Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2014

Jacky Casas

NFC golf ball dispenser

Professeur
Pierre-André Mudry

Expert
David Jilli

Date de la remise du rapport
11.07.2014

HES-SO Valais-Wallis • rte du Rawyl 47 • C.P. • 1950 Sion 2
+41 27 606 85 11 • info@hevs.ch • **www.hevs.ch**

Valais excellence

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2013/14**	No TD / *Nr. DA* **it/2014/60**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☒ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Jacky Casas** Professeur / *Dozent* **Pierre-André Mudry**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **David Jilli** Av. Montagibert 8 | 1005 Lausanne	

Titre / *Titel*

NFC golf ball dispenser

Description / *Beschreibung*

On désire réaliser un système embarqué permettant de libérer des balles de golf pour s'entraîner dans un driving range. Pour recevoir des balles, le client possède une carte NFC ou un mobile Android NFC qui permet de réaliser la facturation en se reliant à une base de données distante. Il faut d'une part réaliser un système embarqué basé sur un Rapsberry PI libérant les balles aux clients authentifiés via NFC et faire une application Android pour l'authentification via NFC.

Objectifs / *Ziele*

- Récupération et amélioration des résultats du projet de semestre pour le NFC
- Écriture d'un logiciel embarqué sur Android pour l'authentification via NFC
- Écriture / portage du logiciel sur plateforme embarquée pour l'authentification via NFC
- Déploiement d'un middleware pour la gestion des droits avec une base de données
- Réplication de la database en local en cas de pertes de communication (si le temps le permet)
- Mise en place d'un démonstrateur complet avec interface graphique.

Signature ou visa / *Unterschrift oder Visum* Responsable de l'orientation *Leiter der Vertiefungsrichtung:* ..... [1] Etudiant / Student :	Délais / *Termine* Attribution du thème / *Ausgabe des Auftrags:* 12.05.2014 Remise du rapport / *Abgabe des Schlussberichts:* 11.07.2014, 12:00 Expositions / *Ausstellungen der Diplomarbeiten:* 27 – 29.08.2014 Défense orale / *Mündliche Verfechtung:* Semaine | Woche 36

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

NFC Golf Ball Dispenser

Diplômant Jacky Casas

Objectif du projet

Le but du projet est de concevoir un système embarqué permettant de libérer des balles de golf pour s'entraîner dans un driving range. Pour acheter des balles, le client possède une carte NFC ou un mobile Android NFC qui permet de réaliser la facturation en se reliant à une base de données distante.

Méthodes | Expériences | Résultats

Le système embarqué, un Raspberry Pi, est connecté à un lecteur de carte NFC. Un logiciel avec interface graphique a été développé en Python pour permettre de détecter et communiquer avec une carte NFC. Le système communique et enregistre les transactions sur une base de données distante de manière sécurisée.

Une application Android a également été développée. Elle permet au client de créer un compte et de se connecter à celui-ci. Avec cette application, le client peut retirer des balles de golf en approchant le smartphone du lecteur. Les informations du compte comme le solde, le statut du compte et les dernières transactions sont affichées sur l'application.

Le smartphone accède à des APIs qui ont été déployées sur un serveur distant pour s'authentifier et aller chercher les différentes informations.

Travail de diplôme
| édition 2014 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
Dr. Pierre-André Mudry
pierre-andre.mudry@hevs.ch

Partenaire
Digiclever Sàrl

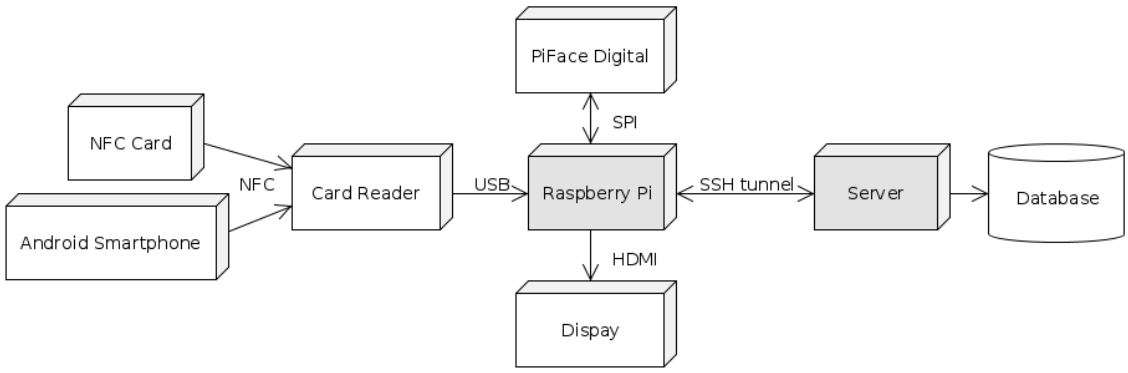


Schéma bloc du dispositif.

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Mon travail de Bachelor de la formation d'ingénieur en Systèmes Industriels - Infotronics à la HES-SO de Sion consiste à concevoir un distributeur de balles de golf NFC.

Ce projet est réalisé sous la supervision du Dr. Pierre-André Mudry, professeur à la HES-SO de Sion, et a été mandaté par l'entreprise Digiclever. Digiclever est une entreprise vaudoise basée à Gland qui est spécialisée dans le développement web et la création de produits utilisant la communication sans contact (RFID, NFC).

1.2 Description

Le but est de réaliser un système embarqué permettant de libérer des balles de golf pour s'entraîner au driving range. Pour recevoir des balles, le client possède une carte NFC ou un mobile Android NFC qui permet de réaliser la facturation en se liant à une base de données distante. Il faut d'une part réaliser un système embarqué basé sur un Raspberry Pi libérant les balles aux clients authentifiés via NFC et faire une application Android pour l'authentification via NFC.

Les objectifs sont les suivants :

- Définition de la plate-forme hardware/software pour le NFC
- Ecriture d'un logiciel embarqué sur Android pour l'authentification via NFC
- Ecriture / portage du logiciel sur plateforme embarquée pour l'authentification via NFC
- Déploiement d'un middleware pour la gestion des droits avec une base de données
- Mise en place d'un démonstrateur complet avec interface graphique

1.3 MATÉRIEL À DISPOSITION

Pour le développement de ce projet, le matériel suivant est à disposition :

- un kit de développement ACR122-SDK, ou "ACR122U NFC Contactless Smart Card Reader Software Development Kit", de la marque ACS (Advanced Card Systems Ltd.) qui contient :
  - un lecteur de carte ACR122U
  - un lecteur de carte ACR122T
  - 5 cartes NFC Mifare 1K
  - un CD-ROM qui contient des utilitaires et des drivers pour Windows, des programmes d'exemple et le manuel de référence.

FIGURE 1.1 – ACR122-SDK

- un Raspberry Pi
- une carte d'extension PiFace Digital pour Raspberry Pi
- une carte SD de 8 Go
- un cable micro USB pour alimenter le Raspberry Pi
- un adaptateur HDMI/VGA

Chapitre 2

Communication NFC

2.1 NFC, qu'est-ce que c'est ?

Le NFC est un protocole de communication, dont voici la définition qu'en fait Wikipedia :

> La communication en champ proche (en anglais near field communication, NFC) est une technologie de communication sans-fil à courte portée et haute fréquence, permettant l'échange d'informations entre des périphériques jusqu'à une distance d'environ 10 cm. [1]

Cette technologie est présente sur la plupart des smartphones et tablettes sortis ces trois dernières années. Les produits Apple n'ont pas cette capacité, par contre ils utilisent iBeacon, qui est un protocole basé sur le Bluetooth 4.0 (ou Bluetooth Low Energy "BLE"). BLE est également présent sur quelques terminaux Android récents.

Le NFC est utilisé pour s'échanger des documents, des pages internet, des photos entre terminaux Android. Il est également utilisé pour l'identification ou le paiement grâce à des bornes NFC. Il existe des tags NFC passifs sous forme d'auto-collants ou de cartes au format carte de crédit qui sont utilisés pour de l'identification ou pour automatiser des tâches sur un smartphone Android.

Le NFC peut être utilisé de trois façons différentes, comme le montre la figure 2.1 :

1. Le mode émulation de carte NFC
2. Le mode pairage d'appareils
3. Le mode lecture-écriture

Nous allons utiliser le troisième mode pour communiquer entre le lecteur de carte ACR122U et les cartes NFC (p.ex Mifare 1K). Et nous allons utiliser le premier mode pour communiquer entre le lecteur de carte et un smartphone Android.

1. Source : `http://fr.wikipedia.org/wiki/Near_Field_Communication`, page consultée en mai 2014.

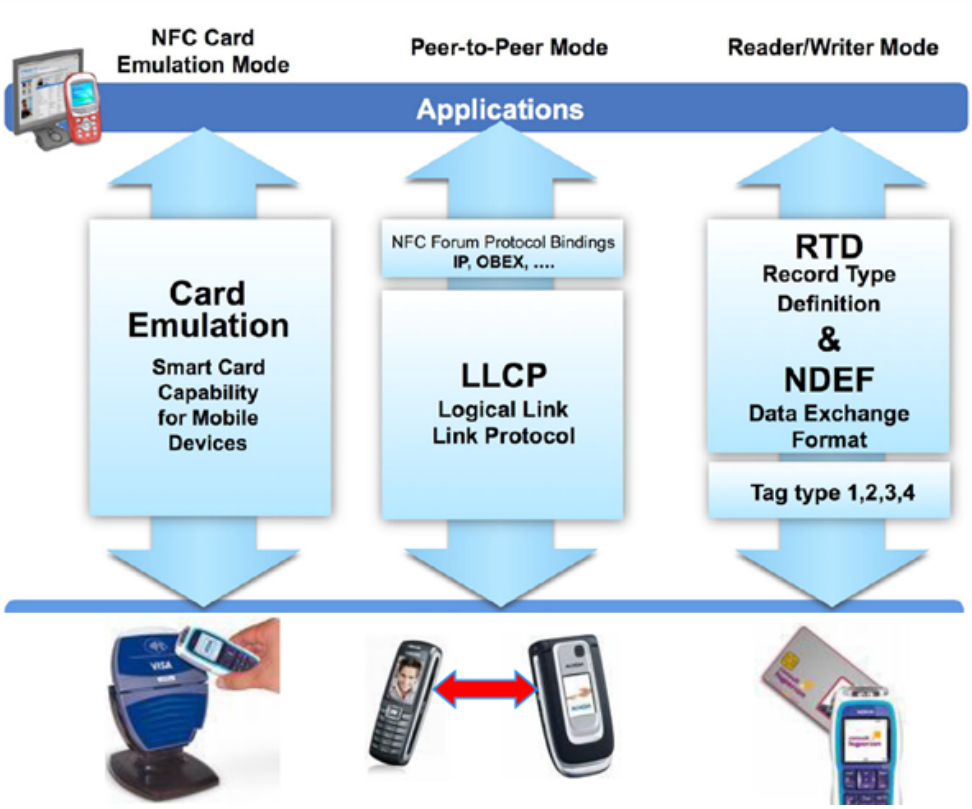


FIGURE 2.1 – Modes d'utilisation du NFC

source : `http://developer.nokia.com/community/wiki/Inside_NFC:_Usages_and_Working_Principles`

2.2 CHOIX DE LA LIBRAIRIE

2.2.1 Contexte

Pour manier le lecteur de cartes NFC ACR122U que nous avons à disposition, nous devons utiliser une interface de programmation (ou API "Application Programming Interface" en anglais). Dans le datasheet de ce lecteur il est indiqué le support de PC/SC et de CT-API, qui est un wrapper de PC/SC. Mais nous pouvons également utiliser LibNFC, une librairie NFC open source bien connue pour la gestion du NFC.

En sachant que nous allons développer notre application en Python, nous devons tenir compte de l'API mais également de la disponibilité de libraires en Python pour y accéder.

2.2.2 LibNFC

LibNFC est une librairie bas niveau permettant la gestion de différents appareils utilisant le NFC. Il s'agit d'une librairie libre sous licence *GNU Lesser General Public License* (ou GNU LGPL). Elle est compatible avec beaucoup de systèmes d'exploitation, notamment Windows, Mac OS X et Linux. La dernière version est la version 1.7.0 et date de septembre 2013.

Le site officiel est accessible à l'adresse suivante : `http://nfc-tools.org/index.php?title=Libnfc`.

LibNFC est utilisable en langage C. Pour pouvoir l'utiliser en Python, il existe deux librairies : *pynfc* ou de *nfcpy*.

Pynfc fourni une interface Python pour la librairie LibNFC, mais la dernière version disponible est la version 0.0.4 et date de février 2010. Elle est compatible avec LibNFC 1.3. Plusieurs fonctions de la nouvelle version ne sont pas accessibles.

Nfcpy est un module Python permettant également de communiquer avec des cartes NFC. Cette librairie est plus récente et encore mise à jour puisque la dernière version date du 13 février 2014. Il s'agit de la version 0.9.1.

2.2.3 PC/SC

PC/SC est l'abréviation de *"Personal Computer/Smart Card"*, il s'agit d'une librairie qui permet d'utiliser des cartes intelligentes, comme des cartes NFC, depuis un ordinateur. Cette librairie est développée par le *PC/SC Workgroup* qui est composé de Microcosft, Toshiba, Gemalto, Infineon, Oracle et d'autres entreprises. Une version libre de la librairie appelée "PC/SC Lite" est également disponible pour les systèmes Linux et Unix. Advanced Card Systems fait partie des membres associés au PC/SC Workgroup. Le site web du PC/SC Workgroup est accessible à l'adresse suivante : `http://www.pcscworkgroup.com/`.

Il existe différentes librairies pour utiliser PC/SC, les langages disponibles sont entre autres le C, Java, PHP, Python, Ruby, Perl. Le module Python correspondant s'appelle *Pyscard* pour Python Smart Card Library.

Advanced Card Systems indique le support de l'API PC/SC pour son lecteur ACR122U.

Notre choix se porte sur l'utilisation de PC/SC Lite avec le module python Pyscard.

2.3 COMMUNICATION AVEC LE LECTEUR DE CARTE

2.3.1 Manuel

Le datasheet officiel de l'API PC/SC du lecteur ACR122U détaille bien la façon de communiquer avec le lecteur. Il suffit de lui envoyer une série d'octets bien précis pour qu'il fasse ce que l'on veut. Le datasheet explique comment savoir des informations sur le lecteur lui-même, comment charger un identifiant qu'on va envoyer à la carte, comment lire et écrire dans la mémoire de la carte, etc. Avec chaque réponse, le lecteur envoie deux octets de contrôle. Si ces octets sont `0x90` et `0x00`, tout s'est bien passé. Pour toute autre valeur, il faut se référer au datasheet pour savoir la source du problème.

2.3.2 ATR

L'ATR (Answer-To-Reset) est une sorte d'identifiant d'un appareil NFC. Lorsqu'on approche la carte d'un lecteur, cet ATR est envoyé, ce qui permet de savoir à quel type de carte nous avons affaire.

L'ATR d'une carte Mifare 1K est le suivant :

```
1  3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6A
```

FIGURE 2.2 – ATR d'une carte Mifare 1K

En décomposant cette suite d'octets grâce au datasheet on trouve ceci :

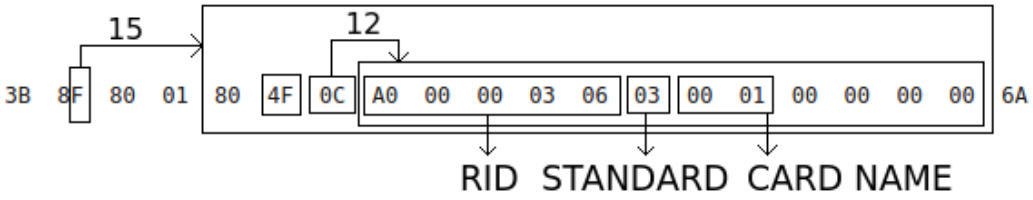


FIGURE 2.3 – ATR d'une carte Mifare 1K analysé

Le RID (Registered Application Provider Identification) est `A0 00 00 03 06`, ce qui correspond à l'identifiant du PC/SC Workgroup. Le standard `03` indique que la carte est ISO 14443A Part 3[2]. Et pour finir le nom de la carte `00 01` correspond à une carte Mifare 1K. Le tableau 2.1 montre les identifiants de plusieurs cartes.

Identifiant	Carte
00 01h	Mifare 1K
00 02h	Mifare 4K
00 03h	Mifare Ultralight
00 04h	Mifare Mini
F0 04h	Topaz & Jewel
F0 11h	FeliCa 212K
F0 12h	FeliCa 424K
FF xxh	indéfini

TABLE 2.1 – identifiants pour différentes cartes

2.4 CARTE MIFARE 1K

Cette carte fait partie de la série des cartes Mifare. Il s'agit du type de cartes les plus utilisées dans le monde. Le "1K" signifie que la carte possède 1ko de mémoire. Nous pouvons donc stocker de l'information dans la carte.

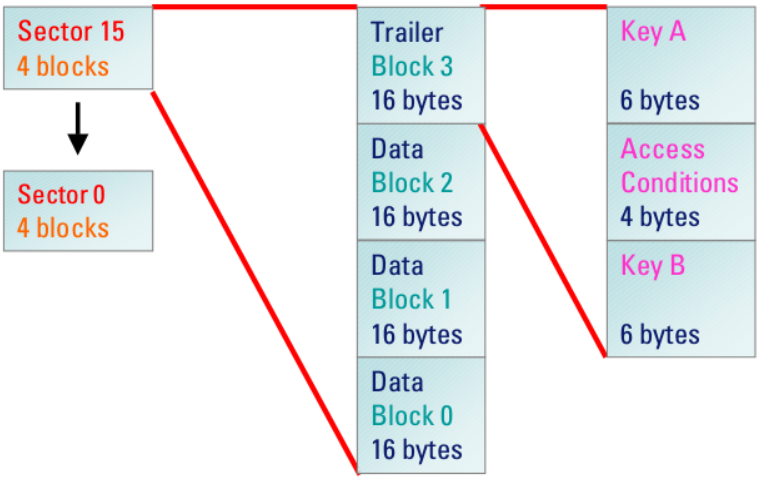


FIGURE 2.4 – Formation de la mémoire

source : tiré du datasheet `www.gemalto.com/products/hybrid_card_body/download/mifare1K_datasheet.pdf`

La mémoire (figure 2.4) de cette carte est composée de 16 secteurs. Chaque secteur est divisé en 4 blocs de 16 octets chacun. 16 secteurs multipliés par 4 blocs, multipliés par 16 octets nous donne 1024, donc bien 1ko.

Dans chaque secteur, trois blocs sont dédiés à stocker de l'information et sont accessibles en lecture et en écriture. Le quatrième bloc est un peu spécial, il contient des informations comme la clé d'authentification du secteur et d'autres informations d'accessibilité. Pour accéder en lecture ou en écriture à un secteur, il faut donc d'abord s'identifier avec le bon mot de passe.

2. Page Wikipedia : `http://en.wikipedia.org/wiki/ISO/IEC_14443`, page consultée en mai 2014.

Le premier bloc du premier secteur est le seul qui n'est accessible qu'en lecture. Dans ce bloc sont écrites les informations du constructeur de la carte, et notamment l'identifiant unique de la carte codé sur 4 octets. La figure 2.5 montre en détail la composition de la mémoire.

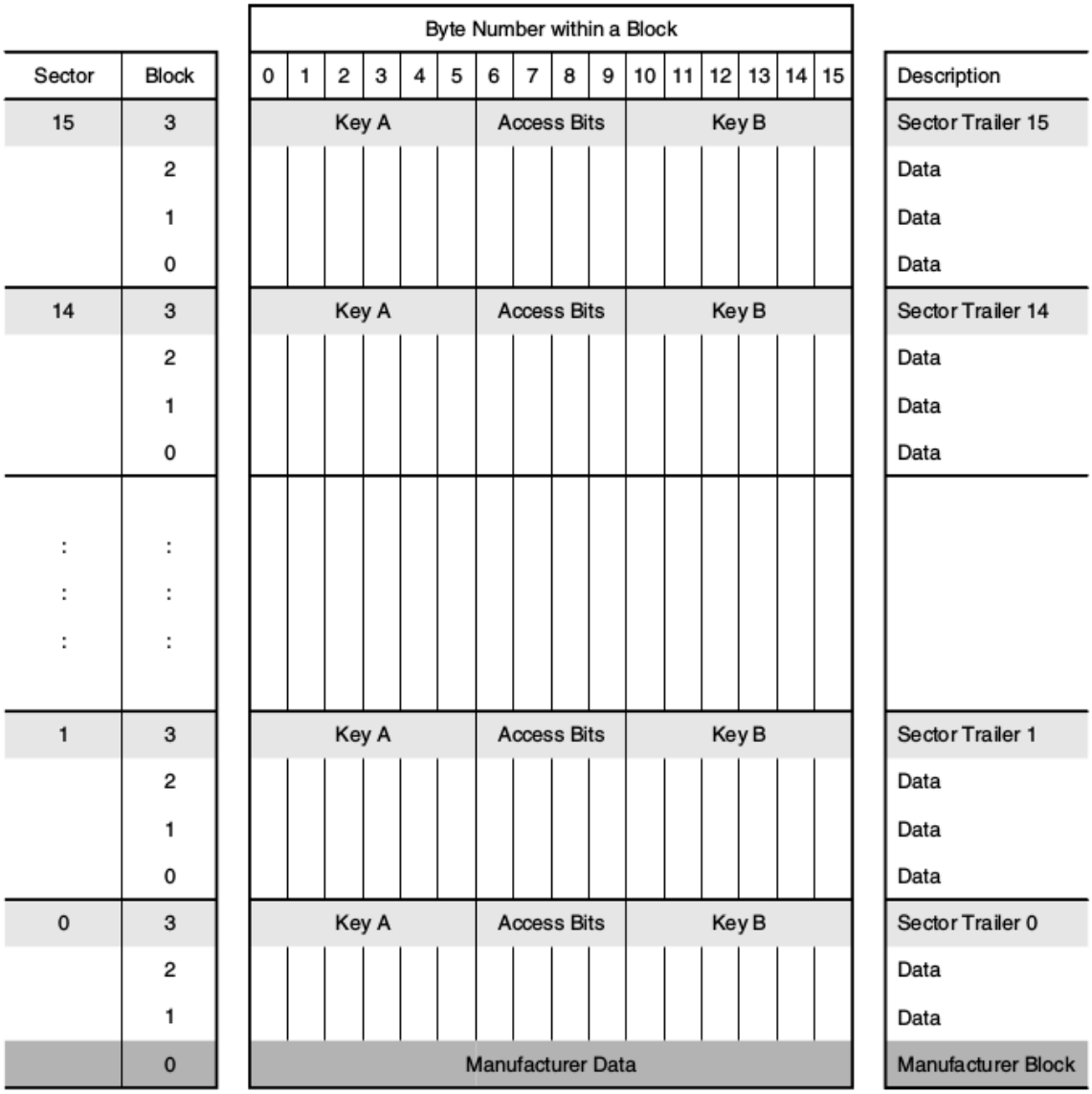


FIGURE 2.5 – Schéma de la mémoire d'une carte Mifare 1K

source : tiré du datasheet `www.nxp.com/documents/data_sheet/MF1S503x.pdf`

Pour pouvoir lire ou écrire une information sur la mémoire d'une carte, il faut envoyer plusieurs commandes au lecteur NFC. La première est une commande qui permet de charger la clé d'accès à la mémoire dans le lecteur. Ensuite il faut envoyer une commande pour que le lecteur essaie de s'identifier auprès de la carte. Et pour finir il est possible d'envoyer la commande pour lire ou écrire des informations. Dans ces commandes il faut préciser plusieurs paramètres comme le numéro du secteur mémoire à accéder, le nombre d'octets à écrire, la type de clé utilisé, etc. Si toutes ces commandes renvoient le résultat `0x90 0x00`, cela voudra dire que tout s'est bien passé. Si ce n'est pas le cas, nous recevons un code d'erreur qui peut vouloir dire que l'authentification s'est mal passée ou autre. Tous les codes d'erreur sont détaillés dans le datasheet.

Une fois l'identification établie pour un bloc mémoire, nous pouvons y accéder autant de fois que nous voulons. Par contre si la carte est retirée du lecteur et à nouveau détectée, il faudra s'identifier à nouveau.

Chapitre 3

Raspberry Pi

3.1 Introduction

Un Raspberry Pi est un ordinateur de la taille d'une carte de crédit (voir figure 3.1) sur lequel on peut installer différents systèmes d'exploitation basés sur GNU/Linux. Il tourne avec un processeur ARM à 700 MHz, possède 512 Mo de RAM, 2 ports USB, un port HDMI, un port RCA, une prise Jack, un port RJ45 et est alimenté par micro-USB. La mémoire de stockage de ce petit ordinateur est une carte SD, qu'on branche sur le port adéquat sur la face inférieure du Raspberry Pi.

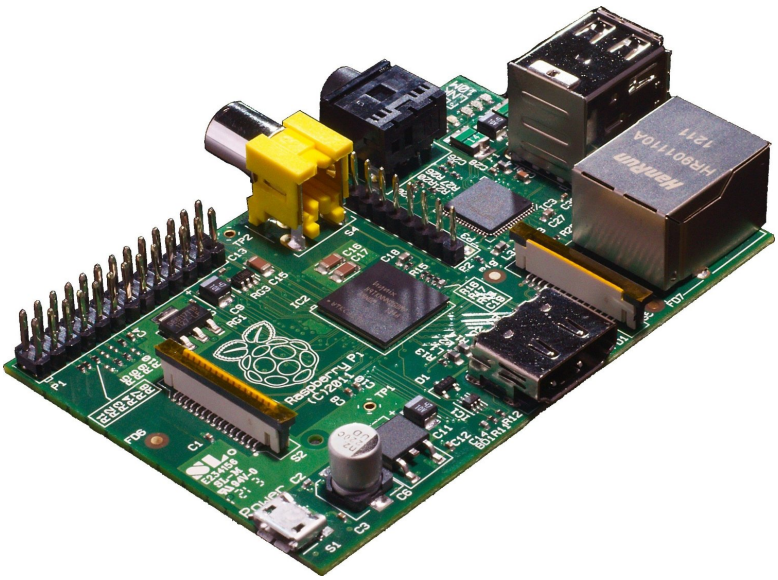


FIGURE 3.1 – Raspberry Pi

source : http://i.pcworld.fr/1297119-raspberry-pi.jpg

Plusieurs systèmes d'exploitation sont à disposition sur le site officiel. Raspbian, Pidora, OpenELEC ou RaspBMC en sont des exemples. La marche à suivre pour installer un système d'exploitation sur la carte SD se trouve en annexe B.1.

Pour utiliser le Raspberry Pi, il est possible de le connecter à un écran, à un clavier et à une souris, à la manière d'un ordinateur standard. Mais il est également possible de s'y connecter par SSH et de le contrôler à distance. En annexe B.2 se trouve un explicatif de la connexion par SSH.

3.2 INSTALLATION DES OUTILS POUR LE NFC

3.2.1 PC/SC

Plusieurs logiciels doivent être installés pour faire fonctionner PC/SC. Voici les commandes à écrire pour les installer les uns après les autres grâce à l'outil `apt-get` :

```
1 apt-get install libusb-dev libusb++-0.1-4c2
2 apt-get install libccid
3 apt-get install pcscd
4 apt-get install libpcsclite1
5 apt-get install libpcsclite-dev
6 apt-get install pcsc-tools
```

FIGURE 3.2 – Commandes pour installer PC/SC

En cas de refus d'installation, pensez à passer en mode administrateur en faisant précéder la commande d'un `sudo`.

Pour voir si le lecteur de carte NFC est bien connecté, on peut entrer la commande `lsusb`. Et pour voir si le service pcscd est bien lancé, on peut entrer `ps -A`.

L'outil *pcsc-tools* contient un utilitaire nommé *pcsc_scan* qui permet de voir si un lecteur de carte NFC est bien connecté et reconnu. Il faut simplement entrer la commande `pcsc_scan` pour le lancer. Il est fort probable que cela ne fonctionne pas du premier coup car le kernel Linux veut lancer les drivers par défaut alors que PC/SC n'a pas besoin de ces drivers pour l'utiliser. Une petite manipulation, fournie par M. Marc Pignat, permet de régler ce problème. Elle empêche le kernel de lancer les drivers par défaut du NFC ainsi que du PN533 (le chip contenu dans le lecteur). Pour cela il faut créer un fichier de blacklist.

```
1 sudo touch /etc/modprob.d/blacklist-nfc.conf
```

FIGURE 3.3 – Création du fichier

Avec comme contenu les deux drivers à ne pas lancer.

```
1 blacklist pn533
2 blacklist nfc
```

FIGURE 3.4 – Contenu du fichier

Pour finaliser le tout, il faut redémarrer le Raspberry Pi ou alors supprimer les deux modules avec les commandes `sudo rmmod pn533` et `sudo rmmod nfc`. En relançant ensuite la commande *pcsc_scan*, on voit que le lecteur est détecté.

3.2.2 Pyscard

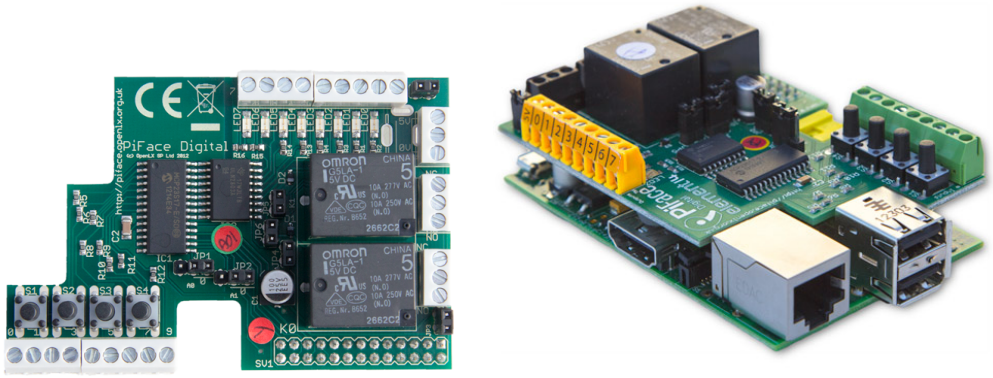
Il faut également installer `pyscard`, la librairie Python pour gérer PC/SC.

```
1 apt-get install python-pyscard
```

FIGURE 3.5 – Commande pour installer pyscard

3.3 PiFace Digital

PiFace Digital est un module qu'on peut brancher sur le Raspberry Pi. Ce module fourni des entrées digitales, des sorties open-collector, des leds, des switches et des relais. La figure 3.6 montre à quoi ressemble cette carte seule (a), ainsi que branchée au Pi (b).



(a) (b)

FIGURE 3.6 – (a) Carte PiFace Digital (b) PiFace branchée au Raspberry Pi

source : http://www.piface.org.uk/products/piface_digital

Pour installer ce module, il suffit d'entrer ces quelques commandes :

```
1  sudo apt-get install python{,3}-pifacedigitalio
```

FIGURE 3.7 – Installation de PiFace

A présent il faut activer le SPI pour la communication entre le Raspberry Pi et le PiFace en allant dans le menu de configuration : `sudo raspi-config`. Dans ce menu, il faut aller dans `Advanced Options`, puis `SPI` et sélectionner `Yes` pour activer le SPI. La figure 3.8 montre la démarche.

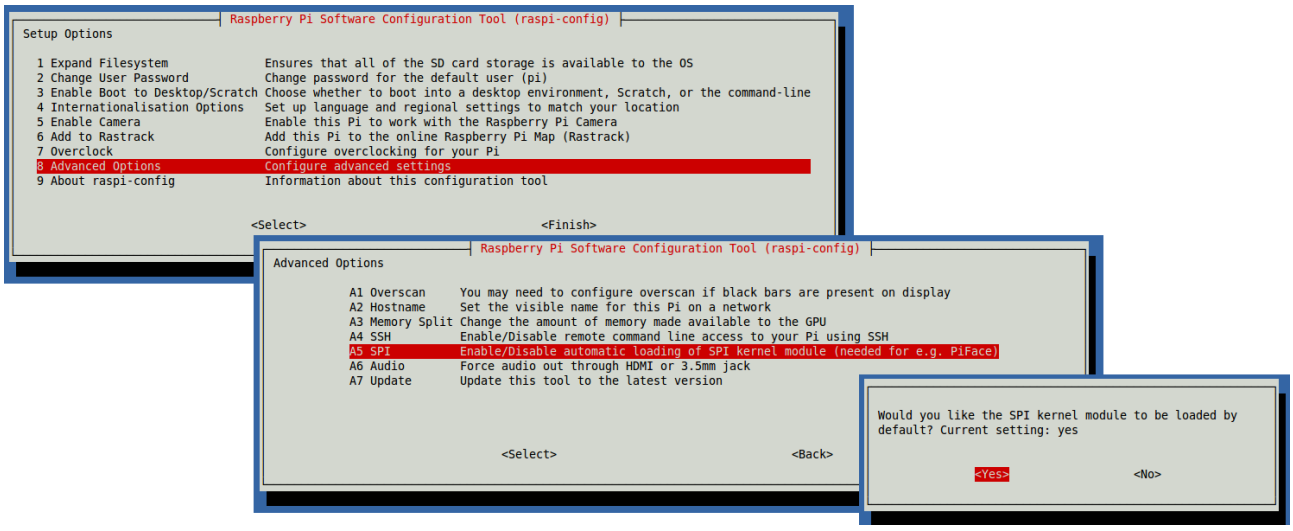


FIGURE 3.8 – Activation du SPI

Pour vérifier que tout fonctionne bien, nous pouvons lancer un petit programme d'exemple. Si une led clignote, c'est bon.

```
1 python3 /usr/share/doc/python3-pifacedigitalio/examples/blink.py
```

FIGURE 3.9 – Test du bon fonctionnement

Nous allons utiliser ce module pour donner un retour à l'utilisateur grâce aux LEDs. Ainsi les LEDs indiqueront si la transaction s'est passée correctement ou si une erreur s'est produite durant l'exécution.

Chapitre 4

Base de données

4.1 Types de base de données

Il existe différents types de base de données. Les plus utilisées étant les bases de données relationnelles. Ces bases peuvent être exploitées grâce au langage SQL qui est un langage de requête structurée. Il existe plusieurs systèmes de gestion de base de données relationnelles (SGBDR), les plus connus étant MySQL, PostgreSQL, SQLite et Oracle Database.

Un autre type de base de données, désigné sous le nom NoSQL (pour "not only SQL"), n'utilise pas le language SQL pour ses requêtes, par opposition aux bases de données relationnelles. Elles utilisent différentes techniques dépendant de la base de données. MongoDB utilise des requêtes sous forme de fichiers JSON, Redis utilise un langage créé spécialement pour cette base de données avec beaucoup de commandes [1], etc.. Ces bases de données peuvent stocker énormément d'informations, mais elles peuvent être moins structurées. Voici quelques systèmes de gestion de bases de données NoSQL connus :

- MongoDB : système orienté documents, utilisé par le CERN, Forbes ou SAP
- CouchDB : également orienté documents, utilisé par Ubuntu pour son défunt service Ubuntu One
- Redis : système clé-valeur, utilisé par GitHub et Stack Overflow notamment
- Cassandra : système orienté colonnes, développé par la Fondation Apache, utilisé par Twitter
- BigTable : développé et utilisé par Google (solution propriétaire)

Que ce soit des bases SQL ou NoSQL, elles sont pour la plupart "scalable" et "distributed", c'est-à-dire qu'on peut les agrandir en ajoutant des serveurs au fil du temps, de plus les données peuvent être sur différents serveurs, tous liés entre eux.

Une correspondance de nommage peut se faire entre une base de données relationnelle et une base de données orientée document comme l'est MongoDB. Dans les deux cas, nous avons affaire à des bases de données. Mais une table dans une base relationnelle correspond à une collection. Et les entrées d'une table, ou enregistrements, correspondent à des documents.

1. Commandes Redis : `http://redis.io/commands`, page consultée en juillet 2014.

4.2 MySQL

FIGURE 4.1 – Logo de MySQL

source : http://www.mysql.fr/common/logos/logo-mysql-170x115.png

Pour modéliser notre système de distributeur de balles de golf, nous pouvons créer plusieurs tables : une qui stocke les informations d'un utilisateur, une autre qui stocke les appareils (cartes ou smartphones), une autre les transactions et une dernière qui contient les distributeurs de balles. Ces tables sont liées entre elles par des relations qui permettent de faire des requêtes avec des jointures. La figure 4.2 montre le schéma SQL d'une telle base de données relationnelle.

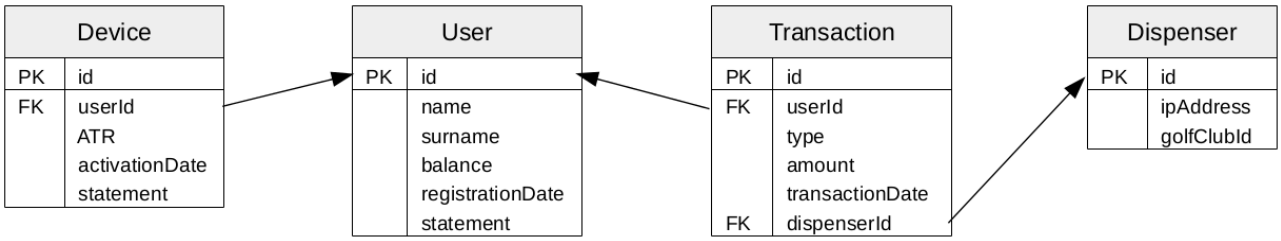


FIGURE 4.2 – Schéma SQL

En annexe C.1 se trouve un explicatif pour installer un serveur MySQL sur une machine Linux.

4.3 MongoDB

MongoDB est un système de gestion de base de données orienté document. Ce qui veut dire que chaque entrée dans la base est un document, ici en format BSON. Le BSON est une version binaire du JSON, qui est un format de fichier utilisé principalement en JavaScript, mais qui tend à se généraliser et à remplacer l'utilisation du XML pour certaines applications.

FIGURE 4.3 – Logo de MongoDB

source : http://info.mongodb.com/rs/mongodb/images/MongoDB_Logo_Full.png

La particularité de ce genre de base de données est que les collections (tables en relationnel) n'ont pas de structure prédéfinie. C'est-à-dire que nous pouvons ajouter n'importe quel document avec un nombre différent d'entrées. Ce qui ne veut pas dire que c'est ce que nous allons faire.

Une façon de modéliser ce que nous voulons pour le distributeur de balles de golf : nous utilisons seulement deux collections, à la place de quatre tables dans la version relationnelle. Une collection *User* et une collection *Transaction*. Tous les appareils d'un utilisateur seront stockés dans l'entrée *devices* de la collection *User*. Et pour chaque transaction, nous stockons les informations du distributeur de balles, comme ça nous économisons également cette table. L'idée étant de faire le moins de requêtes possibles, il est cohérent ici d'avoir

uniquement deux collections. Dans la première nous irons chercher des informations pour savoir si le client peut faire une transaction et dans la deuxième nous irons écrire les transactions du client. La figure 4.4 montre schématiquement la structure de notre base de données.

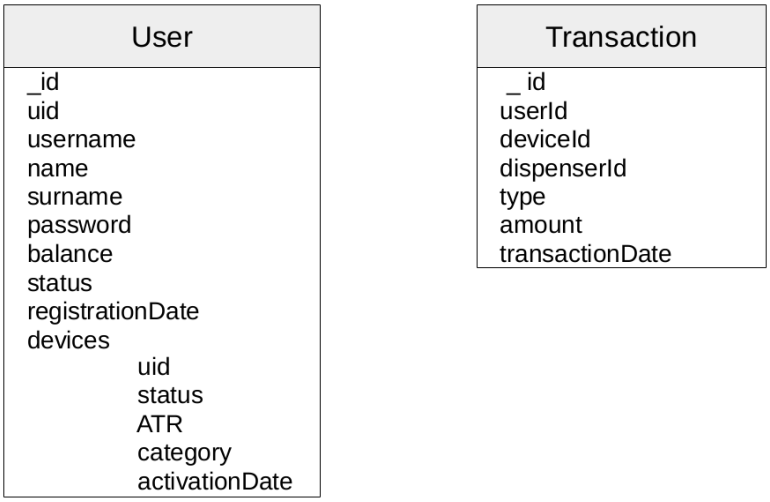


FIGURE 4.4 – Schéma de la base de données MongoDB

Pour interagir avec la base de données, nous n'utilisons pas SQL vu que c'est un SGBD NoSQL. Il s'agit d'un système orienté document. Les données sont stockées sous forme de documents, et les requêtes sont également sous forme de document JSON !

Pour utiliser MongoDB, nous utilisons la librairie *PyMongo*. L'installation d'un serveur MongoDB ainsi que de PyMongo sur une machine Linux est détaillé en annexe C.2.

4.4 NOTRE CHOIX

Les bases de données SQL sont plus répandues et permettent de faire des requêtes très précises en faisant des jointures entre les tables. Les bases NoSQL sont plus souples et permettent de stocker énormément de données.

Les bases de données NoSQL sont des technologies beaucoup plus récentes que les bases de données relationnelles. Les deux types fournissent des librairies pour pouvoir développer en plusieurs langages dont le C, le C++, le Java, le PHP, le Python et le JavaScript.

Nous choisissons ici de travailler avec MongoDB parce que c'est une technologie plus récente et plus souple, mais également pour en apprendre plus sur ce type de base de données et ses particularités.

Chapitre 5

Sécurisation et accessibilité

5.1 Problématique

Il peut y avoir plusieurs Raspberry Pi disséminés dans la nature, avec leur lecteur de carte. Par contre, il faut un endroit unique où stocker les données des clients pour que chaque distributeur de balles puisse accéder à la base de données. Nous allons donc installer la base de données sur un serveur distant.

Il faut que la communication entre les distributeurs et le serveur soit sécurisée car il ne faudrait pas que n'importe qui puisse connaître les identifiants ou numéros de compte d'un client en écoutant le réseau. Pour ce faire, nous allons mettre en place un tunnel SSH entre chaque distributeur et le serveur, qui se trouve sur un réseau privé (ici : le réseau interne de la HES-SO de Sion).

5.2 Caractéristiques d'un tunnel SSH

Un tunnel SSH utilise la technologie SSH. Voici ce que Wikipédia nous dit de SSH :

> Secure Shell (SSH) est à la fois un programme informatique et un protocole de communication sécurisé. Le protocole de connexion impose un échange de clés de chiffrement en début de connexion. Par la suite, tous les segments TCP sont authentifiés et chiffrés. Il devient donc impossible d'utiliser un sniffer pour voir ce que fait l'utilisateur. [1]

Ce protocole est donc sécurisé, mais en plus, avec un tunnel SSH, nous pouvons faire du *port forwarding*, ou de la redirection de ports. En effet, nous voulons envoyer des requêtes sur un port local qui est redirigé dans le tunnel. Et à l'autre extrémité du tunnel, nous voulons que ce qui est reçu sur le port 22 soit redirigé ailleurs, selon notre utilité.

5.3 Mise en place d'un tunnel SSH

Pour mettre en place un tunnel SSH avec redirection locale, il faut entrer une commande qui a la syntaxe suivante :

```
1  ssh -L [port d entree du tunnel]:[adresse du client]:[port de sortie du tunnel]
2      [utilisateur@adresse du serveur distant]
```

FIGURE 5.1 – Syntaxe de la commande pour créer un tunnel SSH

1. Source : `http://fr.wikipedia.org/wiki/Secure_Shell`, page consultée en mai 2014.

Dans notre cas, nous choisissons le port 28082 par exemple comme port d'entrée du tunnel. Notre message va passer dans le tunnel et arriver sur le serveur SSH sur le port 22. Ce serveur va rediriger la requête vers le port 27017, qui est le port par défaut pour accéder à la base de données MongoDB qui se trouve sur le serveur. Voici à quoi ressemble la commande :

```
1  ssh -L 28082:localhost:27017 admin@vlenfc.hevs.ch
```

FIGURE 5.2 – Commande pour créer notre tunnel SSH

La figure 5.3 montre schématiquement la manière dont le script Python accède à la base de donées MongoDB hébergée sur le serveur `vlenfc.hevs.ch` qui se trouve dans le réseau de l'école. Dans notre script Python, il suffit de changer la connection à la base de données `"mongodb://localhost:27017"` en `"mongodb://localhost:28082"`. Ainsi la requête est dirigée dans le tunnel.

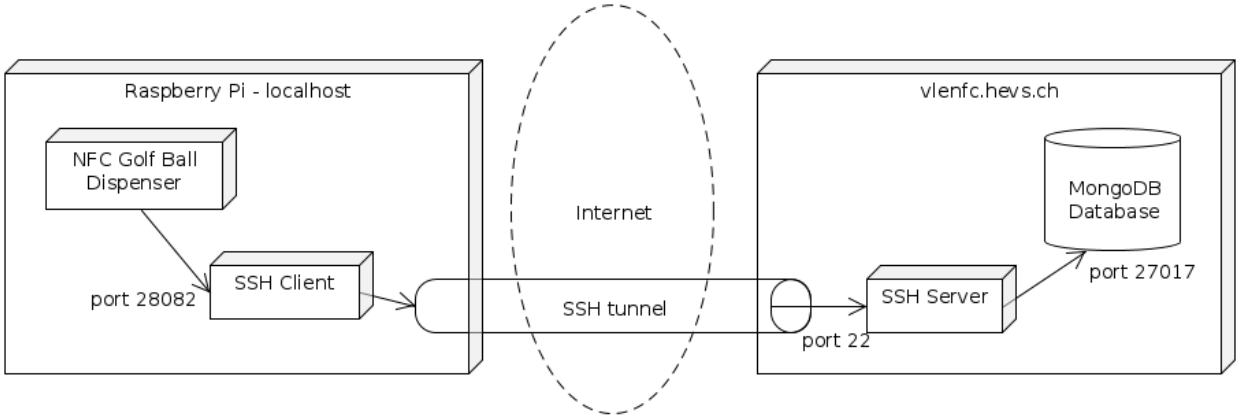


FIGURE 5.3 – Schéma de notre tunnel SSH

Pour détruire ce tunnel, il faut entrer la commande `exit`. Si nous voulons sauvegarder la commande de création du tunnel, nous pouvons créer un alias, comme ci-dessous. Pour créer le tunnel par la suite, il suffit d'entrer `tunnelnfc`.

```
1  alias tunnelnfc='ssh -L 28082:localhost:27017 admin@vlenfc.hevs.ch'
```

FIGURE 5.4 – Exemple d'alias pour la création du tunnel SSH

Chapitre 6

Logiciel sur le Raspberry Pi

6.1 Schéma bloc

Pour schématiser physiquement le projet, nous utilisons un schéma bloc. Celui-ci se trouve à la figure 6.1.

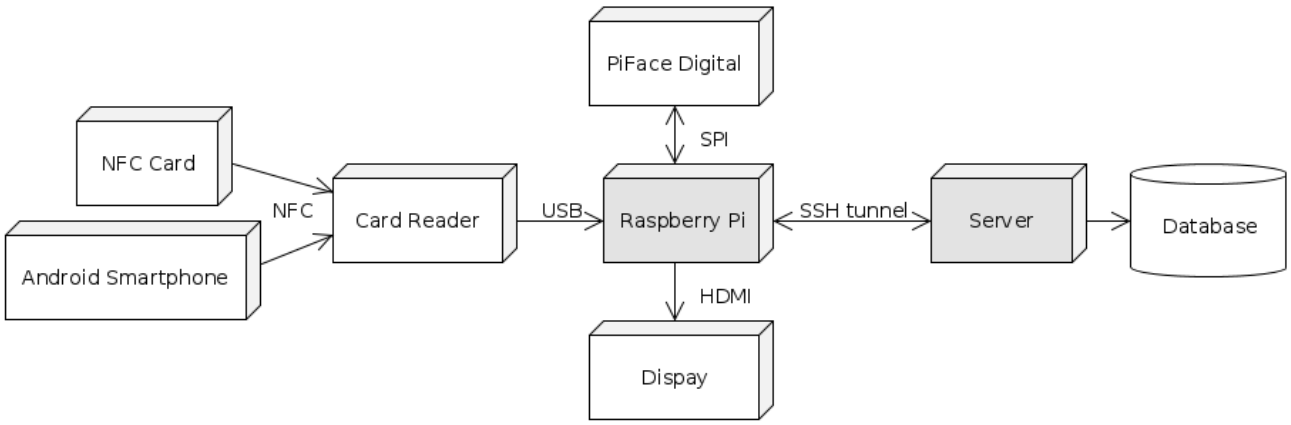


FIGURE 6.1 – Schéma bloc

Le logiciel principal se trouve sur le Raspberry Pi. Celui-ci gère le lecteur NFC qui attend qu'une carte NFC ou un smartphone Android NFC s'approche. Il communique avec un serveur distant relié à une base de données. L'utilisateur peut presser les boutons du module *PiFace* pour interagir avec le système et les résultats sont affichés sur l'écran et les LEDs du *PiFace*.

6.2 Conception du logiciel

Pour mettre en place le logiciel, nous devons modéliser le tout. Nous avons donc notre logiciel en Python qui tourne sur le Raspberry Pi auquel est connecté le lecteur de carte. Lorsqu'une carte s'approche du lecteur, elle est lue (son identifiant ainsi que son ATR sont lus). Là l'utilisateur peut choisir ce qu'il veut faire. Il peut débiter des crédits de son compte pour recevoir des balles de golf, voir les dernières transactions, ajouter un appareil à son compte. Peu importe l'action qu'il choisit, on va contrôler s'il a le droit de le faire en faisant la bonne requête à la base de données qui se trouve de l'autre côté d'un tunnel SSH, sur le serveur. S'il a le droit, on va exécuter l'action et notifier à l'utilisateur que tout s'est bien passé et s'il n'a pas le droit, l'action n'est pas effectuée et le client est notifié également. La notification se fait sur l'écran qui est branché au Raspberry Pi mais également grâce aux leds du module PiFace. La figure 6.2 montre le diagramme de classe auquel nous sommes arrivés. En annexe se trouve un diagramme de classe complet avec les attributs, les méthodes, les signaux et les slots. Il s'agit de l'annexe H.2.1.

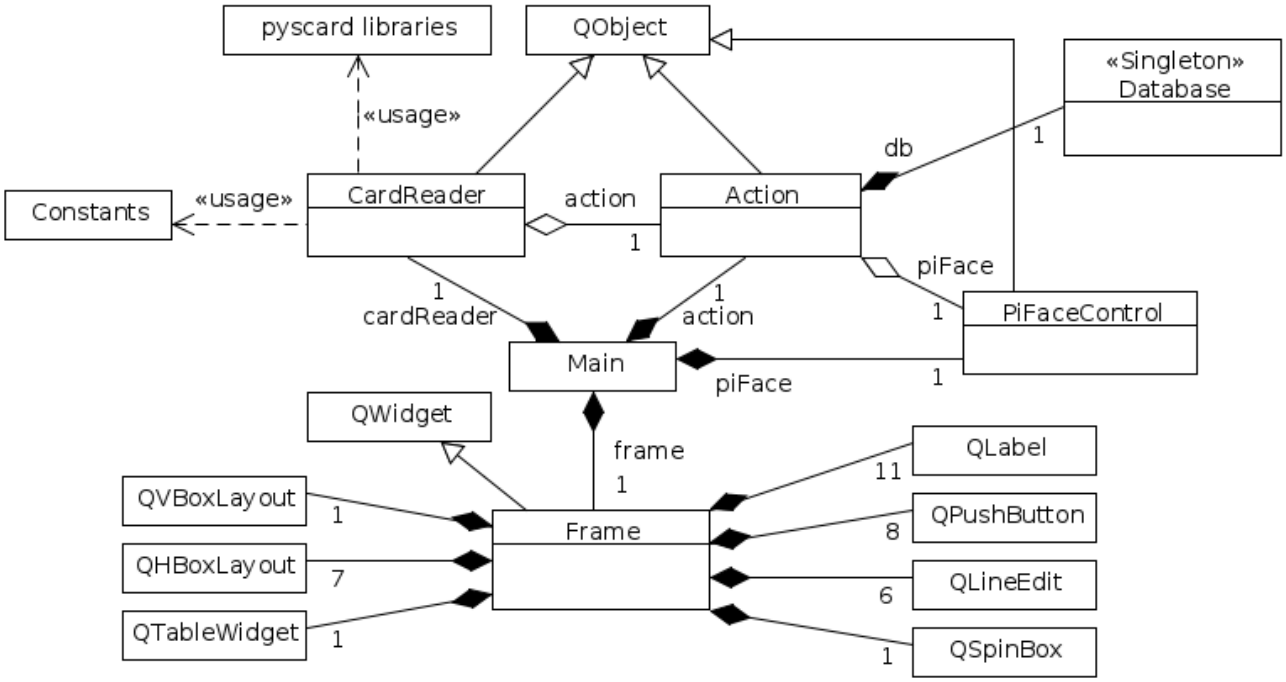


FIGURE 6.2 – Diagramme de classe

Le `main`, au centre, fait office de `factory`. Il crée une instance de la classe `CardReader`, une instance de la classe `Action` et une instance de la classe `Frame` et lie l'instance d'`Action` avec celle de `CardReader`. Ensuite le `main` s'occupe de lier les *"signals"* aux *"slots"* (voir chapitre 6.3). Et pour finir il entre dans une boucle d'exécution pour la fenêtre graphique.

L'instance de `CardReader` attend qu'une carte s'approche du lecteur et lorsqu'une carte est détectée, il collecte l'identifiant et l'ATR et attend que l'utilisateur ordonne une action. Pour communiquer avec le lecteur de carte, cette classe utilise les librairies `pyscard`. Une fois que l'utilisateur a choisi ce qu'il veut faire, l'action correspondante est appelée dans l'objet de la classe `Action`. Dans l'action, on accède à la base de données par l'unique instance de la classe `Database`, on fait les contrôles nécessaires et on indique le résultat à l'utilisateur sur l'interface graphique.

Le détail de chaque classe ainsi que la documentation générale du logiciel a été faite. Pour la visionner, il suffit d'ouvrir le fichier `index.html` qui se trouve à l'adresse `doc/html/index.html` dans le dossier du projet nommé `dispenserui`. Le code est également disponible sur Github, voir annexe H pour plus d'informations.

6.3 PySide - INTERFACE GRAPHIQUE

6.3.1 Présentation et installation

Pour créer notre interface graphique nous allons utiliser *PySide*. Il s'agit d'un wrapper Python de la librairie Qt, qui est normalement utilisé en C++. Qt est un framework qui permet de développer facilement des interfaces graphiques qui sont déployables sur plusieurs systèmes d'exploitation comme Linux, Windows, OSX, Maemo et MeeGo. Une autre alternative aurait été *PyQt*, qui est également un wrapper de Qt. *PyQt* possède une licence moins permissive que *PySide*, c'est pour cette raison que Nokia a développé *PySide*. *Pyqt* est sous licence GNU GPL [1] (GNU General Public License), tandis que *PySide* est sous licence GNU LGPL [2] (GNU Lesser General Public License).

1. Licence GNU GPL : `https://www.gnu.org/licenses/gpl.html`, page consultée en juin 2014.
2. Licence GNU LGPL : `https://www.gnu.org/licenses/lgpl.html`, page consultée en juin 2014.

FIGURE 6.3 – Logo de PySide

source : http://qt-project.org/wiki/PySide_Logo

Pour installer *PySide*, la commande à écrire est la suivante :

```
1  sudo apt-get install python-pyside
```

FIGURE 6.4 – Installation de PySide

Cette commande est valide sur une machine Ubuntu et Debian (y compris le Raspberry Pi sous Raspbian) et fonctionne avec Python 2.x.

6.3.2 Fonctionnement

Pour créer une application avec interface graphique en Qt c'est assez simple. Voici le code minimal pour y arriver (figure 6.5) et le résultat obtenu (figure 6.6) :

```
1  import sys
2  from PySide import QtCore, QtGui
3
4  app = QtGui.QApplication(sys.argv)
5  window = QtGui.QWidget()
6
7  window.resize(300, 100)
8  window.setWindowTitle("Hello, World!")
9  window.show()
10
11 sys.exit(app.exec_())
```

FIGURE 6.5 – Application minimale avec *PySide*

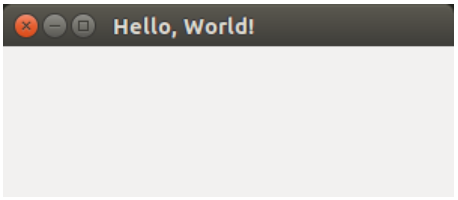


FIGURE 6.6 – Application minimale avec *PySide*

Nous obtenons une simple fenêtre avec le titre "Hello, World !" et qui fait 300x100 pixels. Le style de la fenêtre s'adapte au style de l'interface du système d'exploitation surlequel l'application est exécutée. Dans l'aperçu précédent, il s'agit de l'interface de *Unity* sur Ubuntu. Nous pouvons voir le résultat sur Windows 7 et sur Raspbian, le système d'exploitation du Raspberry Pi, à la figure 6.7.

(a) (b)

FIGURE 6.7 – (a) Fenêtre sur Windows 7 (b) Fenêtre sur Raspbian

Il suffit d'importer la bibliothèque *PySide*, de créer une `QApplication` puis un `QWidget` qui sera notre fenêtre. Il faut ensuite paramétrer un peu notre fenêtre et l'afficher avec `window.show()`. Et pour finir la dernière instruction fait entrer le programme dans une boucle d'exécution pour empêcher le programme de se terminer.

Il faut ensuite mettre des éléments dans cette fenêtre. Pour ce faire, plusieurs `QWidget` sont disponibles dans la librairie. En exemple nous pouvons citer les boutons, les labels, les lignes de texte éditables, les tableaux, les listes, etc. Et pour structurer ces éléments, nous avons besoin de *layouts*. Ils permettent de définir si les éléments sont affichés en tableau, les uns en-dessous des autres, sur la même ligne, etc.

Si nous voulons par exemple un champ de texte suivi d'un bouton, et que les deux sont positionés à la verticale, il nous faut le code suivant (figure 6.8) :

```
1  import sys
2  from PySide import QtCore, QtGui
3
4  app = QtGui.QApplication(sys.argv)
5  window = QtGui.QWidget()
6
7  window.resize(300, 100)
8  window.setWindowTitle("Hello, World!")
9
10 textField = QtGui.QLineEdit()
11 button = QtGui.QPushButton('Enter')
12
13 verticalLayout = QtGui.QVBoxLayout()
14 verticalLayout.addWidget(textField)
15 verticalLayout.addWidget(button)
16
17 window.setLayout(verticalLayout)
18
19 window.show()
20
21 sys.exit(app.exec_())
```

FIGURE 6.8 – Ajout d'éléments graphiques

Nous créons un `QLineEdit` ainsi qu'un `QPushButton` que nous insérons dans un *layout* vertical de la classe `QVBoxLayout`. Et pour finir nous attribuons ce *layout* à la fenêtre qu'on a créé précédemment. Le résultat est visible à la figure 6.9.

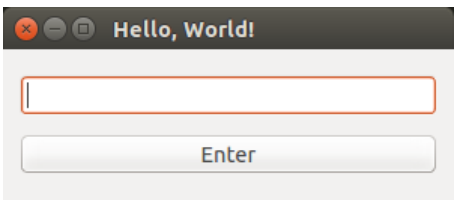


FIGURE 6.9 – Résultat

6.3.3 Signal/Slot

Une des particularités de *Qt* est sa possibilité d'utiliser des *"signals"* et des *"slots"* pour faire de la programmation événementielle. En effet, une interface graphique doit être réactive et réagir en fonction d'événements, par exemple quand on clique sur un bouton, une action est déclenchée.

- Signal : Il s'agit d'un message qui est envoyé lorsqu'un événement se produit.
- Slot : C'est une méthode qui est appelée lorsqu'un signal a été envoyé.

Concrètement, on doit connecter un *signal* à un *slot* pour que lorsque l'événement se produit, le *signal* appelle le *slot*. Cela permet également de faire un lien entre l'interface graphique et les classes métiers sans faire d'aggrégation de classes. Ainsi les fonctionnalités sont bien séparées.

Le code de la figure 6.10 montre comment connecter un bouton à un slot. Lorsqu'on clique sur le bouton `button` de l'objet `window` précédemment créé, la méthode `doSomething()` de l'objet `action` va être appelée.

```
1 window.button.clicked.connect(action.doSomething)
```

FIGURE 6.10 – Connexion d'un bouton

La méthode `doSomething()` est un slot, il faut donc la déclarer en conséquence. La figure 6.11 montre un exemple de code d'un slot. Il faut précéder le nom de la méthode du décorateur `@Slot()`.

```
1 @Slot()
2 def doSomething(self):
3     print 'Im doing something'
```

FIGURE 6.11 – Déclaration d'un *slot*

Ce qu'il est possible également de faire avec ce principe est de lancer une méthode à la fin d'un *timer*. La connexion entre le timer et le slot `updateCounter()` est visible à la figure 6.12. Il faut préalablement créer le timer `timer1`, ce qui est fait à la figure 6.13. Dans l'exemple, le timeout est réglé à 500 ms, donc chaque demi-seconde la méthode `updateCounter()` sera appelée.

```
1 window.connect(window.timer1, SIGNAL("timeout()"), window.updateCounter)
```

FIGURE 6.12 – Connexion d'un timer

```
1 timer1 = QTimer()
2 timer1.start(500)
```

FIGURE 6.13 – Déclaration et lancement d'un timer

Finalement, nous pouvons avoir un vrai *signal* relié à un *slot*. Un *signal* est un attribut static d'une classe, nous pouvons le déclarer comme à la figure 6.14. Il faut également déclarer le *slot* (figure 6.15). Cette fois-ci, la méthode à un argument sous forme d'une chaîne de caractères. La figure 6.16 montre comment connecter le *signal* au *slot*. Et pour finir, dans l'objet de la classe `Action` il faut émettre le *signal* comme le montre le code de la figure 6.17.

```
1  status = Signal()
```

FIGURE 6.14 – Déclaration d'un *signal*

```
1  @Slot(str)
2  def displayStatus(self, message):
3      self.statusLabel.setText(message)
```

FIGURE 6.15 – Déclaration du *slot* avec un argument de type `String`

```
1  action.status.connect(window.displayStatus)
```

FIGURE 6.16 – Connexion d'un *signal* à un *slot*

```
1  self.status.emit('Status : Please place a card in front of the reader.')
```

FIGURE 6.17 – Emission d'un *signal*

6.4 UTILISATION

Avant de lancer le logiciel, il faut bien évidemment créer le tunnel SSH entre le Raspberry Pi et le serveur qui contient la base de données. La création de ce tunnel est expliquée au chapitre 5.3.

Une fois cette étape terminée, il faut se déplacer dans le dossier du projet et exécuter la commande "`python main.py`". Le logiciel s'ouvre. Il suffit ensuite de le mettre en plein écran ou de l'adapter selon votre préférence.

Un utilisateur normal, c'est-à-dire la personne qui vient retirer des balles de golf et veut voir ses dernières transactions, n'a pas besoin d'un clavier ni d'une souris. En effet, les quatre boutons d'action situés au fond de l'interface ont le même effet que la pression sur les quatre boutons physiques situés sur le module PiFace Digital connecté au Raspberry Pi (figure 6.18). Il suffit donc de presser sur le premier ou le deuxième bouton pour retirer des balles et sur le troisième pour voir les transactions.

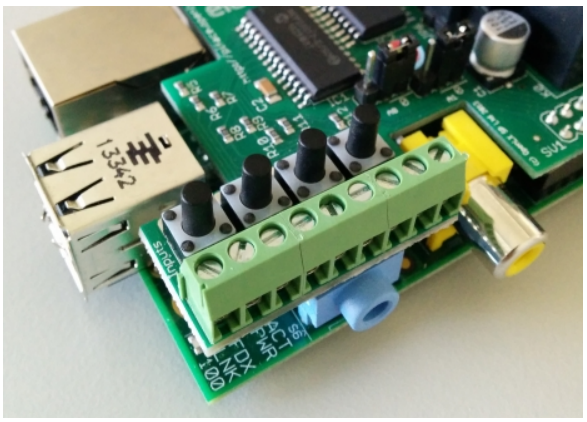


FIGURE 6.18 – Les 4 boutons du module PiFace Digital

L'interface d'administration est également accessible en cliquant sur le quatrième bouton pressoir, mais une fois dans l'admin, il est nécessaire d'avoir un clavier et éventuellement une souris pour entrer le texte dans les différents champs.

Pour fermer le logiciel, il est possible de cliquer sur la petite croix au sommet de la fenêtre, comme un programme standard, ou alors d'appuyer sur la touche *ESC*.

6.5 RÉSULTAT

La figure 6.19 montre le résultat final du logiciel. Ce logiciel est conçu de façon à adapter le contenu de la fenêtre en fonction de sa taille. Nous pouvons donc avoir un visuel cohérent, peu importe l'écran qu'on veut utiliser.

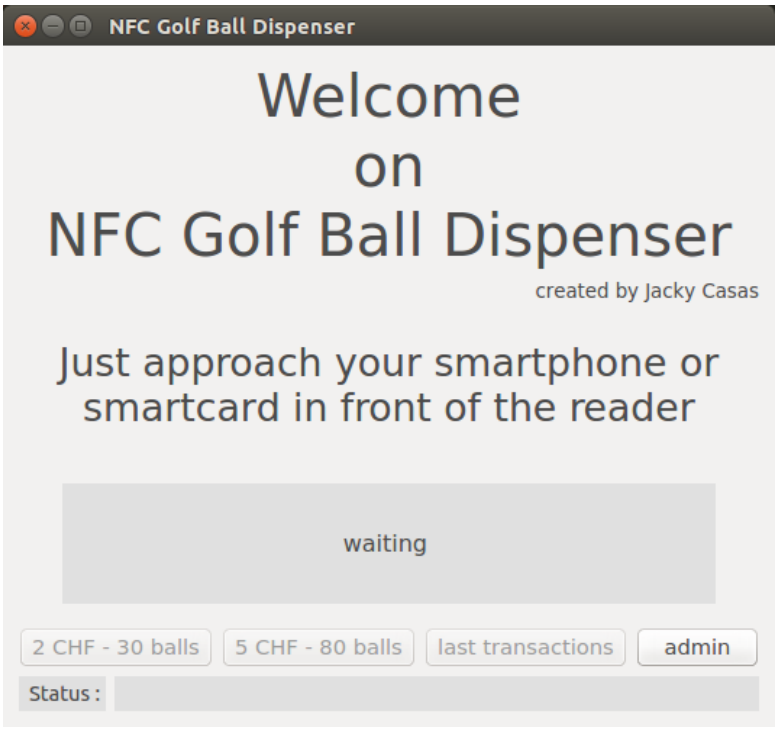


FIGURE 6.19 – Aperçu du logiciel

La structure du logiciel est visible à la figure 6.20. Nous avons un texte d'accueil ainsi qu'un texte explicatif qui indique à l'utilisateur d'approcher sa carte ou son smartphone du lecteur (point 1). Ensuite nous voyons un encadré gris dans lequel il est écrit *waiting* avec des petits points qui s'écrivent et s'effacent pour montrer l'état d'attente (point 2). En dessous de cet encadré se trouvent quatre boutons (point 3), dont trois sont pour l'instant grisés car ce sont des actions possibles uniquement lorsqu'une carte est lue par le lecteur. Puis pour finir nous avons un dernier encadré gris qui indique le statut de l'application (point 4). Des messages seront affichés en fonction de l'état du programme et des actions effectuées.

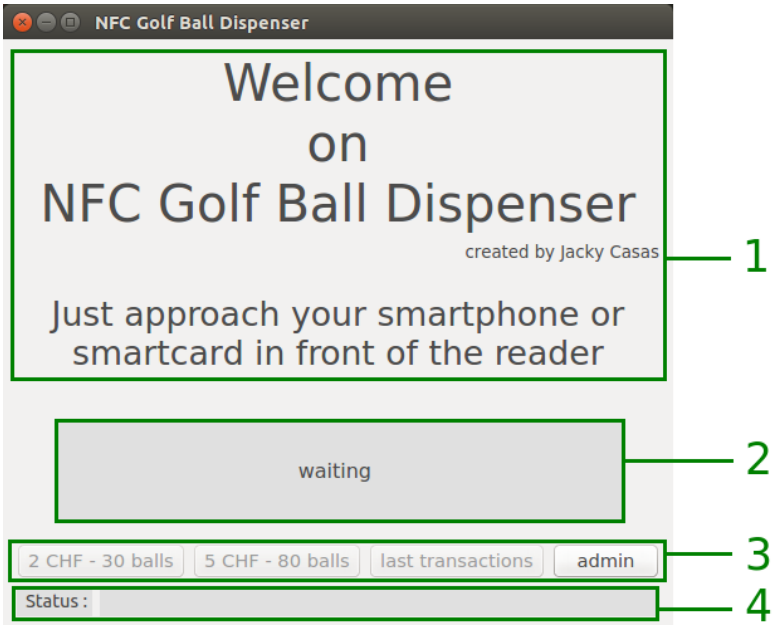


FIGURE 6.20 – Structure du logiciel

Pour construire cette interface, des *layouts* ont été utilisés. La fenêtre elle-même est un *layout* vertical, `QVBoxLayout` en Qt, c'est à dire que chaque élément ajouté va se positionner en dessous du précédent. Dans un *layout*, nous pouvons mettre des éléments, ou `QWidget`, mais également d'autres *layouts*. Le texte d'accueil est par exemple un simple `QLabel` que nous avons ajouté au *layout*, tandis que la ligne des quatre boutons est un *layout* horizontal, `QHBoxLayout`, qui contient quatre `QPushButton`. Chaque élément de cette interface n'a pas de taille fixe mais adapte sa taille en fonction du nombre d'éléments à afficher et du contenu de l'élément.

Tant qu'aucune carte n'est détectée, le lecteur est en attente. Pour montrer cette attente, le mot *"waiting"* est affiché et les trois petits points s'affichent et se cachent selon le cycle visible à la figure 6.21. On passe d'une étape à l'autre chaque demi-seconde.

FIGURE 6.21 – Cycle d'affichage pour l'attente

Lorsqu'un utilisateur approche sa carte NFC vers le lecteur, le logiciel va la détecter, va récupérer l'identifiant et l'ATR sur la carte puis va chercher en base de données des informations sur la carte. La figure 6.22 montre l'affichage lorsque la carte correspond à un compte. Dans l'encadré gris se trouve le solde du compte et les trois boutons précédemment bloqués sont accessibles. Les boutons physiques du Raspberry Pi sont également activés. Le premier sert à débiter 2 CHF sur le compte et à libérer 30 balles de golf, le deuxième permet de débiter 5 CHF et de libérer 80 balles et le troisème permet d'afficher les dix dernières transactions effectuées sur le compte, que ça soit une recharge du compte ou un débit d'argent. Ce fonctionnement est illustré par le diagramme de séquence de la figure 6.23.

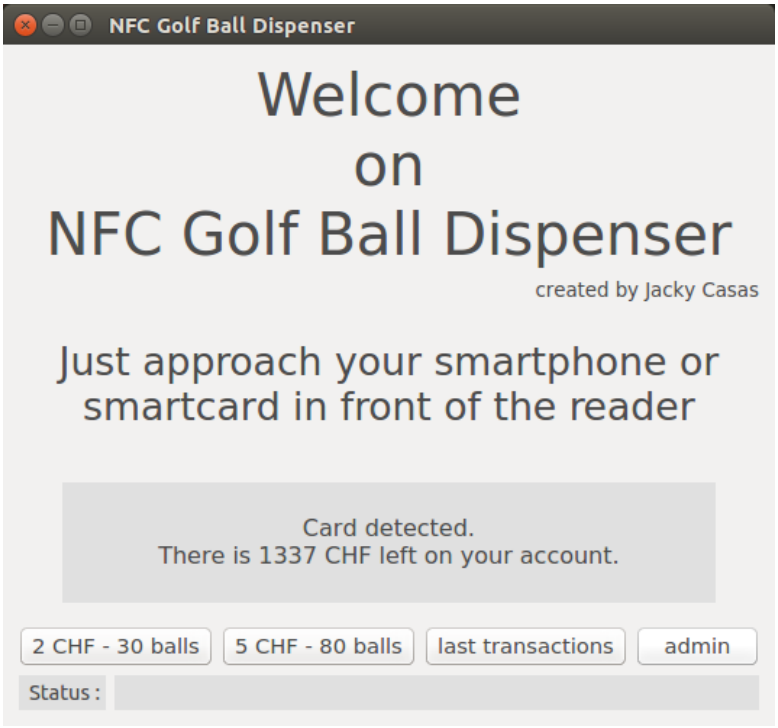


FIGURE 6.22 – Une carte est détectée

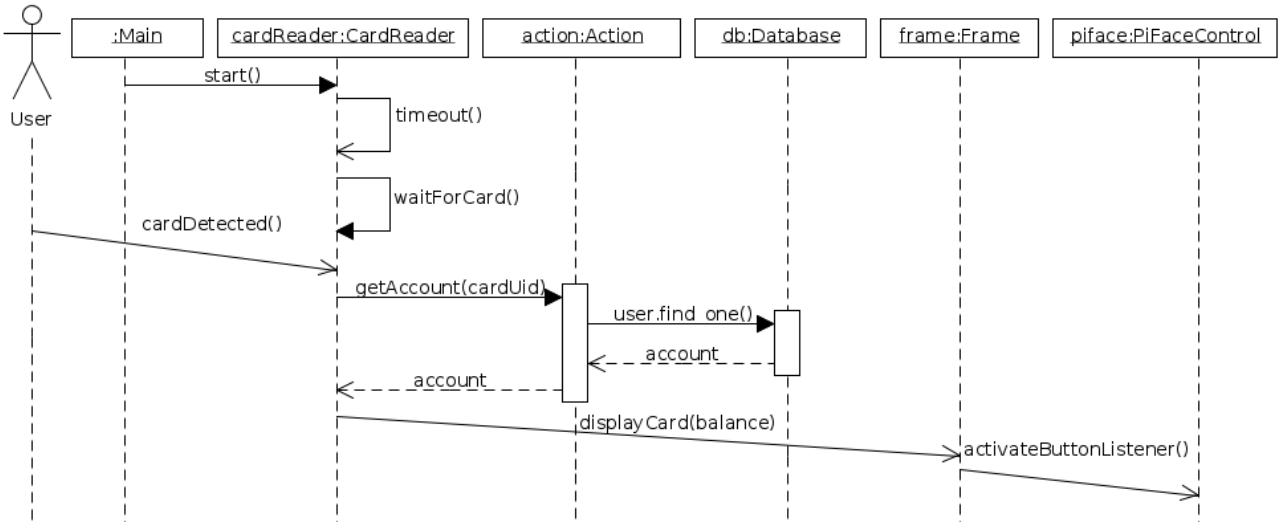


FIGURE 6.23 – Diagramme de séquence : détection d'une carte NFC

Si par contre la carte n'est reliée à aucun compte, le programme ne va pas la trouver dans la base de données et un avertissement est affiché (figure 6.24). De plus les boutons ne sont pas activés, ce qui est logique.

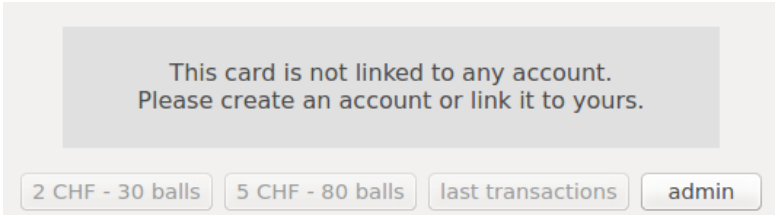


FIGURE 6.24 – La carte est liée à aucun compte

Admettons maintenant que la carte soit valide. Si l'utilisateur presse sur le bouton *"5 CHF - 80 balls"*, la transaction va s'effectuer après vérification du solde du compte. Une nouvelle entrée dans la collection des transactions de la base de données est créée et l'utilisateur va être notifié de la réussite ou non dans la barre de statut située au fond de la fenêtre (voir figure 6.25).

Si la transaction se passe bien, le montant est débité, le nouveau solde est affichée et les LEDs du module PiFace Digital clignotent. A l'inverse, si la transaction est refusée, la raison de ce refus est affichée et les LEDS clignotent d'une autre façon pour indiquer l'échec de la transaction. Un refus de transaction peut être dû au fait qu'il n'y a plus d'argent sur le compte, que la carte de l'utilisateur a été bloquée ou autre. Un cas de réussite de la transaction est imagé par un diagramme de séquence à la figure 6.26.

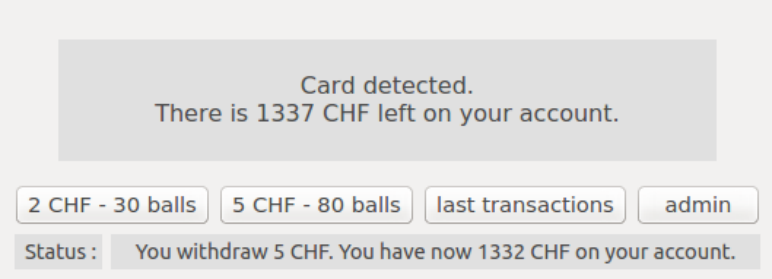


FIGURE 6.25 – Retrait de 5 CHF et libération de 80 balles de golf

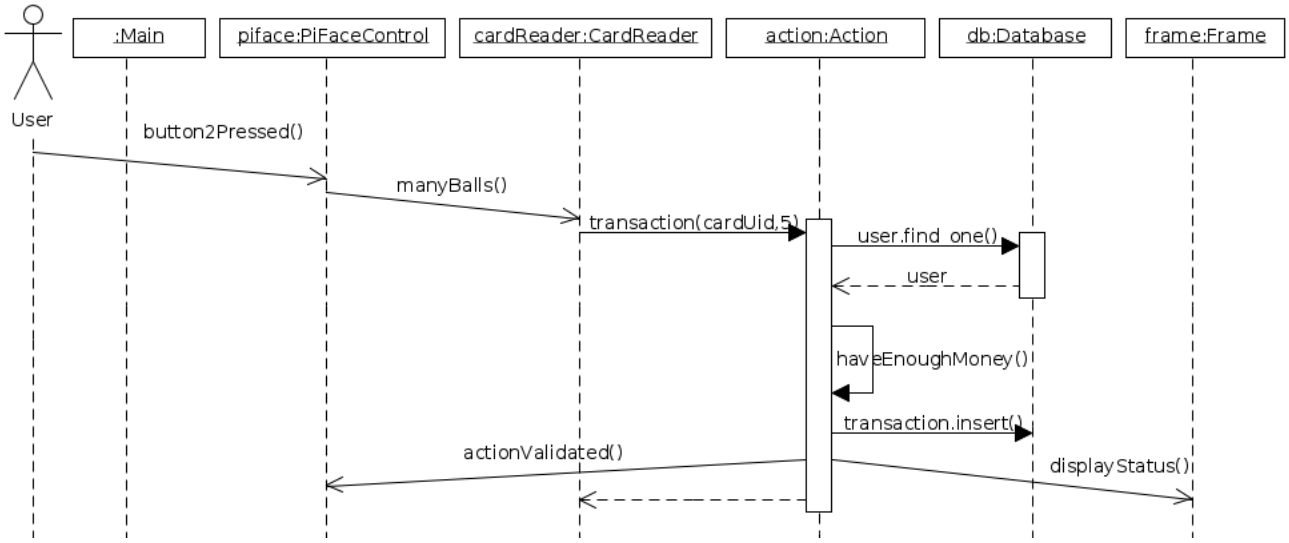


FIGURE 6.26 – Diagramme de séquence : transaction de 5 CHF

Ensuite en cliquant sur le bouton *"last transactions"*, le logiciel affiche les 10 dernières transactions de l'utilisateur (figure 6.27). Pour masquer à nouveau les transactions, il suffit de cliquer à nouveau sur le bouton, comme indiqué dans la barre de statut. Pour afficher ces transactions, une nouvelle requête en base de données est effectuée.

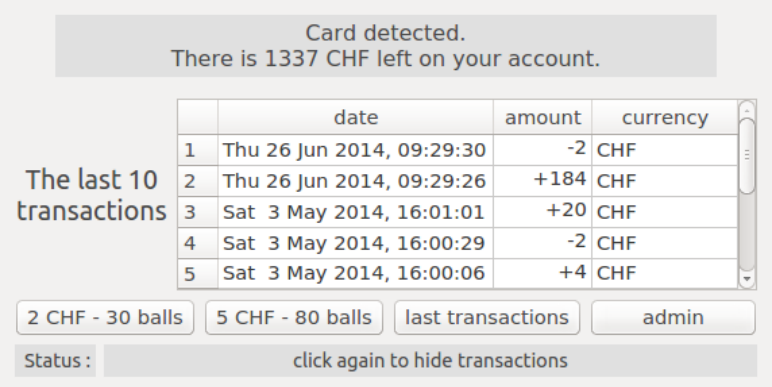


FIGURE 6.27 – Affichage des dernières transactions

L'utilisateur standard peut uniquement faire ces quelques actions et n'a pas besoin de plus. Par contre il peut être utile d'avoir accès à des fonctions plus avancées, ces fonctions sont accessibles uniquement par un administrateur et elles sont accessibles en cliquant sur le bouton *"admin"*. La pression de ce bouton changera le design de l'application et proposera des champs de login et mot de passe que l'on voit à la figure 6.28. Dans cette version, l'identifiant ainsi que le mot de passe sont stockés en dur dans le code de l'application. Pour une mise en production il sera préférable d'ajouter la possibilité de créer un compte administrateur et de laisser choisir le nom ainsi que le mot de passe. En effet, plusieurs personnes doivent pouvoir administrer les distributeurs de balles.

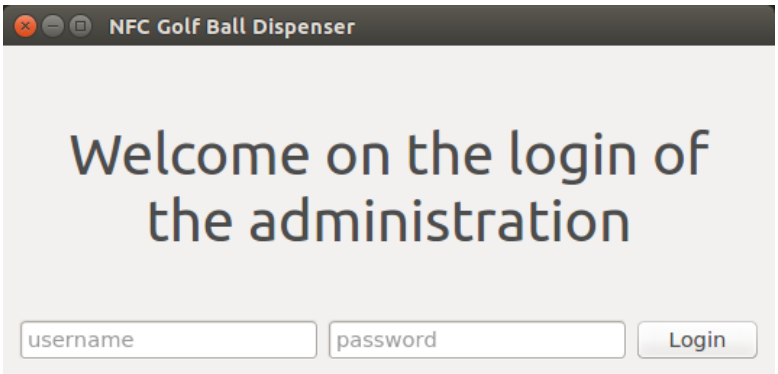


FIGURE 6.28 – Menu de log de l'espace d'administration

Une fois le nom d'utilisateur et le mot de passe entrés correctement, nous arrivons sur l'interface d'administration visible à la figure 6.29. Dans cet espace nous avons toujours les fonctionnalités précédentes, mais des nouvelles se sont ajoutées. Il y a premièrement la possibilité de recharger un compte. Pour ce cas, nous imaginons que le golfeur qui souhaite recharger son compte s'adresse à un administrateur. Cet adminstrateur encaisse le montant souhaité et il va ensuite dans l'administration, entre le montant et clique sur le bouton *"recharge"*. Pour que cela fonctionne, la carte du client doit bien évidemment être détectée par le lecteur pour que le logiciel sache sur quel compte créditer le montant.

La deuxième action est la possibilité de créer un compte pour un nouveau client. Pour ce faire, il suffit d'entrer un identifiant dans la case *"username"* et de cliquer sur *"Create account"*. Le nom et le prénom sont facultatifs. L'identifiant quant à lui est obligatoire et il doit être unique. S'il est déjà utilisé, une indication sera faite dans la barre de statut. Cette action ne requiert pas d'avoir de carte NFC.

Et pour finir, nous pouvons lier une carte NFC à un compte. Il est possible d'ajouter un carte à un compte nouvellement créé qui n'a encore aucune carte liée ou alors d'ajouter une nouvelle carte à un compte qui a déjà une ou plusieurs. Ce dernier cas peut s'illustrer comme ceci : un golfeur veut donner accès à son compte à son fils, ou alors il a perdu sa précédente carte. Pour ce faire, nous approchons la nouvelle carte du lecteur, indiquons le nom d'utilisateur et cliquons sur *"Add smartcard"*. A ce moment, le logiciel va vérifier si le compte

existe et si la carte n'est pas déjà attribuée à un autre compte. Si une erreur se produit, la barre de statut indique la source du problème et sinon l'ajout est validé.

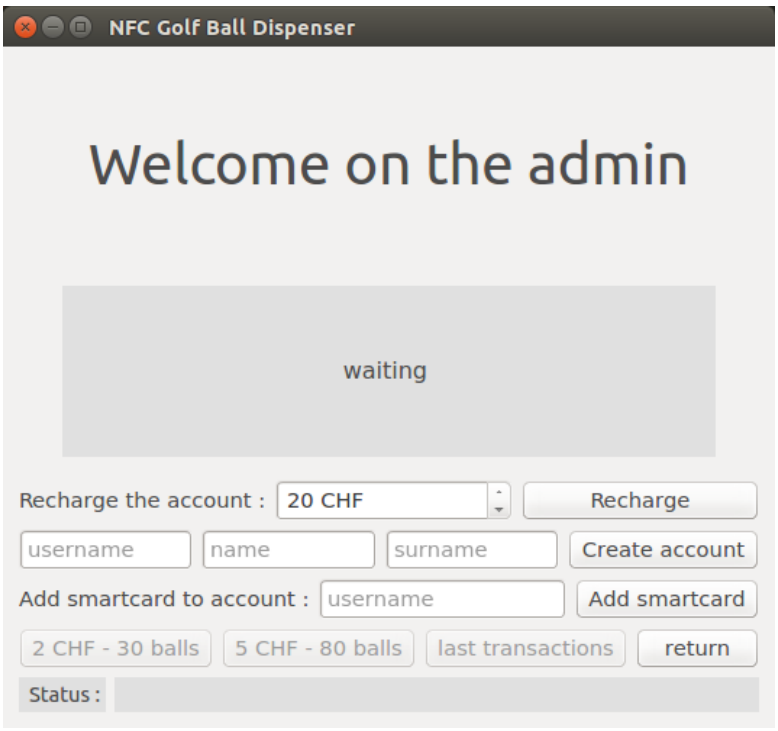


FIGURE 6.29 – Espace d'administration

Chapitre 7

API

7.1 Contexte

Jusqu'à présent, seul le lecteur NFC pouvait communiquer avec la base de données. Mais nous avons également besoin d'y accéder depuis l'application Android pour afficher les informations du client et surtout pour se connecter au compte et émuler une device NFC qui correspond au client. Pour ce faire, nous avons développé des APIs sur le serveur. Ces APIs sont codées en Python avec le framework Flask et sont accessibles par HTTP.

7.2 Flask et les autres librairies

Figure 7.1 – Logo de Flask

source : `http://flask.pocoo.org/static/logo/flask.png`

Flask est une librairie, ou plus spécifiquement un *microframework* comme le site de Flask l'explique, qui permet de faire du développement web en Python de manière plus aisée. Nous n'utilisons pas toute la puissance de Flask, mais spécifiquement la possibilité d'implémenter facilement des APIs REST. Donc toute la partie de génération de page web ne nous intéresse pas. Voyons plus en détail de quoi nous avons besoin :

- **Flask** : Microframework pour développement web.
- **Flask-HTTPAuth** : Librairie pour faire de l'authentification par HTTP, ce qui nous permettra de donner l'accès à certaines ressources uniquement si l'utilisateur est authentifié.
- **Flask-PyMongo** : Librairie permettant de lier l'application Flask à notre base de données MongoDB. Flask est extensible et permet la gestion de plusieurs types de bases de données, ce qui le rend très flexible.
- **passlib** : Librairie Python qui fournit des outils pour hacher des mots de passe. Nous l'utilisons pour hacher les mots de passe avant de les enregistrer en base de données.

- **itsdangerous** : Librairie utilisant HMAC et SHA1 pour hacher et signer des informations. Nous l'utilisons pour hacher la clé secrète avec une durée et lier tout ça à l'identifiant de l'utilisateur. Cela constitue un token qui sera valable seulement pour un utilisateur et pour un temps déterminé.

Pour installer toutes ces librairies Python, quelques appels à la commande `pip` suffisent :

```
1 pip install Flask
2 pip install Flask-HTTPAuth
3 pip install Flask-PyMongo
4 pip install passlib
5 pip install itsdangerous
```

FIGURE 7.2 – Installation des librairies Python utilisées

Notons que pour se simplifier, il suffit de faire un *"pip install"* sur le fichier *"requirements.txt"* du projet comme ceci :

```
1 pip install -r requirements.txt
```

FIGURE 7.3 – Installation des librairies Python utilisées grâce au fichier *requirements.txt*

Ce fonctionnement est expliqué dans les annexes concernant *pip*, voir annexe E.2.

7.3 MOT DE PASSE

Le mot de passe d'un utilisateur va être haché avant d'être stocké sur le serveur. Cela évite premièrement à des administrateurs d'aller voir des mots de passe dans la base de données, deuxièmement cela évite à quiconque qui pourrait trouver le mot de passe de manière légale ou non d'avoir accès aux mots de passe. Et troisièmement ça évite de pouvoir faire pression sur un administrateur pour y avoir accès, puisqu'il ne sont pas stockés en clair.

Voici donc la méthode d'encodage des mots de passe. Nous utilisons la librairie *"passlib"* énoncée plus haut, ou plus précisément sa classe `custom_app_context`. La figure 7.4 montre un exemple d'utilisation de cette librairie dans la console Python.

```
1 >>> from passlib.apps import custom_app_context
2 >>> pwd_hash = custom_app_context.encrypt('ThisIsThePasswordIWantToEncrypt')
3 >>> pwd_hash
4 '$6$rounds=90601$mGWjlqsrSIK4ZHMY$aeYyI85Q9Wx.W4fIvXKYlz46Ko7xZTFIoTzLuV7Mnlu.7KsxZkiyn19GaBrMDfxf2
5                                                                    PEhpp.N7CxaE67ztWlw/1'
6 >>> custom_app_context.verify('ThisIsThePasswordIWantToEncrypt', pwd_hash)
7 True
8 >>> custom_app_context.verify('ThisIsTheAWrongPassword', pwd_hash)
9 False
```

FIGURE 7.4 – Hachage de mot de passe avec *passlib*

Nous voyons que le mot de passe a été haché en une longue chaîne de caractères et que lorsque nous vérifions avec la méthode `verify()`, le résultat est bien juste. *Custom_app_context* fourni une méthode de chiffrage très bien faite de manière simple. L'algorithme utilisé est du SHA-256 sur un système 32-bit et du SHA-512 sur un système 64-bit. Il utilise un sel ainsi qu'un système de tour pour bien encrypter le mot de passe.

```
${hash_type}$rounds={rounds}${salt}${checksum}
```

FIGURE 7.5 – Structure du hash

Voyons ensuite la structure du mot de passe haché (figure 7.5). Plusieurs parties distinctes sont séparées par des `$`. Le premier caractère entouré par des dollars est, dans notre cas, `"6"`. Cela correspond à l'algorithme SHA-512. Si ce chiffre était `"5"`, ça aurait été du SHA-256. Cela permet de rendre le système compatible si un mot de passe a été encodé sur une machine 32-bit et qu'il est vérifié sur une machine 64-bit. Ensuite nous avons la chaîne de caractère *"rounds="* suivi d'un nombre entre 1000 et 999999999. Il s'agit du nombre de fois que l'algorithme a été appliqué. Ici nous avons `"90601"`. La chaîne suivante est le sel, ici `"mGWjlqsrSIK4ZHMY"`, composé de 0 à 16 caractères alphanumériques minuscules et majuscules, plus le point "." et le slash "/". Et pour finir, la somme de contrôle, ou *checksum* du hash, de 43 caractères pour le SHA-256 et de 86 caractères pour le SHA-512. Dans notre cas, ce *checksum* est `"aeYyI85Q9Wx.W4fIvXKYlz46Ko7xZTFIoTzLuV7Mnlu.7KsxZkiyn19GaBrMDfxf2PEhpp.N7CxaE67ztWlw/1"`.

Pour plus d'informations, veuillez vous rendre à l'adresse :
`https://pythonhosted.org/passlib/lib/passlib.apps.html#custom-applications`

7.4 TOKEN

Lorsque l'utilisateur veut accéder à des informations au travers de l'API, il doit, dans certains cas, s'authentifier. Pour ce faire, il va tout d'abord demander un *token*, ou jeton, qui va ensuite servir durant dix minutes. Pour demander un *token*, il doit envoyer son nom d'utilisateur et son mot de passe. Le *token* remplace donc ces identifiants par la suite. Dans cette partie, nous utiliserons la librairie *itsdangerous*, et plus précisément la classe `TimedJSONWebSignatureSerializer`.

Le *JWS*, ou *JSON Web Signature*, est un message qui a été encodé et signé et qui tire partie du format JSON. La classe contient le mot *Timed* en plus, ce qui signifie qu'il y a une notion de temps, et donc qu'une notion de validité peut être ajoutée. C'est ce que nous avons besoin pour générer un *token* qui soit valable uniquement durant un certain temps. Et le *Serializer* permet de créer et de manipuler de telles données.

Le constructeur de ce *Serializer* prend deux arguments : une clé secrète ainsi qu'un temps d'expiration en secondes. Nous pouvons ensuite y ajouter des données au format JSON avec la méthode `dumps()`. Cette méthode prend donc un objet JSON, et peut également prendre un sel pour améliorer l'encryption. Le tout est encrypté et signé et une chaîne est retournée, nous l'utilisons comme *token*.

Pour y extraire des données, il faut ensuite appeler la méthode `loads()` avec le *token* en argument. Cela fonctionne en utilisant l'objet `Serializer` précédemment utilisé, mais également en utilisant une nouvelle instance créée avec la même clé secrète. Cette méthode retourne l'objet JSON qu'on avait inséré avec `dumps()`. Il suffit donc d'extraire l'information utile du JSON, dans notre cas il s'agit de l'identifiant. La figure 7.6 montre un exemple d'exécution de cette technique dans la console Python.

```
>>> from itsdangerous import TimedJSONWebSignatureSerializer as Serializer
>>> secret_key = 'MySecretKey'
>>> s = Serializer(secret_key, expires_in=600)
>>> token = s.dumps({'id':'MyIdentifier'})
>>> token
'eyJhbGciOiJIUzI1NiIsImV4cCI6MTQwNDgwOTc1MCwiaWF0IjoxNDA0ODA5MTUwfQ.eyJpZCI6Ik15SWRlbnRpZmllciJ9.oIIvX
                                                                    Hlfc1Dtg8s89Uz2NAiOKCr-Jhy2_UjW-8GrfB4'
>>> s1.loads(token)
{'id': 'MyIdentifier'}
```

FIGURE 7.6 – Génération d'un *token* avec validité limitée et stockage d'un tableau JSON

Dans le code de notre API, nous vérifions que le *token* soit valide et en plus qu'il ne soit pas expiré avec la fonction qui figure dans le code 7.7. Les exceptions `SignatureExpired` et `BadSignature` sont en fait des

classes qu'il faut également importer du package *itsdangerous* et qui sont émises lorsque la validité du *token* est expirée ou que le *token* est faux. Dans cette méthode, si le *token* est correct, un tableau JSON est retourné. Nous en extrayons l'identifiant qui permet de créer une instance de `User`.

```
1  def verify_auth_token(token):
2      s = Serializer(app.config['SECRET_KEY'])
3      try:
4          data = s.loads(token)
5      except SignatureExpired:
6          return None    # valid token, but expired
7      except BadSignature:
8          return None    # invalid token
9      user = User(uid=data['id'])
10     return user
```

FIGURE 7.7 – Fonction qui vérifie le token et retourne l'identifiant qu'il contient

7.5 STRUCTURE DE LOGICIEL

Le logiciel qui fournit les API et qui est implémenté en Python tourne sur le serveur et utilise le port 80 (HTTP). Pour lancer ce serveur, des indications sont fournies en annexe. En ce qui concerne l'implémentation c'est assez simple avec Flask. Il faut créer cette application, il faut ensuite régler quelques configurations, lier la base de données MongoDB avec l'application, et finalement lancer le service. Voici comment procéder :

```
1  app = Flask(__name__)
2
3  app.config['SECRET_KEY'] = 'This is my secret key, or passphrase can we say, long enough? I think so.'
4  app.config['MONGO_PORT'] = 27017
5  app.config['MONGO_DBNAME'] = 'golfBallDispenserDatas'
6
7  mongo = PyMongo(app)
8
9  if __name__ == '__main__':
10   app.run(host='0.0.0.0', port=80)
```

FIGURE 7.8 – Lancer un serveur Flask

Nous avons ensuite une classe `User` qui permet de stocker les informations d'un utilisateur qu'on va chercher en base de données en fonction de son identifiant unique ou de son nom d'utilisateur. Et finalement, le plus important, les APIs. Pour faire une API avec Flask, il suffit d'ajouter un décorateur Python sur une méthode. Voici un exemple :

```
1  @app.route('/')
2  def home_page():
3      return 'Bienvenue'
4
5  @app.route('/api/users/<username>')
6  def get_users(username):
7      return jsonify({'username': username})
```

FIGURE 7.9 – Exemple d'APIs

Nous voyons que la première méthode est précédée d'un décorateur `@app.route('/')`. Grâce à ça, la méthode va être appelée lorsqu'une requête est faite à l'adresse du serveur. La deuxième méthode, quant à elle, va être appelée lorsque l'URL est celle du serveur suivie de l'argument du décorateur `/api/users/<username>`, où `<username>` peut être remplacé par le nom d'utilisateur souhaité. Dans le

premier cas, l'utilisateur recevra juste le message "Bienvenue" et dans le deuxième cas des données au format JSON renvoyant le nom d'utilisateur demandé.

Il est possible de récupérer des requêtes POST et GET en contrôlant les arguments et les URLs.

En ajoutant le décorateur `@auth.login_required`, nous pouvons autoriser l'accès à l'API uniquement à un utilisateur authentifié. Pour s'authentifier, il doit envoyer son nom d'utilisateur ainsi que son mot de passe avec la requête. Il s'agit là de la méthode d'authentification HTTP dite *"Basic"*[1]. A la place du nom d'utilisateur et du mot de passe, il peut également envoyer un *token*, qu'il a préalablement demandé en s'authentifiant également. Un *token* est valable 10 minutes dans notre cas. Passé ce délai, il devra en demander un nouveau. Cela permet de minimiser le nombre de fois que le mot de passe est transmis par le réseau.

Pour ce projet, nous utilisons le protocole HTTP, mais s'il s'avérait qu'il passe en production, il faudra préférer le HTTPS pour plus de sécurité.

Voici donc la liste des APIs que nous avons implémentées :

- **`/api/newaccount`** : Crée un nouveau compte utilisateur avec un nom d'utilisateur et un mot de passe. Il s'agit d'une requête POST. Cette requête retourne le code HTTP 201 (CREATED) si le compte a bien été créé, et le code HTTP 403 (FORBIDDEN) si le nom d'utilisateur est déjà utilisé.
- **`/api/token`** : Retourne un *token* qui sera valable durant 10 minutes.
- **`/api/deviceid/<androidid>`** : Retourne l'identifiant unique du smartphone Android en fonction de son identifiant Android. Si l'appareil n'est pas encore enregistré sur le compte de l'utilisateur, l'identifiant est généré et l'appareil est ajouté.
- **`/api/user`** : Retourne des informations sur l'utilisateur : nom d'utilisateur, nom, prénom, solde du compte, état du compte, date d'inscription.
- **`/api/transactions`** : Retourne les 10 dernières transactions de l'utilisateur. Chaque transaction comprend le type de transaction (retrait ou recharge), le montant, la date, l'identifiant de l'appareil NFC et l'identifiant du lecteur de carte NFC.

D'autres APIs ont été implémentées mais ne sont pas utilisées par l'application Android. Elles sont tout de même disponibles. Il s'agit par exemple d'une API qui permet de définir un nouveau mot de passe pour un compte, ou d'une autre qui permet de savoir le solde.

Mis à part la requête de création de compte, toutes les autres APIs renvoient les informations en format JSON.

1. Page Wikipedia : `http://en.wikipedia.org/wiki/Basic_access_authentication`, page consultée en juin 2014.

Chapitre 8

Android

8.1 Structure

Passons maintenant à l'application Android. Celle-ci est assez simple puisqu'elle comporte uniquement trois activités, une pour créer un compte, une pour se connecter à son compte et une autre pour afficher les informations de l'utilisateur. Par contre sa construction est moins simple, comme le montre le diagramme de classe de cette application à la figure 8.1.

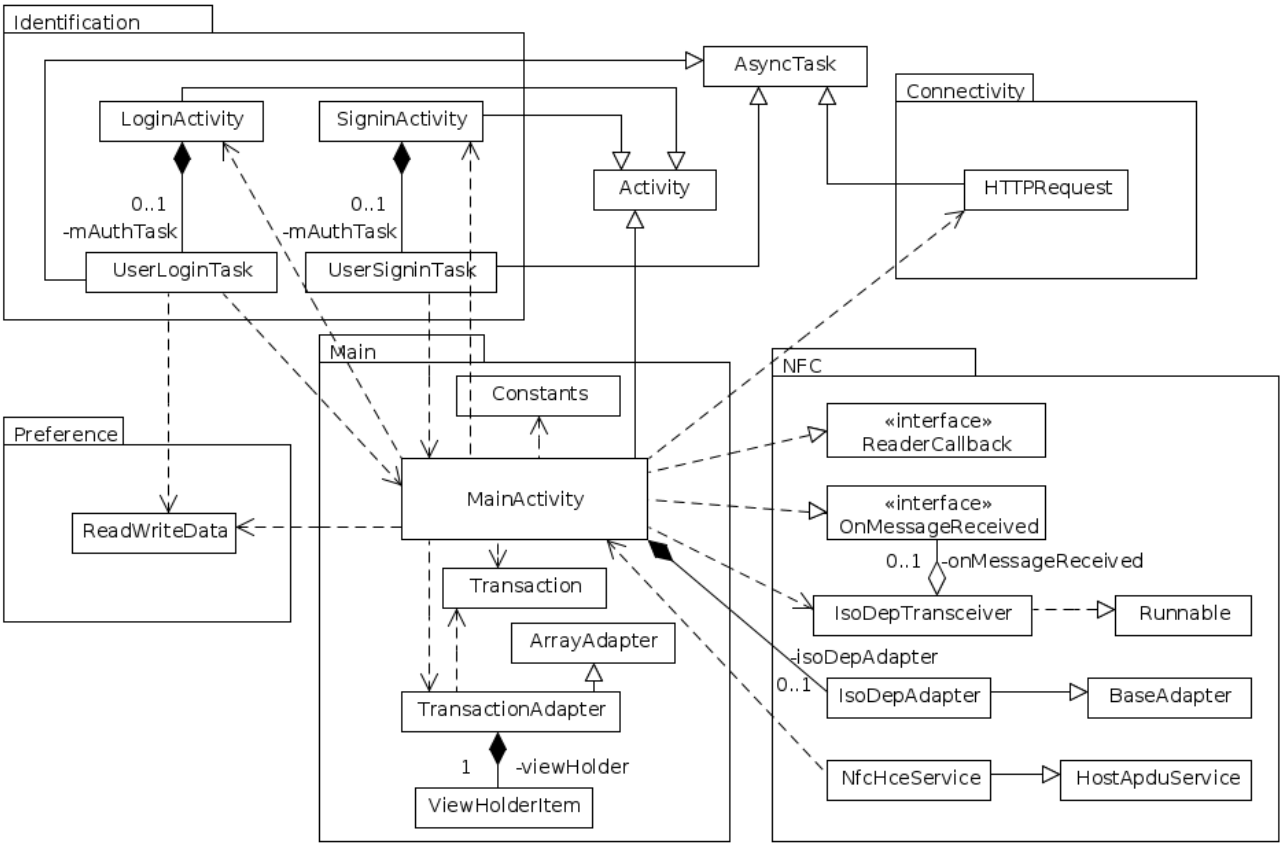


FIGURE 8.1 – Diagramme de classe de l'application

La documentation du code contient le détail de toutes les classes, les diagrammes d'héritage ainsi que la descritption de tous les attributs et méthodes du projet. Elle se trouve dans la dossier du projet `GolfWallet` à l'adresse `GolfWallet/src/doc/html/index.html`. Pour voir le code, rendez-vous à l'annexe H.

8.2 EMULATION DE CARTE NFC

8.2.1 Service Android

Il faut ensuite émuler une carte NFC avec le smartphone pour que le lecteur le détecte comme tel. Avant la version 4.4 d'Android, ou KitKat, il était possible d'émuler une carte NFC avec un "secure element", c'est-à-dire un élément qui fourni un espace de stockage sécurisé. Souvent il s'agissait d'une carte SIM ou d'une carte SD qu'on insérait dans le smartphone. Mais depuis KitKat, il est possible de se passer de cet élément et de faire communiquer le contrôleur NFC directement avec le CPU. Cette méthode s'appelle "Host-based Card Emulation" ou "HCE". Ces deux fonctionnements sont illustrés à la figure 8.2.

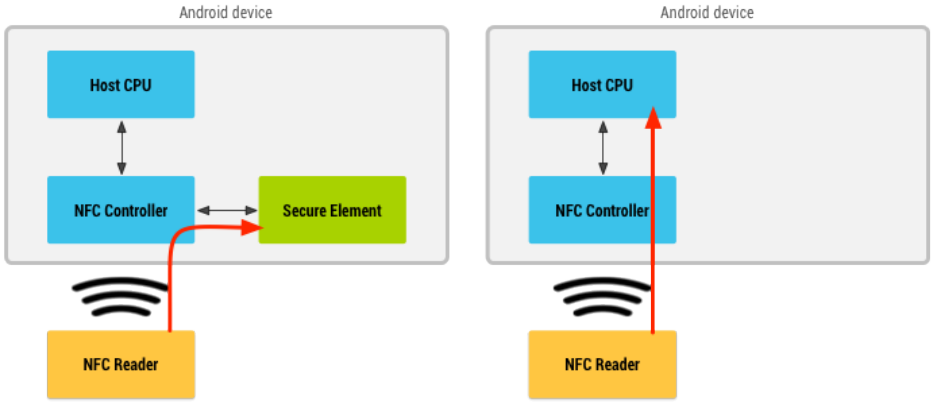


FIGURE 8.2 – (a) Secure Element (b) Host-based Card Emulation

source : tiré de la documentation Android `http://developer.android.com/guide/topics/connectivity/nfc/hce.html`

Pour mettre en place cette méthode, il faut créer un *service* Android. Un service est un programme qui tourne en tâche de fond, il n'a pas besoin d'interface graphique pour être exécuté. Le service qu'on utilise est `HostApduService`. Pour le mettre en place, il faut le déclarer dans le manifest en indiquant le nom du service (figure 8.3) et les permissions qu'il a (utilisation du NFC, figure 8.4). Il faut également exporter ce service pour qu'il puisse être utilisé même quand l'application n'est pas lancée. Il faut ensuite créer un fichier XML (figure 8.5) qui contient les *AID* ou *Application ID* qui permettent de définir différents identifiants qui peuvent accéder à ce service. Cela permet d'avoir plusieurs services NFC sur un même appareil.

```
1  <service
2    android:name=".NfcHceService"
3    android:exported="true"
4    android:permission="android.permission.BIND_NFC_SERVICE" >
5    <intent-filter>
6      <action android:name="android.nfc.cardemulation.action.HOST_APDU_SERVICE" />
7    </intent-filter>
8
9    <meta-data
10     android:name="android.nfc.cardemulation.host_apdu_service"
11     android:resource="@xml/hceservice" />
12 </service>
```

FIGURE 8.3 – Déclaration du service dans le manifest

```
1  <uses-permission android:name="android.permission.NFC" />
```

FIGURE 8.4 – Permission dans le manifest

```
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
  android:description="@string/servicedesc"
  android:requireDeviceUnlock="false" >

  <aid-group
      android:category="other"
      android:description="@string/aiddescription" >
      <aid-filter android:name="F0010203030201" />
  </aid-group>

</host-apdu-service>
```

FIGURE 8.5 – hceservice.xml

8.3 APPEL À L'API DEPUIS ANDROID

Dans l'application Android, nous créons la classe `HTTPRequest` qui hérite de la classe `AsyncTask`. Cette classe permet de faire les appels aux APIs. Ces appels se font de manière asynchrone, l'interface graphique n'est donc pas bloquée durant les appels.

Il faut créer un nouvel objet de cette classe à chaque fois qu'on en a besoin car chaque événement asynchrone s'exécute une seule fois. Voici donc le code de création et d'appel à l'API principale :

```
new HTTPRequest().execute("http://vlenfc.hevs.ch");
```

FIGURE 8.6 – Création et lancement d'une requête asynchrone

8.4 CRÉATION ET REMPLISSAGE D'UN TABLEAU

La création et le remplissage d'un tableau sur une interface Android n'est pas aussi simple qu'on pourrait le croire, c'est pourquoi une explication est fournie ici.

Nous voulons créer une liste de transactions. Sur chaque ligne se trouve une transaction et il y a deux colonnes, la première contient la date et la seconde le montant de la transaction. Pour commencer nous avons créé une classe `Transaction` qui contient toutes les informations d'une transactions (même celles que nous n'avons pas besoin pour remplir le tableau) ainsi que des *getters* pour pouvoir accéder à ces informations.

Nous avons ensuite implémenté la classe `TransactionAdapter` dont le code est visible à la figure 8.7. Cette classe hérite de la classe `ArrayAdapter<Transaction>` et contient une classe statique `ViewHolderItem` qui stocke un champ de texte pour la date et un champ de texte pour le montant. Cela évite que le tableau soit rempli aléatoirement par des transactions lorsque la méthode `getView()` est appelée. Pour que les transactions soient dans l'ordre, un tag est sauvegardé pour chaque ligne. La méthode `getView()` est appelée lorsqu'on crée la vue, mais également lorsqu'une ligne cachée du tableau apparaît à l'écran (lors d'un scroll par exemple). Le constructeur de cette classe prend en argument la liste complète des transactions. Et pour remplir la vue, on va appeler les *getters* de chaque transaction.

```java
public class TransactionAdapter extends ArrayAdapter<Transaction> {

    static class ViewHolderItem {
        TextView dateTextView;
        TextView amountTextView;
    }

    public TransactionAdapter(Context context, int resource, List<Transaction> items) {
        super(context, resource, items);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        ViewHolderItem viewHolder;

        if (convertView == null) {
            LayoutInflater inflater = ((Activity) getContext()).getLayoutInflater();
            convertView = inflater.inflate(R.layout.transaction_row, parent, false);

            viewHolder = new ViewHolderItem();
            viewHolder.dateTextView = (TextView) convertView.findViewById(R.id.date);
            viewHolder.amountTextView = (TextView) convertView.findViewById(R.id.amount);

            // store the holder with the view.
            convertView.setTag(viewHolder);
        } else {
            viewHolder = (ViewHolderItem) convertView.getTag();
        }

        Transaction p = getItem(position);

        if (p != null) {
            viewHolder.dateTextView.setText("" + p.getDate());
            viewHolder.amountTextView.setText("" + p.getSignedAmount());
        }

        return convertView;
    }
}
```

FIGURE 8.7 – TransactionAdapter.java

Ensuite, il faut remplir le tableau avec des transactions. Ceci se fait dans la méthode `onCreate()` de l'activité dans laquelle on veut afficher le tableau. Le code de la figure 8.8 montre comment procéder. Dans cet exemple, nous chargeons la vue puis nous créons une liste qui va contenir les transactions. Ensuite dans le *"try {} catch() {}"* nous créons des instances de `Transaction` grâce à un fichier JSON. Les transactions sont ensuite ajoutées à la liste des transactions. Pour finir, un `TransactionAdapter` est instancié avec la liste des transactions et est attribué à la vue.

A la ligne 21, un layout *"R.layout.transaction_row"* est utilisé. Celui-ci permet de définir la disposition des deux `TextView` dans une ligne du tableau. Ce fichier XML de *layout* contient simplement un `LinarLayout` horizontal contenant les deux `TextView`. Quelques attributs y sont décrits pour positionner et formater ces éléments.

```
1  ListView transactionListView = (ListView)findViewById(R.id.transactionList);
2  List<Transaction> transactionList = new ArrayList<Transaction>();
3  try {
4      JSONObject jsonObj = new JSONObject(result);
5      JSONArray json = (JSONArray) jsonObj.get("transactions");
6
7      for(int i = 0; i < json.length(); i++) {
8          JSONObject c = json.getJSONObject(i);
9          int amount = c.getInt("amount");
10         int type = c.getInt("transactionType");
11         String transactionDate = c.getString("transactionDate");
12         String deviceId = c.getString("deviceId");
13         String dispenserId = c.getString("dispenserId");
14
15         transactionList.add(new Transaction(amount, type, transactionDate, deviceId, dispenserId));
16     }
17 } catch (JSONException e) {
18     e.printStackTrace();
19 }
20 TransactionAdapter transactionAdapter =
21         new TransactionAdapter(this, R.layout.transaction_row, transactionList);
22 transactionListView.setAdapter(transactionAdapter);
```

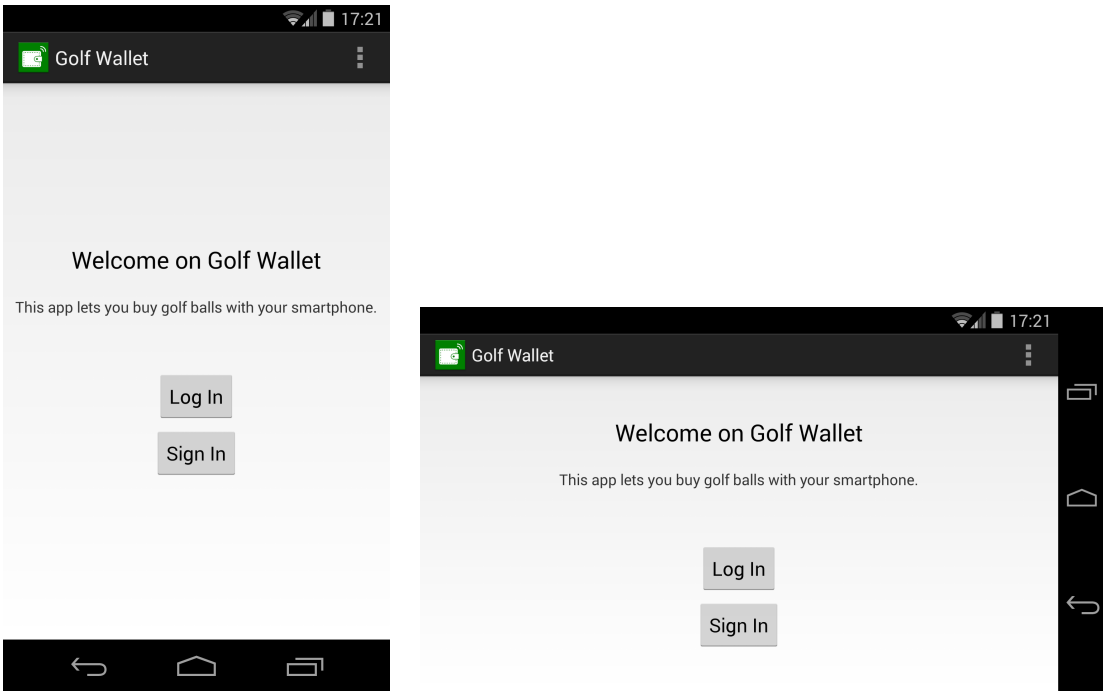
FIGURE 8.8 – Remplissage de la vue dans le *onCreate()* de l'activité

8.5 INTERFACE ET UTILISATION

Voici tout d'abord le menu d'Android avec l'application **Golf Wallet** à la figure 8.9. L'icône de l'application représente un porte-feuille qui émet des ondes, ce qui image bien l'utilité du logiciel, de plus la couleur verte est utilisée pour rappeler l'herbe, donc le golf. L'appareil utilisé ici est un Nexus 5 avec Android KitKat 4.4.4.

FIGURE 8.9 – Icône de l'application

Lorsque nous cliquons sur l'application pour la lancer, nous arrivons sur le menu d'accueil qui propose de se connecter ou de créer un compte (figure 8.10). Le téléphone peut se tenir horizontalement ou verticalement, l'affichage s'adapte.



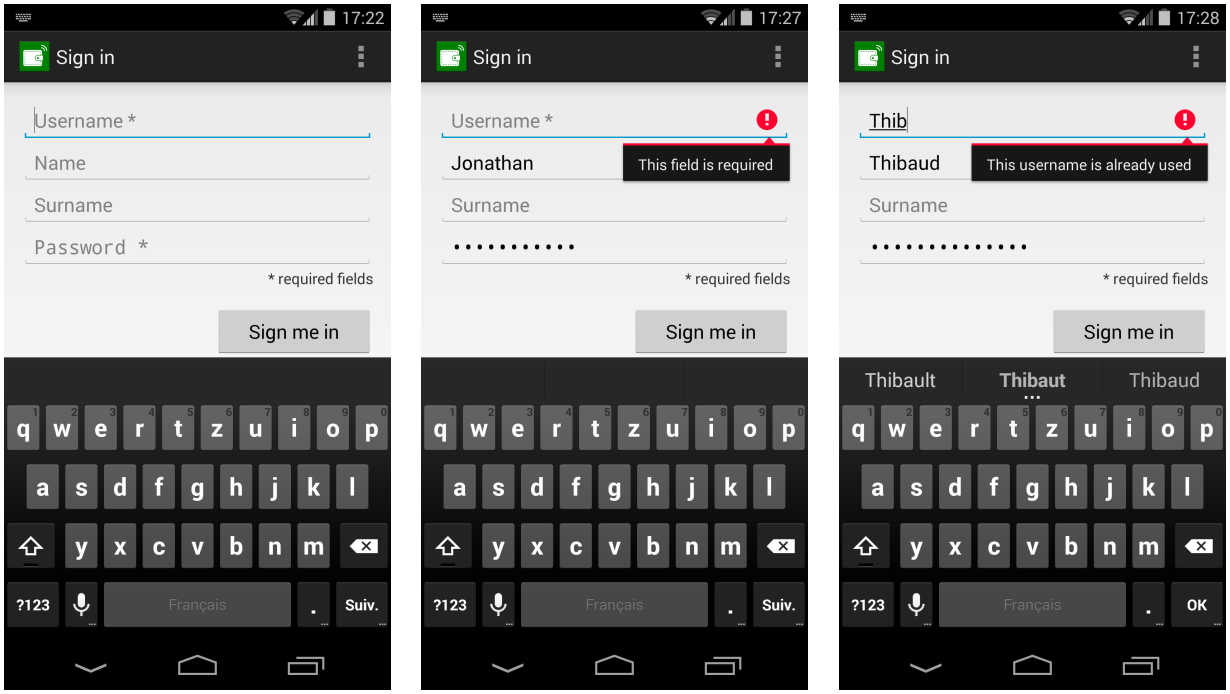
(a) (b)

FIGURE 8.10 – (a) Position verticale (b) Position horizontale

Si l'utilisateur n'a pas encore de compte, il va donc cliquer sur *"Sign In"*. Il va être redirigé vers un écran qui contient un formulaire d'inscription avec quatre champs (voir figure 8.11). Le premier champ est le nom d'utilisateur. Les deuxième et troisième champs sont le nom et le prénom. Ces champs sont facultatifs, il permettent juste à l'application de pouvoir afficher le nom de l'utilisateur lorsqu'il sera connecté. Et pour finir le mot de passe, qui doit contenir au minimum quatre caractères. Lorsque le mot de passe est tapé, les lettres sont remplacées par des petits ronds, pour que le mot de passe ne soit pas visible par une personne tierce lors de l'inscription.

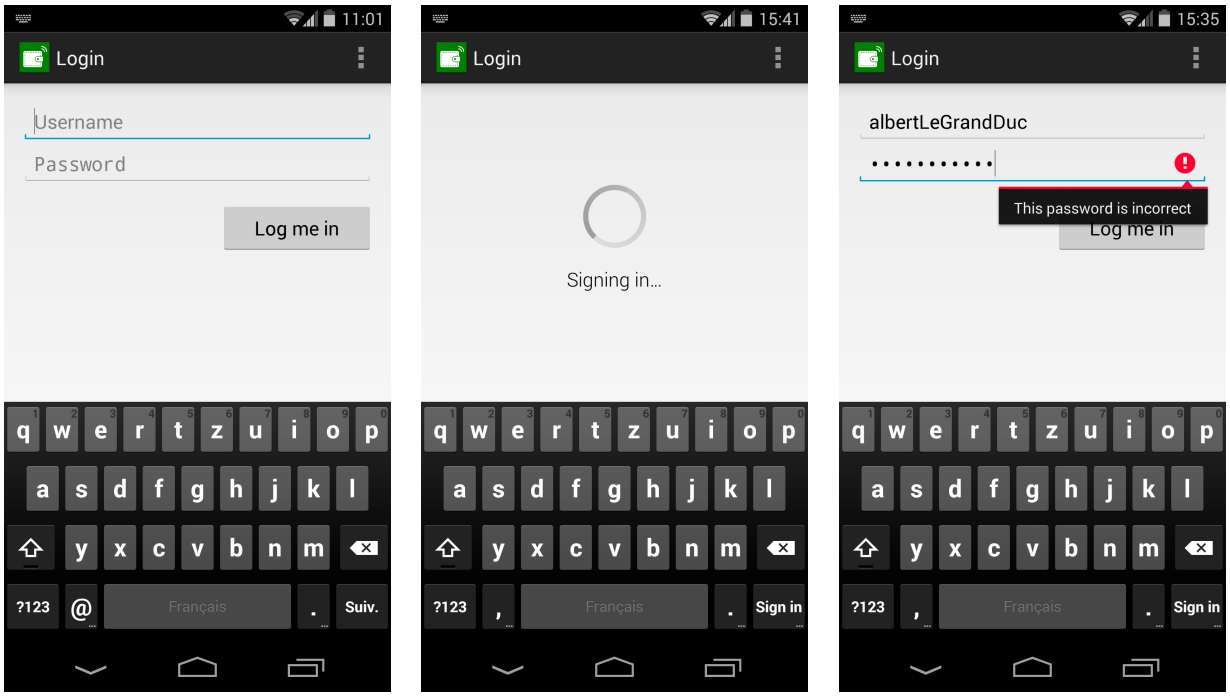
Une fois ces champs remplis, il suffit de cliquer sur le bouton pour créer son compte. Une première vérification contrôle que les champs obligatoires ont bien été remplis, si ce n'est pas le cas une petite boîte de dialogue s'ouvre pour en notifier l'utilisateur (image b). La première vérification validée, une requête est envoyée au serveur par le biais d'une API. Si le nom d'utilisateur est déjà utilisé par un autre client, le compte ne sera pas créé et l'utilisateur est prié de choisir un autre mot de passe (image c), sinon le compte est bien créé. Une redirection vers la première page est effectuée et une notification indique que le compte est créé et qu'un montant de 10 CHF est disponible sur le compte, cela permet au client de pouvoir directement essayer le produit, mais également de lui faire un petit cadeau de bienvenue.

En cliquant sur le bouton *"Log In"*, nous arrivons sur une nouvelle interface (figure 8.12) qui demande à l'utilisateur d'entrer son nom d'utilisateur ainsi que son mot de passe. Quand on valide, quelques contrôles sont effectués, comme précédemment, avant d'envoyer la requête au serveur. Un écran de chargement (image c) est affiché durant le traitement. La réponse de la requête indique si la connexion est acceptée ou non. Si c'est le cas, nous sommes dirigés vers l'activité principale.



(a) (b) (c)

FIGURE 8.11 – (a) Menu de création de compte (b) Champ requis (c) Nom d'utilisateur déjà existant



(a) (b) (c)

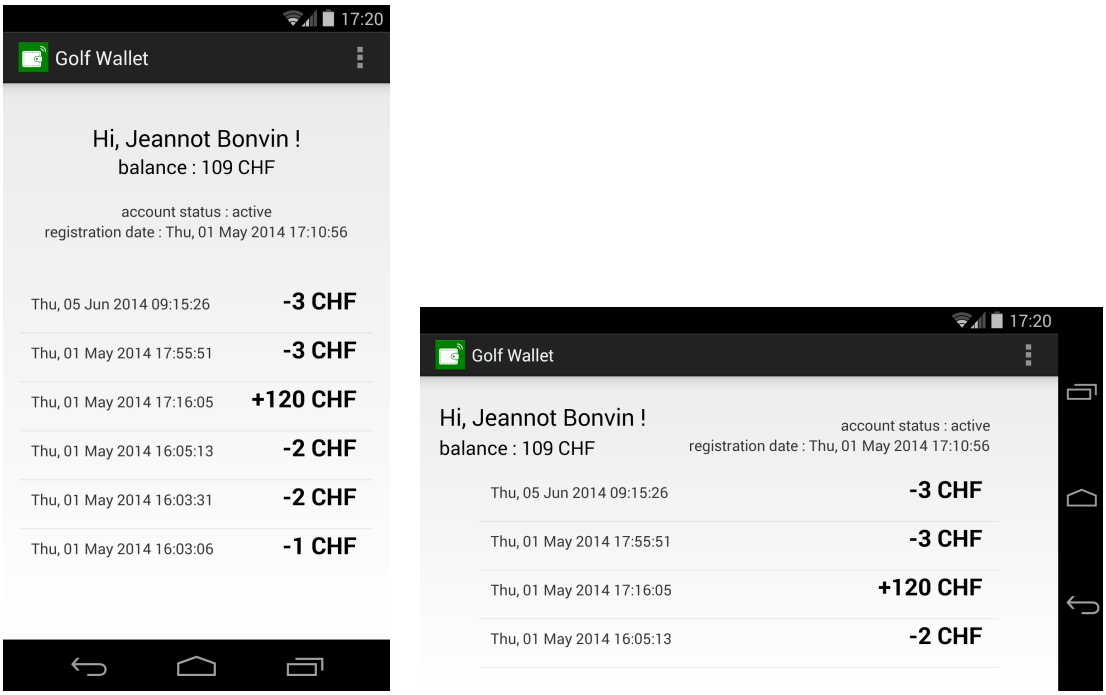
FIGURE 8.12 – (a) Menu de connexion (b) Vérification (c) Erreur de saisie

Lors de la connexion, l'application a recu un identifiant que le smartphone doit émuler lors du contact avec le lecteur NFC. Les identifiants du compte ainsi que l'identifiant du smartphone sont sauvegardés dans les préférences, les `SharedPreferences`. Ainsi lors de la prochaine ouverture de l'application, l'utilisateur sera

déjà connecté.

L'identifiant du smartphone reçu va être envoyé au lecteur NFC lorsque le smartphone s'approche du lecteur. C'est grâce à cet identifiant que la transaction va être effectuée sur le bon compte. Si l'utilisateur n'est pas connecté, le smartphone va envoyer l'identifiant *"AA AA AA AA"* par défaut, ainsi le lecteur pourra dire à l'utilisateur qu'il faut créer un compte avant de pouvoir utiliser le service.

Quand l'activité principale est chargée, plusieurs appels à des APIs sont faits. Premièrement on va demander un *token* de session pour faire les requêtes suivantes. Ensuite on va appeler l'API qui renvoie les informations du compte comme le nom et le prénom de l'utilisateur, la date de création du compte ainsi que le solde de celui-ci. Et pour finir on appelle l'API qui retourne les dernières transactions effectuées. Toutes ces informations vont permettre de remplir le contenu de l'activité comme montré à la figure 8.13. Deux *layouts* différents ont été créés pour que la place soit bien utilisée si le téléphone est en position verticale mais également en position horizontale. Quand le smartphone est en position verticale, les éléments sont les uns en dessous des autres, par contre en position horizontale, le statut du compte et la date de création se positionnent à droite du solde et du nom.



(a) (b)

FIGURE 8.13 – Activité principale : (a) Position verticale (b) Position horizontale

Pour se déconnecter, un bouton *Log out* est disponible quand l'utilisateur clique sur les trois petits points qui se situent en-haut à droite de l'écran. De plus, des *Toasts* s'affichent lorsque l'utilisateur fait des actions, comme lors de la connexion, de la déconnexion et du chargement des transactions. Cela donne un retour de ce qui se passe pour que l'utilisateur comprenne bien. Un *Toast* est un petit texte qui s'affiche au-dessus des autres éléments de l'interface et qui disparaît au bout d'un peu plus d'une seconde.

Chapitre 9

Conclusion

9.1 Synthèse

Un logiciel a été développé sur un système embarqué connecté à un lecteur de carte NFC. Il permet à un utilisateur possédant une carte NFC enregistrée de retirer une certaine quantité d'argent de son compte simplement en approchant la carte du lecteur. Une interface graphique simple et fonctionnelle donne différentes informations à l'utilisateur. Pour interagir avec l'interface, l'utilisateur peut employer les quatre boutons à disposition ou alors un clavier et une souris. Une interface administrateur est disponible pour créer des comptes, ajouter une carte à un compte utilisateur, ou recharger un compte. Toutes les informations sont stockées dans une base de données MongoDB qui se trouve sur un serveur distant qui est accessible de manière sécurisée. Le système pourra s'adapter à un distributeur de balles de golf, mais pour le moment son contrôle est simulé avec quelques LEDs.

De plus, un utilisateur disposant d'un smartphone Android de dernière génération (Android 4.4 ou supérieur avec support NFC) peut utiliser le produit en téléchargeant une application qui va se substituer à une carte NFC. L'application permet de se connecter à un compte utilisateur facilement puis de visualiser les informations du compte. L'application accède aux informations en utilisant des APIs REST de manière authentifiée. Les APIs sont localisées sur le même serveur que la base de données, ce qui lui permet d'atteindre facilement la base de données.

Un démonstrateur a donc été mis en place et il est totalement fonctionnel.

L'objectif est donc correctement atteint.

9.2 Améliorations

Il existe différents types de cartes NFC. Seules les cartes MiFare ont pour l'instant été testées avec le système développé. Il pourrait donc être utile de tester avec d'autres cartes, comme par exemple les cartes FeliCa, qui sont les rivales niponnes de la MiFare.

Lors de l'appel aux APIs, un token de session est demandé lors du premier appel. Pour cela, le nom d'utilisateur ainsi que le mot de passe transitent sur le réseau. Ils sont encodés en base64 et sont situés dans l'en-tête de la requête HTTP. Pour que ces informations soit chiffrées, il serait prudent d'utiliser le protocole sécurisé de HTTP, le HTTPS. C'est une chose importante à mettre en place lors de la mise en production du système pour éviter le vol d'informations relatives aux comptes clients.

Bien qu'une adminstration ait été mise en place dans le logiciel, il serait judicieux de l'améliorer, notamment en proposant un système de création de comptes administrateurs avec une gestion des droits. Une autre amélioration concernant la partie d'administration serait de développer une interface web qui tirerait parti des différents APIs développées pour avoir une vue générale de toutes les informations citées ci-dessous :

- Vision d'ensemble des comptes clients avec pagination et tri en fonction de différents critères (nom, solde du compte, dernière activité, etc.)

- Affichage de toutes les informations d'un client. Cela comprend ses informations personnelles mais également les divers appareils liés à son compte (cartes NFC, smartphones)
- Gestion des utilisateurs : bloquer une carte pour cause de vol, changer les informations d'un compte, etc.
- Statistiques d'utilisation du produit en fonctions de divers critères (proportion carte/smartphone, age, etc.)
- Impression d'une facture pour un client
- Exportation d'une liste de clients selon divers critères
- Envoi groupé d'emails

Et pour finir, il faudrait un site web vitrine qui présente le projet. Ce site serait une publicité permanente mais il permettrait également aux personnes intéressées de suivre l'avancement du développement et des mises à jour. Il pourrait servir de portail d'entrée pour un espace client où l'utilisateur pourrait avoir un aperçu de son compte. Nous pouvons même imaginer que le client puisse recharger son compte avec un paiement en ligne.

9.3 REMERCIEMENTS

Je voudrais remercier plusieurs personnes qui ont contribué de près ou de moins près au développement de ce projet. Tout d'abord je remercie M. Pierre-André Mudry qui est le professeur qui a supervisé le projet. Il a été disponible et m'a proposé/conseillé d'utiliser certaines technologies utiles au projet que je n'avais encore jamais utilisées. Je remercie également Christopher Métrailler pour les précieuses astuces du développement Android qu'il m'a fournies, ainsi que Marc Pignat pour l'astuce concernant les drivers du lecteur NFC. Merci à l'entreprise Digiclever qui est le mandant de ce projet et qui a su satisfaire les attentes que j'avais par rapport au travail de diplôme. J'aimerais ensuite remercier Dominique Casas qui a fait une relecture de ce dossier pour traquer les fautes d'orthographe et les fautes de français. Et pour finir, merci à l'équipe du bureau A304 pour les neuf agréables semaines passées ensemble.

Sion, le 10 juillet 2014

Jacky CASAS

Chapitre 10

Références

- Guide de démarrage Raspberry Pi *"quick-start-guide-v2_1.pdf"* disponible sur le site officiel de Raspberry Pi à l'adresse : `www.raspberrypi.org/new-quick-start-guide/`
- Datasheet des APIs du ACR122U *"API-ACR122U-2.02.pdf"* disponible sur le site officiel d'ACS (Advanced Card Systems) : `www.acs.com.hk/en/products/3/acr122u-usb-nfc-reader/`
- Datasheet de Gemalto des cartes Mifare 1K *"mifare1K_datasheet.pdf"* disponible à l'adresse : `www.gemalto.com/products/hybrid_card_body/download/mifare1K_datasheet.pdf`
- Datasheet de NXP des cartes Mifare 1K *"MF1S503x.pdf"* disponible à l'adresse : `www.nxp.com/documents/data_sheet/MF1S503x.pdf`
- Tutoriel Python sur le site OpenClassrooms pour apprendre les rudiments de Python : `www.openclassrooms.com/informatique/cours/apprenez-a-programmer-en-python`
- Documentation de Python : `https://docs.python.org`
- Documentation de PiFace Digital : `http://piface.github.io/pifacedigitalio`
- Documentation de Pyscard : `http://pyscard.sourceforge.net`
- Documentation de MongoDB : `http://docs.mongodb.org`
- Documentation de PyMongo : `http://api.mongodb.org/python/2.7`
- Documentation de Flask : `http://flask.pocoo.org/`
- Documentation de Flask-PyMongo : `http://flask-pymongo.readthedocs.org/en/latest/`
- Documentation de la librairie Python passlib : `https://pythonhosted.org/passlib/lib/passlib.apps.html#custom-applications`
- Dépôt de la librairie Python itsdangerous : `https://github.com/mitsuhiko/itsdangerous`
- Documentation d'Android : `https://developer.android.com/develop/index.html`
- Documentation du *Host-based Card Emulation* pour le NFC sur Android : `https://developer.android.com/guide/topics/connectivity/nfc/hce.html`
- Documentation de Qt : `http://qt-project.org/doc/`
- Documentation de PySide : `http://qt-project.org/wiki/PySideDocumentation`
- Site de Doxygen : `http://doxygen.org`

Annexe A

Logiciels utilisés

Différents logiciels ont été utilisés pour le projet. En voici une liste avec leur utilité :

1. **Sublime Text** : Tout le code Python, que ça soit pour des tests, pour le logiciel du distributeur de balles ou pour l'implémentation des APIs, a été rédigé sur Sublime Text 3 (Build 3059). Sublime Text est disponible à cette adresse : `http://www.sublimetext.com`.
2. **Guake** : Le terminal utilisé est Guake. C'est un terminal déroulant qui apparaît lors de la pression de la touche F12. En savoir plus : `http://doc.ubuntu-fr.org/guake`.
3. **VIM** et **nano** : Ces deux éditeurs de texte en console ont été utilisés pour faire de simples modifications sur des fichiers de code ou de configuration sur le Raspberry Pi et sur le serveur.
4. **OpenSSH** : Ce client SSH en console a permis de se connecter au Raspberry Pi et au serveur ainsi qu'à créer le tunnel SSH entre le Raspberry Pi et le serveur.
5. **Eclipse** : Pour l'application Android, nous avons utilisé les outils de développement Android (ADT - Android Developer Tools) standards en version 64-bit.
   Pour les télécharger, suivez le lien suivant : `http://developer.android.com/sdk/index.html`.
6. **UMLet** : Les diagrammes de classe ainsi que les diagrammes de séquence ont été conçus avec UMLet, un outil de conception de diagrammes UML.
   UMLet est disponible à l'adresse `http://www.umlet.com/`
7. **LaTeX** : Le rapport que vous êtes en train de lire a été créé avec LaTeX. L'éditeur utilisé est Kile.
8. **Doxygen** : La documentation du code Python et du code Java a été faite avec Doxygen.
   Pour en savoir plus : `http://doxygen.org`.
9. **Git** : Git (`http://git-scm.com`) a été utilisé pour gérer les versions du logiciel développé pour le Raspberry Pi, des APIs, de l'application Android ainsi que du rapport. Ces différents projets sont stockés sur les serveurs de Bitbucket (`https://bitbucket.org`) dans des dépôts privés, puis en version finale sur Github (`https://github.com`). Git a également permis de synchroniser les fichiers en local avec les fichiers qui se trouvaient sur le Raspberry Pi ou le serveur.

De plus, tout le projet a été développé sur une machine avec un système d'exploitation Linux. Il s'agit plus précisément d'un Ubuntu 14.04 (Trusty Tahr) en 64-bit avec Unity 7.2.0 comme environnement de bureau.

Annexe B

Raspberry Pi

B.1 Installation d'un système d'exploitation

Sur le site officiel de Raspberry Pi se trouve un document indiquant la marche à suivre pour installer un système d'exploitation (`www.raspberrypi.org/quick-start-guide`). Il suffit de formater la carte SD et d'y copier NOOBS (New Out of Box Software) que l'on trouve également sur le site officiel (`www.raspberrypi.org/downloads`). NOOBS permet de choisir le système d'exploitation que l'on veut installer au premier démarrage. Plusieurs OS sont disponibles : Raspbian, Pidora, RaspBMC, OpenELEC, Risc OS et Arch.

Pour démarrer, il suffit d'insérer la carte SD dans le Raspberry Pi, de connecter l'alimentation, un clavier et une souris. Ensuite on choisit le système d'exploitation voulu. Ici nous installons Raspbian. Une fois l'installation faite, nous arrivons dans la console principale. L'identifiant par défaut est `pi` et le mot de passe `raspberry`. Pour démarrer l'interface graphique, il suffit de taper `startx`.

B.2 Contrôler le Raspberry Pi à distance

Pour commander le Raspberry Pi a distance, nous pouvons nous connecter via SSH. Pour savoir l'IP du Raspberry, il faut taper la commande `ifconfig`. Il faut également installer SSH avec la commande `sudo apt-get install ssh`.

L'accès à SSH se fait de cette manière : `service ssh` ou `/etc/init.d/ssh` suivi de la commande voulue, par exemple `status, start, restart, ...`

Sur le Raspberry Pi, il faut entrer la commande `sudo raspi-config` pour entrer dans le menu de configuration, ensuite aller dans le menu *Advanced Options* puis *SSH*. Là il est possible de permettre la connexion en SSH.

Ensuite pour se connecter il suffit de taper `ssh login@machine` où le login est *pi* et la machine est l'adresse IP (par exemple 153.109.5.120). Nous nous trouvons alors dans la console du Raspberry Pi depuis une autre machine. Pour se déconnecter, il faut entrer la commande `logout`.

Pour transférer des fichiers depuis une machine vers le Raspberry, nous pouvons utiliser SCP avec par exemple la commande suivante : `scp utils.hpp pi153.109.5.120:/home/pi/Desktop/utils.hpp` où *utils.hpp* est le fichier à transférer et ensuite il y a les identifiants de la machine avec le chemin où on veut le mettre.

Admettons que nous connectons le Raspberry Pi sur un réseau local. Pour savoir son IP, la commande `nmap -sP 192.168.1.*` scanne le réseau et donne l'IP des appareils connectés sur le même sous-réseau. Pour connaître le sous-réseau dans lequel on se situe, un `ifconfig` suffit.

Annexe C

Base de données

C.1 Installation d'un serveur MySQL

Voici comment installer un serveur MySQL sur une machine Linux (que ça soit un ordinateur ou le Raspberry Pi).

```
1  apt-get install mysql-server python-mysqldb
```

FIGURE C.1 – Commande pour installer MySQL

Ensuite il faut définir un mot de passe pour l'utilisateur root.

Pour entrer dans MySQL il faut taper `mysql -u root -p`. Ensuite il faut créer la base de données ainsi qu'un utilisateur qui pourra y accéder. Cet utilisateur sera le script qui contrôle le lecteur de carte. Nous le nommons *"reader"* et lui attribuons le mot de passe *"iLoveCards"*. Voici comment procéder :

```
1  CREATE DATABASE golfUser;
2  USE golfUser;
3  CREATE USER 'reader'@'localhost' IDENTIFIED BY 'iLoveCards';
4  GRANT ALL PRIVILEGES ON golfUser.* TO 'reader'@'localhost'
5  FLUSH PRIVILEGES;
```

FIGURE C.2 – Création d'une base de données MySQL

On peut ensuite quitter MySQL avec la commande `quit`. Pour créer une table, on se connecte à nouveau à MySQL, mais cette fois avec le compte que nous venons de créer : `mysql -u reader -p`.

```
1  CREATE TABLE test (id NUMERIC, tdate DATE, ttime TIME, zone TEXT, temperature NUMERIC);
```

FIGURE C.3 – Commande pour installer MySQL

Pour voir toutes les bases de données qu'il y a sur le serveur, il faut entrer la commande `show databases;` dans le prompt mysql.

C.2 Installation de MongoDB

Pour installer MongoDB sur un systèmes Linux, il suffit d'entrer les commandes suivantes :

```
1 sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
2 sudo echo "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen" >> /etc/apt/sources.list
3 sudo apt-get update
4 sudo apt-get install mongodb-10gen
```

FIGURE C.4 – Installation de MongoDB

Il faut ensuite créer un répertoire où stocker les bases de données.

```
1 sudo su
2 mkdir -p /data/db
3 exit
```

FIGURE C.5 – Création d'un dossier à la racine pour stocker les bases

Il faut changer les droits sur ce dossier avec `sudo chown mongodb:mongodb /data/db`. Et pour lancer le serveur, il suffit de taper `mongod`. Cela va créer quelques fichiers dans */data/db*. Il est possible que ça ne fonctionne pas tout de suite car les fichiers et dossiers créés n'ont pas les bons droits. Il faut faire en sorte d'avoir les droits suivants (figure C.6), cela se fait avec la commande *chown*. Les fichiers à l'intérieur du dossier *journal* doivent avoir les mêmes droits que le dossier.

```
jacky@grid: ~/Documents/Hes/code
jacky@grid:/$ ls -la /data/db
total 81936
drwxr-xr-x 3 mongodb mongodb     4096 Mar 27 10:33 .
drwxr-xr-x 3 root    root        4096 Mar 27 10:34 ..
drwxr-xr-x 2 mongodb nogroup     4096 Mar 27 10:33 journal
-rw------- 1 mongodb nogroup 67108864 Mar 27 10:33 local.0
-rw------- 1 mongodb nogroup 16777216 Mar 27 10:33 local.ns
-rwxr-xr-x 1 mongodb nogroup        0 Mar 27 10:33 mongod.lock
```

FIGURE C.6 – Droits du contenu du dossier data

A présent l'utilisateur *mongodb* peut accéder à ce dossier.

C.2.1 Installation de PyMongo

PyMongo est une librairie pour utiliser MongoDB en Python.

Pour l'installer, rien de plus simple. On utilise pip qui est un outil pour installer les packages Python facilement. Si `pip` n'est pas déjà installé, il faut le faire :

```
1 sudo apt-get install python-pip
```

FIGURE C.7 – Installation de pip

Puis installer PyMongo :

```
1 sudo pip install pymongo
```

FIGURE C.8 – Installation de PyMongo avec pip

Annexe D

API

D.1 Faire tourner le programme en tâche de fond

Lorsqu'on lance le script des API, nous écrivons :

```
1 sudo python /var/www/nfcapi/api.py
```

FIGURE D.1 – Lancer le script

Si une erreur disant que l'adresse est déjà utilisée, cela veut dire qu'un service est déjà connecté au port 80. Il faut donc le désactiver. Si c'est un serveur Apache qui l'utilise, il faut taper la commande suivante pour l'arrêter :

```
1 sudo service apache2 stop
```

FIGURE D.2 – Arrêter le serveur Apache

Il suffit ensuite de relancer le script et le serveur écoute le port 80. Mais lorsque nous quittons le terminal, le script s'arrête. Pour éviter ce problème et laisser tourner le script, il faut utiliser la commande `screen` et lancer le script à l'intérieur.

```
1 screen
2 sudo python /var/www/nfcapp/api.py
```

FIGURE D.3 – Lancer le script dans l'environnement screen

Ensuite il faut taper "`ctrl + a`" puis "`d`", ce qui a pour effet de sortir de l'environnement tout en laissant le script continuer.

Pour voir les différents environnements, il faut entrer :

```
1 screen -ls
```

FIGURE D.4 – Lister les environnements `screen`

Avec cette commande, on voit toutes les instances de "`screen`". Une instance est soit `Attached`, soit `Detached`. Pour arrêter le script, il faut entrer dans le screen, ou le rendre "Attached". Pour ce faire, il faut taper la commande qui suit, où "3053.pts-0.vlenfc" est le numéro du screen auquel on veut accéder.

```
1 screen -r 3053.pts-0.vlenfc
```

FIGURE D.5 – Entrer dans un `screen`

Annexe E

Astuces Python

E.1 VIRTUALENV

Virtualenv est un outil qui permet de créer un environnement isolé du système pour pouvoir lancer un logiciel Python et y installer les librairies utiles au projet.

Pour l'installer, il suffit de taper la commande :

```
1 sudo pip install virtualenv
```

FIGURE E.1 – Installation de *virutalenv* avec *pip*

Le but est donc de créer un environnement dans le projet courant. Donc premièrement il faut se déplacer dans le dossier du projet à coup de `cd` et créer l'environnement (ici nous appelons l'environnement *"venv"*) :

```
1 virtualenv venv
```

FIGURE E.2 – Création d'un environnement nommé *"venv"*

Après l'exécution de cette commande, un dossier a été créé, il contient des raccourcis vers les différentes version de Python ainsi que vers les librairies Python de base. Pour rendre cet environnement effectif, il faut l'activer avec la commande :

```
1 source venv/bin/activate
```

FIGURE E.3 – Activation de l'environnement *"venv"*

Nous voyons dans l'invite de commande que la ligne est précédée par le nom de l'environnement entouré de parenthèses.

```
1 before :    jacky@grid:~/Documents/Projects/MyProject
2 after :    (venv)jacky@grid:~/Documents/Projects/MyProject
```

FIGURE E.4 – Etat de l'invite de commande à l'état normal et dans un environnement

Nous pouvons maintenant installer tous les packages utiles et lancer notre programme Python. Ainsi les packages ne seront pas installés sur tout le système.

Pour sortir de l'environnement, il suffit de taper :

```
1 deactivate
```

FIGURE E.5 – Quitter l'environnement

Si l'envie nous vient de supprimer l'environnement, il suffit juste de supprimer le dossier au nom de l'environnement créé.

Pour plus d'informations sur *virtualenv*, dirigez-vous sur le site officiel :

`http://virtualenv.readthedocs.org/en/latest/virtualenv.html`

E.2 PIP

Le logiciel *pip* a souvent été utilisé dans ce projet pour installer des librairies et packages Python. Voici donc plus en détail son fonctionnement et quelques astuces d'utilisation.

Premièrement il faut installer *pip* avec la commande :

```
1 sudo apt-get install python-pip
```

FIGURE E.6 – Installation de *pip*

Pip nous fournit ensuite la possibilité d'installer un package simplement en tapant :

```
1 pip install libName
```

FIGURE E.7 – Installation du package *"libName"* (nom fictif) avec *pip*

Pour lister tous les packages installés avec *pip*, une commande est disponble. En ajoutant l'argument "`--outdated`", nous pouvons savoir tous les packages dont une nouvelle version est disponible. Voici ces deux commandes :

```
1 pip list
2 pip list --outdated
```

FIGURE E.8 – Listing des tous les packages et des packages périmés

Pour mettre à jour une librairie que nous avons vu dans la liste *outdated*, il suffit de faire :

```
1 pip install --upgrade libName
```

FIGURE E.9 – Mise à jour d'une librairie

Pour avoir des informations comme la version installée, la localisation, et les dépendances, la commande `show` suivie du nom du package est là pour ça :

```
1 pip show libName
```

FIGURE E.10 – Mise à jour d'une librairie

S'il arrivait qu'un package devait être supprimé, il est possible de le faire :

```
1 pip uninstall libName
```

FIGURE E.11 – Suppression d'un package

Une fonctionnalité très intéressante de *pip* est sa capacité à installer plusieurs librairies à la volée en lisant un fichier texte *"requirements.txt"*. Ce fichier doit contenir sur chaque ligne le nom du package voulu. Si seul le nom du package est écrit, la dernière version disponible sera installée. Si par contre il est précisé une version séparé par un "==", c'est cette version qui sera installée. Et si la version est séparée par ">=", il s'agit de la version minimale qui sera installée.

Voici un exemple d'un fichier *"requirements.txt"* :

```
1 Flask==0.10.1
2 pymongo>=2.6.3
3 oauthlib>=0.6.1
```

FIGURE E.12 – Exemple d'un fichier *"requirements.txt"*

Pour installer tous les packages du fichier il faut entrer la commande qui suit :

```
1 pip install -r requirements.txt
```

FIGURE E.13 – Installation de tous les packages du fichier *"requirements.txt"*

Il peut être utile d'utiliser un fichier de ce type pour faire une installation facile d'un programme sur différentes machines. Couplé à *virtualenv* il est encore plus utile.

De plus *pip* offre la possibilité de télécharger des dépôts Git, SVN, Bazaar ou Mercurial assez facilement.

Pour plus d'information sur *pip*, suivez le lien suivant : `https://pip.pypa.io/en/latest/`

Annexe F

Git

F.1 Présentation

Wikipédia nous donne comme explicatif le texte suivant :

> Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, créateur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. [1]

Git est donc un logiciel de gestion de versions de fichiers, tout comme Subversion, Bazaar ou Mercurial. Il permet de garder un historique des fichiers et de tous les changements effectués sur eux, qu'on travaille seul ou à plusieurs.

FIGURE F.1 – Logo de Git

source : `http://git-scm.com/images/logos/downloads/Git-Logo-2Color.png`

F.2 Dépôts

Il existe des sites qui fournissent des hébergements pour les dépôts, ou *repositories*. Cela facilite la mise en place d'un projet. Mais il est également possible d'héberger des dépôts sur un serveur privé.

Le plus connu des hébergeurs de dépôts est *Github*. Il s'agit même d'une sorte de réseau social sur lequel chaque utilisateur possède un profil. Ce profil montre les projets personnels de l'utilisateur ainsi que les projets auxquels il a participé. Le profil affiche également quelques statistiques ainsi qu'une visualisations du nombre de contributions publiques par jour de l'utilisateur. La figure F.2 montre un exemple de cette visualisation d'un utilisateur pris aléatoirement sur le site. Chaque case représente un jour, et plus la case est foncée plus le nombre de contributions est grand.

1. Source : `http://fr.wikipedia.org/wiki/Git`, page consultée en juillet 2014.

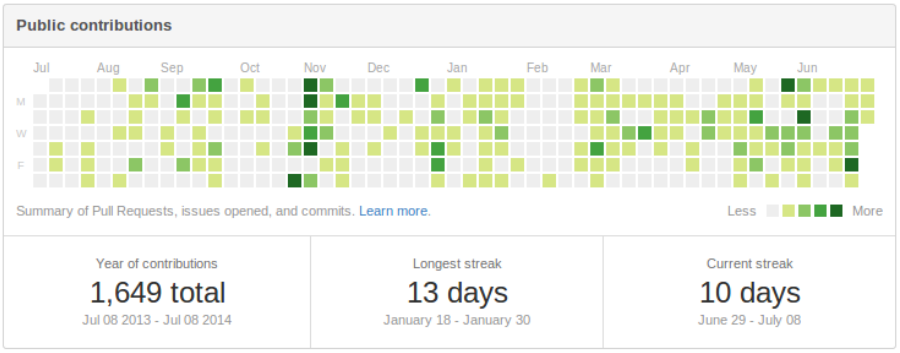


FIGURE F.2 – Visualisation des contributions publiques d'un utilisateur de *Github*

Un autre site web hébergeur de dépôts est *Bitbucket*. Celui-ce est un peu moins *"social"*, mais il offre tous les outils nécessaires à une bonne gestion du code.

Ces deux services ont des versions gratuites et payantes dont les conditions diffèrent. Par exemple la version gratuite de *Github* ne permet pas de faire de dépôts privés, au contraire de la version gratuite de *Bitbucket*. Par contre la version gratuite de *Bitbucket* ne permet pas de collaborer à plus de cinq utilisateurs.

F.3 UTILISATION USUELLE

Pour installer Git sur Linux, un simple *apt-get* suffit :

```
1 apt-get install git
```

FIGURE F.3 – Installation de Git

Voici un petit aperçu des commandes *Git* utilisées constamment lorsqu'on travaille sur un projet versionné.

Premièrement, pour savoir si un fichier a changé dans notre projet, il faut taper :

```
1 git status
```

FIGURE F.4 – Savoir l'état du projet

Si une modification du code est terminée et fonctionne, il est temps de faire un *commit*, soit de regrouper ces modifications dans une sorte de paquet auquel on peut donner un nom et qui possèdera un numéro unique pour pouvoir y référer plus tard. Pour ce faire il faut d'abord ajouter les fichiers modifiés avec la commande *"add"*, puis *"commiter"* ces fichiers comme ceci :

```
1 git add .
2 git commit -m "mon message de commit"
```

FIGURE F.5 – Création d'un *commit*

Le point qui suit le *add* signifie le dossier courant, donc tous les fichiers modifiés qui sont dans le dossier courant. Mais si seuls certains fichiers doivent appartenir à un *commit*, il est possible de les ajouter un par un en écrivant leurs noms séparés par des espaces après la commande *add*. Ensuite le paramètre *"-m"* permet d'ajouter un commentaire au *commit*. Celui-ci devra être en anglais par convention et décrire la modification faite au code en quelques mots. Si le message est oublié, un éditeur de texte en console s'ouvrira pour que vous écriviez le message.

Pour vérifier que tout a bien été ajouté avant de faire le *commit*, on peut faire un "`git status`". Et si certains fichiers ne s'ajoutent pas avec un simple "`git add .`", comme par exemple les fichiers ajoutés ou supprimés en cours de route au projet, on peut forcer tous les fichiers à s'ajouter avec la commande "`git add -all`".

Un *commit* ne va pas rendre vos changements effectifs sur le serveur (dépôt). Une fois que vous avez fait quelques *commits*, ou en fin de journée, il faudra les pousser sur le serveur. Cela se fait avec la commande "`push`". Voici donc comment *pusher* vos *commits* sur le dépôts principal :

```
1 git push origin master
```

FIGURE F.6 – *push* des modifications sur le serveur

Le paramètre après la commande *push* correspond à la branche à laquelle vous voulez apporter les modifications. Pour plus d'informations sur les branches, rendez-vous sur le site officiel de Git ou sur les différents tutoriaux qui se trouvent sur le net.

Une dernière commande à utiliser à tout prix si vous travaillez à plusieurs ou alors si vous travaillez sur plusieurs machines : le *pull*. En effet, si quelqu'un a poussé des modifications sur le serveur, la version du code que vous avez sur votre ordinateur n'est plus à jour, il faut donc aller chercher les dernières versions sur le serveur. Pour cela il faut aller "tirer" sur le serveur avec la commande *pull* comme ceci :

```
1 git pull
```

FIGURE F.7 – Rapatriement de la dernière version du dépôt

Cette commande est à effectuer avant de commencer à coder. Si vous essayez de *pusher* vos *commits* avant d'avoir *pullé* et qu'il y avait effectivement des nouveautés sur le serveur, Git va vous indiquer qu'il faut d'abord *puller* car des confilts de version peuvent apparaître.

Git est donc simple d'utilisation puisqu'il requiert de connaître seulement cinq commandes basiques. Pour un usage un peu plus avancé, il est par contre conseillé de se renseigner sur les branches, la création de dépôts, les forks et les autres capacités de Git.

Git est utilisable facilement en ligne de commande et est disponible sur beaucoup de systèmes d'exploitation. De plus, des interfaces graphiques sont disponibles pour les allergiques à la console.

F.4 DÉBUTER UN PROJET GIT

Créer un dépôt sur *Bitbucket* ou *Github* est facile, il suffit de cliquer sur un bouton du style *"new"* ou *"create"* et de suivre les indications pour lui donner un nom et quelques autres informations. Mais ensuite il faut le lier avec le projet qui est sur votre ordinateur, que vous ayez déjà du code ou non. Pour ce faire, il va falloir taper ces quelques lignes :

```
1 cd /path/to/your/project
2 git init
3 git remote add origin https://username@hostIp/username/repoName.git
```

FIGURE F.8 – Initialiser Git

La première ligne sert à se déplacer dans le dossier de votre projet. Ensuite "`git init`" initialise Git et pour finir la troisième ligne va lier votre dossier de projet avec le dossier sur le dépôt. Il faut bien évidemment remplacer *"username"* par votre nom d'utilisateur et *"hostIp"* par l'hébergeur du dépôt. Dans le cas de *Github* ça sera *"github.com"* et pour *Bitbucket* c'est *"bitbucket.org"*.

Il ne vous reste ensuite qu'à faire un "`git push origin master`" pour pousser votre code sur le dépôt et vous pouvez commencer à coder.

F.5 CLONER UN PROJET EXISTANT

Si vous voulez cloner un projet existant, c'est-à-dire faire une copie locale d'un dépôt, voici la commande à utiliser :

```
1  git clone https://yourUsername@hostIp/projectOwnerUsername/repoName.git
```

FIGURE F.9 – Cloner un dépôt

Dans le cas ci-dessus, un dossier nommé *"repoName"* va être créé à l'endroit où vous vous trouvez. Il est également possible de changer le nom du dossier créé en ajoutant simplement ce nom à la suite de la commande, comme ceci :

```
1  git clone https://yourUsername@hostIp/projectOwnerUsername/repoName.git myNewName
```

FIGURE F.10 – Cloner un dépôt en changeant le nom du dossier de projet en local

F.6 CRÉER DES ALIAS

Lorsqu'on utilise Git, beaucoup de commandes sont tapées et ce sont très souvent les cinq commandes basiques présentées plus haut. Il existe un moyen de créer des *alias* pour raccourcir vos commandes.

```
1  git config --global alias.myAlias theCommand
```

FIGURE F.11 – Créer un alias

Le paramètre "`-global`" permet de pouvoir utiliser cet alias dans tous les projets de votre machine. Si vous ne le mettez pas, il sera utilisable uniquement pour ce projet. Si la commande que vous voulez remplacer contient des espaces, il faut l'entourer de guillemets. Voici par exemple des alias qui pourraient être utiles :

```
1  git config --global alias.st status
2  git config --global alias.pl pull
3  git config --global alias.ps 'push origin master'
4  git config --global alias.ad 'add .'
```

FIGURE F.12 – Exemples d'alias utiles

La figure suivante montre un exemple d'utilisation sans les alias, puis avec les alias décrits plus haut :

```
1  # sans les alias
2  git status
3  git pull
4  git add .
5  git commit -m "comment"
6  git push origin master
7
8  # avec les alias
9  git st
10 git pl
11 git ad
12 git commit -m "comment"
13 git ps
```

FIGURE F.13 – Exemples d'utilisation avec et sans alias

Pour une simple modification de code suivie d'un *push*, nous économisons déjà 25 caractères.

Pour donner un ordre de grandeur, ce projet a nécessité envrion 110 *commits* pour tous les logiciels ainsi que le rapport fait en LaTeX.

F.7 Liens

Voici différents liens utiles :

- Site officiel de Git : `http://git-scm.com`
- Github : `https://github.com`
- Bitbucket : `https://bitbucket.org`
- Tutoriel Git : `https://www.atlassian.com/fr/git/tutorial`

Annexe G

Doxygen

Doxygen est un logiciel qui permet de généré la documentation d'un projet dans un format web. C'est-à-dire des fichiers HTML et Javascript pour que la documentation soit facilement visualisable comme un site web. De plus, Doxygen supporte différents langages de programmation tels que le C, le C++, le Java, le Python, le PHP, le VHDL, etc.

Pour l'installation, voici un lien qui explique bien comment faire : `http://www.stack.nl/~dimitri/doxygen/download.html`

Il est possible d'éditer un fichier de configuration et de lancer une commande pour générer la doc, mais il existe également une interface graphique pour le faire : `doxywizard`.

Si la documentation est générée sur du code brut, Doxygen détecte les classes, les attributs, les méthodes et les héritages facilement. Mais le but d'une doc est d'expliquer à quoi sert chaque méthode, classe, etc. Donc une synthaxe spéciale est à ajouter dans le code pour ajouter ces explicatifs. De plus il est utile de décrire la version d'un fichier, le nom de la personne qui l'a implémenté, la date de création ou de dernière modification ainsi que d'autres informations. Tout cela est possible avec Doxygen.

Doxygen peut également générer des graphes de dépendance, d'héritage ou d'implémentation d'interface. Il suffit d'activer cette option dans les configurations.

Voici un exemple de documentation pour du code Python :

```
1  class CardReader(QObject):
2      """! @brief
3      All actions from and to the NFC card reader are handled here.
4      @author CASAS Jacky
5      @date 22.06.14
6      @version 1.0
7      """
8
9      def __init__(self, action):
10         """! @brief Link an Action instance, create a cardrequest and a timer.
11         @param self the CardReader instance
12         @param action an instance of Action
13         """
14         QObject.__init__(self)
15
16         ## DCCardType instance
17         self.cardtype = DCCardType()
18         ## card request to make a connection with a smartcard
19         self.cardrequest = CardRequest(timeout=0.5, cardType=self.cardtype)
```

FIGURE G.1 – Documentation d'un bout de code en Python

Nous voyons dans cet exemple la documentation d'une classe. Elle commence par `"""!` et se termine par `"""`. Elle contient des balises précédées par un `@` ainsi que le contenu. C'est assez intuitif. Pour documenter

un attribut, on utilise un double dièse (##) suivi de la description. Pour du code Java, c'est le même principe. La manière de délimiter les bouts de doc divergent un peu en fonction du langage.

Pour afficher une documentation, il suffit d'ouvrir le fichier *"index.html"* dans un navigateur web. La figure G.2 monte un exemple de rendu d'une documentation générée avec Doxygen.

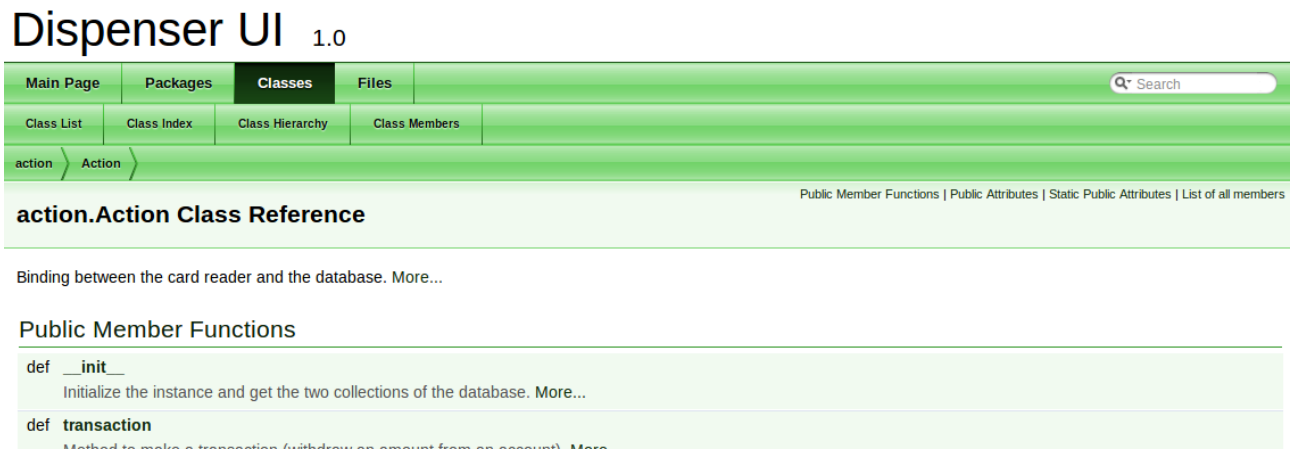


FIGURE G.2 – Exemple de documentation

Voici un lien vers le site web de Doxygen :
`http://doxygen.org`

Annexe H

Code source

H.1 Explications

Pour ce projet, trois logiciels ont été développés, les voici :

1. **dispenserui** : Le logiciel principal développé pour le distributeur de balles de golf sera exécuté sur le Raspberry Pi. Ce logiciel est développé en Python.
2. **nfcapi** : Le système d'API destiné au serveur où se trouve la base de données. Il est également implémenté en Python.
3. **GolfWallet** : L'application Android qui permet d'utiliser son smartphone pour acheter des balles de golf. Elle est développée en Java.

Certains fichiers de code se trouvent dans les chapitres suivants de cette annexe, mais tous les fichiers ne sont pas listés ici par peur de surcharger le contenu de ces annexes et également parce que le projet Android contient beaucoup de fichiers XML. Par contre les trois projets sont disponibles sur *Github* dans des dépôts publics. Pour obtenir un projet, il suffit de le cloner sur votre ordinateur. Une version du code est également disponible sur le CD rendu avec le travail de diplôme.

Voici les liens de ces projets :

1. Projet **dispenserui** : `https://github.com/acknowledge/NFCGolfBallDispenser-UI`
2. Projet **nfcapi** : `https://github.com/acknowledge/NFCGolfBallDispenser-API`
3. Projet **GolfWalet** : `https://github.com/acknowledge/NFCGolfBallDispenser-AndroidApp`

De plus, pour mieux appréhender le code, une documentation a été créée pour le projet *dispenserui* ainsi que pour le projet *GolfWallet*. Pour y accéder, il faut cloner le projet et ouvrir le fichier *"index.html"* dans un navigateur. Ce fichier se trouve aux liens relatifs suivants :

- "`dispenserui/doc/html/index.html`" pour le projet **dispenserui**
- "`GolfWallet/src/doc/html/index.html`" pour le projet **GolfWallet**

H.2 LOGICIEL EN PYTHON POUR LE DISTRIBUTEUR DE BALLES DE GOLF

H.2.1 Diagramme de classe complet

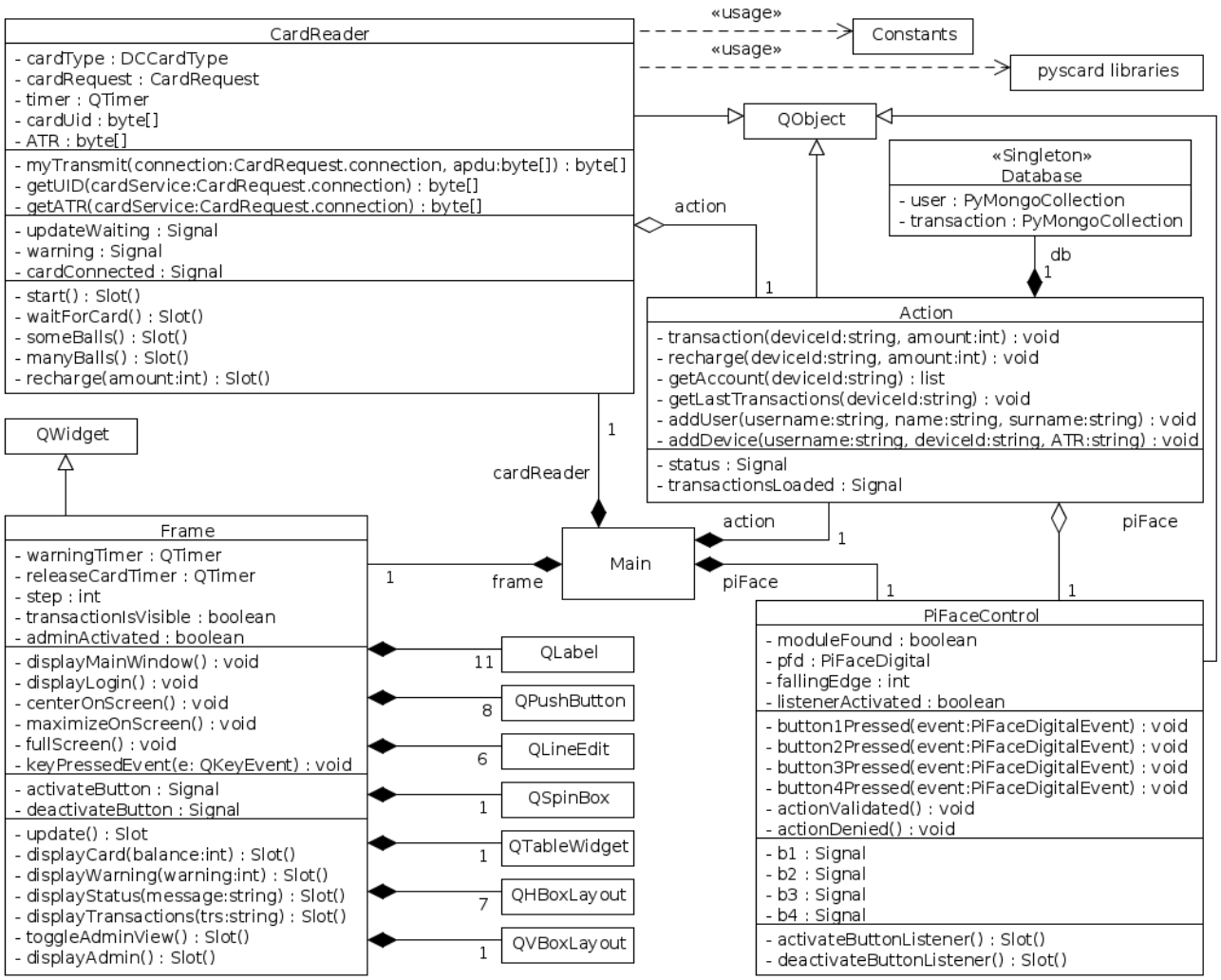


FIGURE H.1 – Diagramme de classe complet

H.2.2 main.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

## @file main.py
#  File containing the main fonction. Entry of the code.

## @package main
#  Main.
#  @author CASAS Jacky
#  @date 22.06.14
#  @version 1.0

## @mainpage Dispenser UI documentation
#  @section intro_sec Introduction
#  This the documentation of the **NFC Golf Ball Dispenser**.
#
#  A NFC Card Reader is plugged in a Raspberry Pi.
#  The Raspberry Pi communicate with a remote MongoDB database and display information to the customer
#  via a screen.
#
#  @section overview_sec Overview
#  This image shows an overview of the interface of this project.
#  ![Overview](../img/overview.png)
#
#  @section classdiagram_sec Class Diagram
#  And here it is the complete class diagram.
#  ![Class Diagram](../img/classDiagram.png)
#
#  @section license_sec License
#  This software is developped by Jacky CASAS and was commissioned by Digiclever.
#  For further information about the license, please contact Digiclever (http://digiclever.com).
#
#  This project was developped at HES-SO//Valais during the bachelor thesis of Jacky Casas in 2014.

from cardreader import CardReader
from ui import Frame
from action import Action
from pifacecontrol import PiFaceControl
import sys
from PySide import QtGui
from PySide.QtCore import *
from functools import partial

def main():
    """! @brief Main of the software"""

    ## Qt Application
    app = QtGui.QApplication(sys.argv)
    app.setOrganizationName('Digiclever')
    app.setApplicationName('NFC Golf Ball Dispenser')
    ## Frame instance
    frame = Frame()
    frame.show()

    ## PiFaceControl instance
    piface = PiFaceControl()
    ## Action instance
    action = Action(piface)
    ## CardReader instance
    cardReader = CardReader(action)

    # connect signals to slots
    frame.b1.clicked.connect(cardReader.someBalls)
    frame.b2.clicked.connect(cardReader.manyBalls)
    frame.transaction.clicked.connect(lambda: action.getLastTransactions(cardReader.cardUid))
    frame.admin.clicked.connect(frame.toggleAdminView)
    frame.log.clicked.connect(lambda: frame.displayAdmin(frame.adminUsername.text(),
        frame.adminPassword.text()))
    frame.bRecharge.clicked.connect(lambda: cardReader.recharge(frame.moneyBox.value()))
    frame.bCreateAccount.clicked.connect(lambda: action.addUser(frame.username.text(),
```

```
71          frame.name.text(), frame.surname.text()))
72      frame.bAddDevice.clicked.connect(lambda: action.addDevice(frame.username2.text(),
73          cardReader.cardUid, cardReader.ATR))
74
75      action.status.connect(frame.displayStatus)
76      action.transactionsLoaded.connect(frame.displayTransactions)
77
78      frame.connect(frame.warningTimer, SIGNAL("timeout()"), cardReader.start)
79      frame.connect(frame.releaseCardTimer, SIGNAL("timeout()"), cardReader.start)
80
81      cardReader.connect(cardReader.timer, SIGNAL("timeout()"), cardReader.waitForCard)
82      cardReader.updateWaiting.connect(frame.update)
83
84      cardReader.cardDetected.connect(frame.displayCard)
85      cardReader.warning.connect(frame.displayWarning)
86
87      frame.activateButton.connect(piface.activateButtonListener)
88      frame.deactivateButton.connect(piface.deactivateButtonListener)
89
90      piface.b1.connect(cardReader.someBalls)
91      piface.b2.connect(cardReader.manyBalls)
92      piface.b3.connect(lambda: action.getLastTransactions(cardReader.cardUid))
93      piface.b4.connect(frame.toggleAdminView)
94
95      cardReader.start()
96      sys.exit(app.exec_())
97
98  if __name__ == '__main__':
99      main()
```

H.2.3 constantes.py

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-
3
4   ## @file constants.py
5   #  File for constants.
6
7   ## @package constants
8   #  Constants.
9   #  @author CASAS Jacky
10  #  @date 22.06.2014
11  #  @version 1.0
12
13  ## dispenser identifier, generate a new one for each dispenser
14  DISPENSER_ID = '84448f2c-f1e8-46db-a637-b813c1e2dd2a'
15
16  ## username for the admin
17  ADMIN_USERNAME = 'admin'
18  ## password for the admin
19  ADMIN_PASSWORD = 'asdf'
20
21  ## transaction type for a recharge
22  RECHARGE = 0
23  ## transaction type for a withdrawal
24  WITHDRAWAL = 1
25
26  ## user statement active
27  STA_USER_ACTIVE = 1
28  ## user statement inactive
29  STA_USER_INACTIVE = 2
30  ## user statement deleted
31  STA_USER_DELETED = 3
32
33  ## device status active
34  STA_DEVICE_ACTIVE = 1
35  ## device status lost
36  STA_DEVICE_LOST = 2
37  ## device status stolen
```

```
38 STA_DEVICE_STOLEN = 3
39 ## device status deleted
40 STA_DEVICE_DELETED = 4
41
42 ## default smartphone id
43 DEFAULT_SMARTPHONE_ID = 'AA AA AA AA'
44
45 ## warning no account
46 WARN_NO_ACCOUNT = 0
47 ## warning account inactive
48 WARN_ACCOUNT_INACTIVE = 1
49 ## warning account deleted
50 WARN_ACCOUNT_DELETED = 2
51 ## warning device lost
52 WARN_DEVICE_LOST = 3
53 ## warning device stolen
54 WARN_DEVICE_STOLEN = 4
55 ## warning device deleted
56 WARN_DEVICE_DELETED = 5
57
58 ## APDU to read reader firmware version
59 READER_FIRMWARE_VERSION = [0xFF, 0x00, 0x48, 0x00, 0x00]
60 ## APDU to get device UID
61 GET_UID = [0xFF, 0xCA, 0x00, 0x00, 0x00]
62
63 ## beginning of the APDU to get Android UID
64 CLA_INS_P1_P2 = [0x00, 0xA4, 0x04, 0x00]
65 ## Android AID, part of the APDU to get Android UID
66 AID_ANDROID = [0xF0, 0x01, 0x02, 0x03, 0x03, 0x02, 0x01]
67 #AID_ANDROID = [0xF0, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06]
```

H.2.4 frame.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  ## @file ui.py
5  #  Contains the class Frame.
6
7  ## @package ui
8  #  Graphical User Interface made with PySide.
9  #  @author CASAS Jacky
10 #  @date 22.06.2014
11 #  @version 1.0
12
13 import sys
14 import os
15 from PySide import QtGui
16 from PySide.QtGui import *
17 from PySide.QtCore import *
18 from constants import *
19
20 class Frame(QtGui.QWidget):
21     """! @brief
22     Main window of the software.
23     @author CASAS Jacky
24     @date 22.06.14
25     @version 1.0
26     """
27     ## Signal used to activate PiFace buttons
28     activateButton = Signal()
29     ## Signal used to deactivate PiFace buttons
30     deactivateButton = Signal()
31
32     def __init__(self, parent=None):
33         """! @brief Frame constructor.
34         @param self the Frame instance
35         @param parent the parent QWidget, optional
36         """
```

```
        QtGui.QWidget.__init__(self, parent)

        #self.resize(600,500)
        self.setFont(QtGui.QFont("Verdana"))
        self.setWindowTitle("NFC Golf Ball Dispenser")

        self.setGeometry(0, 0, 650, 550)
        self.centerOnScreen()
        #self.setMinimumWidth(370)
        self.setMinimumHeight(470)

        ## Label to display a welcome message
        self.l1 = QtGui.QLabel("Welcome<br />on<br />NFC Golf Ball Dispenser", self)
        self.l1.setStyleSheet('font-size:30pt; qproperty-alignment:AlignCenter; qproperty-wordWrap:true;')
        ## Label to display the name of the developer
        self.l2 = QtGui.QLabel('created by Jacky Casas', self)
        self.l2.setStyleSheet('font-size:10pt; qproperty-alignment:AlignRight; qproperty-wordWrap:true;')
        self.l2.setFixedHeight(16)
        ## Label to display instructions
        self.l3 = QtGui.QLabel('Just approach your smartphone or smartcard in front of the reader', self)
        self.l3.setStyleSheet('font-size:20pt; qproperty-alignment:AlignCenter; qproperty-wordWrap:true;')
        ## Label to display the status (waiting for a card or card detected)
        self.l4 = QtGui.QLabel('waiting', self)
        self.l4.setStyleSheet('font-size:12pt; qproperty-alignment:AlignCenter; ',
         'background-color:#E0E0E0; padding:4px; margin:10px 30px;')
        ## Label to display a welcome message in the log view
        self.logl1 = QtGui.QLabel("Welcome on the login of the administration")
        self.logl1.setStyleSheet('font-size:30pt; qproperty-alignment:AlignCenter; qproperty-wordWrap:true;')
        self.logl1.setVisible(False)
        ## Label to display a welcome message in the admin view
        self.adminl1 = QtGui.QLabel("Welcome on the admin")
        self.adminl1.setStyleSheet('font-size:30pt;qproperty-alignment:AlignCenter;qproperty-wordWrap:true;')
        self.adminl1.setVisible(False)
        ## LineEdit to enter the username
        self.adminUsername = QtGui.QLineEdit()
        self.adminUsername.setPlaceholderText('username')
        self.adminUsername.setVisible(False)
        ## LineEdit to enter the password
        self.adminPassword = QtGui.QLineEdit()
        self.adminPassword.setPlaceholderText('password')
        self.adminPassword.setVisible(False)
        self.adminPassword.setEchoMode(QtGui.QLineEdit.Password)

        ## Button to enter in the admin
        self.log = QtGui.QPushButton('Login')
        self.log.setVisible(False)
        hLayoutLogin = QtGui.QHBoxLayout()
        hLayoutLogin.addWidget(self.adminUsername)
        hLayoutLogin.addWidget(self.adminPassword)
        hLayoutLogin.addWidget(self.log)

        ## Button to withdraw 2 CHF
        self.b1 = QtGui.QPushButton('2 CHF - 30 balls', self)
        self.b1.setEnabled(False)
        ## Button to withdraw 5 CHF
        self.b2 = QtGui.QPushButton('5 CHF - 80 balls', self)
        self.b2.setEnabled(False)
        ## Button to show the last 10 transactions
        self.transaction = QtGui.QPushButton('last transactions')
        self.transaction.setEnabled(False)
        ## Button to access the admin view
        self.admin = QtGui.QPushButton('admin')

        hLayoutButtons = QtGui.QHBoxLayout()
        hLayoutButtons.addWidget(self.b1)
        hLayoutButtons.addWidget(self.b2)
        hLayoutButtons.addWidget(self.transaction)
        hLayoutButtons.addWidget(self.admin)

        ## Label indicating the recharge of an account
        self.lRecharge = QtGui.QLabel('Recharge the account :')
        self.lRecharge.setVisible(False)
```

```
109         ## SpinBox to enter the amount we want to recharge
110         self.moneyBox = QtGui.QSpinBox()
111         self.moneyBox.setRange(0, 1000)
112         self.moneyBox.setSingleStep(10)
113         self.moneyBox.setSuffix(' CHF')
114         self.moneyBox.setValue(20)
115         self.moneyBox.setVisible(False)
116         ## Button to recharge an account
117         self.bRecharge = QtGui.QPushButton('Recharge', self)
118         self.bRecharge.setVisible(False)
119 
120         hLayoutRecharge = QtGui.QHBoxLayout()
121         hLayoutRecharge.addWidget(self.lRecharge)
122         hLayoutRecharge.addWidget(self.moneyBox)
123         hLayoutRecharge.addWidget(self.bRecharge)
124 
125         ## LineEdit to write the username to create an account
126         self.username = QtGui.QLineEdit()
127         self.username.setPlaceholderText('username')
128         self.username.setVisible(False)
129         ## LineEdit to write the name to create an account
130         self.name = QtGui.QLineEdit()
131         self.name.setPlaceholderText('name')
132         self.name.setVisible(False)
133         ## LineEdit to write the surname to create an account
134         self.surname = QtGui.QLineEdit()
135         self.surname.setPlaceholderText('surname')
136         self.surname.setVisible(False)
137         ## Button to create an account
138         self.bCreateAccount = QtGui.QPushButton('Create account')
139         self.bCreateAccount.setVisible(False)
140 
141         hLayoutAccount = QtGui.QHBoxLayout()
142         hLayoutAccount.addWidget(self.username)
143         hLayoutAccount.addWidget(self.name)
144         hLayoutAccount.addWidget(self.surname)
145         hLayoutAccount.addWidget(self.bCreateAccount)
146 
147         ## Label indicating that we can add a card to an account
148         self.lAddDevice = QtGui.QLabel('Add smartcard to account :')
149         self.lAddDevice.setVisible(False)
150         ## LineEdit to write the username of the account
151         self.username2 = QtGui.QLineEdit()
152         self.username2.setPlaceholderText('username')
153         self.username2.setVisible(False)
154         ## Button to add a smartcart to an account
155         self.bAddDevice = QtGui.QPushButton('Add smartcard')
156         self.bAddDevice.setVisible(False)
157 
158         hLayoutAddDevice = QtGui.QHBoxLayout()
159         hLayoutAddDevice.addWidget(self.lAddDevice)
160         hLayoutAddDevice.addWidget(self.username2)
161         hLayoutAddDevice.addWidget(self.bAddDevice)
162 
163         ## Label indicating that it is the last 10 transactions
164         self.lTransaction = QtGui.QLabel("The last 10<br />transactions")
165         self.lTransaction.setStyleSheet('font-size:15pt; qproperty-alignment:AlignCenter;',
166             ' qproperty-wordWrap:true;')
167         self.lTransaction.setVisible(False)
168         ## TableWidget to display the 10 last transactions
169         self.transactionTable = QtGui.QTableWidget()
170         self.transactionTable.setVisible(False)
171         self.transactionTable.setColumnCount(3)
172         title = ['date', 'amount', 'currency']
173         vheader = QtGui.QHeaderView(Qt.Orientation.Vertical)
174         vheader.setResizeMode(QtGui.QHeaderView.Stretch)
175         self.transactionTable.setVerticalHeader(vheader)
176         hheader = QtGui.QHeaderView(Qt.Orientation.Horizontal)
177         hheader.setResizeMode(QtGui.QHeaderView.ResizeToContents)
178         hheader.setStretchLastSection(True)
179         self.transactionTable.setHorizontalHeader(hheader)
180         self.transactionTable.setHorizontalHeaderLabels(title)
```

```
181         self.transactionTable.setMaximumWidth(430)
182 
183         hLayoutTransaction = QtGui.QHBoxLayout()
184         hLayoutTransaction.addWidget(self.lTransaction)
185         hLayoutTransaction.addWidget(self.transactionTable)
186 
187         ## Label to indicate that it is the status
188         self.lStatus = QtGui.QLabel()
189         self.lStatus.setStyleSheet('background-color:#E0E0E0; padding:4px;')
190         self.lStatus.setText('Status : ')
191         self.lStatus.setFixedHeight(24)
192         self.lStatus.setFixedWidth(60)
193         ## Label to display the status of an action
194         self.status = QtGui.QLabel()
195         self.status.setStyleSheet('background-color:#E0E0E0; padding:4px;')
196         self.status.setAlignment(Qt.AlignCenter);
197         self.status.setFixedHeight(24)
198 
199         hLayoutStatus = QtGui.QHBoxLayout()
200         hLayoutStatus.addWidget(self.lStatus)
201         hLayoutStatus.addWidget(self.status)
202 
203         # link all the layouts to the main vertical layout
204         verticalLayout = QtGui.QVBoxLayout()
205 
206         verticalLayout.addWidget(self.l1)
207         verticalLayout.addWidget(self.logl1)
208         verticalLayout.addWidget(self.adminl1)
209         verticalLayout.addWidget(self.l2)
210         verticalLayout.addLayout(hLayoutLogin)
211         verticalLayout.addWidget(self.l3)
212         verticalLayout.addWidget(self.l4)
213 
214         verticalLayout.addLayout(hLayoutTransaction)
215         verticalLayout.addLayout(hLayoutRecharge)
216         verticalLayout.addLayout(hLayoutAccount)
217         verticalLayout.addLayout(hLayoutAddDevice)
218         verticalLayout.addLayout(hLayoutButtons)
219         verticalLayout.addLayout(hLayoutStatus)
220 
221         self.setLayout(verticalLayout)
222 
223         try:
224             self.setWindowIcon(QtGui.QIcon('icon.png'))
225         except:pass
226 
227         ## Timer used to display a warning during a fixed time
228         self.warningTimer = QTimer(self)
229         self.warningTimer.setSingleShot(True)
230         ## Timer used to release the card after x seconds if the user don't make any action
231         self.releaseCardTimer = QTimer(self)
232         self.releaseCardTimer.setSingleShot(True)
233         ## Step of the *waiting* text
234         self.step = 0
235         ## Transaction visibility
236         self.transactionIsVisible = False
237         ## Admin visibility
238         self.adminActivated = False
239 
240     @Slot()
241     def update(self):
242         """! @brief Slot used to update the *waiting* label.
243         @param self the Frame instance
244         """
245         self.step += 1
246         self.step %= 4
247         self.l4.setText('waiting ' + self.step * '.')
248         self.b1.setEnabled(False)
249         self.b2.setEnabled(False)
250         self.transaction.setEnabled(False)
251 
252         if self.transactionIsVisible:
```

```
253             self.lTransaction.setVisible(False)
254             self.transactionTable.setVisible(False)
255             self.transactionIsVisible = False
256 
257         self.deactivateButton.emit()
258 
259     @Slot(int)
260     def displayCard(self, balance):
261         """! @brief Slot called when a card is detected. It displays the balance of the account.
262         @param self the Frame instance
263         @param balance the balance of the account linked to the card
264         """
265         self.l4.setText('Card detected.<br />There is ' + `balance` + ' CHF left on your account.')
266         self.b1.setEnabled(True)
267         self.b2.setEnabled(True)
268         self.transaction.setEnabled(True)
269 
270         self.activateButton.emit()
271 
272         # 10 secondes timer before releasing the card if no action is make
273         self.releaseCardTimer.start(10000)
274 
275     @Slot(int)
276     def displayWarning(self, warning):
277         """! @brief Slot which displays a warning.
278         @param self the Frame instance
279         @param warning identifier of the warning to be displayed
280         """
281         if (warning == WARN_NO_ACCOUNT):
282             self.l4.setText('This card is not linked to any account.',
283                 '<br />Please create an account or link it to yours.')
284         elif (warning == WARN_ACCOUNT_INACTIVE):
285             self.l4.setText('Your account has been disable.<br />',
286                 'Please contact the administrator for further information.')
287         elif (warning == WARN_ACCOUNT_DELETED):
288             self.l4.setText('Your account has been deleted.<br />You can\'t use your card anymore.')
289         elif (warning == WARN_DEVICE_LOST):
290             self.l4.setText('This device has been lost.<br />Please send it to the administrator.')
291         elif (warning == WARN_DEVICE_STOLEN):
292             self.l4.setText('Stealing is bad !<br />Please send the device to the administrator.')
293         elif (warning == WARN_DEVICE_DELETED):
294             self.l4.setText('This device has been deleted from your account.<br />You can\'t use it anymore')
295         else:
296             self.l4.setText('An error has occurred')
297 
298         self.deactivateButton.emit()
299 
300         self.warningTimer.start(5000)
301 
302     @Slot(str)
303     def displayStatus(self, message):
304         """! @brief Slot which displays the status of an action.
305         @param self the Frame instance
306         @param message test to write on the status bar
307         """
308         self.status.setText(message)
309 
310     @Slot(str)
311     def displayTransactions(self, trs):
312         """! @brief Slot which displays the status of an action.
313         @param self the Frame instance
314         @param trs array with the last transactions
315         """
316         if self.transactionIsVisible == False:
317             self.status.setText('click again to hide transactions')
318             self.transactionTable.setRowCount(len(trs))
319 
320             for i in range(len(trs)):
321                 for j in range(len(trs[0])):
322                     item = QtGui.QTableWidgetItem(trs[i][j])
323                     if j == 1:
324                         item.setTextAlignment(Qt.AlignVCenter | Qt.AlignRight)
```

```
                    self.transactionTable.setItem(i, j, item)

            self.lTransaction.setVisible(True)
            self.transactionTable.setVisible(True)
            self.transactionIsVisible = True
        else:
            self.status.setText('')
            self.lTransaction.setVisible(False)
            self.transactionTable.setVisible(False)
            self.transactionIsVisible = False

    @Slot()
    def toggleAdminView(self):
        """! @brief Slot which displays of hides the administration view
        @param self the Frame instance
        """
        if self.adminActivated:
            self.adminActivated = False
            self.admin.setText('admin')
            self.displayMainWindow()
        else:
            self.adminActivated = True
            self.admin.setText('return')
            self.displayLogin()

    def displayMainWindow(self):
        """! @brief Method which displays the elements for the user view.
        @param self the Frame instance
        """
        self.l1.setVisible(True)
        self.l2.setVisible(True)
        self.l3.setVisible(True)
        self.logl1.setVisible(False)
        self.adminl1.setVisible(False)

        self.adminUsername.setVisible(False)
        self.adminPassword.setVisible(False)
        self.log.setVisible(False)

        self.lRecharge.setVisible(False)
        self.moneyBox.setVisible(False)
        self.bRecharge.setVisible(False)

        self.username.setVisible(False)
        self.name.setVisible(False)
        self.surname.setVisible(False)
        self.bCreateAccount.setVisible(False)

        self.lAddDevice.setVisible(False)
        self.username2.setVisible(False)
        self.bAddDevice.setVisible(False)

    def displayLogin(self):
        """! @brief Method which displays the elements for the login view.
        @param self the Frame instance
        """
        self.l1.setVisible(False)
        self.l2.setVisible(False)
        self.l3.setVisible(False)
        self.logl1.setVisible(True)
        self.adminl1.setVisible(False)

        self.adminUsername.setVisible(True)
        self.adminPassword.setVisible(True)
        self.log.setVisible(True)

        self.lRecharge.setVisible(False)
        self.bRecharge.setVisible(False)
        self.moneyBox.setVisible(False)

        self.username.setVisible(False)
        self.name.setVisible(False)
```

```
397             self.surname.setVisible(False)
398             self.bCreateAccount.setVisible(False)
399 
400             self.lAddDevice.setVisible(False)
401             self.username2.setVisible(False)
402             self.bAddDevice.setVisible(False)
403 
404         @Slot()
405         def displayAdmin(self, username, password):
406             """! @brief Slot which displays the elements for the admin.
407             @param self the Frame instance
408             @param username username to access the admin
409             @param password password to access the admin
410             """
411             if username == ADMIN_USERNAME and password == ADMIN_PASSWORD:
412                 self.l1.setVisible(False)
413                 self.l2.setVisible(False)
414                 self.l3.setVisible(False)
415                 self.logl1.setVisible(False)
416                 self.adminl1.setVisible(True)
417 
418                 self.adminUsername.setVisible(False)
419                 self.adminPassword.setVisible(False)
420                 self.log.setVisible(False)
421 
422                 self.lRecharge.setVisible(True)
423                 self.bRecharge.setVisible(True)
424                 self.moneyBox.setVisible(True)
425 
426                 self.username.setVisible(True)
427                 self.name.setVisible(True)
428                 self.surname.setVisible(True)
429                 self.bCreateAccount.setVisible(True)
430 
431                 self.lAddDevice.setVisible(True)
432                 self.username2.setVisible(True)
433                 self.bAddDevice.setVisible(True)
434 
435                 self.status.setVisible(True)
436 
437         def centerOnScreen(self):
438             """! @brief Method to center the application on the screen.
439             @param self the Frame instance
440             """
441             resolution = QtGui.QDesktopWidget().screenGeometry()
442             self.move((resolution.width() / 2) - (self.frameSize().width() / 2),
443                       (resolution.height() / 2) - (self.frameSize().height() / 2))
444 
445         def maximizeOnScreen(self):
446             """! @brief Method to maximize the application on the screen.
447             @param self the Frame instance
448             """
449             resolution = QtGui.QDesktopWidget().screenGeometry()
450             self.setGeometry(0, 0, resolution.width(), resolution.height())
451 
452         def fullScreen(self):
453             """! @brief Method to make the application full screen.
454             @param self the Frame instance
455             """
456             self.showFullScreen()
457 
458         def keyPressEvent(self, e):
459             """! @brief Called when a key is pressed. Exit the application when *ESC* is pressed.
460             @param self the Frame instance
461             @param e the event
462             """
463             if e.key() == Qt.Key_Escape:
464                 # close the button listener's threads
465                 self.deactivateButton.emit()
466                 # close the window
467                 self.close()
```

H.2.5 cardreader.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

## @file cardreader.py
#  Contains the classes DCCardType and CardReader.

## @package cardreader
#  Communication with a NFC Card Reader.
#  @author CASAS Jacky
#  @date 22.06.2014
#  @version 1.0

from smartcard.CardType import ATRCardType, CardType
from smartcard.CardRequest import CardRequest
from smartcard.util import toHexString, toBytes
from smartcard.CardConnection import CardConnection
from smartcard.CardConnectionObserver import ConsoleCardConnectionObserver, CardConnectionObserver
from smartcard.Exceptions import CardRequestTimeoutException
from smartcard.sw.ISO7816_4ErrorChecker import ISO7816_4ErrorChecker
from smartcard.sw.ISO7816_8ErrorChecker import ISO7816_8ErrorChecker
from smartcard.sw.ISO7816_9ErrorChecker import ISO7816_9ErrorChecker
from smartcard.sw.ErrorCheckingChain import ErrorCheckingChain
from smartcard.sw.SWExceptions import SWException
from smartcard.Card import Card

import sys
from string import replace
from PySide.QtCore import *
from constants import *
from action import Action
from ui import Frame

class DCCardType(CardType):
    """! @brief
    Direct Convention Card class.
    @author CASAS Jacky
    @date 22.06.14
    @version 1.0
    """

    def matches(self, atr, reader=None):
        """! @brief Method that verify if the ATR begins with 0x38.
        @param self the DCCardType instance
        @param atr ATR of the card
        @param reader identifier of the reader, optional
        """
        return atr[0] == 0x3B

class CardReader(QObject):
    """! @brief
    All actions from and to the NFC card reader are handled here.
    @author CASAS Jacky
    @date 22.06.14
    @version 1.0
    """
    ## Signal used to update the waiting of a new card
    updateWaiting = Signal()
    ## Signal used to display different warnings on the UI
    warning = Signal(int)
    ## Signal used to update UI when a card is detected
    cardDetected = Signal(int)

    def __init__(self, action):
        """! @brief Link an Action instance, create a cardrequest and a timer.
        @param self the CardReader instance
        @param action an instance of Action
        """
        QObject.__init__(self)
        ## DCCardType instance
        self.cardtype = DCCardType()
```

```
71          ## card request to make a connection with a smartcard
72          self.cardrequest = CardRequest(timeout=0.5, cardType=self.cardtype)
73
74          ## link to an Action instance
75          self.action = action
76          ## timer used to update the waiting state
77          self.timer = QTimer(self)
78          ## identifier of the card when detected
79          self.cardUid = None
80          ## ATR of the card when dtected
81          self.ATR = None
82
83      @Slot()
84      def start(self):
85          """! @brief Slot used to start the update timer with 0.5 seconds.
86          @param self the CardReader instance
87          """
88          self.timer.start(500)
89
90      @Slot()
91      def waitForCard(self):
92          """! @brief Slot called every 0.5 seconds. If a card is detected, the card is read.
93          @param self the CardReader instance
94          """
95          try:
96              cardService = self.cardrequest.waitforcard()
97              self.timer.stop()
98
99              cardService.connection.connect()
100
101             self.cardUid = self.getUID(cardService)
102             self.ATR = self.getATR(cardService)
103
104             account = self.action.getAccount(self.cardUid)
105
106             if account is None:
107                 self.warning.emit(WARN_NO_ACCOUNT)
108             else:
109                 if account['statement'] == STA_USER_ACTIVE:
110                     for device in account['devices']:
111                         if device['uid'] == self.cardUid:
112                             if device['status'] == STA_DEVICE_ACTIVE:
113                                 # all is ok
114                                 self.cardDetected.emit(account['balance'])
115                             elif device['status'] == STA_DEVICE_LOST:
116                                 self.warning.emit(WARN_DEVICE_LOST)
117                             elif device['status'] == STA_DEVICE_STOLEN:
118                                 self.warning.emit(WARN_DEVICE_STOLEN)
119                             elif device['status'] == STA_DEVICE_DELETED:
120                                 self.warning.emit(WARN_DEVICE_DELETED)
121                             else:
122                                 self.warning.emit(WARN_DEVICE_DELETED)
123                             break
124                 elif account['statement'] == STA_USER_INACTIVE:
125                     self.warning.emit(WARN_ACCOUNT_INACTIVE)
126                 elif account['statement'] == STA_USER_DELETED:
127                     self.warning.emit(WARN_ACCOUNT_DELETED)
128                 else:
129                     self.warning.emit(WARN_ACCOUNT_DELETED)
130
131         except CardRequestTimeoutException:
132             self.updateWaiting.emit()
133             # init variables
134             self.cardUid = None
135             self.ATR = None
136
137     @Slot()
138     def someBalls(self):
139         """! @brief Slot called when we click on the button to get some balls. Make a transaction.
140         @param self the CardReader instance
141         """
142         self.action.transaction(self.cardUid, 2)
```

```
143         self.start()
144 
145     @Slot()
146     def manyBalls(self):
147         """! @brief Slot called when we click on the button go get many balls. Make a transaction.
148         @param self the CardReader instance
149         """
150         self.action.transaction(self.cardUid, 5)
151         self.start()
152 
153     @Slot(int)
154     def recharge(self, amount):
155         """! @brief Slot called when we want to recharge the account.
156         @param self the CardReader instance
157         @param amount the amount paid to the account
158         """
159         self.action.recharge(self.cardUid, amount)
160         self.start()
161 
162     def myTransmit(self, connection, apdu):
163         """! @brief Method that overrides the standard transmit method.
164         It let us trace the command and response.
165         @param self the CardReader instance
166         @param connection the current connection with the card
167         @param apdu the APDU we want to transmit
168         """
169         # trace request :
170         #print 'sending : \t', toHexString(apdu)
171         response, sw1, sw2 = connection.transmit( apdu )
172         # trace response :
173         #if None == response: response=[]
174         #print 'response : \t', toHexString(response), '\nstatus words : \t', "%x %x" % (sw1, sw2)
175         if sw1 in range(0x61, 0x6f):
176             print "Error: sw1: %x sw2: %x" % (sw1, sw2)
177         return response, sw1, sw2
178 
179     def getUID(self, cardService):
180         """! @brief Method to get the UID of the smartcard or smartphone.
181         @param self the CardReader instance
182         @param cardService the current connection with the smartcard
183         """
184         try:
185             # try to get ID from smartphone
186             aidSize = [0x07]
187             ending = [0x00]
188             apdu = CLA_INS_P1_P2 + aidSize + AID_ANDROID + ending
189             response, sw1, sw2 = self.myTransmit(cardService.connection, apdu)
190             UID = toHexString(response)
191             #apdu = [0x00]
192             #response, sw1, sw2 = self.myTransmit(cardService.connection, apdu)
193             #print 'esasdf : ', toHexString(response)
194         except IndexError:
195             # otherwise get ID from smartcard
196             apdu = GET_UID
197             response, sw1, sw2 = self.myTransmit(cardService.connection, apdu)
198             UID = toHexString(response)
199         return UID
200 
201     def getATR(self, cardService):
202         """! @brief Method to get the ATR of the smartcard or smartphone.
203         @param self the CardReader instance
204         @param cardService the current connection with the smartcard
205         """
206         ATR = toHexString(cardService.connection.getATR())
207         return ATR
```

H.2.6 action.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

## @file action.py
#  Contains the class Action.

## @package action
#  Actions.
#  @author CASAS Jacky
#  @date 22.06.2014
#  @version 1.0

from uuid import uuid4
from datetime import datetime
from PySide.QtCore import *

from constants import *
from database import DataBase
from pifacecontrol import PiFaceControl

class Action(QObject):
    """! @brief
    Binding between the card reader and the database.
    @author CASAS Jacky
    @date 22.06.14
    @version 1.0
    """
    ## Signal used to display the action status on the UI
    status = Signal(str)
    ## Signal used to display the last transactions on the UI
    transactionsLoaded = Signal(list)

    def __init__(self, pifacecontrol):
        """! @brief Initialize the instance and get the two collections of the database.
        @param self the Action instance
        """
        QObject.__init__(self)
        ## contains a link to the database collection *user*
        self.users = DataBase().user
        ## contains a link to the database collection *transaction*
        self.transactions = DataBase().transaction

        self.piFace = pifacecontrol
        self.piFace.activateButtonListener()

    def transaction(self, deviceId, amount):
        """! @brief Method to make a transaction (withdraw an amount from an account).
        @param self the Action instance
        @param deviceId identifier of the device (smartcard or smartphone)
        @param amount amount to withdraw
        """
        canWithdraw = False
        user = self.users.find_one({"devices.uid": deviceId})
        for device in user['devices']:
            if device['uid'] == deviceId:
                if device['status'] == STA_DEVICE_ACTIVE:
                    if amount > 0:
                        if user['balance'] >= amount:
                            canWithdraw = True
                        else:
                            self.status.emit('Please recharge your account,',
                                ' you don\'t have enough money in it.')
                    else:
                        self.status.emit('The amount must be greater than 0.')
                elif device['status'] == STA_DEVICE_LOST:
                    self.status.emit('Impossible transaction : Lost device.')
                elif device['status'] == STA_DEVICE_STOLEN:
                    self.status.emit('Impossible transaction : Stolen device.')
                elif device['status'] == STA_DEVICE_DELETED:
                    self.status.emit('Impossible transaction : Deleted device.')
```

```
                else:
                    self.status.emit('Impossible transaction : Device status unknown.')

                if canWithdraw:
                    # create transaction
                    userId = user['uid']
                    dispenserId = DISPENSER_ID
                    transactionDate = datetime.now()

                    transaction = {"userId":userId,
                        "deviceId":deviceId,
                        "dispenserId":dispenserId,
                        "transactionType":WITHDRAWAL,
                        "amount":amount,
                        "transactionDate":transactionDate}
                    self.transactions.insert(transaction)

                    # debit credit in account
                    query = {"uid": user['uid'] }
                    update = {"$inc": { "balance": -amount}}
                    self.users.update(query, update)

                    self.status.emit('You withdraw ' + `amount` + ' CHF. You have now ' +
                        `user['balance'] - amount` + ' CHF on your account.')
                    self.piFace.actionValidated()
                else:
                    self.piFace.actionDenied()
                break

    def recharge(self, deviceId, amount):
        """! @brief Method to make a recharge (put an amount in an account).
        @param self the Action instance
        @param deviceId identifier of the device (smartcard or smartphone)
        @param amount amount to recharge
        """
        canRecharge = False
        user = self.users.find_one({"devices.uid": deviceId})
        for device in user['devices']:
            if device['uid'] == deviceId:
                if device['status'] == STA_DEVICE_ACTIVE:
                    if amount > 0:
                        canRecharge = True
                    else:
                        self.status.emit('The amount must be greater than 0.')
                elif device['status'] == STA_DEVICE_LOST:
                    self.status.emit('Impossible recharge : Lost device.')
                elif device['status'] == STA_DEVICE_STOLEN:
                    self.status.emit('Impossible recharge : Stolen device.')
                elif device['status'] == STA_DEVICE_DELETED:
                    self.status.emit('Impossible recharge : Deleted device.')
                else:
                    self.status.emit('Impossible recharge : Device status unknown.')

                if canRecharge:
                    # create transaction
                    userId = user['uid']
                    dispenserId = DISPENSER_ID
                    transactionDate = datetime.now()

                    transaction = {"userId":userId,
                        "deviceId":deviceId,
                        "dispenserId":dispenserId,
                        "transactionType":RECHARGE,
                        "amount":amount,
                        "transactionDate":transactionDate}
                    self.transactions.insert(transaction)

                    # recharge account
                    query = {"uid": user['uid'] }
                    update = {"$inc": { "balance": amount}}
                    self.users.update(query, update)

```

```
143                     self.status.emit('Recharge of ' + `amount` + ' CHF. You have now ' +
144                         `user['balance'] + amount` + ' CHF on your account.')
145                     self.piFace.actionValidated()
146                 else:
147                     self.piFace.actionDenied()
148                 break
149 
150     def getAccount(self, deviceId):
151         """! @brief Method to get an account in a JSON format.
152         @param self the Action instance
153         @param deviceId identifier of the device (smartcard or smartphone)
154         """
155         user = self.users.find_one({"devices.uid": deviceId})
156         if user is None:
157             return None
158         else:
159             return user
160 
161     def getLastTransactions(self, deviceId):
162         """! @brief Method to get the last 10 transactions.
163         @param self the Action instance
164         @param deviceId identifier of the device (smartcard or smartphone)
165         """
166         user = self.users.find_one({"devices.uid": deviceId})
167         transactions = self.transactions.find(
168                 {'userId': user['uid']},
169                 {'_id': 0,
170                     'amount': 1,
171                     'transactionType': 1,
172                     'transactionDate': 1,
173                     'deviceId': 1,
174                     'dispenserId': 1}
175             ).limit(10).sort("transactionDate", -1)
176         trs = []
177         for transaction in transactions:
178             tr = []
179             tr.append(transaction['transactionDate'].strftime("%a %e %b %Y, %H:%M:%S"))
180             if transaction['transactionType'] == RECHARGE:
181                 tr.append('+' + `transaction['amount']`)
182             elif transaction['transactionType'] == WITHDRAWAL:
183                 tr.append('-' + `transaction['amount']`)
184             tr.append('CHF')
185             trs.append(tr)
186         self.transactionsLoaded.emit(trs)
187 
188     def addUser(self, username, name=None, surname=None):
189         """! @brief Method to create a new user account.
190         @param self the Action instance
191         @param username unique username
192         @param name name of the user, optional
193         @param surname surname of the user, optional
194         """
195         if not username:
196             self.status.emit('Please enter at least a username.')
197         else:
198             userAlreadyRegistered = self.users.find_one({'username':username})
199             if userAlreadyRegistered is None:
200                 uid = str(uuid4())
201                 balance = 0
202                 registrationDate = datetime.now()
203                 statement = STA_USER_ACTIVE
204                 devices = []
205 
206                 if name is None:
207                     if surname is None:
208                         user = {"uid":uid,
209                             "username":username,
210                             "balance":balance,
211                             "registrationDate":registrationDate,
212                             "statement":statement,
213                             "devices":devices}
214                     else:
```

```
                        user = {"uid":uid,
                            "username":username,
                            "surname":surname,
                            "balance":balance,
                            "registrationDate":registrationDate,
                            "statement":statement,
                            "devices":devices}
                else:
                    if surname is None:
                        user = {"uid":uid,
                            "username":username,
                            "name":name,
                            "balance":balance,
                            "registrationDate":registrationDate,
                            "statement":statement,
                            "devices":devices}
                    else:
                        user = {"uid":uid,
                            "username":username,
                            "name":name,
                            "surname":surname,
                            "balance":balance,
                            "registrationDate":registrationDate,
                            "statement":statement,
                            "devices":devices}

                self.users.insert(user)
                self.status.emit('User successfully added to the database.')
                self.piFace.actionValidated()
            else:
                self.status.emit('User already registered.')

    def addDevice(self, username, deviceId, ATR):
        """! @brief Method to add a device to an account.
        @param self the Action instance
        @param username username of the account owner
        @param deviceId identifier of the device (smartcard or smartphone)
        @param ATR ATR of the card
        """
        if deviceId is None or ATR is None:
            self.status.emit('Please place a card in front of the reader.')
        else:
            if not username:
                self.status.emit('Please enter a username.')
            else:
                userWithDevice = self.users.find_one({"devices.uid": deviceId})
                if userWithDevice is None:
                    user = self.users.find_one({"username": username})

                    if user is None:
                        self.status.emit('This username doesn\'t exist.')
                    else:
                        status = STA_DEVICE_ACTIVE
                        activationDate = datetime.now()
                        category = 'smartcard'

                        # create the new device
                        newDevice = {"uid":deviceId,
                            "status":status,
                            "activationDate":activationDate,
                            "ATR":ATR,
                            "category":category}
                        # add the new device to the list
                        user['devices'].append(newDevice)
                        # remove the _id from user (required to make a $set)
                        del user['_id']

                        # update user
                        query = {"uid": user['uid'] }
                        update = { "$set": user }
                        self.users.update(query, update)

```

```
287                         self.status.emit('Device added to the user ' + user['name'] +
288                             ' ' + user['surname'] + '.')
289                         self.piFace.actionValidated()
290                 else:
291                     self.status.emit('This device already belongs to someone.')
```

H.2.7 database.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  ## @file database.py
5  #  Contains the classes SingletonType and DataBase.
6
7  ## @package database
8  #  Communication with databases.
9  #  @author CASAS Jacky
10 #  @date 22.06.2014
11 #  @version 1.0
12
13 import pymongo
14
15 class SingletonType(type):
16     """! @brief
17     Singleton type.
18     @author CASAS Jacky
19     @date 22.06.14
20     @version 1.0
21     """
22     def __call__(cls, *args, **kwargs):
23         """! @brief Override of the `__call__` method."""
24         try:
25             return cls.__instance
26         except AttributeError:
27             cls.__instance = super(SingletonType, cls).__call__(*args, **kwargs)
28             return cls.__instance
29
30 class DataBase(object):
31     """! @brief
32     Modelisation of the database. This class is a singleton.
33     @author CASAS Jacky
34     @date 22.06.14
35     @version 1.0
36     """
37     __metaclass__ = SingletonType
38
39     def __init__(self):
40         """! @brief
41         The connection to the MongoDB database is made here.
42         """
43         try:
44             # local database
45             client = pymongo.MongoClient('mongodb://localhost:27017/')
46
47             # remote database (through SSH tunnel)
48             #client = pymongo.MongoClient('mongodb://localhost:28082/')
49
50             db = client['golfBallDispenserDatas']
51
52             ## variable containing the collection *user*
53             self.user = db['user']
54             ## variable containing the collection *transaction*
55             self.transaction = db['transaction']
56
57         except pymongo.errors.ConnectionFailure, e:
58            print "Could not connect to MongoDB: %s" % e
```

H.2.8 pifacecontrol.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

## @file pifacecontrol.py
#  Contains the classe PiFaceControl.

## @package pifacecontrol
#  Control of the PiFace Digital module.
#  @author CASAS Jacky
#  @date 22.06.2014
#  @version 1.0

from time import sleep
import imp # to check if a module exists
from PySide.QtCore import *

DELAY = 0.2 #seconds
LONG_DELAY = 0.7

class PiFaceControl(QObject):
    """! @brief
    Control of the PiFace Digital module.
    @author CASAS Jacky
    @date 22.06.14
    @version 1.0
    """
    ## Signal from the button 1
    b1 = Signal()
    ## Signal from the button 2
    b2 = Signal()
    ## Signal from the button 3
    b3 = Signal()
    ## Signal from the button 4
    b4 = Signal()

    def __init__(self):
        """! @brief Initialize the instance if the module is connected.
        @param self the PiFaceControl instance
        """
        QObject.__init__(self)
        try:
            imp.find_module('pifacedigitalio')
            ## tell if the module PiFaceDigital is connected or not
            self.moduleFound = True
            import pifacedigitalio
            ## exportation of the library in the class
            self.pifacedigitalio = pifacedigitalio
            ## instance of PiFaceDigital, access to the leds
            self.pfd = pifacedigitalio.PiFaceDigital()
            for i in range(2, 8):
                self.pfd.leds[i].turn_off()
            ## constant of the library for the falling edge detection
            self.fallingEdge = pifacedigitalio.IODIR_FALLING_EDGE
            ## tell if the button listener is activated or not
            self.listenerActivated = False
        except ImportError:
            print 'The module pifacedigitalio is not installed'
            self.moduleFound = False

    def button1Pressed(self, event):
        """! @brief Method called when button 1 is pressed.
        @param self the PiFaceControl instance
        @param event the event that trigger that method call
        """
        self.b1.emit()

    def button2Pressed(self, event):
        """! @brief Method called when button 2 is pressed.
        @param self the PiFaceControl instance
        @param event the event that trigger that method call
```

```
71          """
72          self.b2.emit()
73
74      def button3Pressed(self, event):
75          """! @brief Method called when button 3 is pressed.
76          @param self the PiFaceControl instance
77          @param event the event that trigger that method call
78          """
79          self.b3.emit()
80
81      def button4Pressed(self, event):
82          """! @brief Method called when button 4 is pressed.
83          @param self the PiFaceControl instance
84          @param event the event that trigger that method call
85          """
86          self.b4.emit()
87
88      def actionValidated(self):
89          """! @brief Method called when an action is validated. It makes a moving line of LEDs.
90          @param self the PiFaceControl instance
91          """
92          if self.moduleFound:
93              self.pfd.leds[7].turn_on()
94              sleep(DELAY)
95              self.pfd.leds[6].turn_on()
96              sleep(DELAY)
97              self.pfd.leds[5].turn_on()
98              self.pfd.leds[7].turn_off()
99              sleep(DELAY)
100             self.pfd.leds[4].turn_on()
101             self.pfd.leds[6].turn_off()
102             sleep(DELAY)
103             self.pfd.leds[3].turn_on()
104             self.pfd.leds[5].turn_off()
105             sleep(DELAY)
106             self.pfd.leds[2].turn_on()
107             self.pfd.leds[4].turn_off()
108             sleep(DELAY)
109             self.pfd.leds[3].turn_off()
110             sleep(DELAY)
111             self.pfd.leds[2].turn_off()
112             sleep(DELAY)
113             self.pfd.leds[2].turn_off()
114
115     def actionDenied(self):
116         """! @brief Method called when an action is denied. It makes LED blinks.
117         @param self the PiFaceControl instance
118         """
119         if self.moduleFound:
120             for i in range(2, 8):
121                 self.pfd.leds[i].turn_on()
122             sleep(DELAY)
123             for i in range(2, 8):
124                 self.pfd.leds[i].turn_off()
125             sleep(DELAY)
126             for i in range(2, 8):
127                 self.pfd.leds[i].turn_on()
128             sleep(DELAY)
129             for i in range(2, 8):
130                 self.pfd.leds[i].turn_off()
131
132             sleep(LONG_DELAY)
133
134             for i in range(2, 8):
135                 self.pfd.leds[i].turn_on()
136             sleep(DELAY)
137             for i in range(2, 8):
138                 self.pfd.leds[i].turn_off()
139             sleep(DELAY)
140             for i in range(2, 8):
141                 self.pfd.leds[i].turn_on()
142             sleep(DELAY)
```

```
143             for i in range(2, 8):
144                 self.pfd.leds[i].turn_off()
145 
146     @Slot()
147     def activateButtonListener(self):
148         """! @brief Slot called to activate the buttons listener.
149         @param self the PiFaceControl instance
150         """
151         if self.moduleFound:
152             if self.listenerActivated == False:
153                 # create new instances of the listener
154                 ## listener fot the button 1
155                 self.listener1 = self.pifacedigitalio.InputEventListener(chip=self.pfd)
156                 ## listener fot the button 2
157                 self.listener2 = self.pifacedigitalio.InputEventListener(chip=self.pfd)
158                 ## listener fot the button 3
159                 self.listener3 = self.pifacedigitalio.InputEventListener(chip=self.pfd)
160                 ## listener fot the button 4
161                 self.listener4 = self.pifacedigitalio.InputEventListener(chip=self.pfd)
162                 # registration of the button to the listener
163                 self.listener1.register(0, self.fallingEdge, self.button1Pressed)
164                 self.listener2.register(1, self.fallingEdge, self.button2Pressed)
165                 self.listener3.register(2, self.fallingEdge, self.button3Pressed)
166                 self.listener4.register(3, self.fallingEdge, self.button4Pressed)
167                 # activation of the listening
168                 self.listener1.activate()
169                 self.listener2.activate()
170                 self.listener3.activate()
171                 self.listener4.activate()
172                 self.listenerActivated = True
173 
174     @Slot()
175     def deactivateButtonListener(self):
176         """! @brief Slot called to activate the buttons listener.
177         @param self the PiFaceControl instance
178         """
179         if self.moduleFound:
180             if self.listenerActivated == True:
181                 # deactivation of the listening, kill the threads
182                 self.listener1.deactivate()
183                 self.listener2.deactivate()
184                 self.listener3.deactivate()
185                 self.listener4.deactivate()
186                 self.listenerActivated = False
```

H.3 API EN PYTHON POUR LE SERVEUR

H.3.1 API.py

```
1  #!/usr/bin/env python
2  
3  import os
4  # Flask imports
5  from flask import Flask, abort, request, jsonify, g, url_for
6  from flask.ext.httpauth import HTTPBasicAuth
7  from passlib.apps import custom_app_context as pwd_context
8  from itsdangerous import (TimedJSONWebSignatureSerializer
9                            as Serializer, BadSignature, SignatureExpired)
10 from flask.ext.pymongo import PyMongo
11 
12 # useful imports
13 import json
14 import random
15 from datetime import datetime
16 from uuid import uuid4
17 
18 # imports from files
```

```
19 import lib
20 import constants
21
22 # initialization of the Flask app
23 app = Flask(__name__)
24 app.config['SECRET_KEY'] = 'This is my secret key, or passphrase can we say, long enough? I think so.'
25 app.config['MONGO_PORT'] = 27017
26 app.config['MONGO_DBNAME'] = 'golfBallDispenserDatas'
27
28 # creation of a mongoDB container for Flask
29 mongo = PyMongo(app)
30
31 # creation of a authentication system for Flask
32 auth = HTTPBasicAuth()
33
34 class User:
35     def __init__(self, username=None, uid=None):
36         if username is None:
37             userFromDB = mongo.db.user.find_one({'uid':uid})
38             if userFromDB is None:
39                 self.uid = None
40                 self.username = None
41             else:
42                 self.uid = uid
43                 self.username = userFromDB.get('username')
44         else:
45             userFromDB = mongo.db.user.find_one({'username':username})
46             if userFromDB is None:
47                 self.username = None
48                 self.uid = None
49             else:
50                 self.username = username
51                 self.uid = userFromDB.get('uid')
52         if userFromDB is not None:
53             self.password_hash = userFromDB.get('password')
54             self.balance = userFromDB.get('balance')
55             self.name = userFromDB.get('name')
56             self.surname = userFromDB.get('surname')
57             self.statement = userFromDB.get('statement')
58             self.devices = userFromDB.get('devices')
59
60     def hash_password(self, password):
61         self.password_hash = pwd_context.encrypt(password)
62
63     def verify_password(self, password):
64         return pwd_context.verify(password, self.password_hash)
65
66     def generate_auth_token(self, expiration=600):
67         s = Serializer(app.config['SECRET_KEY'], expires_in=expiration)
68         return s.dumps({'id': self.uid})
69
70     def get_device_id(self, android_id):
71         userFromDB = mongo.db.user.find_one({'username':self.username, 'devices.androidId':android_id})
72         if userFromDB is None:
73             return self.create_device(android_id)
74         else:
75             for device in userFromDB.get('devices'):
76                 if device.get('androidId') == android_id:
77                     return device.get('uid')
78         return None
79
80     def create_device(self, android_id):
81         deviceId = self.generate_device_id()
82         activation_date = datetime.now()
83         category = 'smartphone'
84         status = constants.STA_DEVICE_ACTIVE
85
86         user = mongo.db.user.find_one({"username": self.username})
87
88         # create the new device
89         newDevice = {"uid":deviceId,
90             "status":status,
```

```
            "activationDate":activation_date,
            "androidId":android_id,
            "category":category}
        # add the new device to the list
        user['devices'].append(newDevice)
        # remove the _id from user (required to make a $set)
        del user['_id']

        # update user
        query = {"uid": user['uid'] }
        update = { "$set": user }
        mongo.db.user.update(query, update)

        return deviceId

    def generate_device_id(self):
        uid = []
        uid.append(random.randint(0,255))
        uid.append(random.randint(0,255))
        uid.append(random.randint(0,255))
        uid.append(random.randint(0,255))
        return lib.toHexString(uid)

    @staticmethod
    def verify_auth_token(token):
        s = Serializer(app.config['SECRET_KEY'])
        try:
            data = s.loads(token)
        except SignatureExpired:
            return None    # valid token, but expired
        except BadSignature:
            return None    # invalid token
        user = User(uid=data['id'])
        return user

    def get_user_info(self):
        user = mongo.db.user.find_one(
                {"uid":self.uid},
                {'_id': 0,
                    'username': 1,
                    'name': 1,
                    'surname': 1,
                    'balance': 1,
                    'statement' : 1,
                    'registrationDate': 1}
            )
        return user

    def get_last_transactions(self, quantity):
        transactions = mongo.db.transaction.find(
                {'userId':self.uid},
                {'_id': 0,
                    'amount': 1,
                    'transactionType': 1,
                    'transactionDate': 1,
                    'deviceId': 1,
                    'dispenserId': 1}
            ).limit(quantity).sort("transactionDate", -1)
        trs = []
        for transaction in transactions:
            trs.append(transaction)
        return trs

@auth.verify_password
def verify_password(username_or_token, password):
    # first try to authenticate by token
    user = User.verify_auth_token(username_or_token)
    if user is None:
        # try to authenticate with username/password
        user = User(username=username_or_token)
        if user.username is None or not user.verify_password(password):
            return False
```

```
163     g.user = user
164     return True
165 
166 @app.route('/')
167 def home_page():
168     return 'Welcome on the NFC GOLF BALL DISPENSER APIs<br />The APIs are accessible via vlenfc.hevs.ch/api/'
169 
170 @app.route('/api/newpassword', methods=['GET'])
171 def set_new_password():
172     username = request.args.get('username', '')
173     password = request.args.get('password', '')
174     password_hash = pwd_context.encrypt(password)
175 
176     query = {"username" : username}
177     newPwd = {"password" : password_hash}
178     update = {"$set" : newPwd}
179     mongo.db.user.update(query, update)
180     return "password registered"
181 
182 @app.route('/api/newaccount', methods=['POST'])
183 def set_new_account():
184     username = request.json.get('username', '')
185     name = request.json.get('name', '')
186     surname = request.json.get('surname', '')
187     password = request.json.get('password', '')
188     password_hash = pwd_context.encrypt(password)
189 
190     # check if the username already exists
191     userFromDB = mongo.db.user.find_one({'username':username})
192     if userFromDB is None:
193         uid = str(uuid4())
194         balance = 10
195         registrationDate = datetime.now()
196         statement = constants.STA_USER_ACTIVE
197         devices = []
198 
199         user = {"uid":uid,
200             "username":username,
201             "password":password_hash,
202             "balance":balance,
203             "registrationDate":registrationDate,
204             "statement":statement,
205             "devices":devices}
206 
207         if name:
208             user['name'] = name
209         if surname:
210             user['surname'] = surname
211 
212         # creation of the user
213         mongo.db.user.insert(user)
214 
215         # Success code 201 = CREATED
216         return "Account sucessfully created", 201
217     else:
218         # Fail code 403 = FORBIDDEN
219         abort(403)
220 
221 @app.route('/api/token')
222 @auth.login_required
223 def get_auth_token():
224     # ex : curl -u username:password -i -X GET http://localhost:5000/api/token
225     # ou : curl -u token:dontcare -i -X GET http://localhost:5000/api/token
226     token = g.user.generate_auth_token(600)
227     return jsonify({'token': token.decode('ascii'), 'duration': 600})
228 
229 @app.route('/api/deviceid/<androidid>')
230 @auth.login_required
231 def get_device_id(androidid):
232     device_id = g.user.get_device_id(androidid)
233     return jsonify({'uid': '%s' % device_id})
234 
```

```
@app.route('/api/balance')
@auth.login_required
def get_balance():
    return jsonify({'balance': '%s' % g.user.balance})

@app.route('/api/devices')
@auth.login_required
def get_devices():
    return jsonify({'devices' : '%s' % g.user.devices})

@app.route('/api/user')
@auth.login_required
def get_user():
    return jsonify({'user': g.user.get_user_info()})

@app.route('/api/transactions')
@auth.login_required
def get_transactions():
    return jsonify({'transactions' : g.user.get_last_transactions(10)})

if __name__ == '__main__':
    #app.run(debug=True, host='127.0.0.1')       # to run locally on port 5000
    app.run(debug=True, host='0.0.0.0', port=80) # to run on a server
```

H.3.2 constants.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# transaction type
RECHARGE = 0
WITHDRAWAL = 1

# user statement
STA_USER_ACTIVE = 1
STA_USER_INACTIVE = 2
STA_USER_DELETED = 3

# device status
STA_DEVICE_ACTIVE = 1
STA_DEVICE_LOST = 2
STA_DEVICE_STOLEN = 3
STA_DEVICE_DELETED = 4

# default smartphone id
DEFAULT_SMARTPHONE_ID = 'AA AA AA AA'
```

H.4 Application Android en Java

H.4.1 MainActivity.java

```
package com.golfwallet.main;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutionException;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
```

```java
import android.nfc.NfcAdapter;
import android.nfc.NfcAdapter.ReaderCallback;
import android.nfc.Tag;
import android.nfc.tech.IsoDep;
import android.os.Bundle;
import android.provider.Settings.Secure;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.golfwallet.R;
import com.golfwallet.connectivity.APITest;
import com.golfwallet.connectivity.HTTPRequest;
import com.golfwallet.identification.LoginActivity;
import com.golfwallet.identification.SigninActivity;
import com.golfwallet.nfc.IsoDepAdapter;
import com.golfwallet.nfc.IsoDepTransceiver;
import com.golfwallet.nfc.IsoDepTransceiver.OnMessageReceived;
import com.golfwallet.preference.ReadWriteData;

/** \mainpage
* \section intro Introduction
* Android application for the NFC Golf Ball Dispenser.
* You can :
* * Create an account
* * Log into your account
* * Watch your last transactions
* * Know your balance and account status
* * Make a transaction using NFC
* \section overview Overview
* \subsection step1 Home view with the app installed
* ![Home view](../img/aHome.png)
* \subsection step2 In-app views
* ![Home view](../img/aViews.png)
* \section classdiagram Class Diagram
* ![Home view](../img/classDiagram.png)
* \section license License
* This software is developped by Jacky CASAS and was commissioned by Digiclever.
* For further information about the license, please contact Digiclever (http://digiclever.com).<br /><br />
* This project was developped at HES-SO//Valais during the bachelor thesis of Jacky Casas in 2014.
*/

/** \brief
 * <b>Main activity of the app.</b>
 * <p>
 * When the user is not logged, there is two button :
 * <ul>
 * <li>Login</li>
 * <li>Signin</li>
 * </ul>
 * </p>
 * <p>
 * And when connected, different information are shown :
 * <ul>
 * <li>User name</li>
 * <li>Balance of the user account</li>
 * <li>Account status</li>
 * <li>Date of registration</li>
 * <li>List of the last transations</li>
 * </ul>
 * </p>
 *  \author Jacky CASAS
 *  \version 1.0
 *  \date 04.06.2014
 */
public class MainActivity extends Activity implements OnMessageReceived,
        ReaderCallback {
```

```
86
87      private NfcAdapter nfcAdapter;          ///< NfcAdapter fot the NFC support
88      private IsoDepAdapter isoDepAdapter;    ///< IsoDepAdapter for the NFC support
89      private static Context context = null;  ///< Context of the application
90
91      private TextView titleTextView;         ///< TextView to display the name of the user
92      private TextView balanceTextView;       ///< TextView to display the account balance
93      private TextView statusTextView;        ///< TextView to display the account status
94      private TextView dateTextView;          ///< TextView to display the date of account creation
95      private ListView transactionListView;   ///< ListView to display last transactions
96
97      private Button loginButton;             ///< Button to access login activity
98      private Button signinButton;            ///< Button to access signin activity
99
100     /**
101      *  \brief Method to get the context of the application
102      *  \return The context of the application
103      */
104     public static Context getAppContext() {
105         return MainActivity.context;
106     }
107
108     /**
109      *  \brief Change the connected status of a user in the application
110      *  \param value True for connected and False to unconnected
111      */
112     public void setConnected(boolean value) {
113         boolean connected = ReadWriteData.getConnectionState(context);
114
115         if (!connected) {
116             // connection
117             ReadWriteData.setConnected(context);
118             Toast.makeText(MainActivity.this, "Connection",Toast.LENGTH_SHORT).show();
119
120             Intent mainIntent = new Intent(MainActivity.this, MainActivity.class);
121             startActivity(mainIntent);
122             finish();
123         } else {
124             // disconnection
125             ReadWriteData.disconnect(context);
126             ReadWriteData.clearAll(context);
127             Intent mainIntent = new Intent(MainActivity.this, MainActivity.class);
128             startActivity(mainIntent);
129             finish();
130         }
131     }
132
133     /**
134      *  \brief
135      *  Method called when the activity is created
136      *  \param savedInstanceState Bundle used to pass information from previous activity to this one
137      */
138     @Override
139     protected void onCreate(Bundle savedInstanceState) {
140         super.onCreate(savedInstanceState);
141
142         MainActivity.context = getApplicationContext();
143
144         isoDepAdapter = new IsoDepAdapter(getLayoutInflater());
145         nfcAdapter = NfcAdapter.getDefaultAdapter(this);
146
147         String username = ReadWriteData.read(context, ReadWriteData.LOGIN_USERNAME);
148         String password = ReadWriteData.read(context, ReadWriteData.LOGIN_PASSWORD);
149
150         if (ReadWriteData.getConnectionState(context) && username != null && password != null) {
151
152             setContentView(R.layout.fragment_main);
153             titleTextView = (TextView)findViewById(R.id.title);
154             balanceTextView = (TextView)findViewById(R.id.balance);
155             statusTextView = (TextView)findViewById(R.id.accountStatus);
156             dateTextView = (TextView)findViewById(R.id.registrationDate);
157             transactionListView = (ListView)findViewById(R.id.transactionList);
```

```
158
159             // GET NEW TOKEN
160             String token = null;
161             try {
162                 token = new HTTPRequest().execute("/api/token", username, password).get();
163
164                 if (token != null) {
165                     ReadWriteData.write(context, ReadWriteData.LOGIN_TOKEN, token);
166                     saveAID();
167                 } else {
168                     Toast.makeText(MainActivity.this, "Connection issues, retry to connect.",
169                         Toast.LENGTH_SHORT).show();
170                     setConnected(false);
171                 }
172
173             } catch (InterruptedException | ExecutionException e) {
174                 e.printStackTrace();
175                 Toast.makeText(MainActivity.this, "Connection issues, retry to connect.",
176                     Toast.LENGTH_SHORT).show();
177                 setConnected(false);
178             }
179
180             // GET ACTIVITY CONTENT FROM API : USER INFORMATIONS
181             String result = "error";
182             try {
183                 if (token == null) {
184                     result = new HTTPRequest().execute("/api/user", username, password).get();
185                 } else {
186                     result = new HTTPRequest().execute("/api/user", token, "").get();
187                 }
188
189                 try {
190                     JSONObject jsonObj = new JSONObject(result);
191                     JSONObject user = (JSONObject) jsonObj.get("user");
192
193                     if (user.has("name") && user.has("surname")) {
194                         titleTextView.setText("Hi, " + user.getString("name") + " " +
195                             user.getString("surname") + " !");
196                     } else {
197                         titleTextView.setText("Hi, " + user.getString("username") + " !");
198                     }
199
200                     balanceTextView.setText("balance : " + user.getInt("balance") + " CHF");
201
202                     switch(user.getInt("statement")) {
203                         case Constants.STA_USER_ACTIVE:
204                             statusTextView.setText("account status : active");
205                             break;
206                         case Constants.STA_USER_DELETED:
207                             statusTextView.setText("account status : deleted");
208                             break;
209                         case Constants.STA_USER_INACTIVE:
210                             statusTextView.setText("account status : inactive");
211                             break;
212                     }
213                     dateTextView.setText("registration date : " +
214                         user.getString("registrationDate").replace(" GMT",""));
215                 } catch (JSONException e) {
216                     e.printStackTrace();
217                 }
218             } catch (InterruptedException | ExecutionException e) {
219                 e.printStackTrace();
220             }
221
222             // GET ACTIVITY CONTENT FROM API : TRANSACTIONS FROM USER
223             result = "error";
224             try {
225                 if (token == null) {
226                     result = new HTTPRequest().execute("/api/transactions", username, password).get();
227                 } else {
228                     result = new HTTPRequest().execute("/api/transactions", token, "").get();
229                 }
```

```
230
231             List<Transaction> transactionList = new ArrayList<Transaction>();
232             try {
233                 JSONObject jsonObj = new JSONObject(result);
234                 JSONArray json = (JSONArray) jsonObj.get("transactions");
235
236                 Toast.makeText(MainActivity.getAppContext(), "Your " + json.length() +
237                     " last transactions", Toast.LENGTH_LONG).show();
238
239                 for (int i = 0; i < json.length(); i++) {
240                     JSONObject c = json.getJSONObject(i);
241                     int amount = c.getInt("amount");
242                     int type = c.getInt("transactionType");
243                     String transactionDate = c.getString("transactionDate");
244                     String deviceId = c.getString("deviceId");
245                     String dispenserId = c.getString("dispenserId");
246
247                     transactionList.add(new Transaction(amount, type,
248                         transactionDate, deviceId, dispenserId));
249                 }
250             } catch (JSONException e) {
251                 e.printStackTrace();
252             }
253
254             TransactionAdapter transactionAdapter = new TransactionAdapter(this,
255                 R.layout.transaction_row, transactionList);
256
257             transactionListView.setAdapter(transactionAdapter);
258
259         } catch (InterruptedException | ExecutionException e) {
260             e.printStackTrace();
261         }
262     } else {
263         setContentView(R.layout.log_layout);
264         loginButton = (Button)findViewById(R.id.loginbutton);
265         signinButton = (Button)findViewById(R.id.signinbutton);
266
267         loginButton.setOnClickListener(new OnClickListener() {
268             @Override
269             public void onClick(View v) {
270                 Intent loginIntent = new Intent(MainActivity.this, LoginActivity.class);
271                 startActivity(loginIntent);
272                 finish();
273             }
274         });
275
276         signinButton.setOnClickListener(new OnClickListener() {
277             @Override
278             public void onClick(View v) {
279                 Intent signinIntent = new Intent(MainActivity.this, SigninActivity.class);
280                 startActivity(signinIntent);
281                 finish();
282             }
283         });
284     }
285 }
286
287 /**
288  *  \brief
289  *  Method which create the option menu of the top right of the app.
290  *  \param menu Menu that contains the different items to be displayed
291  *  \return True if the menu is correctly created
292  */
293 @Override
294 public boolean onCreateOptionsMenu(Menu menu) {
295     getMenuInflater().inflate(R.menu.main, menu);
296
297     if (ReadWriteData.getConnectionState(context)) {
298         menu.findItem(R.id.action_logout).setTitle("Logout");
299     } else {
300         menu.findItem(R.id.action_logout).setTitle("Login");
301     }
```

```
302         return true;
303     }
304 
305     /**
306      *  \brief
307      *  Method called when the user click on a menu item.
308      *  \param item An item of the menu
309      *  \return Say if the action of selection is made correctly or not
310      */
311     @Override
312     public boolean onOptionsItemSelected(MenuItem item) {
313         int id = item.getItemId();
314         if (id == R.id.action_settings) {
315             Toast.makeText(MainActivity.this, "Settings clicked",
316                     Toast.LENGTH_SHORT).show();
317             Intent apiTestIntent = new Intent(MainActivity.this, APITest.class);
318             MainActivity.this.startActivity(apiTestIntent);
319             return true;
320         } else if (id == R.id.action_logout) {
321             if (ReadWriteData.getConnectionState(context)) {
322                 Toast.makeText(MainActivity.this, "Bye bye !", Toast.LENGTH_SHORT).show();
323 
324                 item.setTitle("Login");
325                 setConnected(false);
326             } else {
327                 item.setTitle("Logout");
328                 Intent loginIntent = new Intent(MainActivity.this, LoginActivity.class);
329                 startActivity(loginIntent);
330                 finish();
331             }
332             return true;
333         }
334         return super.onOptionsItemSelected(item);
335     }
336 
337     /**
338      *  \brief
339      *  Method called when the activity is brought to the front.
340      */
341     @Override
342     public void onResume() {
343         super.onResume();
344         nfcAdapter.enableReaderMode(this, this, NfcAdapter.FLAG_READER_NFC_A
345                 | NfcAdapter.FLAG_READER_SKIP_NDEF_CHECK, null);
346     }
347 
348     /**
349      *  \brief
350      *  Method called just before the activity goes on the background.
351      */
352     @Override
353     public void onPause() {
354         super.onPause();
355         nfcAdapter.disableReaderMode(this);
356     }
357 
358     /**
359      *  \brief
360      *  Method called when the service detect a NFC card.
361      *  \param tag Tag that the service has detected
362      */
363     @Override
364     public void onTagDiscovered(Tag tag) {
365         IsoDep isoDep = IsoDep.get(tag);
366         IsoDepTransceiver transceiver = new IsoDepTransceiver(isoDep, this);
367         Thread thread = new Thread(transceiver);
368         thread.start();
369     }
370 
371     /**
372      *  \brief
373      *  Method called when we receive a message from the tag
```

```
374         *  \param message Bytes array containing the message
375         */
376        @Override
377        public void onMessage(final byte[] message) {
378            runOnUiThread(new Runnable() {
379                @Override
380                public void run() {
381                    isoDepAdapter.addMessage(new String(message));
382                }
383            });
384        }
385
386        /**
387         *  \brief
388         *  Method called when an error occured when chatting with the card.
389         *  \param exception Exception
390         */
391        @Override
392        public void onError(Exception exception) {
393            onMessage(exception.getMessage().getBytes());
394        }
395
396        /**
397         *  \brief
398         *  Method that fetch the AID (through API) and store it in the SharedPreferences.
399         */
400        private void saveAID() {
401            // AID = Application ID = unique 4 bytes identifier for each Android devices of an account
402            // the AID is generated in the API
403
404            String username;
405            String password;
406            String token = ReadWriteData.read(context, ReadWriteData.LOGIN_TOKEN);
407            String result = null;
408            String androidId = Secure.getString(MainActivity.getAppContext().getContentResolver(),
409                Secure.ANDROID_ID);
410
411            try {
412                if (token == null) {
413                    username = ReadWriteData.read(context ,ReadWriteData.LOGIN_USERNAME);
414                    password = ReadWriteData.read(context, ReadWriteData.LOGIN_PASSWORD);
415                    result = new HTTPRequest().execute("/api/deviceid", username, password, "/" +
416                        androidId).get();
417                } else {
418                    result = new HTTPRequest().execute("/api/deviceid", token, "", "/" + androidId).get();
419                }
420                ReadWriteData.write(context, ReadWriteData.NFC_AID, result);
421            } catch (InterruptedException | ExecutionException e) {
422                e.printStackTrace();
423                setConnected(false);
424                Toast.makeText(MainActivity.getAppContext(), "An error occured, please try to connect again.",
425                    Toast.LENGTH_SHORT).show();
426            }
427        }
428
429        /**
430         *  \brief
431         *  Method that returns the AID. If the user is connected, the unique AID is returned
432         *  but if not, the default AID is returned.
433         *  \return A byte array that contains the AID
434         */
435        public static byte[] getAID() {
436            String AID = ReadWriteData.read(context, ReadWriteData.NFC_AID);
437            Toast.makeText(MainActivity.getAppContext(), "aid : " + AID, Toast.LENGTH_SHORT).show();
438
439            if (ReadWriteData.getConnectionState(context) && AID != null) {
440                String[] separatedAID = AID.split(" ");
441                byte[] aid = new byte[separatedAID.length];
442
443                for (int i = 0; i < separatedAID.length; i++) {
444                    aid[i] = (byte) (Integer.parseInt(separatedAID[i],16) & 0xff);
445                }
```

```
446
447             return aid;
448         } else {
449             return Constants.DEF_ANDROID_ID;
450         }
451     }
452 }
```

H.4.2 SigninActivity.java

```
1  package com.golfwallet.identification;
2
3  import java.io.ByteArrayOutputStream;
4  import java.io.IOException;
5
6  import org.apache.http.HttpResponse;
7  import org.apache.http.HttpStatus;
8  import org.apache.http.StatusLine;
9  import org.apache.http.client.HttpClient;
10 import org.apache.http.client.methods.HttpPost;
11 import org.apache.http.entity.StringEntity;
12 import org.apache.http.impl.client.DefaultHttpClient;
13 import org.json.JSONException;
14 import org.json.JSONObject;
15
16 import android.animation.Animator;
17 import android.animation.AnimatorListenerAdapter;
18 import android.annotation.TargetApi;
19 import android.app.Activity;
20 import android.content.Intent;
21 import android.os.AsyncTask;
22 import android.os.Build;
23 import android.os.Bundle;
24 import android.text.TextUtils;
25 import android.view.KeyEvent;
26 import android.view.Menu;
27 import android.view.View;
28 import android.view.inputmethod.EditorInfo;
29 import android.widget.EditText;
30 import android.widget.TextView;
31 import android.widget.Toast;
32
33 import com.golfwallet.R;
34 import com.golfwallet.main.MainActivity;
35
36 /** \brief Activity which displays a registration screen to the user
37  *  \author Jacky CASAS
38  *  \version 1.0
39  *  \date 04.06.2014
40  */
41 public class SigninActivity extends Activity {
42
43     private UserSigninTask mAuthTask = null;     ///< Keep track of the login task to ensure
44                                                  ///< we can cancel it if requested.
45
46     // Values for email and password at the time of the login attempt.
47     private String mUsername;                    ///< Value of the username
48     private String mName;                        ///< Value of the name
49     private String mSurname;                     ///< Value of the surname
50     private String mPassword;                    ///< Value of the password
51
52     // UI references.
53     private EditText mUsernameView;              ///< Username EditText
54     private EditText mNameView;                  ///< Name EditText
55     private EditText mSurnameView;               ///< Surname EditText
56     private EditText mPasswordView;              ///< Password EditText
57     private View mLoginFormView;                 ///< Login form View
58     private View mLoginStatusView;               ///< Login status View
59     private TextView mLoginStatusMessageView;    ///< TextView for the status messages
```

```
60
61      /**
62       *  \brief Method called when the activity is created
63       *  \param savedInstanceState Bundle of the last activity
64       */
65      @Override
66      protected void onCreate(Bundle savedInstanceState) {
67          super.onCreate(savedInstanceState);
68
69          setContentView(R.layout.activity_signin);
70
71          // Set up the signin form.
72          mUsername = getIntent().getStringExtra("username");
73          mUsernameView = (EditText) findViewById(R.id.username);
74          mUsernameView.setText(mUsername);
75
76          mName = getIntent().getStringExtra("name");
77          mNameView = (EditText) findViewById(R.id.name);
78          mNameView.setText(mName);
79
80          mSurname = getIntent().getStringExtra("surname");
81          mSurnameView = (EditText) findViewById(R.id.surname);
82          mSurnameView.setText(mSurname);
83
84          mPasswordView = (EditText) findViewById(R.id.password);
85          mPasswordView
86                  .setOnEditorActionListener(new TextView.OnEditorActionListener() {
87                      @Override
88                      public boolean onEditorAction(TextView textView, int id,
89                              KeyEvent keyEvent) {
90                          if (id == R.id.login || id == EditorInfo.IME_NULL) {
91                              attemptLogin();
92                              return true;
93                          }
94                          return false;
95                      }
96                  });
97
98          mLoginFormView = findViewById(R.id.login_form);
99          mLoginStatusView = findViewById(R.id.login_status);
100         mLoginStatusMessageView = (TextView) findViewById(R.id.login_status_message);
101
102         findViewById(R.id.sign_in_button).setOnClickListener(
103                 new View.OnClickListener() {
104                     @Override
105                     public void onClick(View view) {
106                         attemptLogin();
107                     }
108                 });
109     }
110
111     /**
112      *  \brief
113      *  Method called to create the option menu on the top right of the screen
114      *  \param menu Menu containing the menu items
115      */
116     @Override
117     public boolean onCreateOptionsMenu(Menu menu) {
118         super.onCreateOptionsMenu(menu);
119         getMenuInflater().inflate(R.menu.signin, menu);
120         return true;
121     }
122
123     /**
124      *  \brief Attempts to sign in or register the account specified by the signin form.
125      * If there are form errors (invalid email, missing fields, etc.), the
126      * errors are presented and no actual login attempt is made.
127      */
128     public void attemptLogin() {
129         if (mAuthTask != null) {
130             return;
131         }
```

```
132
133         // Reset errors.
134         mUsernameView.setError(null);
135         mNameView.setError(null);
136         mSurnameView.setError(null);
137         mPasswordView.setError(null);
138
139         // Store values at the time of the login attempt.
140         mUsername = mUsernameView.getText().toString();
141         mName = mNameView.getText().toString();
142         mSurname = mSurnameView.getText().toString();
143         mPassword = mPasswordView.getText().toString();
144
145         boolean cancel = false;
146         View focusView = null;
147
148         // Check for a valid password.
149         if (TextUtils.isEmpty(mPassword)) {
150             mPasswordView.setError(getString(R.string.error_field_required));
151             focusView = mPasswordView;
152             cancel = true;
153         } else if (mPassword.length() < 4) {
154             mPasswordView.setError(getString(R.string.error_invalid_password));
155             focusView = mPasswordView;
156             cancel = true;
157         }
158
159         // Check for a valid username.
160         if (TextUtils.isEmpty(mUsername)) {
161             mUsernameView.setError(getString(R.string.error_field_required));
162             focusView = mUsernameView;
163             cancel = true;
164         }
165
166         if (cancel) {
167             // There was an error; don't attempt login and focus the first
168             // form field with an error.
169             focusView.requestFocus();
170         } else {
171             // Show a progress spinner, and kick off a background task to
172             // perform the user login attempt.
173             mLoginStatusMessageView.setText(R.string.progress_signing_in);
174             showProgress(true);
175             mAuthTask = new UserSigninTask();
176             //mAuthTask.execute((Void) null);
177             mAuthTask.execute(mUsername, mName, mSurname, mPassword);
178         }
179     }
180
181     /**
182      *  \brief Shows the progress UI and hides the login form.
183      *  \param show Boolean to display or not the progress
184      */
185     @TargetApi(Build.VERSION_CODES.HONEYCOMB_MR2)
186     private void showProgress(final boolean show) {
187         // On Honeycomb MR2 we have the ViewPropertyAnimator APIs, which allow
188         // for very easy animations. If available, use these APIs to fade-in
189         // the progress spinner.
190         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB_MR2) {
191             int shortAnimTime = getResources().getInteger(
192                     android.R.integer.config_shortAnimTime);
193
194             mLoginStatusView.setVisibility(View.VISIBLE);
195             mLoginStatusView.animate().setDuration(shortAnimTime)
196                     .alpha(show ? 1 : 0)
197                     .setListener(new AnimatorListenerAdapter() {
198                         @Override
199                         public void onAnimationEnd(Animator animation) {
200                             mLoginStatusView.setVisibility(show ? View.VISIBLE
201                                     : View.GONE);
202                         }
203                     });
```

```
204
205             mLoginFormView.setVisibility(View.VISIBLE);
206             mLoginFormView.animate().setDuration(shortAnimTime)
207                     .alpha(show ? 0 : 1)
208                     .setListener(new AnimatorListenerAdapter() {
209                         @Override
210                         public void onAnimationEnd(Animator animation) {
211                             mLoginFormView.setVisibility(show ? View.GONE
212                                     : View.VISIBLE);
213                         }
214                     });
215         } else {
216             // The ViewPropertyAnimator APIs are not available, so simply show
217             // and hide the relevant UI components.
218             mLoginStatusView.setVisibility(show ? View.VISIBLE : View.GONE);
219             mLoginFormView.setVisibility(show ? View.GONE : View.VISIBLE);
220         }
221     }
222
223     /**
224      *  \brief Represents an asynchronous registration task used
225      * to create an account.
226      */
227     public class UserSigninTask extends AsyncTask<String, Void, Boolean> {
228         @Override
229         protected Boolean doInBackground(String... params) {
230
231             String URL = "http://vlenfc.hevs.ch/api/newaccount";
232             HttpResponse response;
233             HttpClient httpclient = new DefaultHttpClient();
234             HttpPost postRequest = new HttpPost(URL);
235
236             try {
237                 String json = "";
238                 JSONObject jsonObject = new JSONObject();
239                 try {
240                     jsonObject.accumulate("username", params[0]);
241                     if(params[1] != null && !params[1].isEmpty()) {
242                         jsonObject.accumulate("name", params[1]);
243                     }
244                     if(params[2] != null && !params[2].isEmpty()) {
245                         jsonObject.accumulate("surname", params[2]);
246                     }
247                     jsonObject.accumulate("password", params[3]);
248                 } catch (JSONException e) {
249                     e.printStackTrace();
250                 }
251
252                 json = jsonObject.toString();
253                 StringEntity se = new StringEntity(json);
254                 postRequest.setEntity(se);
255
256                 postRequest.setHeader("Accept", "application/json");
257                 postRequest.setHeader("Content-type", "application/json");
258
259                 response = httpclient.execute(postRequest);
260                 StatusLine statusLine = response.getStatusLine();
261                 if (statusLine.getStatusCode() == HttpStatus.SC_CREATED ||
262                         statusLine.getStatusCode() == HttpStatus.SC_OK){
263                     ByteArrayOutputStream out = new ByteArrayOutputStream();
264                     response.getEntity().writeTo(out);
265                     out.close();
266                 } else{
267                     //Closes the connection.
268                     response.getEntity().getContent().close();
269                     throw new IOException(statusLine.getReasonPhrase());
270                 }
271             } catch (IOException e) {
272                 e.printStackTrace();
273                 return false;
274             }
275             return true;
```

```
276         }
277
278         @Override
279         protected void onPostExecute(final Boolean success) {
280             mAuthTask = null;
281             showProgress(false);
282
283             if (success) {
284
285                 Intent mainIntent = new Intent(SigninActivity.this, MainActivity.class);
286                 startActivity(mainIntent);
287
288                 Toast.makeText(SigninActivity.this, "Your account is successfully creates." +
289                     " You can now log in to use your Golf Wallet. Your account is credited with" +
290                     " 10 CHF. Enjoy.", Toast.LENGTH_LONG).show();
291
292                 finish();
293             } else {
294                 mUsernameView.setError(getString(R.string.error_username_already_taken));
295                 mUsernameView.requestFocus();
296             }
297         }
298
299         @Override
300         protected void onCancelled() {
301             mAuthTask = null;
302             showProgress(false);
303         }
304     }
305 }
```

H.4.3 HTTPRequest.java

```
1  package com.golfwallet.connectivity;
2
3  import java.io.ByteArrayOutputStream;
4  import java.io.IOException;
5
6  import org.apache.http.HttpResponse;
7  import org.apache.http.HttpStatus;
8  import org.apache.http.StatusLine;
9  import org.apache.http.client.ClientProtocolException;
10 import org.apache.http.client.HttpClient;
11 import org.apache.http.client.methods.HttpGet;
12 import org.apache.http.client.methods.HttpPost;
13 import org.apache.http.entity.StringEntity;
14 import org.apache.http.impl.client.DefaultHttpClient;
15 import org.json.JSONException;
16 import org.json.JSONObject;
17
18 import android.os.AsyncTask;
19 import android.util.Base64;
20
21 /** \brief Asynchronous task used to call the different APIs
22  *  \author Jacky CASAS
23  *  \version 1.0
24  *  \date 04.06.2014
25  */
26 public class HTTPRequest extends AsyncTask<String, String, String> {
27
28     /**
29      *  \brief Method which called the APIs.
30      *  \param uri Array of strings
31      *  \return A string corresponding to the answer.
32      */
33     @Override
34     protected String doInBackground(String... uri) {
35
36         String URL = "http://vlenfc.hevs.ch";
```

```
37          String responseString = null;
38          HttpResponse response;
39          HttpClient httpclient = new DefaultHttpClient();
40
41          HttpGet getRequest;
42          HttpPost postRequest;
43
44          String credentials;
45          String base64EncodedCredentials;
46
47          switch(uri[0]) {
48          case "/":
49              try {
50                  getRequest = new HttpGet(URL);
51                  response = httpclient.execute(getRequest);
52                  StatusLine statusLine = response.getStatusLine();
53                  if(statusLine.getStatusCode() == HttpStatus.SC_OK){
54                      ByteArrayOutputStream out = new ByteArrayOutputStream();
55                      response.getEntity().writeTo(out);
56                      out.close();
57                      responseString = out.toString();
58                  } else{
59                      //Closes the connection.
60                      response.getEntity().getContent().close();
61                      throw new IOException(statusLine.getReasonPhrase());
62                  }
63              } catch (ClientProtocolException e) {
64              } catch (IOException e) {
65              }
66              break;
67          case "/api/newpassword":
68              try {
69                  getRequest = new HttpGet(URL + uri[0] + uri[1]);
70                  response = httpclient.execute(getRequest);
71                  StatusLine statusLine = response.getStatusLine();
72                  if(statusLine.getStatusCode() == HttpStatus.SC_OK){
73                      ByteArrayOutputStream out = new ByteArrayOutputStream();
74                      response.getEntity().writeTo(out);
75                      out.close();
76                      responseString = out.toString();
77                  } else{
78                      //Closes the connection.
79                      response.getEntity().getContent().close();
80                      throw new IOException(statusLine.getReasonPhrase());
81                  }
82              } catch (ClientProtocolException e) {
83              } catch (IOException e) {
84              }
85              break;
86          case "/api/users":
87
88              postRequest = new HttpPost(URL + uri[0]);
89
90              try {
91                  String json = "";
92                  JSONObject jsonObject = new JSONObject();
93                  try {
94                      jsonObject.accumulate("username", "jbonvin");
95                      jsonObject.accumulate("password", "jjjj");
96                  } catch (JSONException e) {
97                      e.printStackTrace();
98                  }
99
100                 json = jsonObject.toString();
101                 StringEntity se = new StringEntity(json);
102                 postRequest.setEntity(se);
103
104                 postRequest.setHeader("Accept", "application/json");
105                 postRequest.setHeader("Content-type", "application/json");
106
107                 response = httpclient.execute(postRequest);
108                 StatusLine statusLine = response.getStatusLine();
```

```
109                 if(statusLine.getStatusCode() == HttpStatus.SC_OK){
110                     ByteArrayOutputStream out = new ByteArrayOutputStream();
111                     response.getEntity().writeTo(out);
112                     out.close();
113                     responseString = out.toString();
114                 } else{
115                     //Closes the connection.
116                     response.getEntity().getContent().close();
117                     throw new IOException(statusLine.getReasonPhrase());
118                 }
119             } catch (IOException e) {
120                 e.printStackTrace();
121             }
122             break;
123         case "/api/users/jbonvin":
124             getRequest = new HttpGet(URL + uri[0]);
125             try {
126                 response = httpclient.execute(getRequest);
127                 StatusLine statusLine = response.getStatusLine();
128                 if(statusLine.getStatusCode() == HttpStatus.SC_OK){
129                     ByteArrayOutputStream out = new ByteArrayOutputStream();
130                     response.getEntity().writeTo(out);
131                     out.close();
132                     responseString = out.toString();
133                 } else{
134                     //Closes the connection.
135                     response.getEntity().getContent().close();
136                     throw new IOException(statusLine.getReasonPhrase());
137                 }
138             } catch (ClientProtocolException e) {
139             } catch (IOException e) {
140             }
141             break;
142         case "/api/token":
143             // uri[1] : username or token
144             // uri[2] : password or empty
145             getRequest = new HttpGet(URL + uri[0]);
146             credentials = uri[1] + ":" + uri[2];
147             base64EncodedCredentials = Base64.encodeToString(credentials.getBytes(), Base64.NO_WRAP);
148             getRequest.addHeader("Authorization", "Basic " + base64EncodedCredentials);
149 
150             try {
151                 response = httpclient.execute(getRequest);
152                 StatusLine statusLine = response.getStatusLine();
153                 if(statusLine.getStatusCode() == HttpStatus.SC_OK){
154                     ByteArrayOutputStream out = new ByteArrayOutputStream();
155                     response.getEntity().writeTo(out);
156                     out.close();
157                     responseString = out.toString();
158 
159                     // get token from JSON object
160                     try {
161                         JSONObject jsonObj = new JSONObject(responseString);
162                         responseString = jsonObj.getString("token");
163                     } catch (JSONException e) {
164                         e.printStackTrace();
165                     }
166 
167                 } else{
168                     //Closes the connection.
169                     response.getEntity().getContent().close();
170                     throw new IOException(statusLine.getReasonPhrase());
171                 }
172             } catch (ClientProtocolException e) {
173                 e.printStackTrace();
174             } catch (IOException e) {
175                 e.printStackTrace();
176             }
177             break;
178         case "/api/user":
179             // uri[1] : username or token
180             // uri[2] : password or empty
```

```
181             getRequest = new HttpGet(URL + uri[0]);
182             credentials = uri[1] + ":" + uri[2];
183             base64EncodedCredentials = Base64.encodeToString(credentials.getBytes(), Base64.NO_WRAP);
184             getRequest.addHeader("Authorization", "Basic " + base64EncodedCredentials);
185 
186             try {
187                 response = httpclient.execute(getRequest);
188                 StatusLine statusLine = response.getStatusLine();
189                 if(statusLine.getStatusCode() == HttpStatus.SC_OK){
190                     ByteArrayOutputStream out = new ByteArrayOutputStream();
191                     response.getEntity().writeTo(out);
192                     out.close();
193                     responseString = out.toString();
194                 } else{
195                     //Closes the connection.
196                     response.getEntity().getContent().close();
197                     throw new IOException(statusLine.getReasonPhrase());
198                 }
199             } catch (ClientProtocolException e) {
200                 e.printStackTrace();
201             } catch (IOException e) {
202                 e.printStackTrace();
203             }
204             break;
205         case "/api/deviceid":
206             // uri[1] : username or token
207             // uri[2] : password or empty
208             // uri[3] : android_id
209             getRequest = new HttpGet(URL + uri[0] + uri[3]);
210             credentials = uri[1] + ":" + uri[2];
211             base64EncodedCredentials = Base64.encodeToString(credentials.getBytes(), Base64.NO_WRAP);
212             getRequest.addHeader("Authorization", "Basic " + base64EncodedCredentials);
213 
214             try {
215                 response = httpclient.execute(getRequest);
216                 StatusLine statusLine = response.getStatusLine();
217                 if(statusLine.getStatusCode() == HttpStatus.SC_OK){
218                     ByteArrayOutputStream out = new ByteArrayOutputStream();
219                     response.getEntity().writeTo(out);
220                     out.close();
221                     responseString = out.toString();
222 
223                     // get AID from JSON object
224                     try {
225                         JSONObject jsonObj = new JSONObject(responseString);
226                         responseString = jsonObj.getString("uid");
227                     } catch (JSONException e) {
228                         e.printStackTrace();
229                     }
230                 } else{
231                     //Closes the connection.
232                     response.getEntity().getContent().close();
233                     throw new IOException(statusLine.getReasonPhrase());
234                 }
235             } catch (ClientProtocolException e) {
236                 e.printStackTrace();
237             } catch (IOException e) {
238                 e.printStackTrace();
239             }
240             break;
241         case "/api/balance":
242             // uri[1] : username or token
243             // uri[2] : password or empty
244             getRequest = new HttpGet(URL + uri[0]);
245             credentials = uri[1] + ":" + uri[2];
246             base64EncodedCredentials = Base64.encodeToString(credentials.getBytes(), Base64.NO_WRAP);
247             getRequest.addHeader("Authorization", "Basic " + base64EncodedCredentials);
248 
249             try {
250                 response = httpclient.execute(getRequest);
251                 StatusLine statusLine = response.getStatusLine();
252                 if(statusLine.getStatusCode() == HttpStatus.SC_OK){
```

```
                    ByteArrayOutputStream out = new ByteArrayOutputStream();
                    response.getEntity().writeTo(out);
                    out.close();
                    responseString = out.toString();

                    // get balance from JSON object
                    try {
                        JSONObject jsonObj = new JSONObject(responseString);
                        responseString = jsonObj.getString("balance");
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                } else{
                    //Closes the connection.
                    response.getEntity().getContent().close();
                    throw new IOException(statusLine.getReasonPhrase());
                }
            } catch (ClientProtocolException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            break;
        case "/api/devices":
            // uri[1] : username or token
            // uri[2] : password or empty
            getRequest = new HttpGet(URL + uri[0]);
            credentials = uri[1] + ":" + uri[2];
            base64EncodedCredentials = Base64.encodeToString(credentials.getBytes(), Base64.NO_WRAP);
            getRequest.addHeader("Authorization", "Basic " + base64EncodedCredentials);

            try {
                response = httpclient.execute(getRequest);
                StatusLine statusLine = response.getStatusLine();
                if(statusLine.getStatusCode() == HttpStatus.SC_OK){
                    ByteArrayOutputStream out = new ByteArrayOutputStream();
                    response.getEntity().writeTo(out);
                    out.close();
                    responseString = out.toString();
                } else{
                    //Closes the connection.
                    response.getEntity().getContent().close();
                    throw new IOException(statusLine.getReasonPhrase());
                }
            } catch (ClientProtocolException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            break;
        case "/api/transactions":
            // uri[1] : username or token
            // uri[2] : password or empty
            getRequest = new HttpGet(URL + uri[0]);
            credentials = uri[1] + ":" + uri[2];
            base64EncodedCredentials = Base64.encodeToString(credentials.getBytes(), Base64.NO_WRAP);
            getRequest.addHeader("Authorization", "Basic " + base64EncodedCredentials);

            try {
                response = httpclient.execute(getRequest);
                StatusLine statusLine = response.getStatusLine();
                if(statusLine.getStatusCode() == HttpStatus.SC_OK){
                    ByteArrayOutputStream out = new ByteArrayOutputStream();
                    response.getEntity().writeTo(out);
                    out.close();
                    responseString = out.toString();
                } else{
                    //Closes the connection.
                    response.getEntity().getContent().close();
                    throw new IOException(statusLine.getReasonPhrase());
                }
            } catch (ClientProtocolException e) {
```

```
325                 e.printStackTrace();
326             } catch (IOException e) {
327                 e.printStackTrace();
328             }
329             break;
330         }
331         return responseString;
332     }
333 
334     /**
335      *  \brief Method called when the asynchronous doInBackground() method is over.
336      *  \param result String resulting of the asynchronous request
337      */
338     @Override
339     protected void onPostExecute(String result) {
340         super.onPostExecute(result);
341     }
342 }
```

H.4.4 ReadWriteData.java

```
1  package com.golfwallet.preference;
2  
3  import android.content.Context;
4  import android.content.SharedPreferences;
5  
6  /** \brief Store et retrieve data from the SharedPreferences
7   *  \author Jacky CASAS
8   *  \version 1.0
9   *  \date 04.06.2014
10  */
11 public class ReadWriteData {
12 
13     public final static String LOGIN_USERNAME = "name";     ///< Constant for the username
14     public final static String LOGIN_PASSWORD = "password"; ///< Constant for the password
15     public final static String LOGIN_TOKEN = "token";       ///< Constant for the token
16     public final static String CONNECTED = "connected";     ///< Constant for the connection
17     public final static String NFC_AID = "aid";             ///< Constant for the AID
18 
19     private static final String LOGIN_FILE_NAME = ReadWriteData.class
20             .getSimpleName();                               ///< Constant for the file
21 
22     /**
23      *  \brief Write a value in the SharedPreferences.
24      *  \param context Context of the app
25      *  \param tag String corresponding with a class constant for the type of data to save
26      *  \param data String containing the data to save
27      */
28     public static void write(Context context, String tag, String data) {
29         /* Open file and editor */
30         SharedPreferences sharedPref = context.getSharedPreferences(
31                 LOGIN_FILE_NAME, 0);
32         SharedPreferences.Editor editor = sharedPref.edit();
33 
34         /* Put string value with a tag in the file */
35         editor.putString(tag, data);
36         /* Close the editor */
37         editor.commit();
38     }
39 
40     /**
41      *  \brief Read a value from the SharedPreferences
42      *  \param context Context of the app
43      *  \param tag Type of data
44      *  \return The data requested
45      */
46     public static String read(Context context, String tag) {
47         SharedPreferences sharedPref = context.getSharedPreferences(
48                 LOGIN_FILE_NAME, 0);
```

```
49          return sharedPref.getString(tag, null);
50      }
51
52      /**
53       *  \brief Remove a data from the SharedPreferences
54       *  \param context Context of the app
55       *  \param tag Type of data
56       */
57      public static void remove(Context context, String tag) {
58          SharedPreferences sharedPref = context.getSharedPreferences(
59                  LOGIN_FILE_NAME, 0);
60          SharedPreferences.Editor editor = sharedPref.edit();
61
62          editor.remove(tag);
63          editor.commit();
64      }
65
66      /**
67       *  \brief Clear all the datas stored in the SharedPreferences
68       *  \param context Context of the app
69       */
70      public static void clearAll(Context context) {
71          SharedPreferences sharedPref = context.getSharedPreferences(
72                  LOGIN_FILE_NAME, 0);
73          SharedPreferences.Editor editor = sharedPref.edit();
74
75          editor.clear();
76          editor.commit();
77      }
78
79      /**
80       *  \brief Store in the SharedPreferences that the user is connected.
81       */
82      public static void setConnected(Context context) {
83          SharedPreferences sharedPref = context.getSharedPreferences(LOGIN_FILE_NAME, 0);
84          SharedPreferences.Editor editor = sharedPref.edit();
85          editor.putBoolean(CONNECTED, true);
86          editor.commit();
87      }
88
89      /**
90       *  \brief Method used to know if the user is connected.
91       *  \param context Context of the app
92       *  \return True if connected and False if not
93       */
94      public static boolean getConnectionState(Context context) {
95          SharedPreferences sharedPref = context.getSharedPreferences(LOGIN_FILE_NAME, 0);
96          return sharedPref.getBoolean(CONNECTED, false);
97      }
98
99      /**
100      *  \brief Disconnect the user by erasing the connected date in the SharedPreferences
101      *  \param context Context of the app
102      */
103     public static void disconnect(Context context) {
104         SharedPreferences sharedPref = context.getSharedPreferences(LOGIN_FILE_NAME, 0);
105         SharedPreferences.Editor editor = sharedPref.edit();
106         editor.remove(CONNECTED);
107         editor.commit();
108     }
109 }
```

H.4.5 NfcHceService.java

```
1  package com.golfwallet.nfc;
2
3  import com.golfwallet.main.MainActivity;
4  import android.nfc.cardemulation.HostApduService;
5  import android.os.Bundle;
```

```java
import android.util.Log;

/** \brief Class NfcHceService.
 *  \author Jacky CASAS
 *  \version 1.0
 *  \date 04.06.2014
 */
public class NfcHceService extends HostApduService {

    private byte[] aid = new byte[4];    ///< AID of the connected device
    private byte[] success = new byte[] {
            (byte) 0x90, 0x00
            };                          ///< Success 2 bytes to put at the end of the response

    /**
     *  \brief Process a command : receive and treat the request and compute the response
     *  \param apdu Byte array containing the request
     *  \param extras Bundle containing information from the activity
     *  \return A byte array containing the response
     */
    @Override
    public byte[] processCommandApdu(byte[] apdu, Bundle extras) {
        if (selectAidApdu(apdu)) {
            Log.i("HCEDEMO", "Application selected");
            aid = MainActivity.getAID();
            return concat(aid, success);
        }

        aid = MainActivity.getAID();
        return concat(aid, success);
    }

    /**
     *  \brief Compare the first two byte to know if it is the expected message
     *  \return True if APDU begins with 0x00 0xA4 and is longer than 2 bytes
     */
    private boolean selectAidApdu(byte[] apdu) {
        return apdu.length >= 2 && apdu[0] == (byte)0 && apdu[1] == (byte)0xa4;
    }

    /**
     *  \brief Called when the service is deactivated
     *  \param reason The reason why it is deactivated
     */
    @Override
    public void onDeactivated(int reason) {
        Log.i("HCEDEMO", "Deactivated: " + reason);
    }

    /**
     *  \brief Concatenate two byte arrays.
     *  \return The concatenate array
     */
    public byte[] concat(byte[] A, byte[] B) {
        int aLen = A.length;
        int bLen = B.length;
        byte[] C= new byte[aLen+bLen];
        System.arraycopy(A, 0, C, 0, aLen);
        System.arraycopy(B, 0, C, aLen, bLen);
        return C;
    }
}
```

H.4.6 IsoDepAdapter.java

```java
package com.golfwallet.nfc;

import java.util.ArrayList;
import java.util.List;
```

```java
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

/** \brief Class IsoDepAdapter.
 *  \author Jacky CASAS
 *  \version 1.0
 *  \date 04.06.2014
 */
public class IsoDepAdapter extends BaseAdapter {

    private LayoutInflater layoutInflater;          ///< The layout inflater
    private List<String> messages = new ArrayList<String>(100); ///< A queue of messages
    private int messageCounter;                     ///< Counter for the message in message list

    /**
     *  \brief Constructor
     *  \param layoutInfater To inflate the adapter
     */
    public IsoDepAdapter(LayoutInflater layoutInflater) {
        this.layoutInflater = layoutInflater;
    }

    /**
     *  \brief Used to add a message in the message queue
     *  \param message String containing the message
     */
    public void addMessage(String message) {
        messageCounter++;
        messages.add("Message [" + messageCounter + "]: " + message);
        notifyDataSetChanged();
    }

    /**
     *  \brief Count the number of messages in the list
     *  \return The number of messages
     */
    @Override
    public int getCount() {
        return messages == null ? 0 : messages.size();
    }

    /**
     *  \brief Give a message from a position
     *  \param position Position of the message in the list
     *  \return The message at the requested position
     */
    @Override
    public Object getItem(int position) {
        return messages.get(position);
    }

    /**
     *  \brief Give the ID of a message
     *  \param position Position of the message
     *  \return The ID of the message
     */
    @Override
    public long getItemId(int position) {
        return 0;
    }

    /**
     *  \brief Getter for a specific view
     *  \param position Position of the view
     *  \param convertView Actual view
     *  \param parent Parent of the view
     *  \return The right view
     */
    @Override
```

```
77      public View getView(int position, View convertView, ViewGroup parent) {
78          if (convertView == null) {
79              convertView = layoutInflater.inflate(android.R.layout.simple_list_item_1, parent, false);
80          }
81          TextView view = (TextView)convertView.findViewById(android.R.id.text1);
82          view.setText((CharSequence)getItem(position));
83          return convertView;
84      }
85  }
```

H.4.7 IsoDepTransceiver.java

```
1   package com.golfwallet.nfc;
2
3   import java.io.IOException;
4   import android.nfc.tech.IsoDep;
5
6   /** \brief Class IsoDepTransceiver.
7    *  \author Jacky CASAS
8    *  \version 1.0
9    *  \date 04.06.2014
10   */
11  public class IsoDepTransceiver implements Runnable {
12
13      /**
14       *  \brief Interface to implement when we want to receive messages
15       */
16      public interface OnMessageReceived {
17          void onMessage(byte[] message);
18          void onError(Exception exception);
19      }
20
21      private IsoDep isoDep;                          ///< IsoDep
22      private OnMessageReceived onMessageReceived;///< Instance of the inner interface
23
24      /**
25       *  \brief Constructor
26       *  \param isoDep
27       *  \param onMessageReceived
28       */
29      public IsoDepTransceiver(IsoDep isoDep, OnMessageReceived onMessageReceived) {
30          this.isoDep = isoDep;
31          this.onMessageReceived = onMessageReceived;
32      }
33
34      private static final byte[] CLA_INS_P1_P2 = {
35          0x00, (byte)0xA4, 0x04, 0x00
36          };                                          ///< Header of the AID
37      private static final byte[] AID_ANDROID = {
38          (byte)0xF0, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06
39          };                                          ///< AID of the Android service
40
41      /**
42       *  \brief Compute the AID APDU
43       *  \param aid Byte array with the AID
44       *  \return The ready to send APDU
45       */
46      private byte[] createSelectAidApdu(byte[] aid) {
47          byte[] result = new byte[6 + aid.length];
48          System.arraycopy(CLA_INS_P1_P2, 0, result, 0, CLA_INS_P1_P2.length);
49          result[4] = (byte)aid.length;
50          System.arraycopy(aid, 0, result, 5, aid.length);
51          result[result.length - 1] = 0;
52          return result;
53      }
54
55      /**
56       *  \brief Loop of execution of the transceiver
57       */
```

```
58      @Override
59      public void run() {
60          int messageCounter = 0;
61          try {
62              isoDep.connect();
63              byte[] response = isoDep.transceive(createSelectAidApdu(AID_ANDROID));
64              while (isoDep.isConnected() && !Thread.interrupted()) {
65                  String message = "Message from IsoDep " + messageCounter++;
66                  response = isoDep.transceive(message.getBytes());
67                  onMessageReceived.onMessage(response);
68              }
69              isoDep.close();
70          }
71          catch (IOException e) {
72              onMessageReceived.onError(e);
73          }
74      }
75  }
```