



Système location Autolib

ANNÉE 2019 | BTS SN | LYCÉE CHRISTOPHE COLOMB

Ben zina Wesley | Pean Théo | Belkhir Malik | Belin Maxime

I. Sommaire

II. Présentation Générale du Projet.....	4
III. Présentation de l'équipe.....	4
IV. Schéma de l'architecture Système.....	5
A. Équipement du terminal embarqué.....	5
B. Station Autolib.....	6
C. Système de géolocalisation.....	Erreur ! Signet non défini.
V. Diagramme de cas d'utilisation d'ensemble.....	8
VI. Acquisition des coordonnées GPS.....	9
A. Introduction.....	9
B. Standard NMEA-0183.....	9
C. Exemple de trames.....	10
a. La trame GGA : Données d'acquisition du FIX - GPS.....	10
b. La trame GLL : Position géographique-Longitude/latitude - GPS.....	10
c. La trame GSV : satellite en vue GPS.....	10
D. Programmation.....	11
E. Acquisition des trames.....	11
F. Diagramme séquentiel.....	12
G. Diagramme de classe.....	12
H. Acquisition de la trame GLL.....	13
VII. Bloc Piface : Entrées/Sorties.....	14
A. Introduction.....	14
a. Sorties de la carte PiFace.....	14
b. Entrées de la carte PiFace.....	14
c. Programmation des sorties.....	14
d. Programmation des entrées.....	15
B. Implémentation dans le projet.....	15
C. Diagramme séquentiel.....	15
D. Diagramme de classe.....	15

<u>VIII. Serveur TCP</u>	16
<u>A. Introduction</u>	16
<u>B. Diagramme séquentiel</u>	16
<u>C. Diagramme de classe</u>	16
<u>D. Interface Serveur</u>	17
<u>IX. Interface Homme-Machine Véhicule</u>	18
<u>A. Diagramme de classe</u>	18
<u>B. Interface graphique</u>	19
<u>a. Partie 1</u>	19
<u>b. Partie 2</u>	20
<u>c. Partie 3</u>	20
<u>d. Partie 4</u>	20
<u>X. Programme de lecture des cartes</u>	21
<u>XI. Programme d'envoi des coordonnées GPS</u>	25
<u>XII. Présentation du lecteur de carte</u>	35
<u>XIII. Système intégré à la station</u>	41
<u>A. Lecture et écriture de la carte</u>	41
<u>B. Exemple de commande APDU</u>	43
<u>a. Préparation de l'environnement</u>	43
<u>b. Détection du ou des lecteurs</u>	43
<u>c. Connexion à une carte</u>	44
<u>d. Envoie d'une commande au lecteur</u>	44
<u>C. Les commandes APDU principales</u>	44
<u>D. Diagramme de classe</u>	46
<u>XIV. Communication avec la base de données</u>	47
<u>A. Introduction</u>	47
<u>B. Base de données</u>	47
<u>C. Diagramme de classe</u>	48
<u>XV. Serveur TCP</u>	49
<u>A. Introduction</u>	49

<u>B. Diagramme de classe</u>	49
<u>XVI. Serveur TCP.</u>	50
<u>A. Diagramme de classe</u>	50
<u>B. Interface graphique</u>	51
<u>XVII. Annexe.</u>	57
<u>A. Codage de l'application embarquée (étudiant 1)</u>	57
<u>a. Classe CGPS.</u>	57
<u>b. Classe CGPIO.</u>	60
<u>c. Classe CGSERVEUR</u>	61
<u>d. Classe CIHM</u>	62
<u>B. Codage de l'application embarquée (étudiant 2)</u>	65
<u>a. Classe NFC.</u>	65
<u>b. Classe GSM.</u>	67
<u>C. Source</u>	72
<u>D. Codage de l'application de la station</u>	73
<u>E. Codage de l'application de supervision</u>	86

II. Présentation générale du projet.

Il s'agit d'un système d'Autolib, permettant au client d'interagir avec une borne (interface homme-machine) et de louer un véhicule totalement électrique afin de se déplacer d'un point A vers un point B.

Chaque station comporte une borne interactive à écran tactile, sur laquelle le client s'identifie (carte RFID sans contact délivrée lors de son inscription avec un code confidentiel), choisit la station d'arrivée et diverses options, ou clôt la location en cours.

Le client pourra réserver une place dans la station de destination (pour une durée de 90 minutes). Cette place devient alors indisponible pour les autres usagers.

Il faut passer la carte sur le lecteur situé sur le pare-brise, cela permet de déverrouiller le véhicule ainsi que la trappe où est branché le câble d'alimentation. Le débranchement initie la période de facturation.

Tout au long du parcours, le bouton assistance présent dans la voiture et sur les bornes de location permet de contacter le Central.

Une fois le véhicule garé sur une place réservée ou libre, il faut le rebrancher et passer la carte sur le lecteur pare-brise pour verrouiller les portes. La borne interactive délivre un reçu récapitulant la location.

L'ensemble du projet est réalisé sous QT. La partie embarqué est réalisée sous Linux et la partie station et supervision sont réalisées sous Windows

III. Présentation de l'équipe.

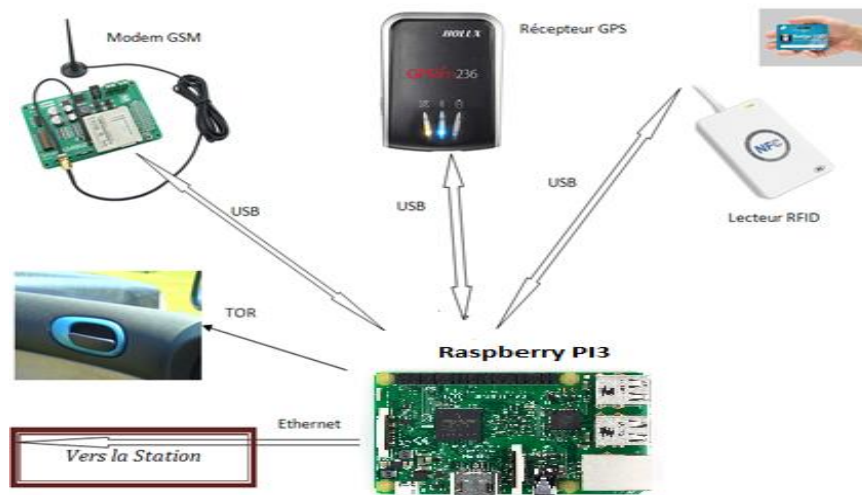
L'équipe de projet est composée de 4 étudiants :

- Ben zina Wesley a pour tâche de réaliser l'application située dans la station.
- Pean Théo est chargé de l'intégration de l'application embarquée dans le véhicule.
- Belkhir Malik doit développer l'application embarquée.
- Belin Maxime réalise une application de supervision au centre de traitement.

IV. Schéma de l'architecture système.

A. Équipement du terminal embarqué

Architecture de l'équipement terminal embarqué

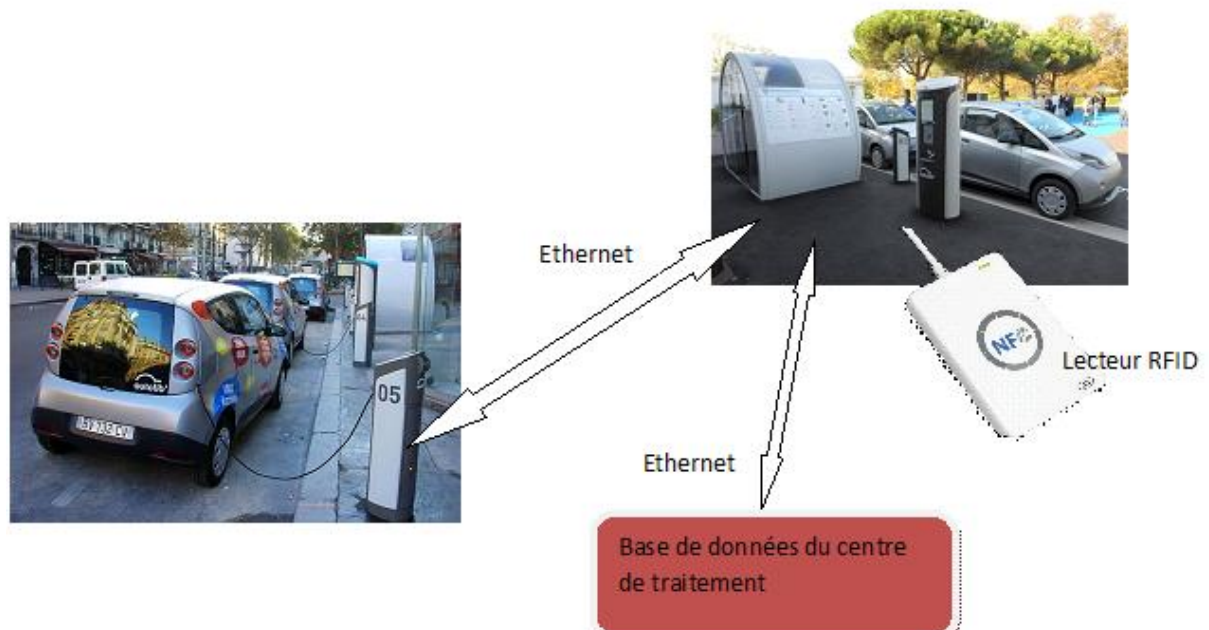


Le système embarqué dans le véhicule est constitué de plusieurs composants :

- Un lecteur de carte RFID pour lire le badge du client.
- Un récepteur GPS pour récupérer les coordonnées GPS en temps réel.
- Un Modem GSM afin de transmettre les valeurs reçues par le récepteur GPS.
- Une carte Raspberry PI3 pour contrôler les composants ci-dessus ainsi que pour créer un serveur TCP.

B. Station Autolib

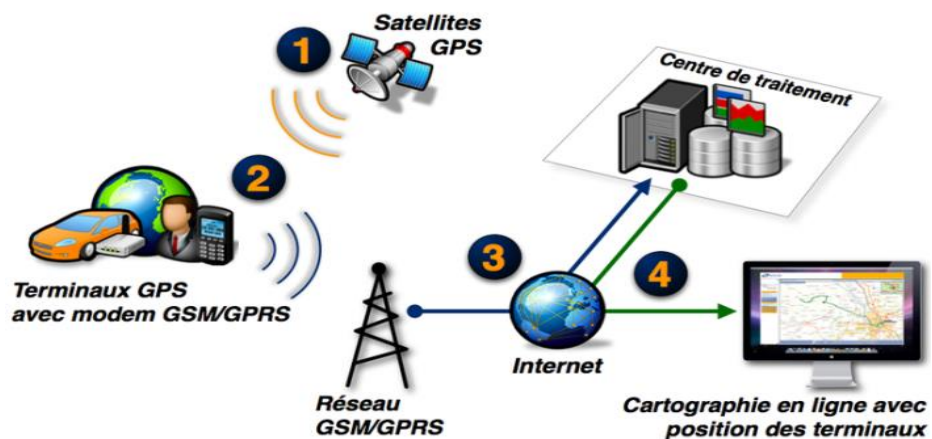
Architecture d'une station AUTOLIB



Le matériel nécessaire à la station, un PC sous Windows pour y intégrer l'application client, un clavier numérique afin que le client puisse saisir son mot de passe à 4 chiffres et pour finir, un lecteur RFID pour que le client s'identifie lorsqu'il y dépose sa carte. La station devra être connectée à Internet pour communiquer avec la base de données.

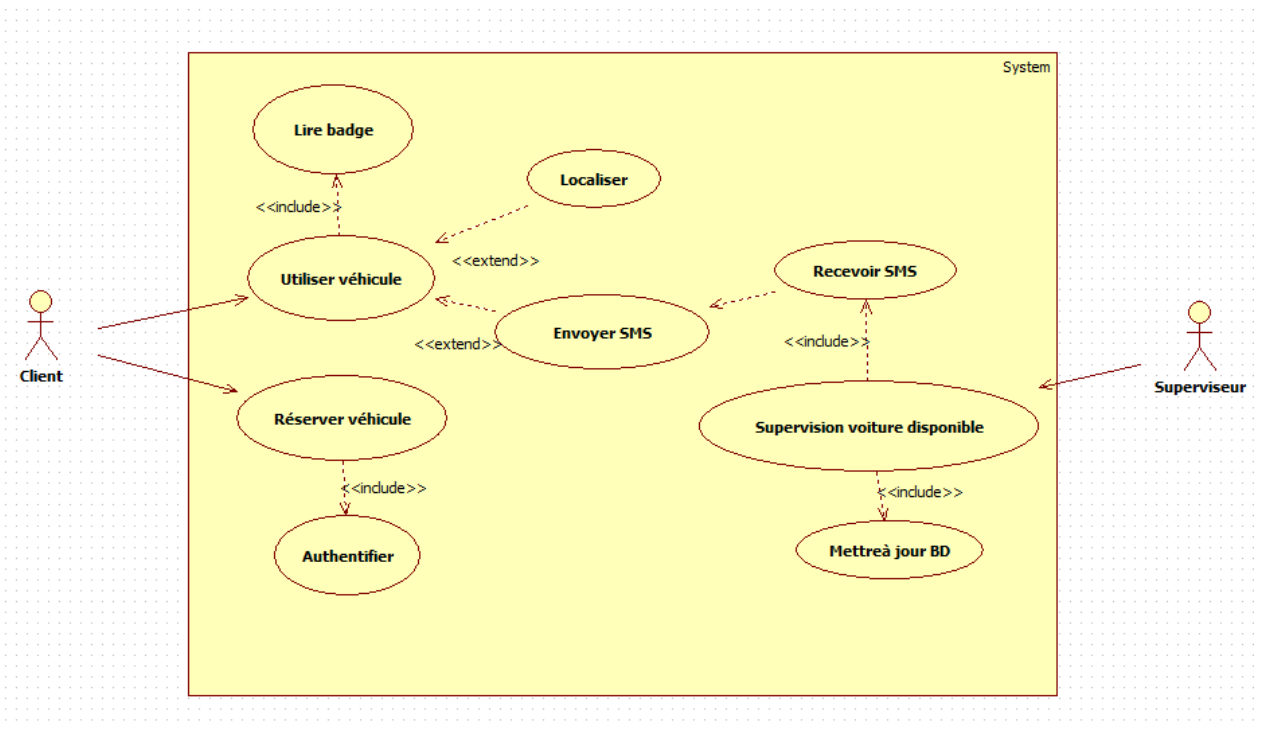
C. Système de géolocalisation

Architecture du système de géolocalisation GPS avec remontée des données via GSM



Le système embarqué reçoit en temps réel ses coordonnées via satellite à intervalle régulier. Les trames de coordonnées sont ensuite transmises par le modem GSM, toujours à intervalle régulier au centre de traitement afin de mettre à jour la position du véhicule dans la base de données ainsi que la position sur la carte du moniteur de la supervision.

V. Diagramme de cas d'utilisation d'ensemble.



VI. Acquisition des coordonnées GPS

A. Introduction

Pour connaître la position du véhicule, on utilise un GPS USB ND100. Cet appareil fonctionne avec le standard NMEA (National Marine & Electronics Association) et reçoit les trames GGA, GGL, GSA, GSV et RMC. Pour connaître la position, on a besoin de 4 satellites visible en permanence minimum.



B. Standard NMEA-0183

Toutes les données sont transmises sous la forme de trames constituées de caractères ASCII ainsi que les caractères [CR] Retour Chariot et [LF] Retour à la ligne. Chaque trame est constituée, au maximum, de 82 caractères à la vitesse de transmission de 4800 bauds.

-Chaque trame commence par le caractère « \$ »

-Suivi par un groupe de 2 lettres pour l'identifiant du récepteur :

GP pour Global Positioning System (Système Américain)

BD ou **GB** pour Beidou (Système Chinois)

GA pour Galileo (Système Européen)

GL pour GLONASS (Système Russe)

- Puis un groupe de 3 lettres pour l'identifiant de la trame

GGA pour GPS Fix and Date

GLL pour Positionnement Géographique Longitude – Latitude

GSA pour DOP et satellites actifs

GSV pour Satellites visibles

VTG pour Direction (cap) et vitesse de déplacement (en nœuds et Km/h)

RMC pour données minimales exploitables spécifiques

-Suivent ensuite un certain nombre de champs séparés par une « , » (séparateur de champs).

-Un champ optionnel : checksum précédé du signe « * » représente le OU exclusif de tous les caractères compris entre le « \$ » et « * ».

-Suivi par la fermeture de la séquence avec un [CR][LF].

C. Exemples de trames

a. La trame GGA : Données d'acquisition du FIX – GPS

\$GPGGA,123519.000,4807.038,N,01131.324,E,1,08,0.9,545.4,M,46.9,M, , *42

123519.000 = Acquisition du FIX à 12:35:19 UTC

4807.0380,N = Latitude 48 deg 07.0380' N

01131.3240,E = Longitude 11 deg 31.3240' E

1 = Fix qualification : (0 = non valide, 1 = Fix GPS, 2 = Fix DGPS)

08 = Nombre de satellites en poursuite.

0.9 = DOP (Horizontal dilution of position) Dilution horizontale.

545.4,M = Altitude, en Mètres, au-dessus du niveau moyen des Océans.

46.9,M = Correction de la hauteur de la géoïde en Mètres par rapport à l'ellipsoïde WGS84 (MSL).

(Champ vide) = nombre de secondes écoulées depuis la dernière mise à jour DGPS.

(Champ vide) = Identification de la station DGPS.

***42** = Checksum

Non représentés **CR et LF**.

b. La trame GLL : Position Géographique – Longitude/Latitude – GPS

\$GPGLL,4916.45,N,12311.12,W,225444,A

4916.46,N = Latitude 49 deg. 16.45 min. Nord.

12311.12,W = Longitude 123 deg. 11.12 min. West (ouest)

225444 = Acquisition du Fix à 22:54:44 UTC

A = Données valides

Pas de checksum

Non représentés **CR et LF**.

c. La trame GSV : satellite en vue - GPS

\$GPGSV,2,1,08,01,40,083,46,02,17,308,41,12,07,344,39,14,22,228,45*75

2 = Nombre de trames GSV avec les données complètes.

1 = Trame 1 de 2 trames (jusqu'à 3 trames)

08 = Nombre de satellites visibles (SV).

01 = N° d'identification du 1er Satellite.

40 = Elévation en degrés du 1er Satellite.

083 = Azimut en degrés du 1er Satellite.

46 = Force du signal du 1er Satellite (Plus grand=meilleur)

(Cette séquence se répète jusqu'à 4 satellites par trames.

On peut donc avoir jusqu'à 3 trames GSV dans une transmission (12 satellites).)

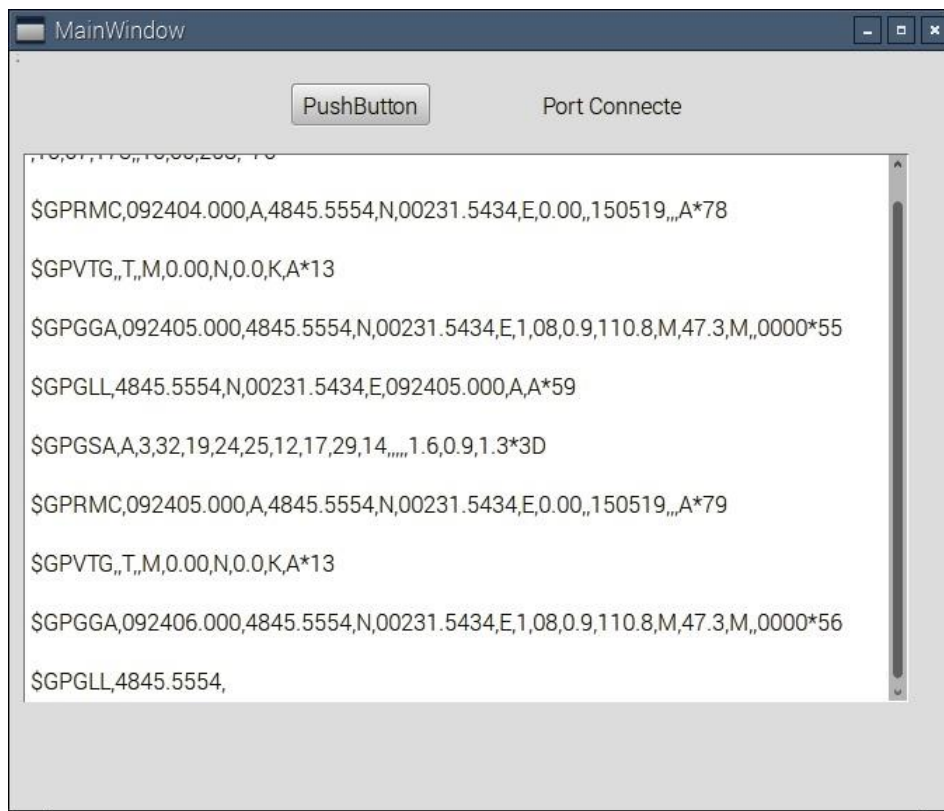
***75** = checksum

D. Programmation

Pour configurer la clé, on doit écrire :

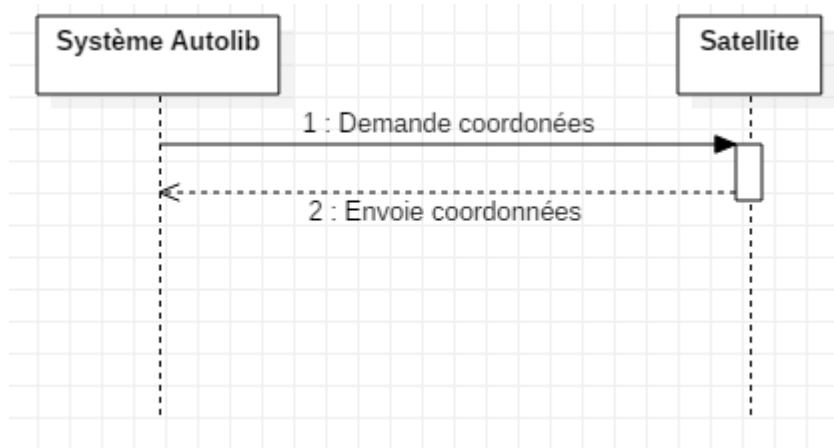
```
if ( (fd = open( "/dev/ttyUSB0" , O_RDONLY | O_NOCTTY )) < 0 ) {}
else{
struct termios tio;
tcgetattr(fd,&tio);
tio.c_cflag = B4800 | CS8 | CLOCAL | CREAD;
tio.c_oflag = 0;
tio.c_lflag = 0;
tio.c_iflag |= IGNPAR;
tio.c_lflag &= ~(ICANON);
tio.c_cc[VTIME] = 0;
tio.c_cc[VMIN] = 1;
tcsetattr(fd,TCSANOW,&tio);
tcflush(fd, TCIOFLUSH);
}
```

E. Acquisition des trames



Voici un exemple d'acquisition de trames

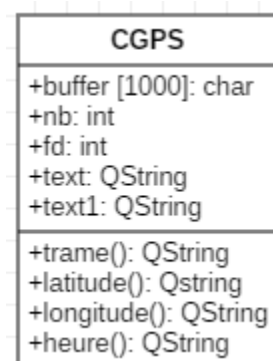
F. Diagramme séquentiel



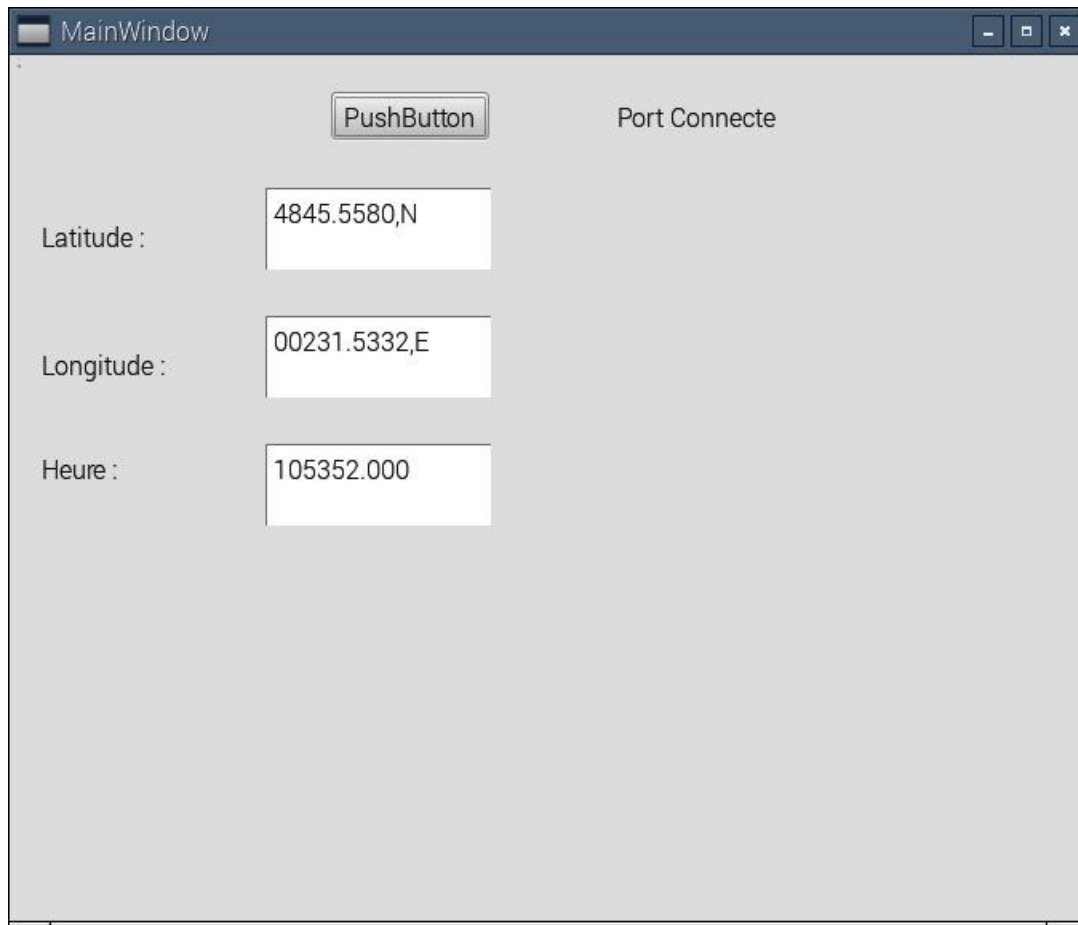
Dans la suite du projet, le système envoie par SMS les coordonnées au centre de traitement à intervalle régulier.

G. Diagramme de classe

La fonction `trame()` permet de récupérer le contenu de la trame GLL (précisé dans le fichier `CGPS.h` présent dans l'annexe). La fonction `latitude()` permet de récupérer la latitude. La fonction `longitude()`, récupérer la longitude et la fonction `heure()` l'heure d'acquisition de la trame. Chaque valeur sont renvoyés dans une `QString` qui stocke les valeurs et permet de les afficher via une interface graphique



H. Acquisition de la trame GLL



The screenshot shows a software window titled "MainWindow" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a "PushButton" at the top center and a label "Port Connecte" to its right. Below these, there are three input fields for data entry:

Label	Value
Latitude :	4845.5580,N
Longitude :	00231.5332,E
Heure :	105352.000

Appuyé sur « PushButton » permet de mettre à jour les coordonnées reçues.

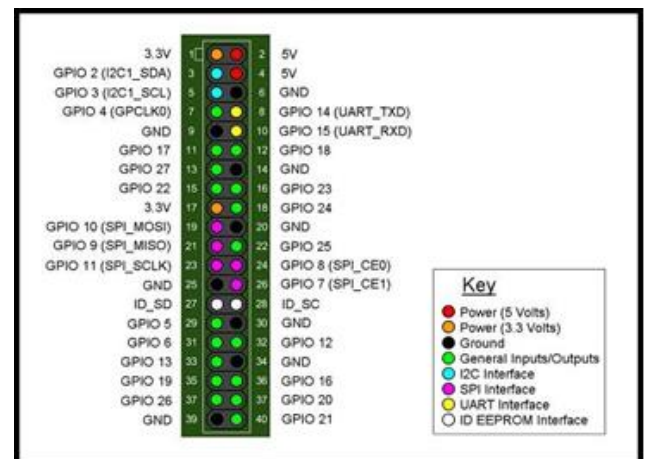
VII. Bloc PiFace : Entrées/Sorties

A. Introduction

La carte PiFace est une création d'Andrew Robinson conçue pour être facilement utilisable, initier les enfants à l'électronique et leur permettre d'interagir rapidement avec le monde réel.



La PiFace se connecte directement sur le connecteur GPIO du Raspberry Pi. Les entrées et sorties se raccordent sur des borniers situés le long des côtés de la carte. La carte est équipée d'un MCP23S17 connecté sur le bus SPI du GPIO et offrant 16 entrées/sorties numériques. Les 16 I/O se répartissent en 8 entrées (dont 4 équipées de boutons poussoirs) et 8 sorties.



a. Sorties de la carte PiFace

Les sorties sont bufferisées par ULN2803 et munies chacune d'une LED de contrôle. Chaque sortie est un Darlington en collecteur-ouvert qui supporte 50V et peut commander une charge jusque 500 mA. Des cavaliers présents sur la carte permettent un certain nombre de réglages : adresse de la carte, alimentation externe, diodes anti-retour, désactivation des relais et désactivation des sorties.

b. Entrées de la carte PiFace

Les entrées sont maintenues à l'état haut (1) par une résistance de tirage activée par défaut. Les entrées sont mises à 0 (activées) par un bouton poussoir présent sur la carte (entrées 0 à 3) ou par un dispositif extérieur mettant l'entrée à la masse (entrées 0 à 7).

c. Programmation des sorties

Pour configurer l'état d'une LED, on doit utiliser la fonction `digital_write(int, int)`

int : numéro du port de la LED, valeur entre 0 et 7.

int : état de la LED, 1 pour l'allumer et 0 pour l'éteindre.

d. Programmation des entrées

Pour lire l'état d'un bouton, on utilise la fonction `digital_read(int)`

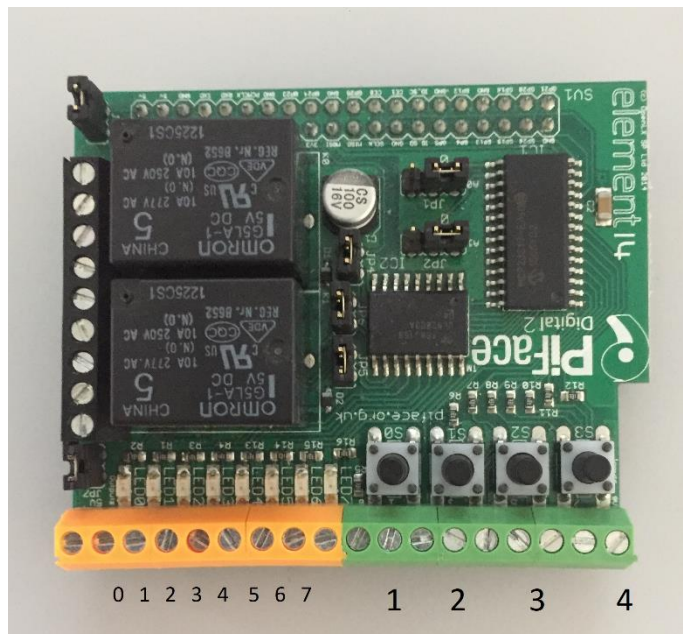
`int` : numéro du bouton

Le programme affichera « 1 » pour l'état « haut » et « 0 » pour l'état « bas ».

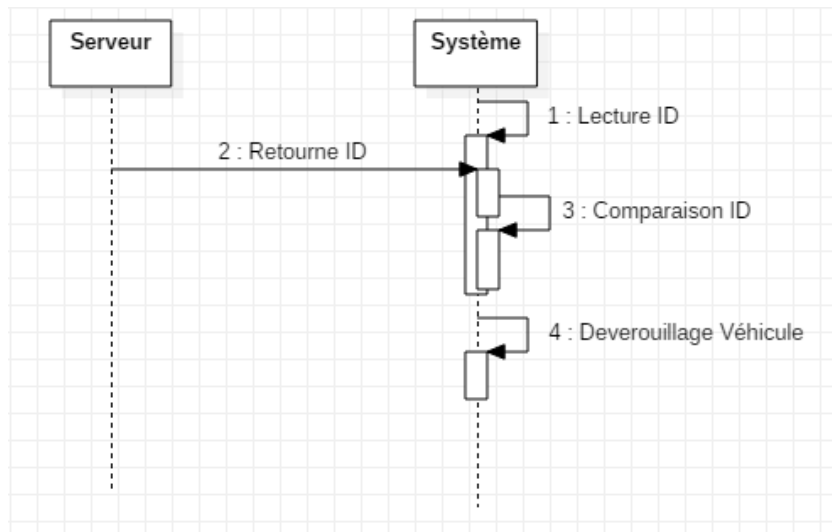
B. Implémentation dans le projet

La carte PiFace permet ici de simuler des éléments du véhicule comme l'état du câble (branché ou non), l'état des portières (fermées ou non), le début/la fin d'une course, ou encore vérifier si le badge qui est présenté sur le pare-brise est bien celui qui a réservé le véhicule sur la borne de la station.

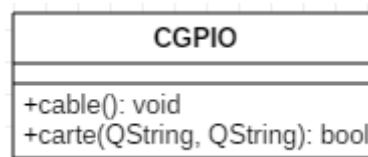
Sur la carte ci-contre, le bouton 4 ne fonctionne pas, les LEDS 1, 3 et 5 sont utilisées ainsi que les boutons 1, 2 et 3.



C. Diagramme séquentiel



D. Diagramme de classe



La fonction `cable()` permet de d'allumer la LED 5 si on appuie sur le bouton 3 et d'éteindre la LED 5 si on appuie sur le bouton 3.

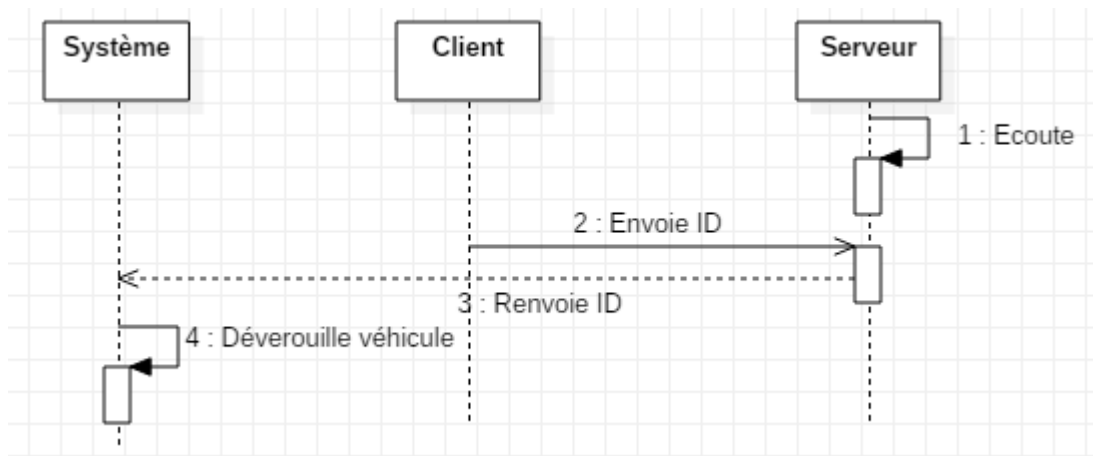
La fonction `carte(QString, QString)` prend deux chaînes de caractères en arguments et compare ces deux chaînes qui correspondent aux valeurs de la carte obtenues sur lors de la réservation ainsi que lors de la lecture sur le pare-brise. Si elles sont égales, la LED 1 s'allume et return « true », sinon les chaînes sont différentes, la LED 3 s'allume et return « false ».

VIII. Serveur TCP

A. Introduction

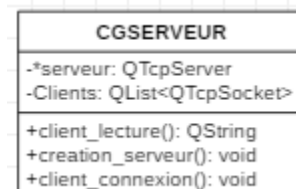
Le serveur TCP a pour but d'assurer les communications entre la station et le véhicule afin que la station puisse transmettre l'ID du badge du client qui réserve le véhicule.

B. Diagramme séquentiel



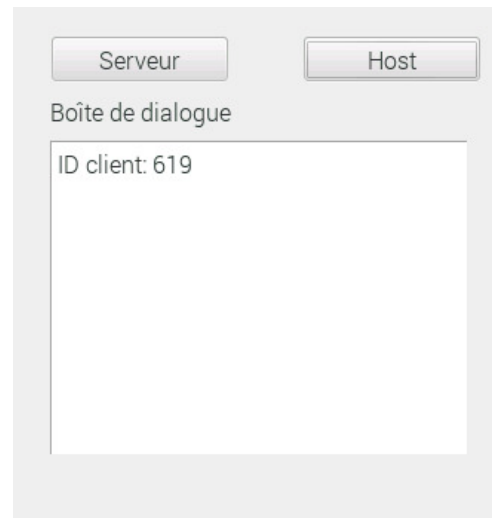
C. Diagramme de classe

creation_serveur() : permet de créer un serveur TCP.
 client_connexion() permet aux machines de se connecter au serveur (dans notre cas : la station).
 client_lecture() : cette fonction sert à lire ce que la station envoie lors de sa connexion (l'ID du badge client), de stocker et renvoyer cette valeur pour la comparer avec celle lue par le lecteur sur le pare-brise.



D. Interface serveur

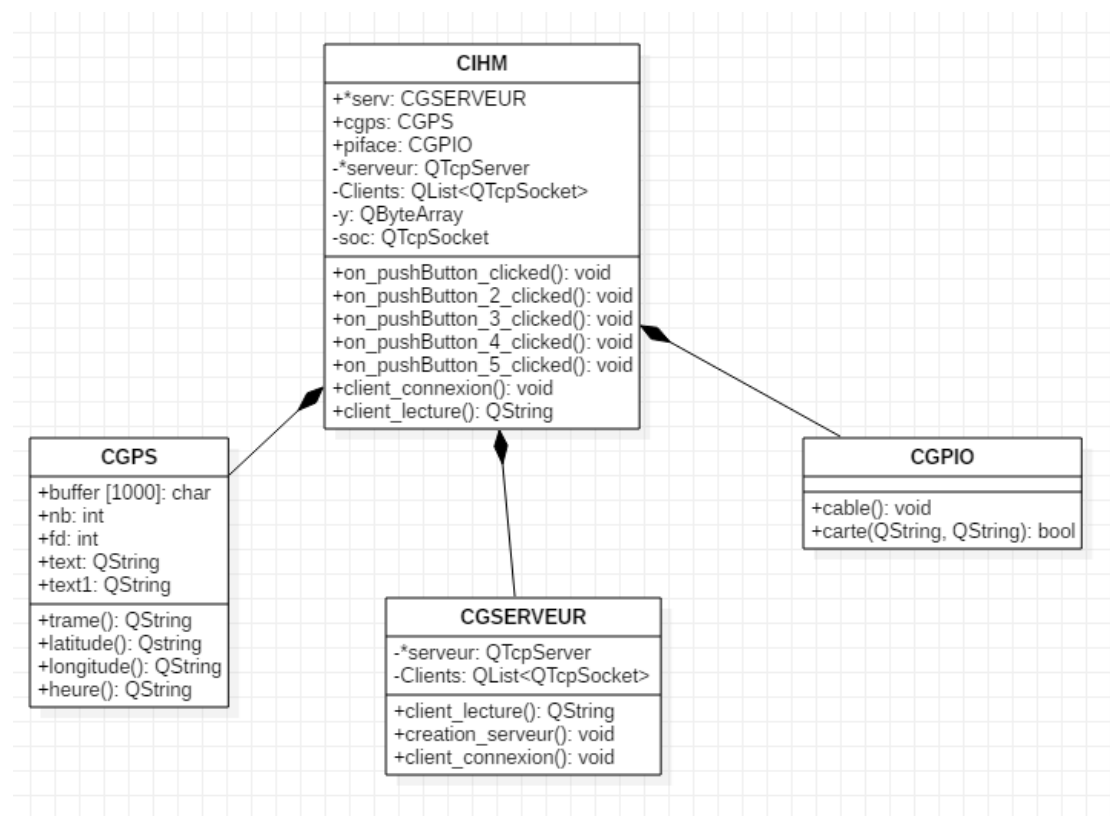
Appuyer sur le bouton « Serveur » permet d'afficher une boîte de dialogue et un bouton « host ». Appuyer sur le bouton host permet de créer un serveur TCP et donc à la station de se connecter. Lors de la connexion, la station envoie un message qui est affiché dans la boîte de dialogue. Ce message contient la valeur de l'ID du client qui a réservé le véhicule.



IX. Interface Homme-Machine Véhicule

L'interface Homme-Machine (IHM) permet d'afficher les informations sur un écran.

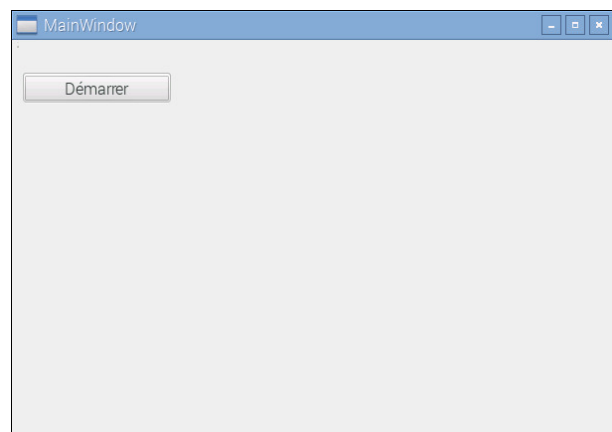
A. Diagramme de classe



B. Interface graphique

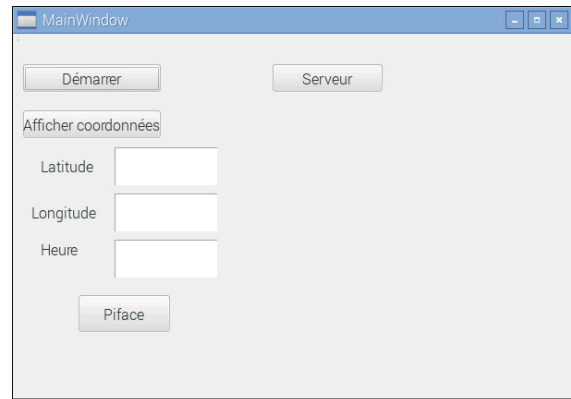
a. Partie 1

Il n'y a qu'un seul bouton au début. Appuyer dessus permet de « démarrer » l'interface.



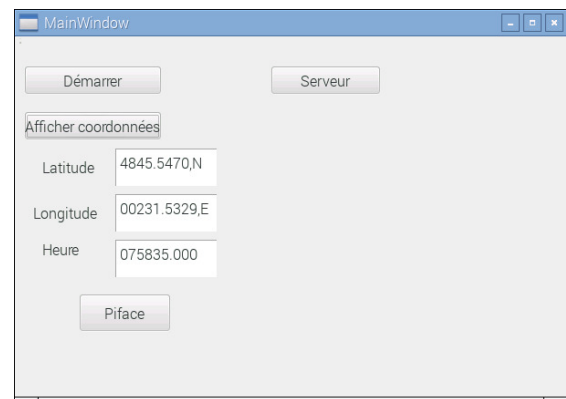
b. Partie 2

Des zones de textes et des boutons supplémentaires sont affichés. Le bouton PiFace permet de simuler les opérations d'entrées et de sorties évoquées précédemment.



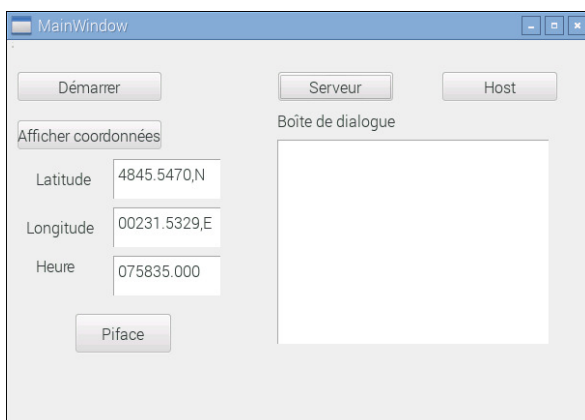
c. Partie 3

Lorsqu'on appuie sur le bouton « Afficher coordonnées », les coordonnées sont affichées dans des zones de textes. Appuyer à nouveau dessus permet de mettre à jour les coordonnées.



d. Partie 4

Le bouton « Serveur » a la même utilité que dans l'interface serveur évoquée précédemment et fonctionne à l'identique.



X. Programme de lecture des cartes

A la fin de cette partie, je dois être en mesure de pouvoir identifier que le numéro correspondant à la carte passée sur le lecteur RFID est bien celle correspondant au numéro qui a été alloué à cette autolib. Des cartes RFID ainsi qu'un lecteur RFID ont donc été mis à ma disposition pour me permettre de pouvoir effectuer l'identification du client.

Je crée alors le programme sur le logiciel QT.

Création de la classe CRFID :

Pour commencer j'ai placé les fichiers de la librairie « winscard » dans le dossier du projet, j'ai ajouté la ligne « `LIBS += winscard.lib` » dans le fichier .pro et j'ai inclus winscard (« `#include "winscard.h"` ») dans le code source. Cela me permettra d'utiliser de nombreuses fonctionnalités liées à la carte et au lecteur.

J'ai alors dû installer les drivers du lecteur pour qu'il puisse être reconnu par la machine et ainsi pouvoir l'utiliser correctement, j'ai alors placé dans mon programme un code me permettant de détecter si les drivers sont correctement installés ou non.

```
int ret = SCardEstablishContext( SCARD_SCOPE_USER, NULL, NULL, &hContext );
if( ret != SCARD_S_SUCCESS ){ui->textEdit->setText("Erreur, le driver n'est pas installé.);}
else{ui->textEdit->setText("driver installé.);};
```

Après cette vérification on passe alors à la détection de la présence du lecteur.

```
size = 256;
ret = SCardListReaders( hContext, NULL, readerName, &size );
if( ret != SCARD_S_SUCCESS ){ui->textEdit_2->setText("Erreur, le lecteur n'est pas détecté.);}
else{ui->textEdit_2->setText("lecteur détecté.);};
```

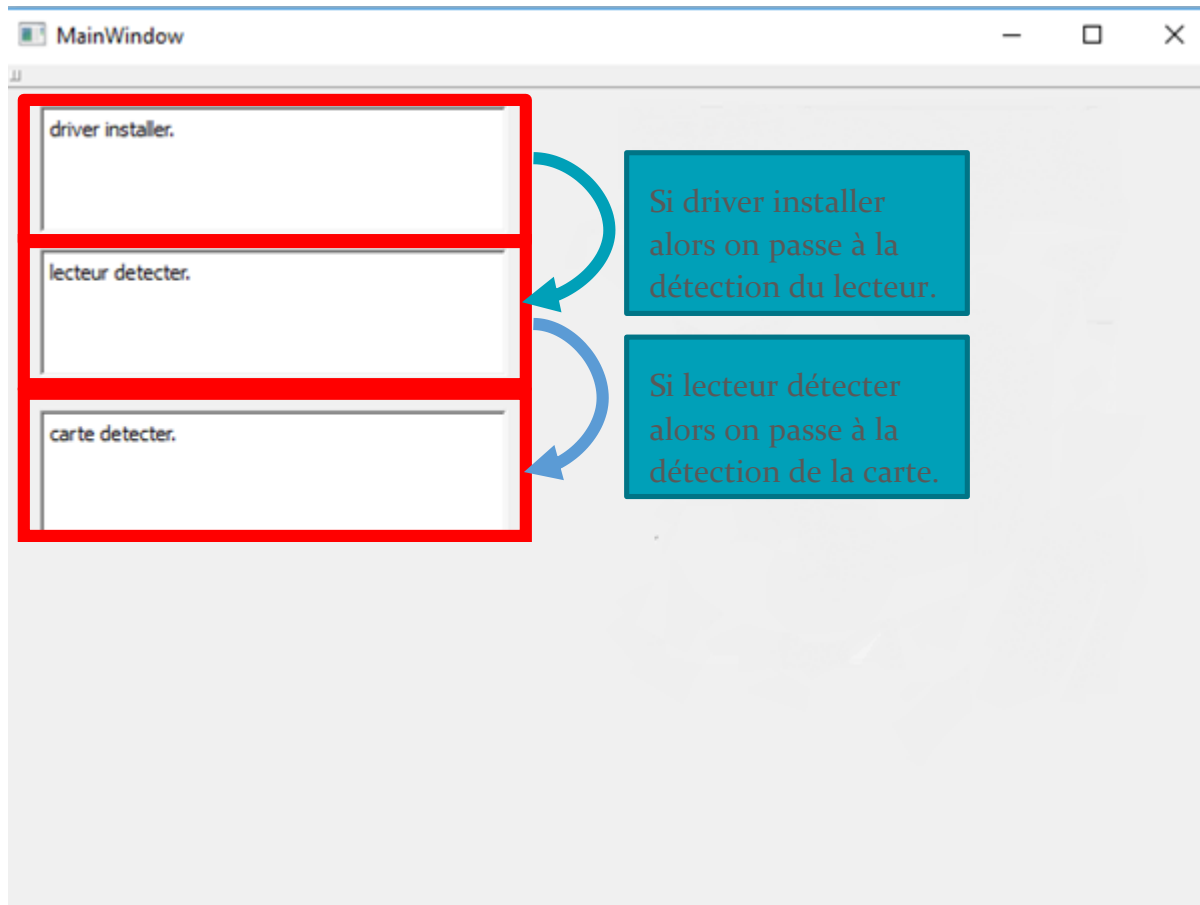
Et une fois que celui-ci est bien présent on peut alors détecter si la carte est présente sur le lecteur.

```
ret=SCardConnect(hContext, readerName, SCARD_SHARE_SHARED, SCARD_PROTOCOL_T0|SCARD_PROTOCOL_T1, &hCard, &pdwActiveProtocol);
if( ret != SCARD_S_SUCCESS ){ui->textEdit_3->setText("Erreur, la carte n'est pas détectée.);}
else{ui->textEdit_3->setText("carte détectée.);};
```

Les fonctions « `ScardEstablishContext()` », « `SCardListReaders()` », `SCardConnect()` ainsi que les variables qu'elles contiennent sont déjà prédéfinies dans le fichier « `winscard.h` ».

Tout ce programme est placé dans un timer qui me permet de pouvoir l'actualiser indéfiniment et ainsi détecter à chaque instant si l'un des outils (lecteur, carte) est manquant.

Après avoir créé l'interface graphique, je place chaque réponse aux nombreuses vérifications dans des textEdit, ce qui me permettra de savoir en temps réel si tout est bien en place.



Après avoir effectué toutes ces tâches, on passe maintenant au chargement des clés d'authentification du lecteur dans le but de pouvoir ensuite lire les données se trouvant dans la carte.

```
SendBuff[0] = 0xFF;
SendBuff[1] = 0x82;
SendBuff[2] = 0x00;
SendBuff[3] = 0; // key store n°
SendBuff[4] = 0x06;
SendBuff[5] = 0xFF; // clé sur 6 octets
SendBuff[6] = 0xFF;
SendBuff[7] = 0xFF;
SendBuff[8] = 0xFF;
SendBuff[9] = 0xFF;
SendBuff[10] = 0xFF;
RecvLen=2;
int ret = SCardTransmit(hCard, NULL, SendBuff, 11, NULL, RecvBuff, &RecvLen);
if ((RecvBuff[RecvLen - 2] == 0x90) && (RecvBuff[RecvLen - 1] == 0x00)) {ui->textEdit_5->setText("c'est ok");}
else {ui->textEdit_5->setText("erreur.");}
```

« SendLen » représente le nombre d'octets envoyé et « RecvLen » représente le nombre d'octets reçu.

On passe ensuite à la fonction qui permet l'authentification pour un bloc. La variable « NbBlock » représente le bloc qu'on sélectionne, on le définira plus tard.

```
for (int i=0;i<262;i++){
    SendBuff[i] = 0;
}
//Chargement du block
SendBuff[0] = 0xFF;
SendBuff[1] = 0x86;
SendBuff[2] = 0x00;
SendBuff[3] = 0x00;
SendBuff[4] = 0x05;
SendBuff[5] = 0x01;
SendBuff[6] = 0x00;
SendBuff[7] = NbBlock;    // n° de bloc
SendBuff[8] = 0x60; //type A    sinon type B = 0x61
SendBuff[9] = 0;
SendLen = 10;
RecvLen = 2;
int ret = SCardTransmit(hCard, NULL, SendBuff, SendLen, NULL, RecvBuff, &RecvLen);
```

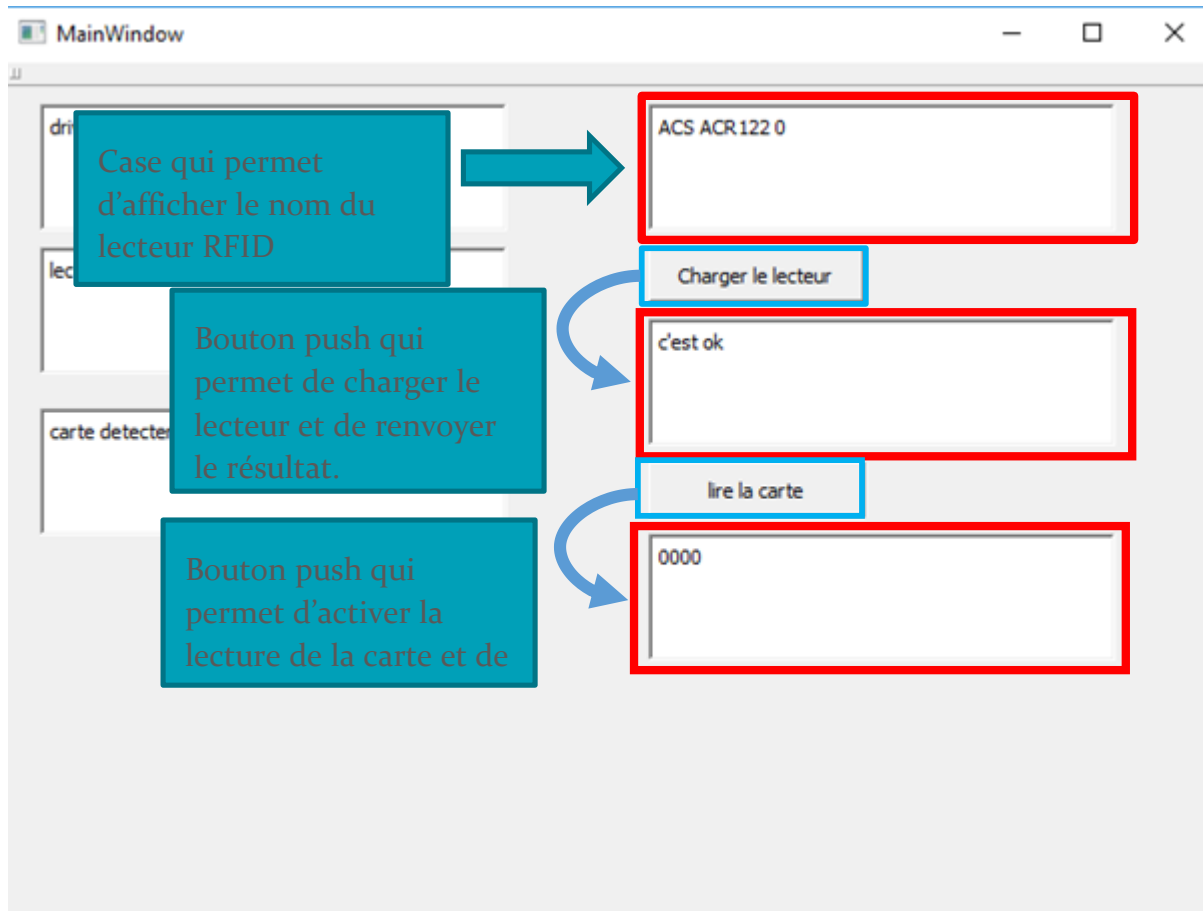
On peut alors commencer à lire les données qui ont été préalablement remplis par mon coéquipier qui se charge de l'enregistrement des clients. On réinitialise le buffer, on envoie la demande de lecture avec « sendbuff », on compare alors les deux dernières valeurs du tableau avec 0x90 et 0x00 et si elles sont équivalentes c'est que la lecture c'est bien dérouler, on remplit alors la QString avec le tableau de « RecvBuff » qui reçoit la lecture et on l'affiche dans un textEdit.


```
for (int i=0;i<262;i++){
    SendBuff[i] = 0;
}
QString a="";
SendLen = 5;
RecvLen = 18;

SendBuff[0]=0xFF;
SendBuff[1]=0xB0;
SendBuff[2]=0x00;
SendBuff[3]=NbBlock;
SendBuff[4]=0x10;

int ret = SCardTransmit(hCard, NULL, SendBuff, SendLen, NULL, RecvBuff, &RecvLen);
if ( (RecvBuff[RecvLen - 2]== 0x90) && (RecvBuff[RecvLen - 1]== 0x00)){
    ui->textEdit_6->setText("Lecture du bloc réussi ");
    for (int i=0;i<16;i++)
    {
        a += RecvBuff[i];
    }
    ui->textEdit_6->setText(a);
}
else
{
    ui->textEdit_6->setText("Lecture Bloc Impossible") ;
}
```

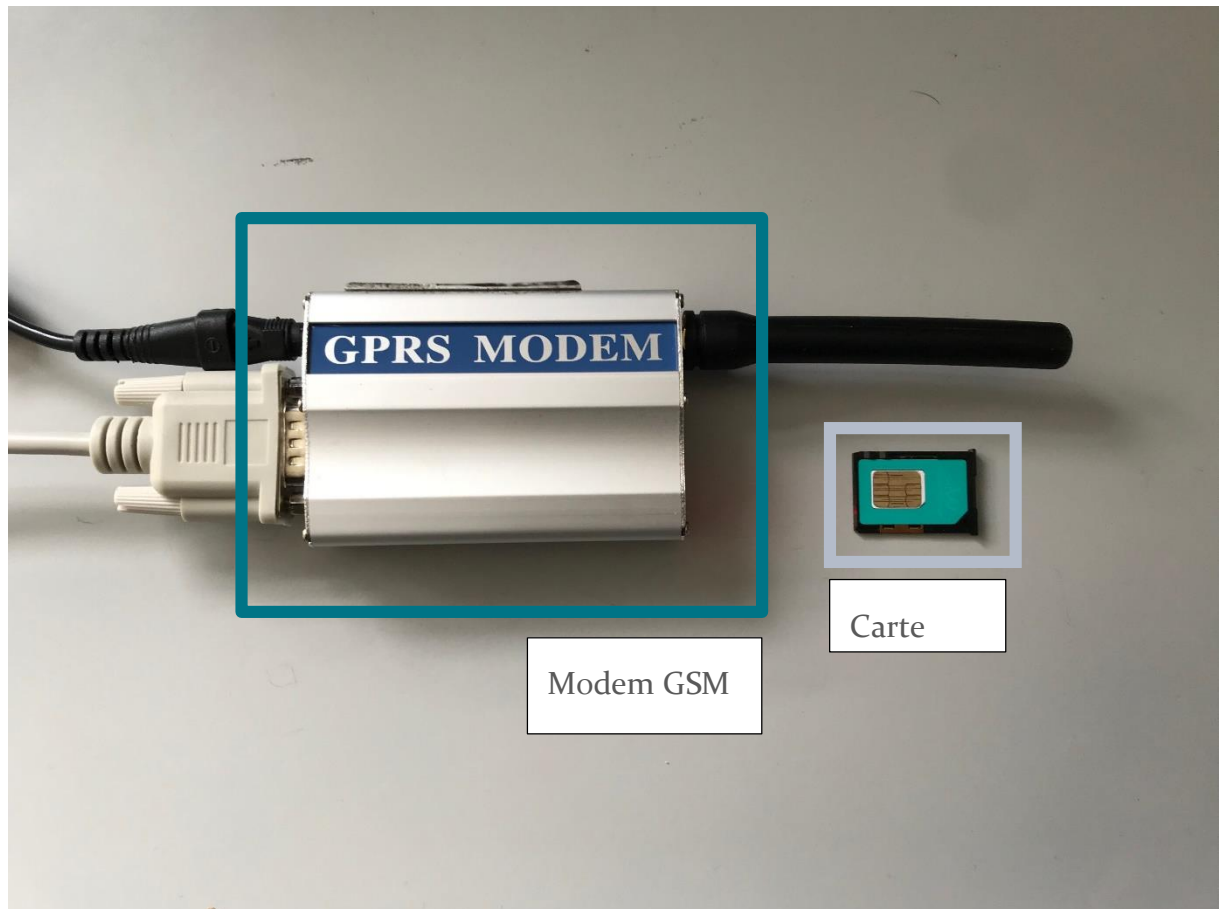
J'ai donc complété mon interface graphique afin d'y ajouté les nouvelles données qui sont mis à notre disposition.



Après avoir effectué toutes ces tests et ces vérifications, je rassemble le tout dans une seule fonction qui me permettra de comparer les valeurs reçues après avoir lu la carte avec la valeur correspondant à celle qui correspond au client qui a bien réservé la voiture, le tout sans avoir à devoir appuyer sur des boutons.

XI. Programme d'envoi des coordonnées GPS

A la fin de cette partie, je dois être en mesure de pouvoir transmettre par sms les coordonnées GPS qui ont été capturé par mon coéquipier. Un modem GSM ainsi qu'une puce téléphonique sont à ma disposition pour effectuer cette tâche.



Je crée alors le programme sur le logiciel QT.

Création de la classe CGSM :

Malheureusement, la version 4 de Qt ne fournit pas de classes pour gérer un port série. On va donc devoir utiliser une bibliothèque logicielle externe à Qt : la classe QextSerialPort. Cette bibliothèque permettra d'avoir accès au port disponible et propose plusieurs fonctionnalités pour pouvoir les utiliser.

On inclut alors cette bibliothèque directement à notre projet (QT += serialport et

`include(../src/qextserialport.pri)`. On peut alors utiliser toutes les fonctions disponibles.

Je commence alors mon programme par le paramétrage du port série selon les données fournies par la doc.

```
port = new QextSerialPort(QLatin1String("COM13"), QextSerialPort::Polling);

port->setBaudRate(BAUD115200);
port->setFlowControl(FLOW_OFF);
port->setParity(PAR_NONE);
port->setDataBits(DATA_8);
port->setStopBits(STOP_2);
```

Ensuite je crée une fonction « émettre () » qui transmettra directement le message que j'aurai mis en argument.

```
if (port == NULL || !port->isOpen())
{
    qDebug("erreur message non transmit");
}

else {
    port->write(message.toLatin1());
}
```

A chaque instruction que j'enverrais en modem GSM, une réponse me sera retourné, je crée alors une fonction recevoir() qui me permettra de les recevoir.

```
int numBytes;

numBytes = port->bytesAvailable();
if(numBytes > 1024)
    numBytes = 1024;

int i = port->read(buff, numBytes);
if (i != -1)
    buff[i] = '\0';
else
    buff[0] = '\0';
msg = QLatin1String(buff);
```

Pour communiquer avec le modem, il faut alors utiliser des commandes AT qui nous sont fournis dans la documentation. Il faut procéder à plusieurs étapes avant d'envoyer un sms.

Pour commencer, une fois que la carte SIM est bien insérer, on envoie au modem le code pin afin de pouvoir avoir accès au réseau téléphonique.

```
timer->stop();  
emettre("AT+CPIN=7518" "\x0D");  
recevoir();  
timer2->start(11000);  
ui->textEdit->setText("Connexion en cours...");
```

On émet la commande « AT+CPIN=7518 » qui entre le code PIN, on demande alors à recevoir la réponse avec la fonction « recevoir() » et on marque dans un TextEdit que la connexion est en cours.

J'expliquerais plus tard l'utilisation des timers

Une fois que le code pin est validé on peut alors passer à l'étape suivante qui consiste à vérifier que la carte SIM est maintenant bien valide.

```
timer2->stop();  
emettre("AT+CPIN?" "\x0D");  
recevoir();  
  
if(msg.contains("OK") || x == 1)  
{  
    x=1;  
    timer3->start(5000);  
    ui->textEdit->setText("Code PIN validé");  
    qDebug(msg.toLatin1());  
}  
else {  
    ui->textEdit->setText("Erreur Code PIN");  
    qDebug(msg.toLatin1());  
}  
  
}
```

C'est dans cette fonction qu'après avoir transmis le code pin précédemment, qu'on vérifie qu'il est bien valide et qu'on puisse passer à l'étape suivante, avec la commande « AT+CPIN=XXXX », le modem retournera « OK ». La fonction « contains() » permet de vérifier si les caractères entrés dans ces parenthèses (ici « OK ») sont contenues dans la variable (ici « msg »). Mais étant donné qu'on aura à effectuer plusieurs fois cette étape, à partir de la deuxième fois, le modem ne retournera plus « OK » à la commande même si le code pin est toujours bon, c'est pour cela que j'ai décidé de placer un « ou logique » dans mon if. En effet j'ai utilisé un entier qui prendra la valeur de 1 que si le code pin est validé la première fois ce qui permettra de passer cette étape la deuxième fois même malgré la réponse du modem.

Une fois le code pin et la carte SIM validé, la led du modem se mettra alors à clignoter.



Passé cette étape, on enregistre le GSM.

```
timer3->stop();
emettre("AT+CREG?" "\x0D");
recevoir();
if(msg.contains("READY"))
{
    timer4->start(5000);
    ui->textEdit->setText("Carte SIM validée");
    qDebug(msg.toLatin1());
}
else {
    ui->textEdit->setText("Erreur Carte SIM");
    qDebug(msg.toLatin1());
}
```

On passe ensuite en mode texte

```
timer4->stop();
emettre("AT+CMGF=1" "\x0D");
recevoir();
if(msg.contains("0,1"))
{
    timer5->start(5000);
    ui->textEdit->setText("GSM enregistré");
    qDebug(msg.toLatin1());
}
else {
    ui->textEdit->setText("Erreur Enregistrement refusé ");
    qDebug(msg.toLatin1());
}
```

On entre maintenant le numéro du destinataire du message.

```
timer5->stop();
emettre("AT+CMGS="+33634235939" "\x0D");
recevoir();
if(msg.contains("OK"))
{
    timer6->start(5000);
    ui->textEdit->setText("Passage en mode texte");
    qDebug(msg.toLatin1());
}
else
{
    ui->textEdit->setText("Erreur texte");
    qDebug(msg.toLatin1());
}
```

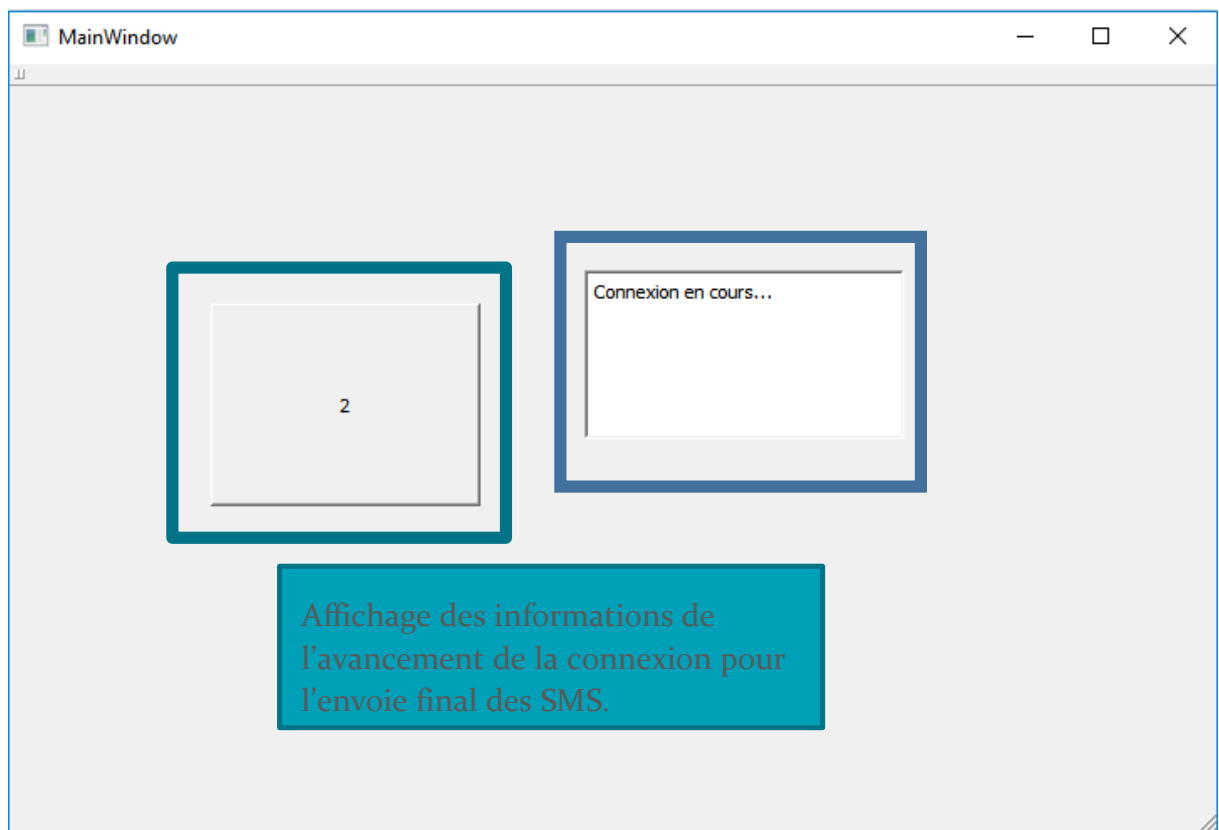
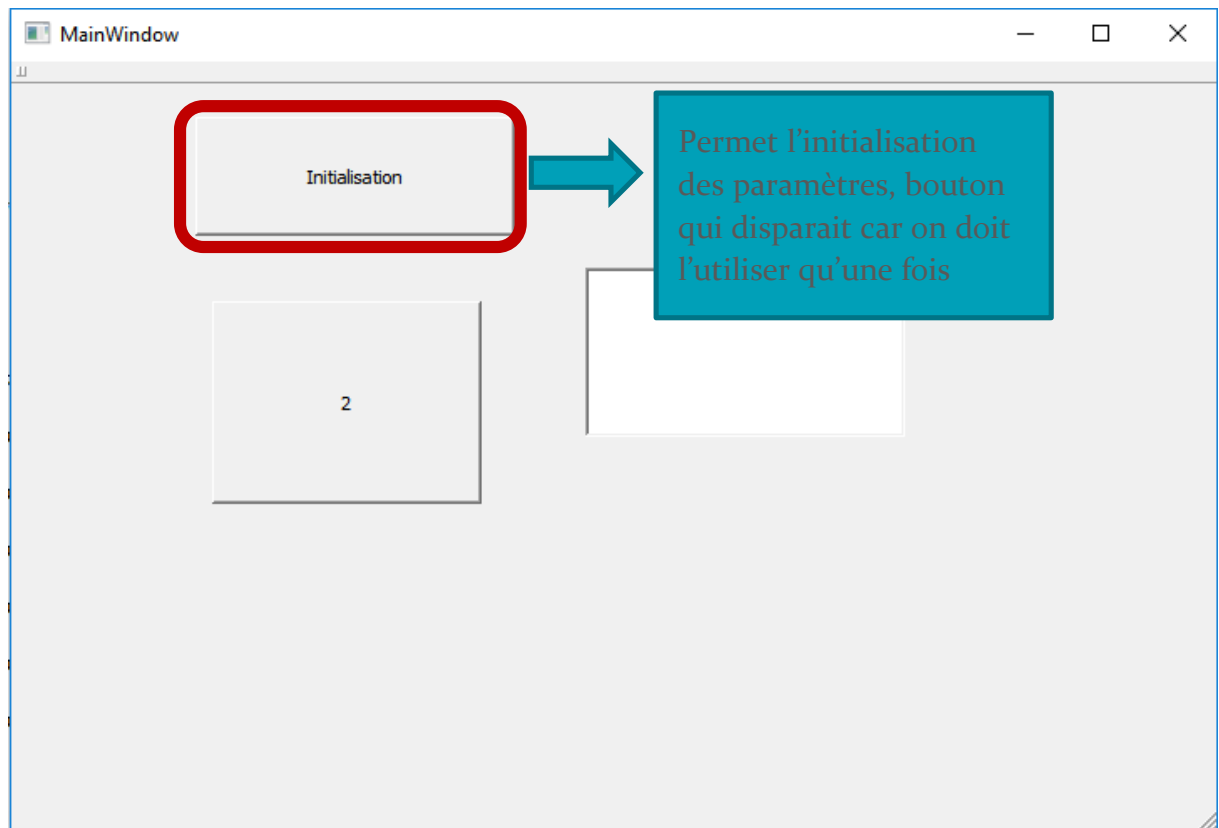
Et on envoie le message

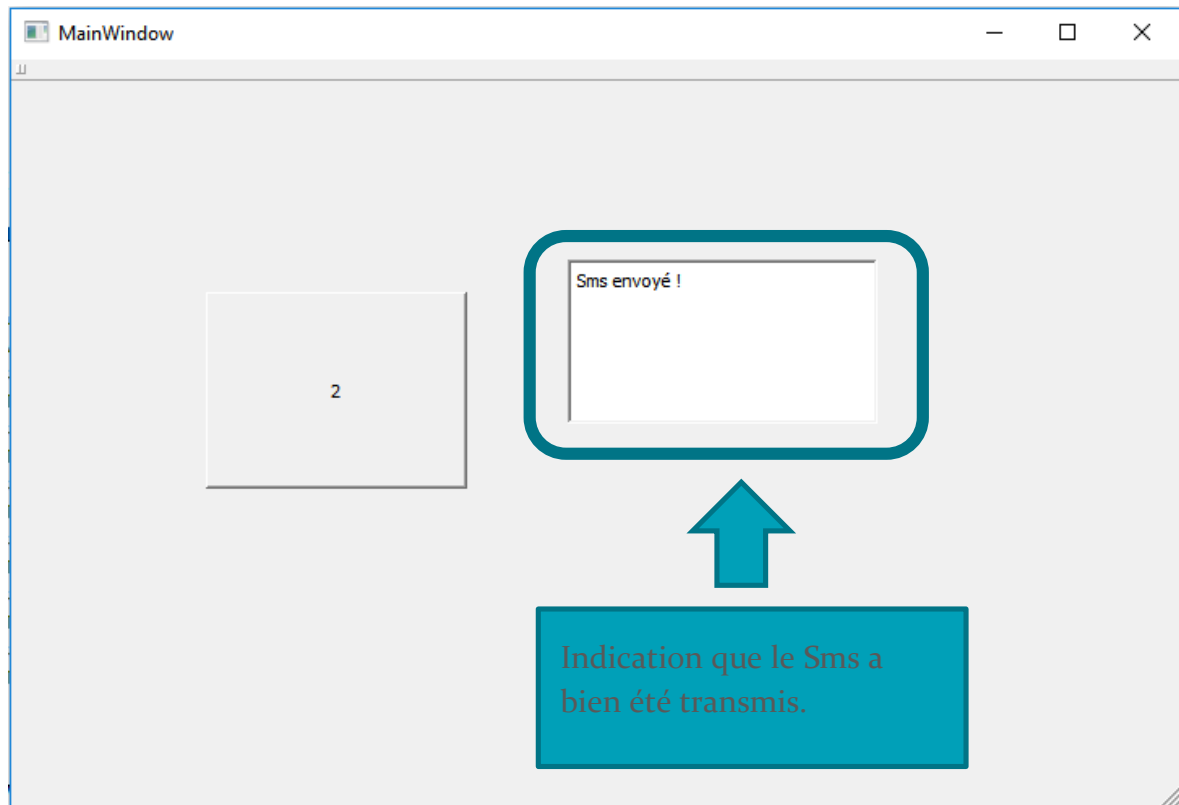
```
timer6->stop();
recevoir();
if(msg.contains(">"))
{
    emettre("94 94" "\x1A");
    ui->textEdit->setText("Sms envoyé !");
    qDebug(msg.toLatin1());
}
else
{
    ui->textEdit->setText("Erreur GMGS message non envoyé !");
    qDebug(msg.toLatin1());
}
```

J'ai ensuite utiliser plusieurs timer afin de pouvoir lancer en différer chaque fonction afin de laisser le temps au modem de me renvoyer la réponse et que je puisse la lire.

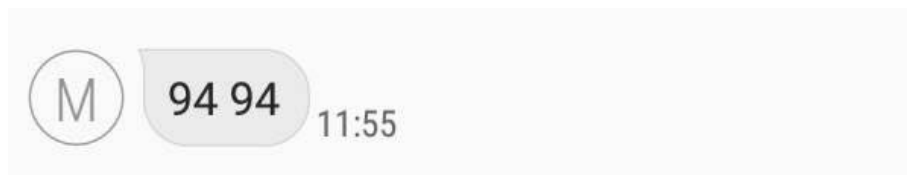
J'ai ensuite placer un timer que je n'arrête jamais afin d'envoyer selon un certain laps de temps les coordonnées GPS

```
timer=new QTimer(this);
connect(timer,SIGNAL(timeout()),this,SLOT(CPIN()));
timer2=new QTimer(this);
connect(timer2,SIGNAL(timeout()),this,SLOT(CPIN2()));
timer3=new QTimer(this);
connect(timer3,SIGNAL(timeout()),this,SLOT(CREG()));
timer4=new QTimer(this);
connect(timer4,SIGNAL(timeout()),this,SLOT(CMGF()));
timer5=new QTimer(this);
connect(timer5,SIGNAL(timeout()),this,SLOT(CMGS()));
timer6=new QTimer(this);
connect(timer6,SIGNAL(timeout()),this,SLOT(envoi()));
timer7=new QTimer(this);
connect(timer6,SIGNAL(timeout()),this,SLOT(transmission()));
```

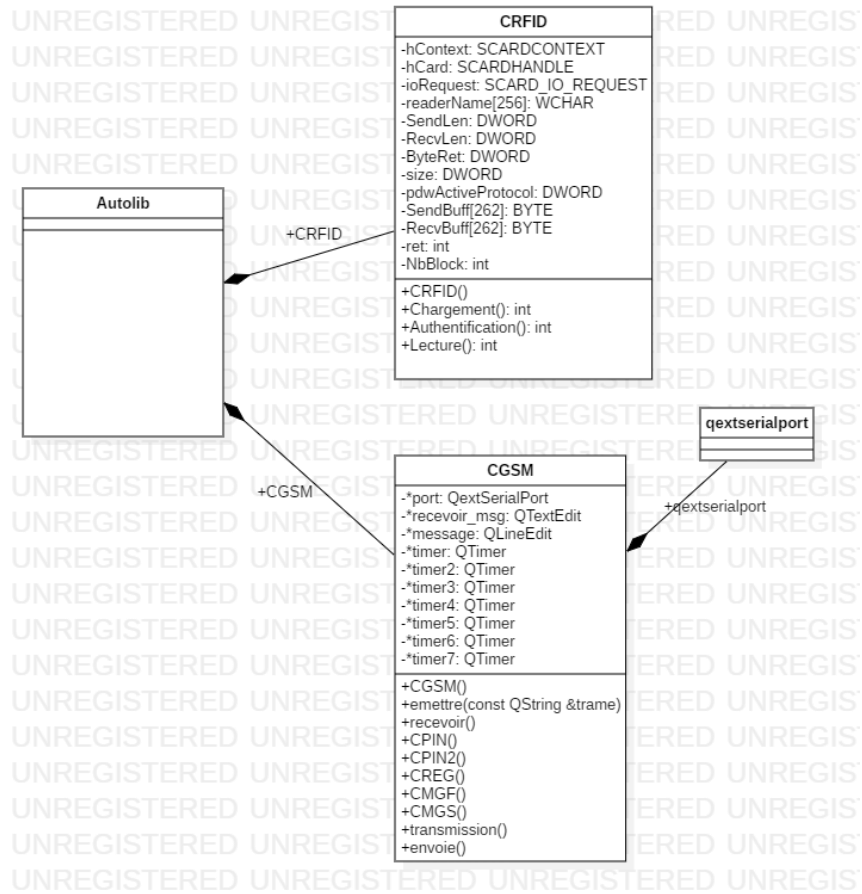





Voila un exemple d'un précédent test que j'avais effectué :



Voici donc mon diagramme de classe:



XII. Présentation du lecteur de carte

Présentation du matériel



- Lecteur : ACR122U
- Norme : RadioFrequencyIDentification/NearFieldCommunication
- Marque : ACS
- Connectique : USB (2.0)

Les cartes que nous utilisons sont des cartes mémoire disposant d'un numéro de série de 32 bits pré-encodé et d'un espace de stockage découpé en segments de données puis en blocs de données.

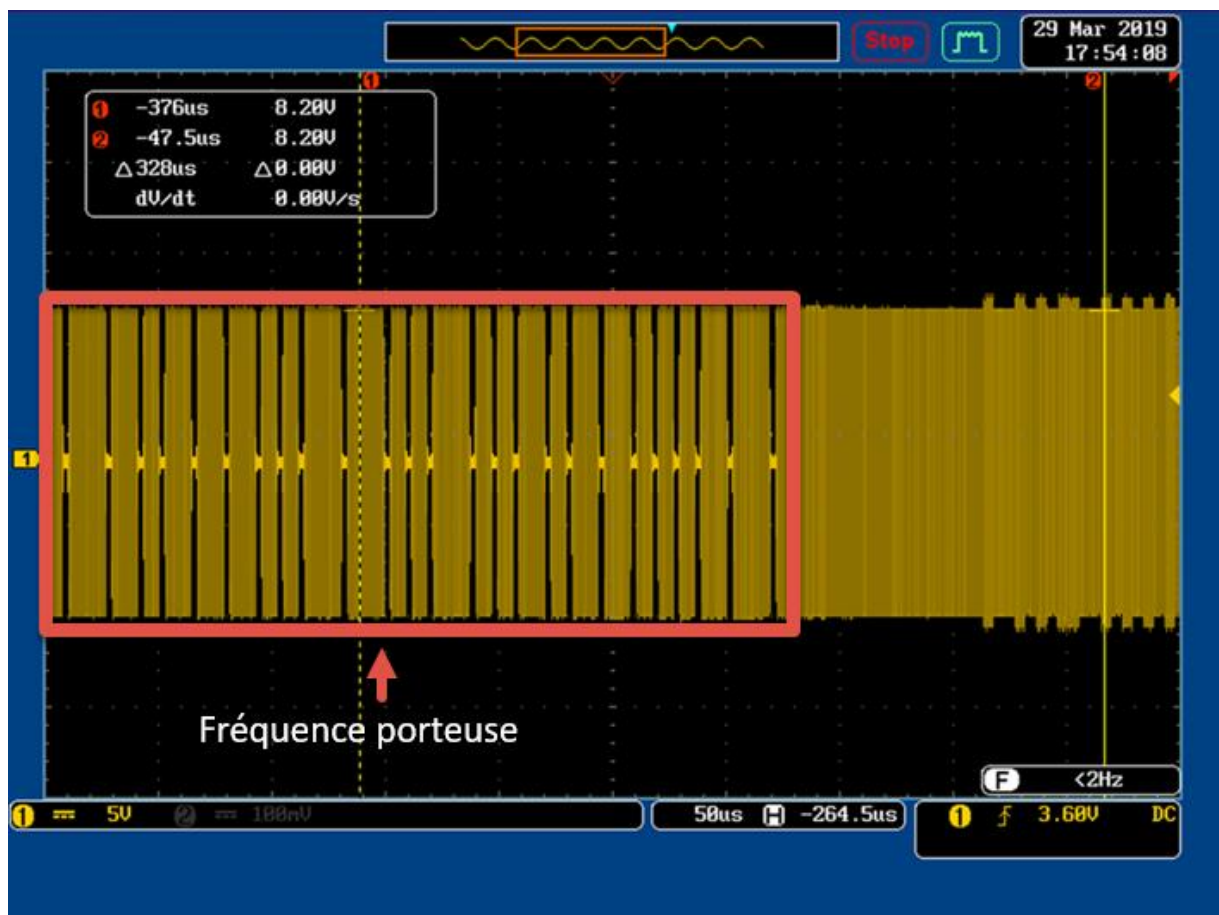
Ces cartes offrent 768 octets de stockage réparties sur 16 secteurs. Chaque secteur est composé de 3 blocs de 16 octets.

Le lecteur est dit actif tandis que les cartes sont passives c'est-à-dire que c'est seulement en étant au contact du lecteur qu'elles reçoivent suffisamment d'énergie pour pouvoir transmettre les données qu'elle contient.

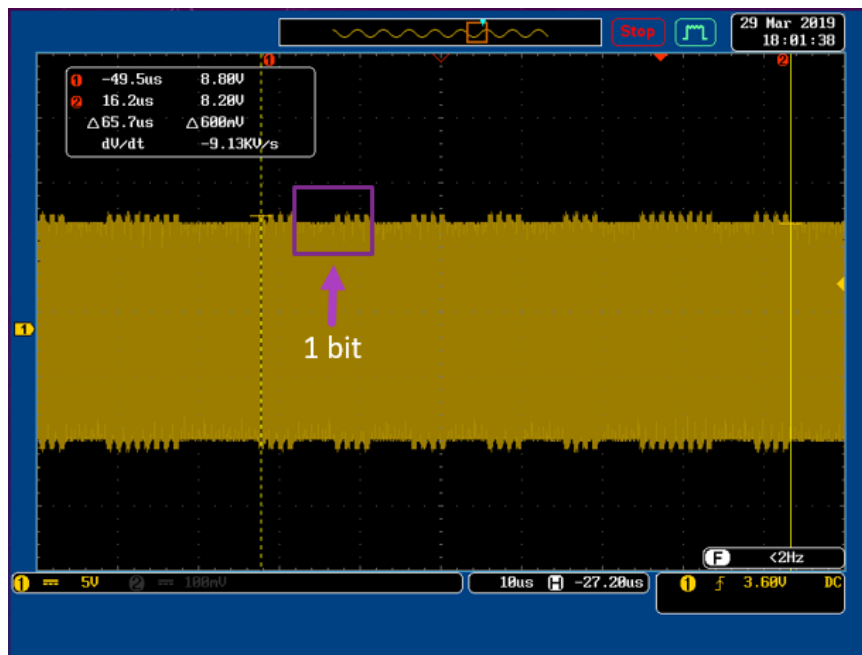
Lecteur RFID Haute Fréquence :

Type de fréquence	Fréquence	Taux de transfert
Basse fréquence	<135 khz	1kb/s
Haute fréquence	13.56 Mhz	106kb/s
Très haute fréquence	863 à 915 Mhz	28kb/s

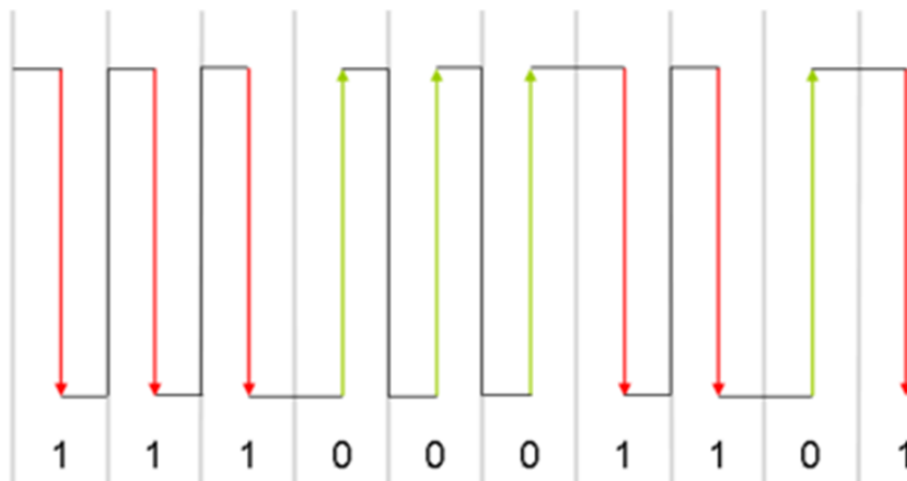
On peut voir ci-dessous la Fréquence porteuse transmise par le lecteur.



Et ci-dessous, on peut voir la réponse de la carte.



La méthode de codage qui est utiliser est le codage Manchester, représenté par :



Le passage de l'état haut à l'état bas représenté par un « 1 » logique et le passage de l'état bas à l'état haut représenté par un « 0 » logique.

On effectue alors le calcul visant à vérifier que le type de fréquence est bien une Haute Fréquence :

1 Tb correspond à l'intervalle entre le début et la fin d'un bit et 7 bit sont présent dans l'intervalle qu'on a sélectionnée.

On a donc $7 T_b = 65,7 \mu s$

$$1 T_b = \frac{65,7 \mu s}{7} = 9,4 \mu s$$

$$D = \frac{1}{T_b} = \frac{1}{9,4 \mu s} = 106,380 \text{ kb.s}$$

On a bien vérifié que le taux de transfert était équivalent.

Présentation du matériel :



- Modem GSM
- Connectique : USB à RS232 Port série 9 broches DB9
- langage de commandes : Commande AT
- vitesse de transmission : 115 200 bauds

Le RS-232 est une norme standardisant une voie de communication de type série sur trois fils minimum. Disponible sur presque tous les PC depuis 1981 jusqu'au milieu des années 2000, il est communément appelé le « port série ».

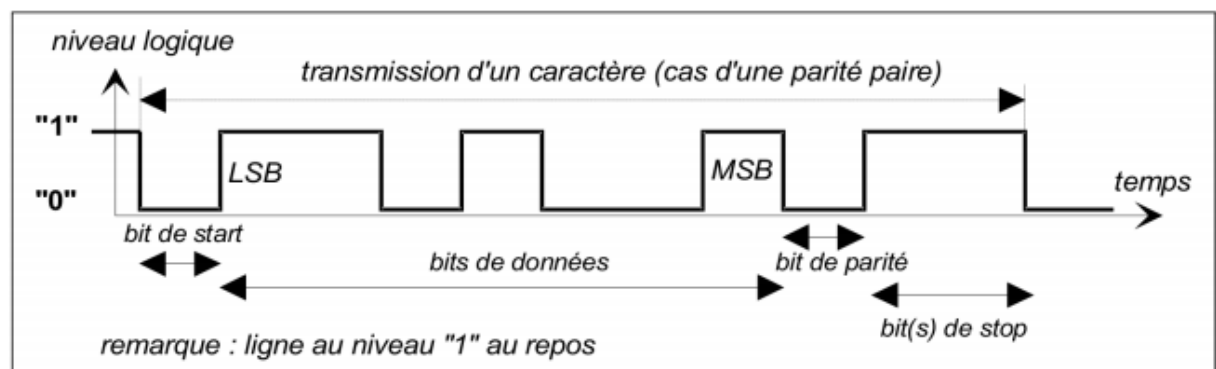
Sur les systèmes d'exploitation MS-DOS et Windows, les ports RS-232 sont désignés par les noms COM1, COM2, etc. Cela leur a valu le surnom de « ports COM ».

On utilise maintenant des adaptateurs USB/RS-232 car les PC ne disposent plus d'interfaces physiques RS-232. Cela revient à exploiter logiciellement un port série virtuel. Ces périphériques utilisent en réalité une transmission série avec

un convertisseur USB <-> RS-232. La prise en charge du périphérique est assurée par le système d'exploitation via un pilote de périphérique (driver).



trame série :



Pour établir une communication effective via un port série physique (RS-232) ou virtuel, il est nécessaire de définir le protocole utilisé : notamment, le débit de la transmission (en bits/s), le codage utilisé, le découpage en trame, etc. La norme RS-232 (couche Physique) laisse ces points libres, mais en pratique on utilise souvent des trames d'un caractère ainsi constituées :

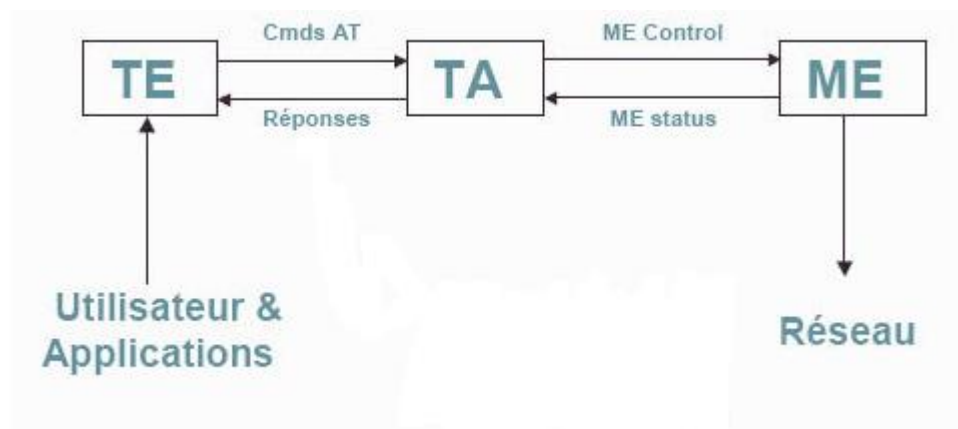
- 1 bit de départ (START)
- 7 à 8 bit de données
- 1 bit de parité optionnel (aucune, paire ou impaire)
- 1 ou plusieurs bits d'arrêt (STOP)

De nombreux périphériques séries expriment encore leur vitesse de transmission en bauds. Le baud est l'unité de mesure du nombre de symboles transmissibles par seconde. Dans le cas d'un signal modulé utilisé dans le domaine des télécommunications, le baud est l'unité de mesure de la rapidité de modulation. Il ne faut donc pas confondre le baud avec le bit par seconde (bit/s) car il est souvent possible de transmettre plusieurs bits par symbole. Si

on transmet un bit par symbole alors le baud et le bit par seconde (bit/s) sont équivalents.

Les commandes AT :

Les commandes AT sont définies dans la norme GSM 07.07 (pour les SMS cf. GSM 07.05). AT est l'abréviation de ATtention. Ces 2 caractères sont toujours présents pour commencer une ligne de commande sous forme de texte (codes ASCII). Les commandes permettent la gestion complète du mobile. Le schéma du fonctionnement est le suivant :



Le pc envoie les commandes AT via l'interface au modem GSM, lequel répondra.

Les commandes AT sont généralement suivies d'un <CR> (un retour chariot).

XIII. Système intégré à la station.



Voici la première image que le client verra à son arrivé. Le client devra positionner sa carte sur le lecteur et appuyer sur le bouton « START ».

A. Lecture et écriture de la carte.

La lecture de la carte se fait via L'ACR122U qui est un lecteur/programmeur RFID Haut Fréquence (13.56MHz).

Le lecteur reçoit de l'ordinateur des **commandes "APDUs"** pour communiquer avec les tags MIFARE et des **commandes "pseudo APDUs"** pour le contrôle de périphériques NFC.

Nous allons voir à présent la programmation d'applications de communication avec tags MIFARE et entre 2 lecteurs NFC qui utilise l'API C de Microsoft (**WINS CARD**), mais est transposable dans le principe pour la communication avec un autre type de lecteur compatible NFC ou en utilisant une autre API



Les commandes APDU :

Commande APDU						
En-tête				Données		
CLA	INS	P1	P2	Lc	Data	Le

CLA : Class de l'application.
 INS : code de l'instruction.
 P1-P2 : Paramètres.
 Lc : nombre d'octets du champ "Data".
 Le : nombre maximum d'octets attendus dans le champ "Data" de la réponse APDU.

Réponse APDU		
Données		Statut
Data	SW1	SW2

SW1, SW2 = 0x90 , 0x00 si pas d'erreur

a. La Bibliothèque.

L'application étant codé sous Qt Creator, nous devons ajouter dans le fichier .pro la ligne « LIBS += winscard.lib », placer les fichiers de l'archive winscard_lib.zip (des .dll).

libeay32.dll
 libmysql.dll
 ssleay32.dll

Enfin il faut aussi inclure le fichier « winscard.h » dans le code source.

b. Les variables utilisées.

```

SCARDCONTEXT hContext;
SCARDHANDLE hCard;

SCARD_IO_REQUEST ioRequest;
WCHAR readerName[256];
DWORD SendLen, RecvLen, ByteRet, size;
BYTE SendBuff[262], RecvBuff[262];
  
```

c. Structure d'une carte RFID

Sector	Block	Byte Number within a Block																Description
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
15	3	Key A				Access Bits				Key B								Sector Trailer 15
	2																	Data
	1																	Data
	0																	Data
14	3	Key A				Access Bits				Key B								Sector Trailer 14
	2																	Data
	1																	Data
	0																	Data
:	:																	
:	:																	
:	:																	
1	3	Key A				Access Bits				Key B								Sector Trailer 1
	2																	Data
	1																	Data
	0																	Data
0	3	Key A				Access Bits				Key B								Sector Trailer 0
	2																	Data
	1																	Data
	0																	Manufacturer Block

B. Exemple de commande APDU.a. Préparation de l'environnement.

```
int ret = SCardEstablishContext( SCARD_SCOPE_USER,NULL,NULL,&hContext );
if( ret != SCARD_S_SUCCESS ) .....
```

Cette commande devra etre executé au debut du programme afin de s'assurer que tout les drivers soit presents.

b. Détection du ou des lecteurs.

```
size = 256;
ret = SCardListReaders( hContext,NULL,readerName,&size );
if( ret != SCARD_S_SUCCESS ) .....
```

Une fois que le lecteur est branché nous pouvons nous assuré qu'il est bien détecté par le programme en réalisant cette commandes. En effet si le lecteur n'est pas détecté ou

non reconnue, nous pouvons afficher un message sur la fenêtre pour en avertir l'utilisateur.

c. Connexion à une carte.

```
ret=SCardConnect(hContext,readerName,SCARD_SHARE_SHARED,SCARD_PROTOCOL_T0/SCARD_PROTOCOL_T1,&hCard,&dwActProtocol );
```

```
if( ret != SCARD_S_SUCCESS ) .....
```

Ici nous allons pouvoir faire une détection de la carte en direct sur le programme. Un Timer pourra être réalisé pour détecter plusieurs fois si la carte est toujours connectée.

A l'inverse on peut faire un Timer avec cette commande (si la carte n'est pas présentée) pour rester en recherche de carte jusqu'à que la carte sera déposée sur le lecteur. Une fois la carte déposée, la détection de carte s'arrête et la suite du programme peut continuer.

Nous pouvons aussi déconnecter la carte avec la commande suivante :

```
ret = SCardDisconnect( hCard, SCARD_UNPOWER_CARD );
```

d. Envoie d'une commande au lecteur.

```
int ret = SCardTransmit(hCard, NULL ou &ioRequest, SendBuff, SendLen, NULL, RecvBuff, &RecvLen);
```

```
if( ret != SCARD_S_SUCCESS ) .....
```

argument 2 = NULL (Windows) ou &ioRequest (Linux)

SendBuff = tableau d'octets contenant la commande APDU

RecvBuff = tableau d'octets qui reçoit la réponse à la commande

SendLen = nombre d'octets de la commande,

RecvLen = nombre d'octets de la réponse

C. Les commandes APDU principales.

- **Chargement des clés d'authentification dans le lecteur** : il suffit de charger la clé par défaut à l'emplacement 0 (n°cle), c'est à dire les 6 octets FF FF FF FF FF FF

Cette clé servira alors pour l'authentification des blocs en type A ou type B.

On peut charger différentes clés pour authentifier différemment les différents blocs de la puce.

$APDU = 0xFF, 0x82, 0x00, n^{\circ}cle, 0x06, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF$

La réponse est sur 2 octets (SW1 et SW2)

- Authentification pour un bloc :

$APDU = 0xFF, 0x86, 0x00, 0x00, 0x05, 0x01, 0x00, n^{\circ}bloc, 0x60$ (typeA) ou $0x61$ (typeB), $n^{\circ}cle$

La réponse est sur 2 octets (SW1 et SW2)

Remarque : le n° de bloc est compris entre 0 et 63 (64 blocs de 16 octets = 1024)

Exemple : secteur 1 | bloc 0 $\rightarrow n^{\circ} bloc = 4$

- Lecture du contenu d'un bloc = 16 octets (page 12 de la doc de l'API) :

$APDU = 0xFF, 0xB0, 0x00, n^{\circ}bloc, 0x10$

La réponse est sur 18 octets : les 16 octets du bloc + SW1 et SW2

Il faut bien sur tester si les 2 derniers octets de la réponse valent 0x90 0x00 pour s'assurer d'une lecture correcte :if ((RecvBuff [RecvLen - 2] == 0x90) && (RecvBuff [RecvLen - 1] == 0x00))

- Ecriture du contenu d'un bloc = 16 octets (page 13 de la doc de l'API) :

$APDU = 0xFF, 0xD6, 0x00, n^{\circ}bloc, 0x10, \underline{x, x, x, x, x, x, x, x, x, x, x, x, x, x, x, x}, \underline{x, x}$ sont les 16 octets à écrire.

La réponse est sur 2 octets (SW1 et SW2).

D. Diagramme de classe.



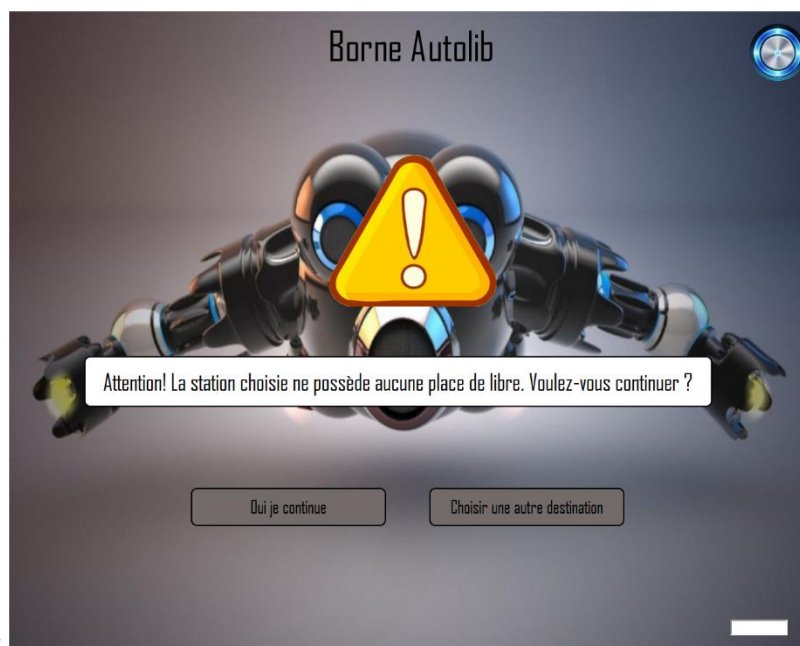
La classe « cardrfid » va nous permettre d'initialiser l'environnement, comme décrit précédemment, grâce à la fonction Initialisationcard(). Dans cette même fonction nous détectons aussi le lecteur et nous pouvons aussi récupérer son nom avec l'argument « readerName ».

La fonction « update » va être utilisé dans un Timer, elle sert à détecter la carte.

XIV. Communication avec la Base de données.

A. Intro

La base de données a pour but d'y insérer les informations nécessaire concernant les clients, les véhicules, les stations et des informations sur le trajet effectué par les véhicules. Nous allons aussi pouvoir récupérer ces informations à partir de notre programme pour qu'il agisse en conséquence.



Exemple :

B. Base de données

numero_badge	heure_de_depart	station_depart	station_arrivee	heure_arrivee	kilometrage
numero-vehicule	immatriculation	latitude	longitude	libre	kilometrage
1	BW-484-QX	0	0	1	25000
2	TP-363-XQ	0	0	1	64591
3	BM-242-BR	0	0	0	15643
4	IM-537-JK	0	0	0	64984

numero_badge	nom	code_secret	facture
1	Patissier	1234	1
2	Hamadouche	1111	1
3	Ben zina	2401	1

id-station	nom-station	nb-places	nb-vehicules	place_libre
3	Trocadero	7	6	1
2	Fosh	7	3	0
1	Bastille	7	7	5

C. Diagramme de classe.

Ici nous donnons un nom à notre base de données par exemple « db » en le déclarant comme un QSqlDatabase.

Nous déclarons aussi les différentes informations que nous allons utiliser en QString

On utilise la fonction « login », cette fonction définira l'adresse où se trouve la base de donnée pour que le programme sache où chercher la base, définira aussi le nom de la base à laquelle il faut se connecter et pour finir nous y insérons les Username et le Password. Dans cette même fonction nous testons aussi si la connexion à la base s'est bien effectuer.

Les fonctions clients, station et véhicules servent à faire des requêtes SQL afin de récupérer les informations dans la base.

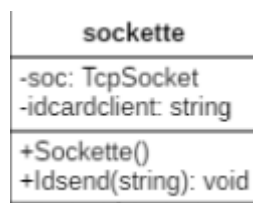
sql
-db: QSqlDatabase -numero_badge[]: QString -nom[]: QString -code_secret[]: QString -facture[]: QString -id_station[]: QString -nom-station[]: QString -nb_place[]: QString -nb_vehicules[]: QString -place_libre[]: QString -numero_vehicule[]: QString -immatriculation[]: QString -libre[]: QString -kilometrage[]: QString
+Sql() +login(): void +clients(QString): void +station(QString): void +vehicules(QString): void +getnumero_badge(int): QString +getnom(int): QString +getcode_secret(int): QString +getfacture(int): QString +getid_station(int): QString +getnom_station(int): QString +getnb_place(int): QString +getnb_vehicules(int): QString +getnumero_vehicule(int): QString +getimmatriculation(int): QString +getlibre(int): QString +getkilometrage(int): QString +getplace_libre(int): QString +getreservation(string): QString

XV. Client TCP

A. Introduction

Le client TCP a pour but d'assurer les communications entre la station et le véhicule afin que la station puisse transmettre l'ID du badge du client qui réserve le véhicule.

B. Diagramme de classe

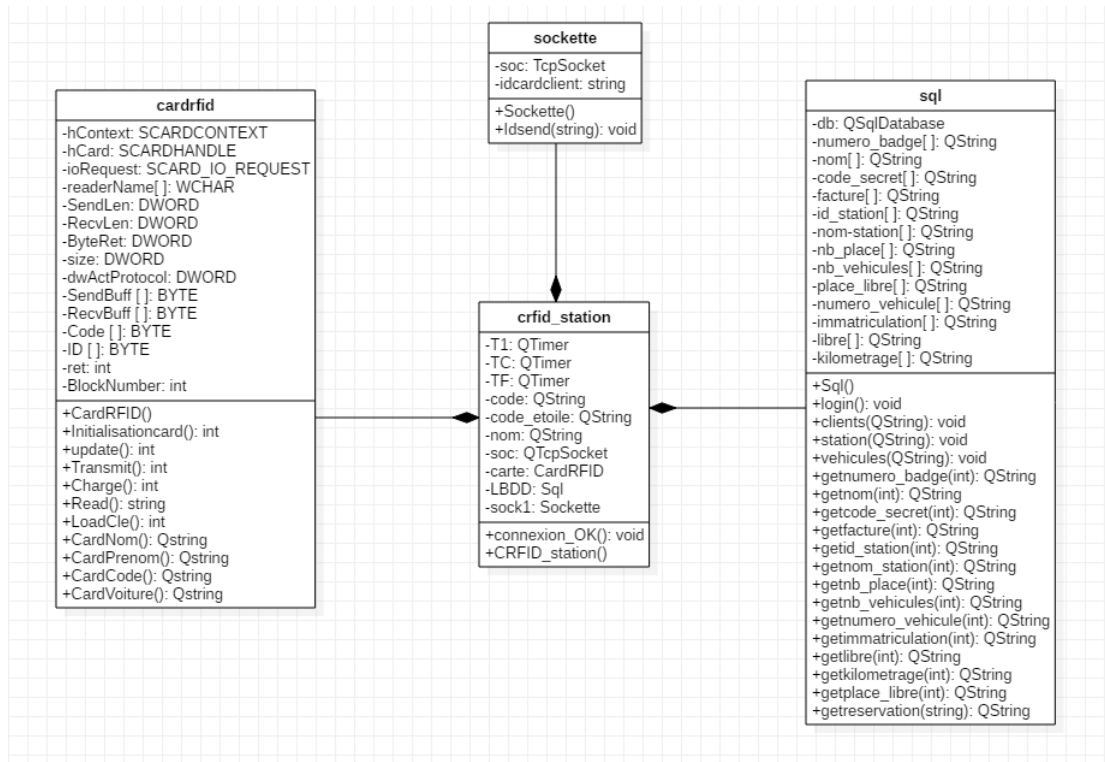


Idsend est une fonction permettant de se connecter au serveur en y insérant son adresse et son port.

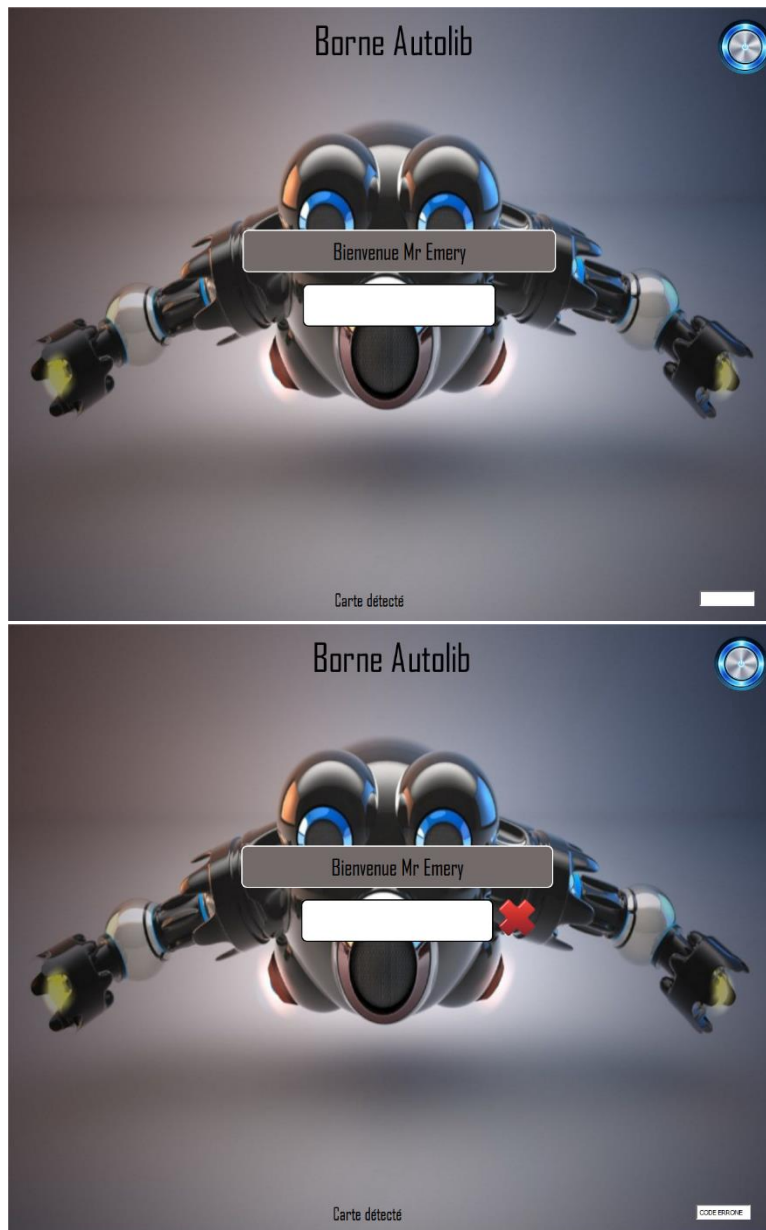
Il y a aussi la fonction « connexion_OK » où l'on insère le message à envoyer c'est-à-dire l'id du client.

XVI. Interface Homme-Machine Station.

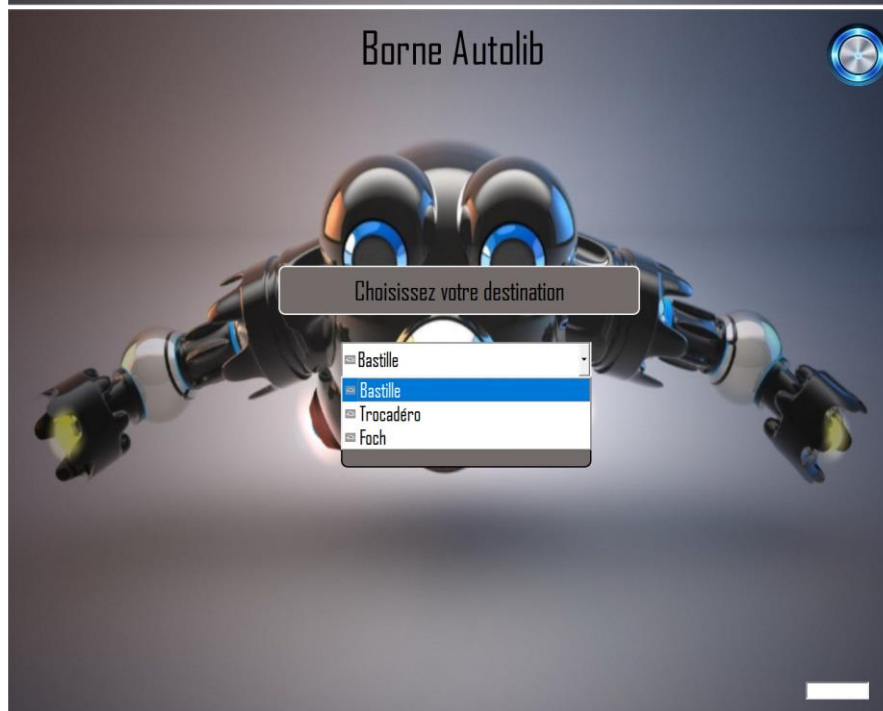
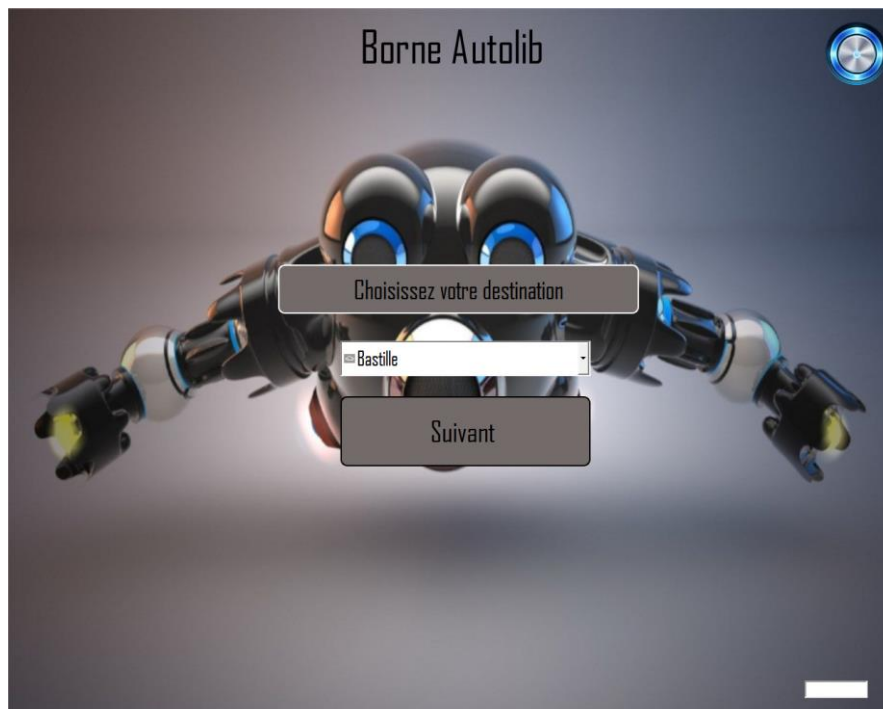
A. Diagramme de classe

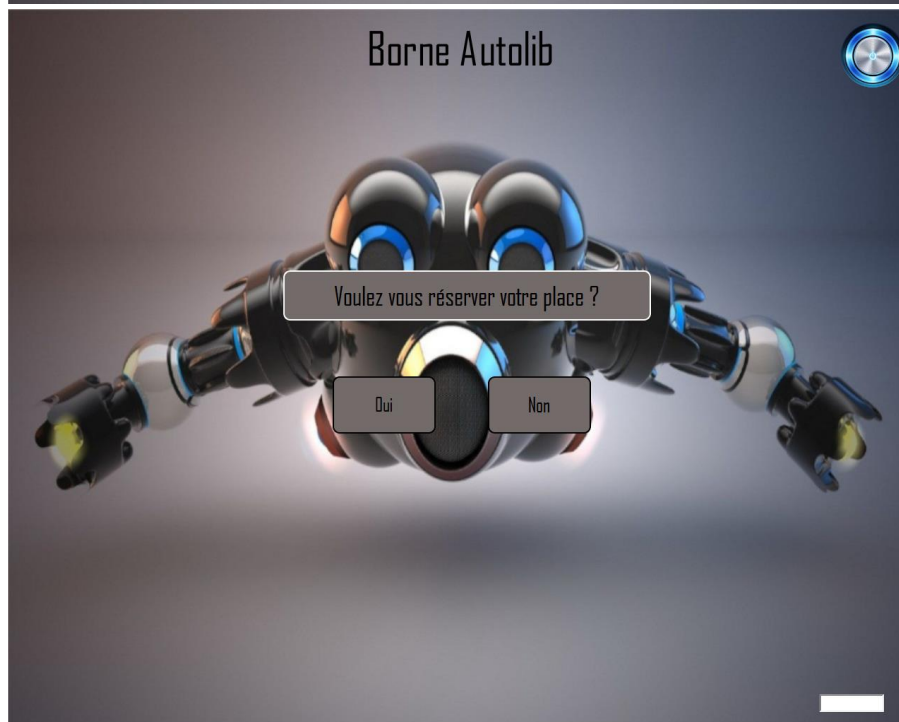
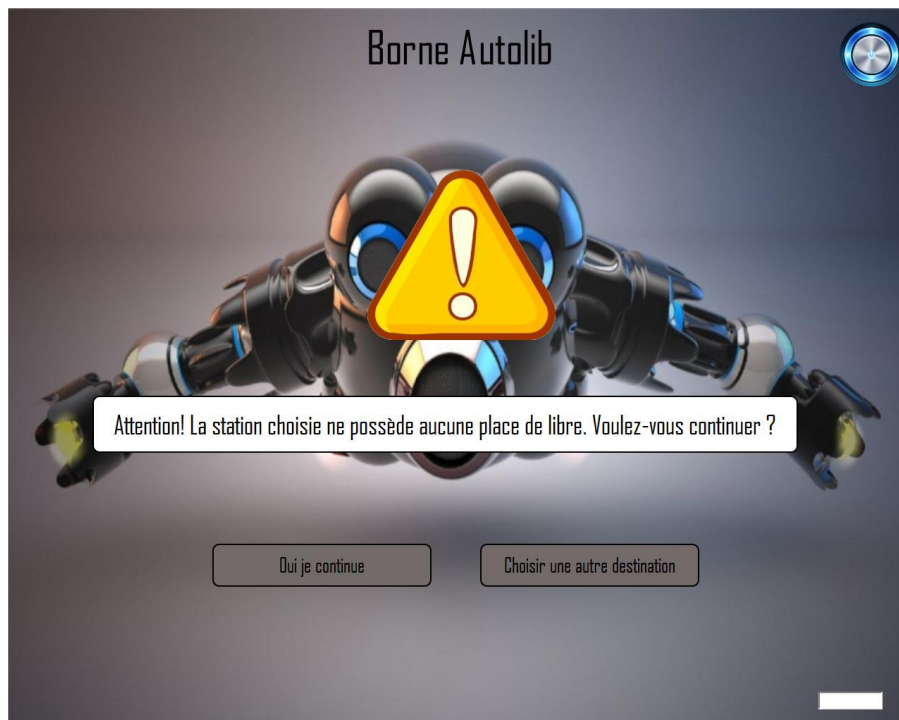


B. Interface graphique











XVII. Annexe

A. Codage de l'application embarquée (étudiant 1)

a. Classe CGPS

1. CGPS.h:

```
#ifndef CGPS_H
#define CGPS_H
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include "fcntl.h"
#include "errno.h"
#include "QString"
class CGPS {
private:
    char buffer [1000];
    int nb;
    int fd;
public:
    CGPS();
    QString trame();
    QString latitude();
    QString longitude();
    QString heure();
};
#endif // CGPS_H
```

2. CGPS.cpp:

```
#include "cgps.h"
#include "termios.h"
CGPS::CGPS()
{
    if((fd=open("/dev/ttyUSB0",O_RDONLY|O_NOCTTY))<0){ }
    else{
        struct termios tio;
        tcgetattr(fd,&tio);
        tio.c_cflag=B4800|CS8|CLOCAL|CREAD;
        tio.c_oflag=0;
        tio.c_lflag=0;
        tio.c_iflag |=IGNPAR;
        tio.c_lflag&=~(ICANON) ;
        tio.c_cc[VTIME]=0 ;
        tio.c_cc[VMIN]=1;
        tcflush(fd,TCIOFLUSH);
        tcsetattr(fd,TCSANOW,&tio);
    }
}
```

```

QString CGPS::trame(){
    nb=500;
    int q=read(fd,buffer,nb);
    QString Q=QString::number(q);
    QString sd;
    for(int i=0;i<500;i++){sd+=buffer[i];}
    std::string text=sd.toUtf8().constData();
    QString pk;
    int pos=text.find("$GPGLL");
    for(int j=pos;j<pos+50;j++){pk+=buffer[j];}
    std::string ok=pk.toUtf8().constData();
    int pos2=ok.find(",");
    QString sk;
    for(int k=pos2;k<pos2+11;k++){sk+=ok[k+1];} // Stockage de la latitude
    int pos3=ok.find("N,");
    QString skt;
    for(int l=pos3;l<pos3+12;l++){skt+=ok[l+2];} // Stockage de la longitude
    int pos4=ok.find("E");
    QString t1;
    for (int m=pos4;m<pos4+10;m++){t1+=ok[m+2];} // Stockage de l'heure
    QString position = sk + "\n" + skt + "\n" + t1;
    return position;
}

QString CGPS::latitude(){
    nb=500;
    int q=read(fd,buffer,nb);
    QString Q=QString::number(q);
    QString sd;
    for(int i=0;i<500;i++){sd+=buffer[i];}
    std::string text=sd.toUtf8().constData();
    QString pk;
    int pos=text.find("$GPGLL");
    for(int j=pos;j<pos+50;j++){pk+=buffer[j];}
    std::string ok=pk.toUtf8().constData();
    int pos2=ok.find(",");
    QString sk;
    for(int k=pos2;k<pos2+11;k++){sk+=ok[k+1];} // Stockage de la latitude
    return sk;
}

QString CGPS::longitude(){
    nb=500;
    int q=read(fd,buffer,nb);
    QString Q=QString::number(q);
    QString sd;
    for(int i=0;i<500;i++){sd+=buffer[i];}
    std::string text=sd.toUtf8().constData();
    QString pk;
    int pos=text.find("$GPGLL");

```

```
for(int j=pos;j<pos+50;j++){pk+=buffer[j];}
std::string ok=pk.toUtf8().constData();
int pos3=ok.find("N,");
QString skt;
for(int l=pos3;l<pos3+12;l++){skt+=ok[l+2];} // Stockage de la longitude
return skt;
}
QString CGPS::heure(){
nb=500;
int q=read(fd,buffer,nb);
QString Q=QString::number(q);
QString sd;
for(int i=0;i<500;i++){sd+=buffer[i];}
std::string text=sd.toUtf8().constData();
QString pk;
int pos=text.find("$GPGLL");
for(int j=pos;j<pos+50;j++){pk+=buffer[j];}
std::string ok=pk.toUtf8().constData();
int pos4=ok.find("E");
QString t1;
for (int m=pos4;m<pos4+10;m++){t1+=ok[m+2];} // Stockage de l'heure
return t1;
}
```

b. Classe CGPIO

1. CGPIO.h

```
#ifndef CGPIO_H
#define CGPIO_H
#include "piface.h"
#include "QString"
class CGPIO
{
public:
    CGPIO();
    void cable();
    bool carte(QString b, QString c);
};
#endif // CGPIO_H
```

2. CGPIO.cpp

```
#include "cgpio.h"
#include "piface.h"
CGPIO::CGPIO(){}
bool CGPIO::carte(QString b, QString c){
    pfio_init();
    QString malik= b;
    QString wesley= c;
    if (malik == wesley){
        pfio_digital_write(1,1);
        return true;
    }
    else {
        pfio_digital_write(3,1);
        return false;
    }
}
void CGPIO::cable(){
    pfio_init();
    pfio_digital_write(1,1);
    while(1){
        int num = pfio_read_input();
        if (num==4){ pfio_digital_write(5,1); }
        if (num==2){pfio_digital_write(5,0);}
        if (num==1){break;}
    }
}
```

c. Classe CGSERVEUR1. CGSERVEUR.h

```

#ifndef CGSERVEUR_H
#define CGSERVEUR_H
#include <QTcpServer>
#include <QTcpSocket>
#include <QtNetwork>
#include <QMainWindow>
class CGSERVEUR : public QObject {
    Q_OBJECT
public:
    CGSERVEUR();
public slots:
    void client_connexion();
    QString client_lecture();
    void creation_serveur();
private:
    QTcpServer *serveur;
    QList<QTcpSocket *> Clients;
};
#endif // CGSERVEUR_H

```

2. CGSERVEUR.cpp

```

#include "cgserveur.h"
CGSERVEUR::CGSERVEUR(){}
void CGSERVEUR::client_connexion( ) {
    QTcpSocket *clientConnection = serveur->nextPendingConnection ( ); // socket de
    service
    connect (clientConnection, SIGNAL (readyRead( )), this, SLOT(client_lecture())); //
    connexion du signal "readyRead( )" indiquant la réception de données en provenance
    du client au slot " client_lecture( )".
    Clients.append(clientConnection);
}
QString CGSERVEUR::client_lecture() {
    QString texte;
    foreach(QTcpSocket *client, Clients) {
        if (client->bytesAvailable()){
            QByteArray s = client->readAll(); // boucle de lecture des données reçues
            texte=s;
        }
    }
    return texte;
}
void CGSERVEUR::creation_serveur(){
    serveur=new QTcpServer;
    int a=5000;
    serveur->listen(QHostAddress::Any,a);
    connect (serveur, SIGNAL(newConnection()),this,SLOT(client_connexion())); }

```

d. Classe CIHM

1. CIHM.h

```
#ifndef CIHM _H
#define CIHM _H
#include "cgps.h"
#include "cgserveur.h"
#include "piface.h"
#include "cgpio.h"
#include <QMainWindow>
namespace Ui {
class MainWindow;
}
class MainWindow: public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow (QWidget *parent = 0);
    ~MainWindow();
    CGPS gps;
    CGSERVEUR *serv;
    CGPIO piface;
public slots:
    void on_pushButton_2_clicked();
    void on_pushButton_clicked();
    void on_pushButton_3_clicked();
    void client_connexion();
    QString client_lecture();
    void on_pushButton_4_clicked();
    void on_pushButton_5_clicked();
private:
    Ui::MainWindow *ui;
    QTcpServer *serveur;
    QList<QTcpSocket *> Clients;
    QByteArray y;
    QTcpSocket soc;
};
#endif // CIHM _H
```

2. CIHM.cpp

```
#include "cihm.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->label_4->setVisible(false);
    ui->pushButton_5->setVisible(false);
```

```

    ui->pushButton_3->setVisible(false);
    ui->pushButton_2->setVisible(false);
    ui->textBrowser->setVisible(false);
    ui->textBrowser_2->setVisible(false);
    ui->textBrowser_3->setVisible(false);
    ui->label->setVisible(false);
    ui->label_2->setVisible(false);
    ui->label_3->setVisible(false);
    ui->textEdit->setVisible(false);
    ui->pushButton_4->setVisible(false);

}
MainWindow::~MainWindow()
{ delete ui;}
void CIHM::on_pushButton_clicked() //bouton de démarrage
{
    if(ui->pushButton_2->isVisible()){
        ui->label_4->setVisible(false);
        ui->pushButton_3->setVisible(false);
        ui->pushButton_5->setVisible(false);
        ui->pushButton_2->setVisible(false);
        ui->textBrowser->setVisible(false);
        ui->textBrowser_2->setVisible(false);
        ui->textBrowser_3->setVisible(false);
        ui->label->setVisible(false);
        ui->label_2->setVisible(false);
        ui->label_3->setVisible(false);
    }
    else{
        ui->pushButton_3->setVisible(true);
        ui->pushButton_5->setVisible(true);
        ui->pushButton_2->setVisible(true);
        ui->textBrowser->setVisible(true);
        ui->textBrowser_2->setVisible(true);
        ui->textBrowser_3->setVisible(true);
        ui->label->setVisible(true);
        ui->label_2->setVisible(true);
        ui->label_3->setVisible(true);
    }
}
void MainWindow::on_pushButton_2_clicked() //bouton "afficher coordonnées"
{
    QString fnatic=gps.latitude();
    QString vitality=gps.longitude();
    QString misfits=gps.heure();
    ui->textBrowser->setText(fnatic);
    ui->textBrowser_3->setText(vitality);
    ui->textBrowser_2->setText(misfits);
}

```

```

}
void MainWindow::on_pushButton_4_clicked() //bouton "host"
{
    //serv->creation_serveur();
    serveur=new QTcpServer;
    int a=5000;
    serveur->listen(QHostAddress::Any,a);
    connect (serveur, SIGNAL(newConnection()),this,SLOT(client_connexion()));
}
QString CIHM::client_lecture() {
QString texte;
foreach(QTcpSocket *client, Clients){
    if (client->bytesAvailable()){
        y=client->readAll();
        ui->textEdit->append(y);
        texte=y;
    }
}
return texte;
}
void MainWindow::client_connexion(){
    //serv->client_connexion();

    QTcpSocket *clientConnection =serveur->nextPendingConnection();
    connect (clientConnection,SIGNAL(readyRead()),this,SLOT(client_lecture()));
    Clients.append(clientConnection);
}
void MainWindow::on_pushButton_3_clicked() //bouton "serveur"
{
    if(ui->pushButton_4->isVisible()){
        ui->pushButton_4->setVisible(false);
        ui->label_4->setVisible(false);
        ui->textEdit->setVisible(false);
    }
    else{
        ui->label_4->setVisible(true);
        ui->pushButton_4->setVisible(true);
        ui->textEdit->setVisible(true);
    }
}
void MainWindow::on_pushButton_5_clicked() // bouton "piface"
{
    QString a =""; //client_lecture();
    QString b ="";
    piface.carte(a,b);
    bool c =piface.carte(a,b);
    if (c){piface.cable();}
}

```


B. Codage de l'application embarquée (étudiant 2)

a. Classe NFC

Crfid.h :

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include "winscard.h"
namespace Ui {
    class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    SCARDCONTEXT      hContext;
    SCARDHANDLE        hCard;
    SCARD_IO_REQUEST   ioRequest;
    WCHAR              readerName[256];
    DWORD              SendLen, RecvLen, ByteRet,
size, pdwActiveProtocol;
    BYTE              SendBuff[262], RecvBuff[262];
    void Lecture();
    void Authentification();
private:
    Ui::MainWindow *ui;
    QTimer *timer;
    int ret;
    int NbBlock;
private slots:
    void on_pushButton_2_clicked();
    void on_pushButton_clicked();
    void update();
};
#endif // MAINWINDOW_H
```

Crfid.cpp :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "QTimer"
#include "winscard.h"
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    timer=new QTimer(this);
    connect(timer,SIGNAL(timeout()),this,SLOT(update()));
    timer->start(500);
}
```

```

MainWindow::~MainWindow()
{
}

void MainWindow::update()
{
    int ret = SCardEstablishContext(
        SCARD_SCOPE_USER, NULL, NULL, &hContext );
    if( ret != SCARD_S_SUCCESS ){ui->textEdit->setText("Erreur, le
driver n'est pas installe.");}
    else{ui->textEdit->setText("driver installer.");
        size = 256;
        ret = SCardListReaders( hContext, NULL, readerName, &size );
        if( ret != SCARD_S_SUCCESS ){ui->textEdit_2->setText("Erreur,
le lecteur n'est pas detecter.");}
        else{ui->textEdit_2->setText("lecteur detecter.");
            //WCHAR *p = readerName; while (*p) p++; p++;
            QString a=QString::fromWCharArray(readerName);
            ui->textEdit_4->setText(a);

ret=SCardConnect(hContext, readerName, SCARD_SHARE_SHARED, SCARD_PROTOCO
L_T0|SCARD_PROTOCOL_T1, &hCard, &pdwActiveProtocol);
            if( ret != SCARD_S_SUCCESS ){ui->textEdit_3-
>setText("Erreur, le carte n'est pas detecter.");}
            else{ui->textEdit_3->setText("carte detecter.");}
        }
    }
}

void MainWindow::on_pushButton_clicked()
{
    SendBuff[0] = 0xFF;
    SendBuff[1] = 0x82;
    SendBuff[2] = 0x00;
    SendBuff[3] = 0;
    SendBuff[4] = 0x06;
    SendBuff[5] = 0xFF;
    SendBuff[6] = 0xFF;
    SendBuff[7] = 0xFF;
    SendBuff[8] = 0xFF;
    SendBuff[9] = 0xFF;
    SendBuff[10] = 0xFF;
    RecvLen=2;
    int ret = SCardTransmit(hCard, NULL , SendBuff, 11, NULL,
RecvBuff, &RecvLen);
    if ( (RecvBuff[RecvLen - 2]== 0x90) && (RecvBuff[RecvLen - 1]==
0x00) ){ui->textEdit_5->setText("c'est ok");}
    else{ui->textEdit_5->setText("erreur chakal.");}
}

void MainWindow::Authentication() {
    for (int i=0;i<262;i++){
        SendBuff[i] = 0;
    }
    //Chargement du block
    SendBuff[0] = 0xFF;
    SendBuff[1] = 0x86;
    SendBuff[2] = 0x00;
    SendBuff[3] = 0x00;
}

```

```

        SendBuff[4] = 0x05;
        SendBuff[5] = 0x01;
        SendBuff[6] = 0x00;
        SendBuff[7] = NbBlock; // n° de bloc
        SendBuff[8] = 0x60; //type A    sinon type B = 0x61
        SendBuff[9] = 0;
        SendLen = 10;
        RecvLen = 2;
        int ret = SCardTransmit(hCard, NULL, SendBuff, SendLen, NULL,
RecvBuff, &RecvLen);
    }
    void MainWindow::Lecture() {
        for (int i=0;i<262;i++){
            SendBuff[i] = 0;
        }
        QString a="";
        SendLen = 5;
        RecvLen = 18;
        SendBuff[0]=0xFF;
        SendBuff[1]=0xB0;
        SendBuff[2]=0x00;
        SendBuff[3]=NbBlock;
        SendBuff[4]=0x10;
        int ret = SCardTransmit(hCard, NULL, SendBuff, SendLen, NULL,
RecvBuff, &RecvLen);
        if ( (RecvBuff[RecvLen - 2]== 0x90) && (RecvBuff[RecvLen - 1]==
0x00)){
            ui->textEdit_6->setText("Lecture du bloc réussi ");
            for (int i=0;i<16;i++)
            {
                a += RecvBuff[i];
            }
            ui->textEdit_6->setText(a);
        }
        else
        {
            ui->textEdit_6->setText("Lecture Bloc Impossible") ;
        }
    }
    void MainWindow::on_pushButton_2_clicked()
    {
        Authentification();
        NbBlock = 0x08;
        Lecture();
    }

```

b. Classe GSM

```

Gsm.h : #ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include "qextserialport.h"
#include <QWidget>
class QLineEdit;
class QTextEdit;
class QextSerialPort;

```

```

namespace Ui {
    class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private:
    Ui::MainWindow *ui;
    QextSerialPort *port;
    QLineEdit *message;
    QTextEdit *received_msg;
    char buff[1024];
    QString msg;
    int x;
    QTimer *timer;
    QTimer *timer2;
    QTimer *timer3;
    QTimer *timer4;
    QTimer *timer5;
    QTimer *timer6;
    QTimer *timer7;
private slots:
    void initialisation();
    void emettre(const QString &trame);
    void recevoir();
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();
    void CPIN();
    void CPIN2();
    void CREG();
    void CMGF();
    void CMGS();
    void transmission();
    void envoie();
};
#endif // MAINWINDOW_H

Gsm.cpp : #include "mainwindow.h"
#include "ui_mainwindow.h"
#include "qextserialport.h"
#include <QTimer>
#include <QLineEdit>
#include <QTextEdit>
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    timer=new QTimer(this);
    connect(timer,SIGNAL(timeout()),this,SLOT(CPIN()));
    timer2=new QTimer(this);
    connect(timer2,SIGNAL(timeout()),this,SLOT(CPIN2()));

```

```

        timer3=new QTimer(this);
        connect(timer3,SIGNAL(timeout()),this,SLOT(CREG()));
        timer4=new QTimer(this);
        connect(timer4,SIGNAL(timeout()),this,SLOT(CMGF()));
        timer5=new QTimer(this);
        connect(timer5,SIGNAL(timeout()),this,SLOT(CMGS()));
        timer6=new QTimer(this);
        connect(timer6,SIGNAL(timeout()),this,SLOT(envoi()));
        timer7=new QTimer(this);
        connect(timer6,SIGNAL(timeout()),this,SLOT(transmission()));
    }
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::initialisation()
{
    port = new QextSerialPort(QLatin1String("/dev/ttyS0"),
QextSerialPort::Polling);
    port = new QextSerialPort(QLatin1String("COM13"),
QextSerialPort::Polling);
    port->setBaudRate(BAUD115200);
    port->setFlowControl(FLOW_OFF);
    port->setParity(PAR_NONE);
    port->setDataBits(DATA_8);
    port->setStopBits(STOP_2);
    if (port->open(QIODevice::ReadWrite)) {
        qDebug("Connecté");
    } else {
        qDebug("Erreur");
    }
}
void MainWindow::emettre(const QString &message)
{
    if (port == NULL || !port->isOpen())
    {
        qDebug("erreur message non transmit");
    }
    else {
        port->write(message.toLatin1());
    }
}
void MainWindow::recevoir()
{
    int numBytes;
    numBytes = port->bytesAvailable();
    if(numBytes > 1024)
        numBytes = 1024;
    int i = port->read(buff, numBytes);
    if (i != -1)
        buff[i] = '\0';
    else
        buff[0] = '\0';
    msg = QLatin1String(buff);
}
void MainWindow::on_pushButton_clicked()
{

```

```
        initialisation();
        ui->pushButton->setVisible(false);
    }
    void MainWindow::on_pushButton_2_clicked()
    {
        transmission();
        timer7->start(300000);
    }
    void MainWindow::transmission()
    {
        timer->start(2000);
    }
    void MainWindow::CPIN()
    {
        timer->stop();
        emettre("AT+CPIN=7518" "\x0D");
        recevoir();
        timer2->start(11000);
        ui->textEdit->setText("Connexion en cours...");
    }
    void MainWindow::CPIN2()
    {
        timer2->stop();
        emettre("AT+CPIN?" "\x0D");
        recevoir();
        if(msg.contains("OK") || x == 1)
        {
            x=1;
            timer3->start(5000);
            ui->textEdit->setText("Code PIN validé");
            qDebug(msg.toLatin1());
        }
        else {
            ui->textEdit->setText("Erreur Code PIN");
            qDebug(msg.toLatin1());
        }
    }
    void MainWindow::CREG()
    {
        timer3->stop();
        emettre("AT+CREG?" "\x0D");
        recevoir();
        if(msg.contains("READY"))
        {
            timer4->start(5000);
            ui->textEdit->setText("Carte SIM validée");
            qDebug(msg.toLatin1());
        }
        else {
            ui->textEdit->setText("Erreur Carte SIM");
            qDebug(msg.toLatin1());
        }
    }
    void MainWindow::CMGF()
    {
        timer4->stop();
        emettre("AT+CMGF=1" "\x0D");
```

```
recevoir();
if(msg.contains("0,1"))
{
    timer5->start(5000);
    ui->textEdit->setText("GSM enregistré");
    qDebug(msg.toLatin1());
}
else {
    ui->textEdit->setText("Erreur Enregistrement refusé ");
    qDebug(msg.toLatin1());
}
}
void MainWindow::CMGS()
{
    timer5->stop();
    emettre("AT+CMGS="+33634235939" "\x0D");
    recevoir();
    if(msg.contains("OK"))
    {
        timer6->start(5000);
        ui->textEdit->setText("Passage en mode texte");
        qDebug(msg.toLatin1());
    }
    else
    {
        ui->textEdit->setText("Erreur texte");
        qDebug(msg.toLatin1());
    }
}
void MainWindow::envoie()
{
    timer6->stop();
    recevoir();
    if(msg.contains(">"))
    {
        emettre("94 94" "\x1A");
        ui->textEdit->setText("Sms envoyé !");
        qDebug(msg.toLatin1());
    }
    else
    {
        ui->textEdit->setText("Erreur GMGS message non envoyé !");
        qDebug(msg.toLatin1());
    }
}
```

C. Sources

https://www.framboise314.fr/wp-content/uploads/2015/11/2014_06_programmez_175_piface_scratch.pdf
<http://rexpfio.sourceforge.net/doc/index.html#robo5>

D. Codage de l'application de la station

1. crfid_station.cpp :

```

2. #include "crfid_station.h"
3. #include "ui_crfid_station.h"
4. #include "sql.h"
5. #include "sockette.h"
6. #include <QtSql>
7. #include <QMessageBox>
8. #include <QKeyEvent>
9. CRFID_station::CRFID_station(QWidget *parent) :
10.     QMainWindow(parent),
11.     ui(new Ui::CRFID_station)
12. {
13.     ui->setupUi(this);
14.     carte.Initialisationcard();
15.     T1 = new QTimer(this);
16.     //TIMER DETECTION DE LA CARTE
17.     connect(T1, SIGNAL(timeout()), this, SLOT(update()));
18.     TC = new QTimer(this);
19.     //TIMER CODE
20.     connect(TC, SIGNAL(timeout()), this, SLOT(codec()));
21.     TF = new QTimer(this);
22.     //TIMER POUR AFFICHAGE EN FIN DE PROGRAMME
23.     connect(TF, SIGNAL(timeout()), this, SLOT(afficheF()));
24.     LBDD.login();
25.     setStyleSheet("QMainWindow{background-
26.         image:url(C:/Users/SN2-GR1/Desktop/CRFID-
27.         B/RESSOURCES/FOND.jpg);}"); //IMAGE DE FOND
28.     /*LE MENU DEROUlant*/
29.     ui->comboBox->hide();
30.     /*LES LABELS*/
31.     ui->label_setcode->hide();
32.     ui->label_indicator->hide();
33.     ui->label_X->hide();
34.     ui->label_V->hide();
35.     ui->label_Recapitulatif->hide();
36.     ui->label_NoPlace->hide();
37.     ui->label_Warning->hide();
38.     /*LES BOUTONS*/
39.     ui->pushButton_refresh->hide();
40.     ui->pushButton_Next->hide();
41.     ui->pushButton_YES->hide();
42.     ui->pushButton_4->hide();
43.     ui->pushButton_5->hide();
44.     ui->pushButton_6->hide();
45.     ui->pushButton_7->hide();
46.     ui->pushButton_8->hide();
47.     ui->pushButton_9->hide();
48.     ui->pushButton_new_destination->hide();
49.     ui->textBrowser_info->hide();
50. }
51. CRFID_station::~CRFID_station()
52. {
53.     delete ui;
54. }
55. void CRFID_station::on_pushButton_clicked() //BUTTON EXIT

```

```

51. {
52.     close();
53.     qDebug() << "Programme fermé";
54. }
55. void CRFID_station::on_pushButton_start_clicked()
56. {
57.     TC->start(100);
58.     T1->start();
59.     ui->pushButton_start->hide();
60.     ui->label_First_indicator->hide();
61.     ui->label_setcode->show();
62.     ui->label_indicator->show();
63.     ui->pushButton_9->hide();
64.     ui->label_Recapitulatif->hide();
65.     ui->textBrowser_info->show();
66.     if( carte.update()==0) {
67.         ui->pushButton_refresh->show();
68.         ui->label_Cardconnected->setText("Carte absente ou
non reconnue...");
69.         ui->label_indicator->hide();
70.         ui->label_setcode->hide();
71.     };
72.     else {ui->label_Cardconnected->setText("Carte détectée");
73.         ui->label_indicator->setText("Bienvenue Mr
"+carte.CardNom());
74.         qDebug() << "Le client est Mr "+carte.CardNom();
75.         T1->stop();}
76. }
77. void CRFID_station::on_pushButton_refresh_clicked()
78. {
79.     if(carte.update()==0) {
80.         ui->pushButton_refresh->show();
81.         ui->label_Cardconnected->setText("Carte absente ou
non reconnue...");return;
82.     }
83.     else {ui->label_Cardconnected->setText("carte présente");
84.         ui->pushButton_refresh->hide();
85.         on_pushButton_start_clicked();
86.         T1->stop();}
87. }
88. void CRFID_station::codec() {
89.     ui->label_setcode->setText(code_etoile);
90. }
91. void CRFID_station::keyPressEvent( QKeyEvent *e )
92. {
93.     if (e->key()==Qt::Key_0){if
(code.size()<4){code+="0";code_etoile+="*";}}
94.     if (e->key()==Qt::Key_1){if
(code.size()<4){code+="1";code_etoile+="*";}}
95.     if (e->key()==Qt::Key_2){if
(code.size()<4){code+="2";code_etoile+="*";}}
96.     if (e->key()==Qt::Key_3){if
(code.size()<4){code+="3";code_etoile+="*";}}
97.     if (e->key()==Qt::Key_4){if
(code.size()<4){code+="4";code_etoile+="*";}}
98.     if (e->key()==Qt::Key_5){if
(code.size()<4){code+="5";code_etoile+="*";}}

```

```

99.         if (e->key()==Qt::Key_6){if
        (code.size()<4){code+="6";code_etoile+="*";}}
100.        if (e->key()==Qt::Key_7){if
        (code.size()<4){code+="7";code_etoile+="*";}}
101.        if (e->key()==Qt::Key_8){if
        (code.size()<4){code+="8";code_etoile+="*";}}
102.        if (e->key()==Qt::Key_9){if
        (code.size()<4){code+="9";code_etoile+="*";}}
103.        if (e->key()==Qt::Key_Enter or e-
        >key()==Qt::Key_Return){codeS();}
104.        if (e->key()==Qt::Key_Delete or e-
        >key()==Qt::Key_Backspace){code="";code_etoile="";}
105.    }
106. void CRFID_station::codeS(){
107.     if (code==carte.CardCode()){
108.         ui->textBrowser_info->setText("CODE BON");
109.         qDebug()<<"Code bon";
110.         ui->label_Cardconnected->setText(" ");
111.         ui->label_X->hide();
112.         ui->label_V->show();
113.         ui->label_indicator->setText("Choisissez votre
        destination");
114.         choice();
115.     }
116.     else {
117.         ui->textBrowser_info->setText("CODE ERRONE");
118.         ui->label_X->show();
119.         ui->label_V->hide();
120.         code="";code_etoile="";
121.     }
122. }
123. void CRFID_station::choice(){
124.     ui->pushButton_5->show();
125.     ui->pushButton_6->show();
126.     ui->label_setcode->hide();
127.     ui->label_X->hide();
128.     ui->label_V->hide();
129.     ui->label_indicator->hide();
130. }
131. void CRFID_station::SDest(){
132.     ui->label_setcode->hide();
133.     ui->label_X->hide();
134.     ui->label_V->hide();
135.     ui->pushButton_Next->show();
136.     ui->comboBox->show();
137.     ui->label_indicator->show();
138.     ui->comboBox->addItem(QIcon("C:/Users/SN2-
        GR1/Desktop/CRFID-B/RESSOURCES/logo_auto.png"),"Bastille");
139.     ui->comboBox->addItem(QIcon("C:/Users/SN2-
        GR1/Desktop/CRFID-B/RESSOURCES/logo_auto.png"),"Trocadéro");
140.     ui->comboBox->addItem(QIcon("C:/Users/SN2-
        GR1/Desktop/CRFID-B/RESSOURCES/logo_auto.png"),"Foch");
141. }
142. void CRFID_station::on_pushButton_Next_clicked()
143. {
144.     QString a;
145.     if(ui->comboBox->currentText()=="Bastille"){a="1";}

```

```

146.     if(ui->comboBox->currentText()=="Foch"){a="2";}
147.     if(ui->comboBox->currentText()=="Trocadéro"){a="3";}
148.     LBDD.stations(a);
149.     LBDD.getplace_libre(0);
150.     qDebug()<<LBDD.getplace_libre(0);
151.     if(LBDD.getplace_libre(0)==0){
152.         ui->comboBox->hide();
153.         ui->label_indicator->hide();
154.         ui->pushButton_Next->hide();
155.         ui->label_NoPlace->show();
156.         ui->label_Warning->show();
157.         ui->label_NoPlace->setText("Attention ! La station
choisie ne possède aucune place de libre. Voulez-vous continuer
?");
158.         ui->pushButton_7->show();
159.         ui->pushButton_new_destination->show();
160.     }
161.     else{
162.         ui->pushButton_Next->hide();
163.         ui->comboBox->hide();
164.         ui->label_indicator->setText("Voulez vous réserver
votre place ?");
165.         ui->pushButton_YES->show();
166.         ui->pushButton_4->show();
167.     }
168. }
169. void CRFID_station::on_pushButton_YES_clicked()
170. {
171.     ui->pushButton_4->hide();
172.     ui->pushButton_YES->hide();
173.     sock1.Idsend(carte.CardVoiture());
174.     TF->start(5000);
175.     qDebug()<<"Fin client";
176.     ui->label_indicator->setText("Votre place est
réservée.");
177.     // sock1.Idsend();
178. }
179. void CRFID_station::on_pushButton_5_clicked()
180. {
181.     ui->pushButton_5->hide();
182.     ui->pushButton_6->hide();
183.     SDest();
184.     ui->textBrowser_info->setText("");
185.     qDebug()<<"Debut une course";
186. }
187. void CRFID_station::on_pushButton_6_clicked()
188. {
189.     ui->pushButton_5->hide();
190.     ui->pushButton_6->hide();
191.     ui->label_indicator->show();
192.     ui->label_indicator->setText("Fin de la location .");
193.     ui->pushButton_8->show();
194.     ui->textBrowser_info->setText("");
195.     qDebug()<<"Met fin à une course";
196. }
197. void CRFID_station::on_pushButton_8_clicked()
198. {

```

```
199.     ui->label_Recapitulatif->show();
200.     ui->pushButton_9->show();
201.     ui->label_indicator->hide();
202.     ui->pushButton_8->hide();
203. }
204. void CRFID_station::on_pushButton_9_clicked()
205. {
206.     ui->pushButton_start->show();
207.     ui->pushButton_9->hide();
208.     ui->label_Recapitulatif->hide();
209. }
210. void CRFID_station::on_pushButton_new_destination_clicked()
211. {
212.     ui->pushButton_7->hide();
213.     ui->pushButton_new_destination->hide();
214.     ui->label_NoPlace->hide();
215.     ui->label_Warning->hide();
216.     ui->pushButton_Next->show();
217.     ui->comboBox->show();
218.     ui->label_indicator->show();
219. }
220. void CRFID_station::on_pushButton_4_clicked()
221. {
222.     ui->pushButton_4->hide();
223.     ui->pushButton_YES->hide();
224.     TF->start(5000);
225.     sock1.Idsend(carte.CardVoiture());
226.     ui->label_indicator->setText("Vous pouvez prendre l'auto
3.");
227.     qDebug() << "Fin client";
228. }
229. void CRFID_station::on_pushButton_7_clicked()
230. {
231.     ui->pushButton_7->hide();
232.     ui->pushButton_new_destination->hide();
233.     ui->label_NoPlace->hide();
234.     ui->label_Warning->hide();
235.     ui->label_indicator->show();
236.     sock1.Idsend(carte.CardVoiture());
237.     ui->label_indicator->setText("Vous pouvez prendre l'auto
3.");
238.     qDebug() << "Fin client";
239.     TF->start(5000);
240. }
241. void CRFID_station::afficheF()
242. {
243.     qDebug() << "Nouveau client";
244.     ui->pushButton_start->show();
245.     ui->label_First_indicator->show();
246.     ui->label_indicator->hide();
247.     ui->textBrowser_info->hide();
248.     code="";
249.     code_etoile="";
250.     ui->comboBox->clear();
251.     TF->stop();
252. }
```

3. crfid_station.h :

```

4. #ifndef MAINWINDOW_H
5. #define MAINWINDOW_H
6. #include "WINS CARD.h"
7. #include "scarderr.h"
8. #include <QMainWindow>
9. #include <QTimer>
10. #include "cardrfid.h"
11. #include "sql.h"
12. #include "sockette.h"
13. namespace Ui {
14.     class CRFID_station;
15. }
16. class CRFID_station : public QMainWindow
17. {
18.     Q_OBJECT
19. public:
20.     explicit CRFID_station(QWidget *parent = 0);
21.     void connexion_OK ();
22.     ~CRFID_station();
23. private:
24.     Ui::CRFID_station *ui;
25.     QTimer *T1;
26.     QTimer *TC;
27.     QTimer *TF;
28.     QString code;
29.     QString code_etoile;
30.     QString nom;
31.     QTcpSocket soc;
32.     CardRFID carte;
33.     Sql LBDD;
34.     Sockette sock1;
35. private slots:
36.     void on_pushButton_7_clicked();
37.     void on_pushButton_4_clicked();
38.     void on_pushButton_new_destination_clicked();
39.     void on_pushButton_9_clicked();
40.     void on_pushButton_8_clicked();
41.     void on_pushButton_6_clicked();
42.     void on_pushButton_5_clicked();
43.     void on_pushButton_YES_clicked();
44.     void on_pushButton_Next_clicked();
45.     void on_pushButton_refresh_clicked();
46.     void on_pushButton_start_clicked();
47.     void on_pushButton_clicked();
48.     void codec();
49.     void codeS();
50.     void SDest();
51.     void choice();
52.     void afficheF();
53.     virtual void keyPressEvent( QKeyEvent *e );
54. };
55. #endif // CRFID_STATION_H

```

3. Cardrfid.cpp

```

4. #include "cardrfid.h"
5. CardRFID::CardRFID() {
6. }
7. int CardRFID::update()
8. {
9.     ret = SCardConnect(
        hContext, readerName, SCARD_SHARE_SHARED, SCARD_PROTOCOL_T0|SCARD_
        PROTOCOL_T1, &hCard, &dwActProtocol );
10.     if (ret != SCARD_S_SUCCESS) {
11.         //qDebug() << "carte absente ou non reconnue...";
12.         return 0; }
13.     else {
14.         //qDebug() << "carte présente";
15.         ret = LoadCle();
16.         if (ret != SCARD_S_SUCCESS) {
17.             //qDebug() << "erreur de chargement";
18.             return 0; }
19.         else {
20.             return 1; } }
21. }
22. int CardRFID::Initialisationcard() //initialisation de
    l'environnement
23. {
24.     ret =
        SCardEstablishContext(SCARD_SCOPE_USER, NULL, NULL, &hContext);
25.     if (ret != SCARD_S_SUCCESS) {
26.         //qDebug() << "Erreur de l'environnement";
27.         return 0; }
28.     else {
29.         //DETECTION DU LECTEUR ET NOM
30.         DWORD size = 256;
31.         ret =
            SCardListReaders(hContext, NULL, readerName, &size);
32.         if (ret != SCARD_S_SUCCESS) {
33.             //qDebug() << "Erreur lecteur";
34.             return 0; }
35.         else {
36.             QString lecteur =
                QString::fromWCharArray(readerName);
37.             //qDebug() << lecteur;
38.             return 1; } }
39. }
40. int CardRFID::Transmit()
41. {
42.     ret =
        SCardTransmit(hCard, NULL, SendBuff, SendLen, NULL, RecvBuff, &RecvLe
        n);
43.     if (ret != SCARD_S_SUCCESS) {
44.         //qDebug() << "erreur transmission...";
45.         return ret; }
46. }
47. QString CardRFID::Read() //Fonction de lecture d'un bloc
    de la carte
48. {
49.     for(int i=0; i<262; i++){
50.         SendBuff[i] = 0; }

```

```
51.     QString a="";
52.     SendLen = 5;
53.     RecvLen = 18;
54.     SendBuff[0]=0xFF;
55.     SendBuff[1]=0xB0;
56.     SendBuff[2]=0x00;
57.     SendBuff[3]=BlockNumber;
58.     SendBuff[4]=0x10;
59.     ret = Transmit();
60.     if((RecvBuff[RecvLen - 2]== 0x90) && (RecvBuff[RecvLen -
1]== 0x00)){
61.         //qDebug()<<"Lecture du bloc réussi ";
62.         for (int i=0;i<16;i++){
63.             if (RecvBuff[i] != 0){
64.                 a += RecvBuff[i];}}
65.         return a;}
66. }
67. int CardRFID::LoadCle()
68. {
69.     for(int i=0;i<262;i++){
70.         SendBuff[i] = 0;}
71.     SendBuff[0] = 0xFF;
72.     SendBuff[1] = 0x82;
73.     SendBuff[2] = 0x00;
74.     SendBuff[3] = 0; //Key Store n°
75.     SendBuff[4] = 0x06;
76.     SendBuff[5] = 0xFF; // clé sur 6 octets
77.     SendBuff[6] = 0xFF;
78.     SendBuff[7] = 0xFF;
79.     SendBuff[8] = 0xFF;
80.     SendBuff[9] = 0xFF;
81.     SendBuff[10] = 0xFF;
82.     SendLen = 11;
83.     RecvLen = 2;
84.     ret = Transmit();
85.     return ret;
86. }
87. void CardRFID::Charge()
88. {
89.     for(int i=0;i<262;i++){
90.         SendBuff[i] = 0;}
91.     //Chargement du block
92.     SendBuff[0] = 0xFF;
93.     SendBuff[1] = 0x86;
94.     SendBuff[2] = 0x00;
95.     SendBuff[3] = 0x00;
96.     SendBuff[4] = 0x05;
97.     SendBuff[5] = 0x01;
98.     SendBuff[6] = 0x00;
99.     SendBuff[7] = BlockNumber; // n° de bloc
100.    SendBuff[8] = 0x60; //type A sinon type B = 0x61
101.    SendBuff[9] = 0; //Key Store n°
102.    SendLen = 10;
103.    RecvLen = 2;
104.    ret = Transmit();
105. }
106. QString CardRFID::CardCode()
```



```

107. {
108.     BlockNumber = 0x08;
109.     Charge();
110.     return Read();
111. }
112. QString CardRFID::CardPrenom()
113. {
114.     BlockNumber = 0x09;
115.     Charge();
116.     return Read();
117. }
118. QString CardRFID::CardNom()
119. {
120.     BlockNumber = 0x0A;
121.     Charge();
122.     return Read();
123. }
124. QString CardRFID::CardVoiture()
125. {
126.     BlockNumber = 0x0C;
127.     Charge();
128.     return Read();
129. }

```

3. cardrfid.h :

```

4. #ifndef CARDRFID_H
5. #define CARDRFID_H
6. #include "WINS CARD.h"
7. #include "scarderr.h"
8. #include <QString>
9. #include <stdlib.h>
10. #include <stdio.h>
11. #include <QDebug>
12. class CardRFID
13. {
14. public:
15.     CardRFID();
16.     int Initialisationcard();
17.     int update();
18.     int Transmit();
19.     void Charge();
20.     QString Read();
21.     int LoadCle();
22.     QString CardNom();
23.     QString CardPrenom();
24.     QString CardCode();
25.     QString CardVoiture();
26. private:
27.     SCARDCONTEXT hContext;
28.     SCARDHANDLE hCard;
29.     SCARD_IO_REQUEST ioRequest;
30.     WCHAR readerName[256];
31.     DWORD SendLen, RecvLen, ByteRet, size, dwActProtocol;
32.     BYTE SendBuff[262], RecvBuff[262], Code[4], ID[16];
33.     int ret;
34.     int BlockNumber;
35. };
36. #endif // CARDRFID_H

```

4. sockette.cpp :

```
#include "sockette.h"
Sockette::Sockette() {
}
void Sockette::Idsend(QString mess)
{
    idcardclient=mess;
    QString IP ="192.168.1.98";
    int a =5000;
    connect (&soc,SIGNAL(connected()),this, SLOT(connexion_OK()));
    soc.connectToHost(IP, a);
}
void Sockette::connexion_OK() {
    QTextStream texte (&soc);
    texte<<"ID client: "+idcardclient<<endl;
    qDebug()<<"ID client: "+idcardclient;
}
```

5. sockette.h :

```
6. #ifndef SOCKETTE_H
7. #define SOCKETTE_H
8. #include <QtNetwork/QTcpSocket>
9. class Sockette : public QObject
10. {
11.     Q_OBJECT
12. public:
13.     Sockette();
14.     void Idsend (QString mess);
15. private:
16.     QTcpSocket soc;
17.     QString idcardclient;
18. private slots:
19.     void connexion_OK();
20. };
21. #endif // SOCKETTE_H
```

6. sql.cpp :

```
7. #include "sql.h"
8. Sql::Sql() {
9. }
10. void Sql::login()
11. {
12.     //On ouvre la connexion
13.     db =
        QSqlDatabase::addDatabase("QMYSQL3","Connection");
14.     db.setHostName("192.168.1.115");
15.     db.setDatabaseName("autolib");
16.     db.setUserName("root");
17.     db.setPassword("");
```

```

18.         //Teste connexion base de donnée Mysql
19.         if(db.open()) {
20.             qDebug() << "Connexion BDD: opérationnel";
21.         } else {
22.             qDebug() << "Connexion BDD: erreur";
23.             db.close();
24.         }
25.     void Sql::clients(QString num)
26.     {
27.         //QString nom = "clients";
28.         QString requete = "SELECT * FROM clients WHERE
`numero-badge` = "+num;
29.         QSqlQuery query(db);
30.         query.exec(requete);
31.         for(int i=0; query.next(); i++) {
32.             numero_badge[i] = query.value(0).toString();
33.             nom[i] = query.value(1).toString();
34.             code_secret[i] = query.value(2).toString();
35.             facture[i] = query.value(3).toString();
36.         }
37.     }
38.     void Sql::stations(QString idS)
39.     {
40.         //QString nom = "stations";
41.         QString requete = "SELECT * FROM stations WHERE
`id-station` = "+idS;
42.         QSqlQuery query(db);
43.         query.exec(requete);
44.         for(int i=0; query.next(); i++) {
45.             id_station[i] = query.value(0).toString();
46.             nom_station[i] = query.value(1).toString();
47.             nb_places[i] = query.value(2).toString();
48.             nb_vehicules[i] = query.value(3).toString();
49.             place_libre[i] = query.value(4).toString();
50.         }
51.     void Sql::vehicules(QString nbrveh)
52.     {
53.         QString requete = "SELECT * from vehicules WHERE
`numero-vehicule` = "+nbrveh;
54.         QSqlQuery query(db);
55.         query.exec(requete);
56.         for(int i=0; query.next(); i++) {
57.             numero_vehicule[i] =
query.value(0).toString();
58.             immatriculation[i] =
query.value(1).toString();
59.             libre[i] = query.value(2).toString();
60.             kilometrage[i] = query.value(3).toString();
61.         }
62.     }
63.     QString Sql::getnumero_badge(int nb)
64.     {
65.         return numero_badge[nb];
66.     }
67.     QString Sql::getnom(int nb)
68.     {
69.         return nom[nb];

```

```
70.     }
71.     QString Sql::getcode_secret(int nb)
72.     {
73.         return code_secret[nb];
74.     }
75.     QString Sql::getfacture(int nb)
76.     {
77.         return facture[nb];
78.     }
79.     QString Sql::getid_station(int nb)
80.     {
81.         return id_station[nb];
82.     }
83.     QString Sql::getnom_station(int nb)
84.     {
85.         return nom_station[nb];
86.     }
87.     QString Sql::getnb_places(int nb)
88.     {
89.         return nb_places[nb];
90.     }
91.     QString Sql::getnb_vehicules(int nb)
92.     {
93.         return nb_vehicules[nb];
94.     }
95.     QString Sql::getplace_libre(int nb)
96.     {
97.         return place_libre[nb];
98.     }
99.     QString Sql::getnumero_vehicule(int nb)
100.    {
101.        return numero_vehicule[nb];
102.    }
103.    QString Sql::getimmatriculation(int nb)
104.    {
105.        return immatriculation[nb];
106.    }
107.    QString Sql::getlibre(int nb)
108.    {
109.        return libre[nb];
110.    }
111.    QString Sql::getkilometrage(int nb)
112.    {
113.        return kilometrage[nb];
114.    }
115.    //QString Sql::getreservation(QString ids)
116.    //{
117.    //    QString requete = "UPDATE `stations` SET
118.    //        `place_libre`="+
119.    //        getplace_libre(0).toInt().getplace_libre(0).toInt()-1 +
120.    //        "WHERE `id-station`= "+ids;
121.    //    QSqlQuery query(db);
122.    //    query.exec(requete);
123.    //    return query.value(0).toString();
124.    // }
```

7. sql.h :

```
8. #ifndef SQL_H
9. #define SQL_H
10. #include <QtSql/QtSqlDatabase>
11. #include <QtSql/QtSqlQuery>
12. #include <QModelIndex>
13. #include <QCoreApplication>
14. #include <QtSql>
15. #include <QMessageBox>
16. #include <QDebug>
17. class Sql
18. {
19. public:
20.     Sql();
21.     void login();
22.     void clients(QString num);
23.     void stations(QString idS);
24.     void vehicules(QString nbrveh);
25.     QString getnumero_badge(int nb);
26.     QString getnom(int nb);
27.     QString getcode_secret(int nb);
28.     QString getfacture(int nb);
29.     QString getid_station(int nb);
30.     QString getnom_station(int nb);
31.     QString getnb_places(int nb);
32.     QString getnb_vehicules(int nb);
33.     QString getnumero_vehicule(int nb);
34.     QString getimmatriculation(int nb);
35.     QString getlibre(int nb);
36.     QString getkilometrage(int nb);
37.     QString getplace_libre(int nb);
38.     QString getreservation(QString idS);
39. private:
40.     QSqlDatabase db;
41.     QString numero_badge[50];
42.     QString nom[50];
43.     QString code_secret[50];
44.     QString facture[50];
45.     QString id_station[50];
46.     QString nom_station[50];
47.     QString nb_places[50];
48.     QString nb_vehicules[50];
49.     QString place_libre[50];
50.     QString numero_vehicule[50];
51.     QString immatriculation[50];
52.     QString libre[50];
53.     QString kilometrage[50];
54. };
55. #endif // SQL_H
```

E. Codage de l'application de supervision