capture ci-dessous :

```
int logctl_get(): open '/dev/hwlog_switch' fail -1, 13. Permission denied

Note: log switch off, only log_main and log_events will have logs!
"adb -s 4TE7N16B17000721 logcat -c"
```

J'ai donc essayé d'effectuer l'opération de rootage sur mon smartphone. Il faut savoir qu'en fonction de la marque et du modèle du téléphone l'opération est très différente. De plus, cette opération est très peu recommandé car des problèmes peuvent survenir sur le smartphone comme des soucis d'exécution d'applications ou des défaillances de sécurité. J'ai pris le risque de tenter l'opération, sans succès malheureusement, malgré le nombre d'essais que j'ai pu faire.

Dans les prochaines semaines, je compte alors tenté une nouvelle méthode, notamment le développement d'une relation Client/Serveur entre le smartphone et l'ordinateur pour déclencher la deuxième partie de la synchronisation.

5.6 - Test unitaire

Condition du test		
État initial du module		**Environnement du test**
Programme	Le projet ControleurDeRondes est exécuté et le responsable se connecte.	Smartphone ayant activé le mode débogage. Ordinateur possédant l'outil *adb* et Qt.
Conditions initiales		
Responsable de sécurité connecté et téléphone connecté.		
Procédure de test		

Repère	**Opérations**	**Résultats attendus**
1	Ouverture de l'IHM de la Synchronisation.	Affichage des éléments de l'interface.
2	Ne pas détecter de smartphone.	Aucun.
3	Appuie sur le bouton *Détecter* pour détecter un nouveau smartphone.	Message qui s'affiche pour attribuer un nom au smartphone.
4	Appuie sur valider.	Enregistrement du smartphone.
5	Si nom de smartphone existe déjà.	Message d'erreur qui demande un nouveau nom unique.
6	Détection d'un smartphone déjà identifié.	Affichage du nom du téléphone dans la liste déroulante.
7	Sélection d'un smartphone dans la liste déroulante.	Le bouton Synchroniser devient actif.
8	Appuie sur le bouton synchroniser.	Copie de la base de données de l'ordinateur dans la base de donnée « ControleurDeRonde.db ». Envoi des fichiers de l'ordinateur vers le smartphone au chemin /sdcard/Android/data/com.project.rondierprojet/files/ Récupération des fichiers présents sur le smartphone au chemin indiqué sur l'ordinateur dans le build de l'application. Suppression des fichiers présents sur le smartphone.

9	Si problème lors de la synchronisation.	Affichage de la barre d'état de la synchronisation en rouge.
10	Si pas de problème lors de la synchronisation.	Message synchronisation terminé et barre d'état de la synchronisation de l'interface de l'onglet Synchronisation en vert à 100 %.
Erreur		**Source de problème et solution envisagée**
Impossible de lancer la récupération des données du smartphone.		Pas d'accès root sur le smartphone, impossible d'accéder au fichier log du smartphone. Je vais changer la méthode de déclenchement de la réception des données du smartphone.

6 - Bilan du développement effectué

Pendant le développement de la synchronisation il m'a été possible de rendre fonctionnelle la plus-value que j'ai ajouté au programme, le bouton détection qui permet de détecter un smartphone après le lancement de l'application.

Sur la partie de synchronisation il me reste encore le problème de réception à régler mais je pense avoir assez de possibilité pour régler ce dernier. Concernant l'application en général les requêtes sql fonctionne, il manque seulement quelques modifications qui seront à faire pendant l'intégration avec Jérémi Guillaumin et Antoine Cheruel.

Je trouve le projet vraiment complet dans l'ensemble car nous utilisons plusieurs de nouveaux types de technologie que je ne connaissais pas tel que le NFC ou l'outil adb. De plus, l'utilisation de bases de données de différent format enrichit mes connaissances sur le sujet et multiplie les possibilités d'utilisation. De même pour l'application Android qui permet de nous habituer au langage QML mixé avec du C++.

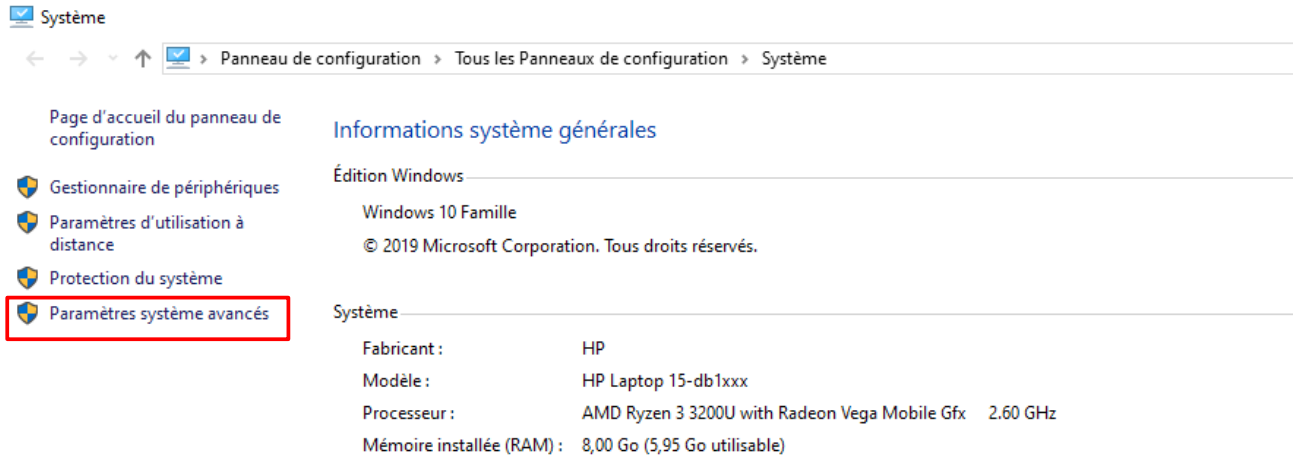
L'intégration sera très intéressante car nous allons nous rendre compte du réel avancement du projet lorsque toutes les parties seront réunis et que le projet commencera à prendre véritablement forme.

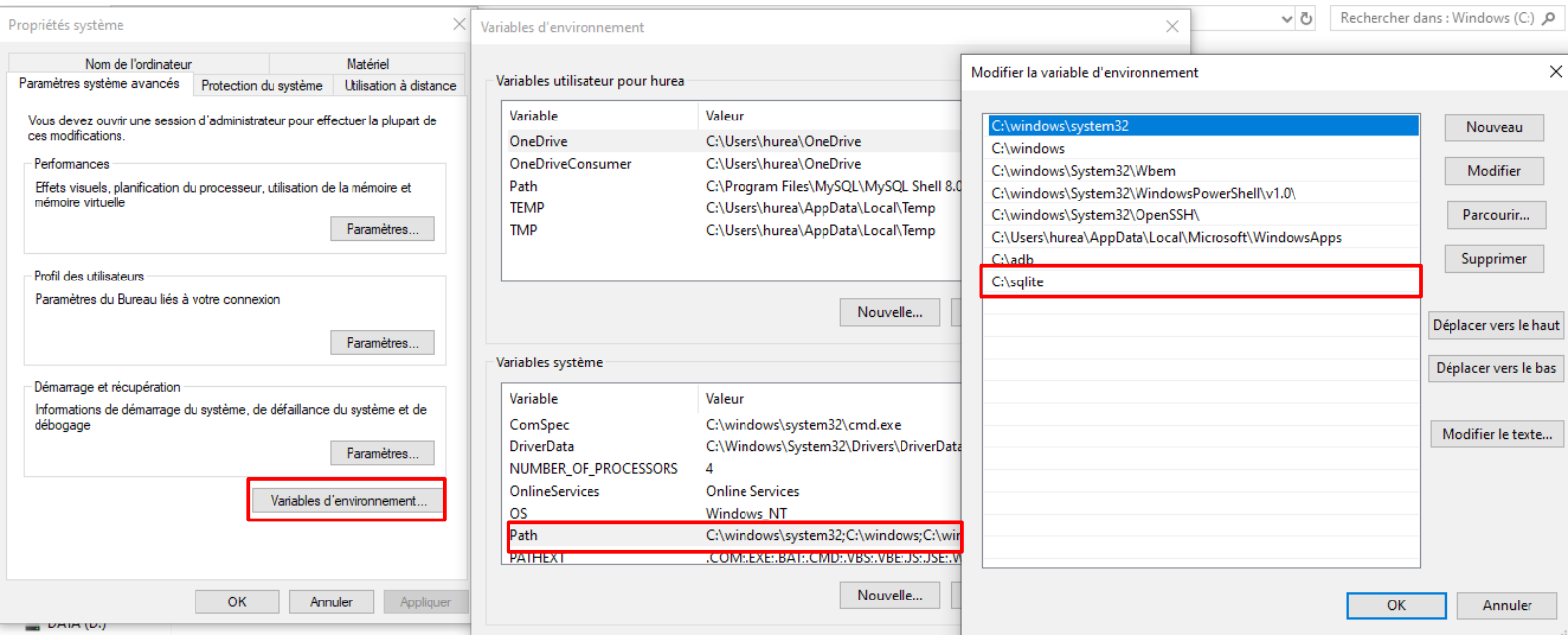
7 - Annexes

7.1 - Installation de sqlite sur l'ordinateur

Après avoir installé sqlite en suivant ce lien il est nécessaire d'ajouter l'outil dans les variables d'environnement de l'ordinateur pour lancer l'application sans utiliser le .exe.

Il suffit de copier le chemin du dossier sqlite créé à la racine du disque C: et coller le chemin dans le path comme ci-dessous :





7.2 - Installation de adb sur l'ordinateur

Il faut effectuer la même manipulation que la partie 7.1 pour utiliser adb sans lancer l'exécutable.