Institut Supérieur d'Informatique
de Modélisation et de leurs Applications

Campus des Cézeaux
1 Rue de la Chebarde
63178 AUBIERE Cedex

Développement d'une application Android sur les tags NFC

Projet de Deuxième Année
Fillière : Systèmes d'Information et Aide à la Décision

Présenté par : Matthieu Basse & Marvin Volga

Responsables ISIMA :
Philippe Lacomme
Maxime Chassaing

Soutenu le 18/03/2016
Durée : 120h

Remerciements

Nous tenons à remercier M. Philippe LACOMME, notre référent, tout d'abord pour sa proposition de projet et surtout pour sa disponibilité tout au long de notre travail.

De plus nous souhaitons remercier M. Maxime CHASSAING qui nous a accompagné durant ce projet et nous a apporté une précieuse aide dans la réalisation de celui-ci.

Résumé

L'objectif du projet était de développer une **application Android** capable de gérer les évènements de **lecture** et d'**écriture** sur les tags **NFC**. Après s'être familiarisé avec les applications Android existantes, nous avons pu mieux visualiser notre objectif et commencer à faire des recherches sur le NFC dont nous ne savions pas grand-chose.
Notre application devait être capable de reconnaitre tous les types de tags NFC et de modifier leur contenu si besoin. Cela impliquait qu'elle devait également fonctionner pour des cartes de paiement ou encore des cartes étudiantes qui n'ont pas le même format que les tags NFC programmables. Les applications existantes ne proposent pas toutes la lecture de ce type de tag et la modification du contenu n'est proposée dans aucune d'entre elle.
Nous avons utilisé le logiciel de développement **Android Studio** pour réaliser notre application, nommé NFCeditor. Cette dernière possède trois fonctionnalités : la lecture, l'écriture et l'effacement d'un tag NFC et permet de lire les types de tags contenant du **NDEF**.

Mots clés : application Android, lecture, écriture, NFC, Android Studio, NDEF

Abstract

The purpose of this project was to develop an **Android application** able to handle different events on **NFC** tags such as **reading** and **writing**. After becoming familiar with all the existing Android applications, we were able to better visualize our goal and begin to make our researches on NFC.
Our application had to be able to recognize all the types of NFC tag and to modify their content if needed. That involved that the application had also to function with payment cards and student cards which has not the same format as programmable NFC tags. Not all of the existing applications propose to read this kind of tag and none of them propose to modify the content.
We used the software development **Android Studio** to realize our application, named NFCeditor. The application has three functionalities: reading, writing and erasing a NFC tag and allows reading the types of tags that contains **NDEF**.

Keywords: Android application, NFC, reading, writing, Android Studio, NDEF

Table des matières

Table des figures

Lexique

API: Application Programming Interface. C'est un ensemble d'outils aidant à la construction d'un logiciel ou d'une application. Le niveau d'API Android (API Level) correspond à une version d'Android. Par exemple la version 1.6 d'Android correspond au niveau d'API 4.

EDI: Les Environnement de Développement Intégrés rassemblent dans un même outil tous les éléments nécessaires à la programmation (éditeur de texte, compilateur, débogueur…).

IntelliJ IDEA: C'est un EDI Java pour le développement de logiciel développé par JetBrains.

ISO 14443: Norme définissant un mécanisme de communication et impose la taille des objets qui communiquent

ISO 14443-4: Norme définissant le protocole de transmission des cartes à puce sans contact.

ISO 15693: Norme ISO pour les cartes de voisinage c'est-à-dire celles qui peuvent être lues à une distance plus grande que les cartes sans contact.

JIS 6319-4: norme dérivée de l'ISO 18092 pour la communication en mode passif.

LIMOS: « Le Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes est une Unité Mixte de Recherche (UMR 6158) en informatique, et plus généralement en Sciences et Technologies de l'Information et de la Communication (STIC). »

MIFARE Classic: Protocole propriétaire partiellement conforme au standard de l'ISO 14443-A car possède un protocole de sécurité propriétaire NXP pour l'authentification et le cryptage.

Package : Unique identifiant que possède chaque application Android. Il existe une convention pour le nommage des packages : nom_domaine.nom_entreprise.nom_application. Pour notre application le nom de package est « fr.isima.nfceditor »

RFC: Les Requests For Comments sont une série numérotée de documents officiels décrivant les aspects techniques d'Internet, ou de différents matériels informatiques (routeurs, serveur DHCP).

SDK: Software Developpement Kit. Il s'agit d'un ensemble complet d'outils de développement logiciel (débogueur, bibliothèques, émulateur…).

TNF: Type Name Format. Correspond à une valeur numérique indiquant comment interpréter le Type du Record.

URI: Uniform Resource Identifier est une chaîne de caractère qui identifie une ressource web.

Références webographiques

- [Android Developer] http://developer.android.com (date de consultation 2015/2016)
- [NFC Forum] http://nfc-forum.org (date de consultation 2015/2016)
- [Android Headlines] http://www.androidheadlines.com (date de consultation 2015)
- [Developpez.com] http://www.developpez.com (date de consultation 2015)
- [NXP] http://www.nxp.com (date de consultation 2015/2016)
- [Wikipédia] https://fr.wikipedia.org (date de consultation 2016)
- [Intel] https://software.intel.com (date de consultation 2015)
- [AndyTags] http://www.andytags.com/nfc (date de consultation 2015/2016)
- [Fragmentos Developers] http://www.framentos.com (date de consultation 2015/2016)
- [JavaTechig] http://javatechig.com (date de consultation 2016)
- [Nfcpy] http://nfcpy.readthedocs.org (date de consultation 2016)
- [Safari] https://www.safaribooksonline.com (date de consultation 2016)
- [GoToTags] https://gototags.com (date de consultation 2016)
- [OpenClassRooms] http://www.openclassrooms.com (date de consultation 2016)
- [OpenClipArt] https://openclipart.org (date de consultation 2016)

Introduction

Le projet dont ce rapport fait l'objet a été réalisé par deux étudiants en deuxième année à l'ISIMA (Institut Supérieur d'Informatique de Modélisation et de leurs Applications), à savoir Matthieu Basse et Marvin Volga. Ceux-ci ont été supervisés par M. Philippe Lacomme, qui a proposé le projet, et M. Maxime Chassaing.

M. Lacomme a testé plusieurs applications du Google Play Store permettant d'interagir avec des tags NFC, qui est une technologie de communication sans fil. Il s'est rendu compte qu'aucune d'entre elles ne lisait tous les types de tags NFC dont il avait connaissance. Aussi, c'est dans ce contexte qu'il nous a confié le projet de réaliser une application Android dont l'objectif serait de pouvoir lire et écrire des tags NFC.

L'application devrait pouvoir communiquer avec le plus de types de tag possible et être ergonomique car elle allait être publiée sur le Google Play Store.

Pour situer le sujet plus en détail, il nous a fallu prendre connaissance des applications existantes et les tester. Ensuite nous nous sommes renseignés sur les différents types de tags NFC existants et sur la manière d'échanger avec eux.

Nous commencerons par présenter les bases et le cadre du projet avant d'expliquer sa mise en œuvre au travers des outils utilisés et de sa planification. Afin de présenter au mieux sa réalisation, nous verrons la structure de l'application demandée avant de détailler ses fonctionnalités. Enfin nous évoquerons les améliorations possibles à apporter à l'application avant de conclure ce rapport par un bilan du projet.

1. Les bases et le cadre du projet

Le projet consistant en une application Android, nous allons présenter ce système d'exploitation avant d'expliquer ce qu'est la technologie NFC, qui est au cœur du projet, et définir les objectifs de ce dernier.

1.1 A propos d'Android

1.1.1 Système d'exploitation

Android est un système d'exploitation mobile pour smartphones et tablettes notamment. Développé en Java par l'Open Handset Alliance, il appartient aujourd'hui à Google et est en Open Source.

Figure 1 : Logo d'Android

1.1.2 Android API

L'API[1] Android est une collection de packages qui définissent des classes facilitant l'utilisation des fonctionnalités et des appareils Android.

De plus le SDK[2] Android propose les librairies et outils de développements essentiels pour construire, tester et débugger les applications Android.

1.1.3 Développement d'application Android

1.1.3.1 Composants de l'application

Une application Android est structurée en blocks. Ses composants sont de quatre types : *Activities*, qui représentent un unique écran avec une interface utilisateur ; *Services* qui fonctionnent en arrière-plan pour effectuer des opérations de longue durée ou effectuer des travaux pour les processus distants ; *Content Providers* qui gèrent un ensemble de données d'applications partagées et *Broadcast Receivers* qui sont des composants répondant aux annonces diffusées à l'échelle du système.

Chaque type a des buts distincts et un cycle de vie qui définit comment créer et détruire le composant. Le système du téléphone utilise ces composants comme points d'entrée dans l'application.

L'application réalisée au cours de ce projet est composée uniquement d'*Activities* (Activités).

1.1.3.2 Activités

Une application Android est généralement composée de plusieurs activités liées les unes aux autres. Normalement, l'activité affichée au lancement de l'application est appelée « activité main ».

Chaque activité peut en lancer une autre pour effectuer différentes actions et respecte un cycle de vie comme le présente la figure ci-après, extraite du site http://www.openclassrooms.com.

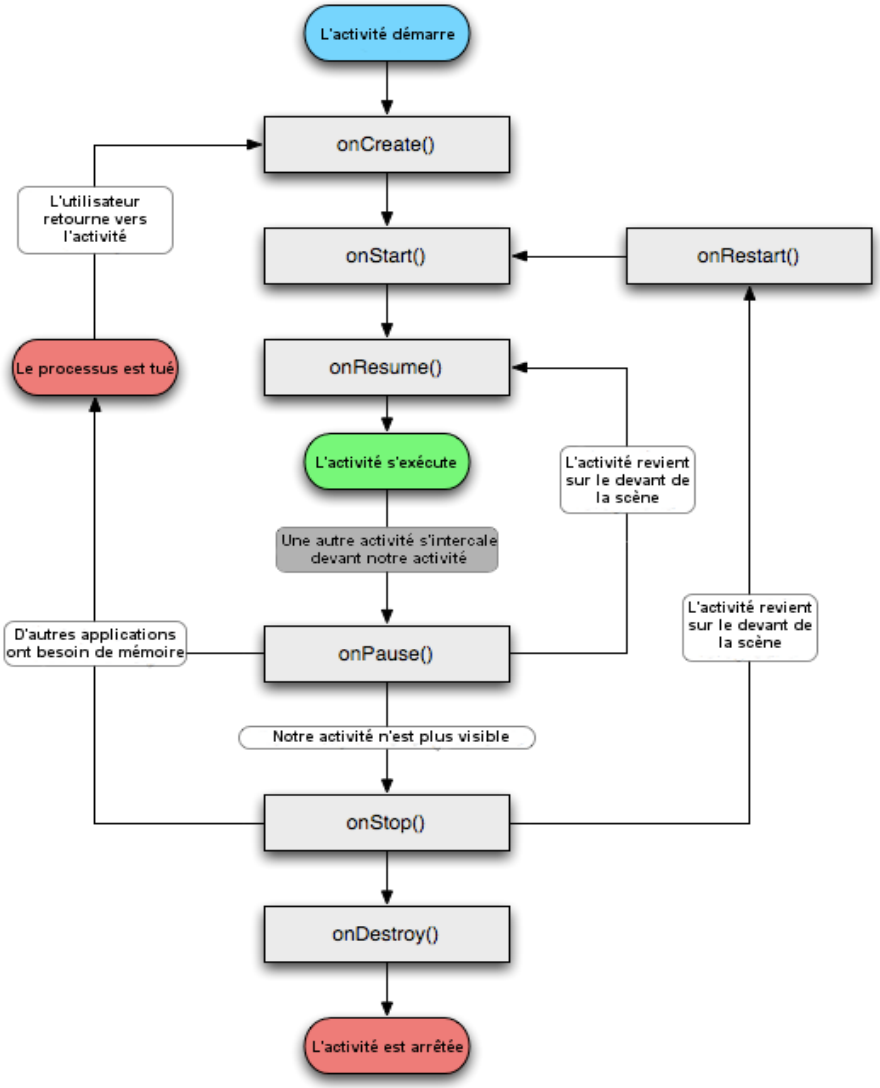


Figure 2 : Cycle de vie d'une activité

Nous allons décrire l'ensemble des fonctions liées au cycle de vie d'une application :

- *onCreate()* : est appelée au premier lancement de l'activité, ou si celle-ci est ressuscitée.
- *onStart()* : est exécutée après chaque *onCreate()* ou *onRestart()*, elle permet de charger les données lorsque l'activité est visible par l'utilisateur.
- *onRestart()* : est lancée lorsque l'activité repasse au premier plan après avoir été arrêtée via *onStop()*
- *onResume()* : est exécutée lorsque l'activité est passée en avant plan (permet la mise à jour des données)
- *onPause()* : est appelée chaque fois que l'utilisateur change d'activité ou quand celui-ci ferme l'activité (permet la sauvegarde des données)
- *onStop()* : est lancée avant chaque mise en sommeil de l'activité (permet la libération des ressources)
- *onDestroy()* : est exécutée lors de l'arrêt de l'activité, elle met fin au cycle de vie.

1.1.3.3 Layout

La structure visuelle d'une activité est décrite dans un fichier .xml appelé layout.

Ainsi, chaque activité a son propre fichier layout pour la décrire. Par exemple, on associera l'activité MainActivity au fichier activity_main.xml qui est généré automatiquement lors de la création de l'activité dans le projet.

1.1.3.4 Fichier Manifest

Le fichier Manifest indique au système qu'un composant qui veut se lancer existe déjà. L'application doit déclarer tous ses composants, notamment les activités, dans un fichier AndroidManifest.xml.

Ce fichier comprend également les informations concernant les permissions requises par l'application, le niveau minimum de l'API, les caractéristiques logicielles et matérielles et les librairies requises.

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="fr.isima.android.nfceditor">
4
5      <uses-sdk android:minSdkVersion="10" />
6
7      <uses-permission android:name="android.permission.NFC" />
8
9      <uses-feature
10         android:name="android.hardware.nfc"
11         android:required="true" />
12
13     <application
14         android:allowBackup="true"
15         android:icon="@mipmap/ic_launcher"
16         android:label="@string/app_name"
17         android:supportsRtl="true"
18         android:theme="@style/AppTheme">
19         <activity
20             android:name=".MainActivity"
21             android:theme="@style/AppTheme">
22         </activity>
23     </application>
24 </manifest>
```

Niveau minimum de l'API

Permission recquise

Caractéristique matérielle

Déclaration d'une Activity

Figure 3 : Exemple de fichier AndroidManifest.xml

1.1.3.5 Ressources de l'application

En complément du code, une application Android a besoin de ressources séparées du code comme des images, des fichiers son, etc.

Pour chaque ressource inclue dans l'application, les outils de construction SDK définissent un unique entier nommé ID qui est utilisé pour faire référence à la ressource à partir du code ou d'autres ressources .xml.

1.2 Qu'est-ce que le NFC ?

Le NFC, Near Field Communication, (ou communication en champ proche) est une technologie permettant d'échanger des données à une distances très faible (moins de 5cm), entre deux appareils équipés de ce dispositif. Le NFC est intégré dans certains téléphones portables sous forme de puce, ainsi que sur certaines cartes de transport ou de paiement (permettant le paiement sans contact).

Le NFC a trois modes de fonctionnement différents.

1.2.1 Le mode lecteur

Le mobile équipé du NFC est capable de lire des tags (étiquettes électroniques), pour récolter des informations pratiques, ou pour lancer une action de manière automatique sur un Smartphone.

Exemples d'utilisations :

- Parcours dans un musée
- Automatisation d'une tâche : lancer une application à l'approche du tag NFC par exemple, ou désactiver certaines fonctionnalités du mobile.

Ce mode permet également de donner la fonctionnalité de commutateur à un tag NFC. En effet, il est possible sur certaines applications de programmer le tag de façon à ce que la première fois qu'on passe le téléphone dessus, il réalise une certaine action et qu'en repassant le téléphone une deuxième fois, il réalise une autre action.

Exemple d'utilisation :

- Au premier passage du téléphone, le tag active la fonction GPS du téléphone, et au deuxième passage, le tag désactive cette même fonction.

1.2.2 Le mode pair-à-pair

Ce mode de fonctionnement permet l'échange d'informations entre deux appareils équipés du NFC.

Exemple d'utilisation :

- Un échange de photos entre une tablette et un Smartphone.
- Récupération des contacts téléphonique lors d'un changement de portable.

1.2.3 Le mode émulation de carte

Le terminal mobile fonctionne comme une carte sans contact. La carte SIM du portable peut être utilisée pour stocker des informations chiffrées, et les sécuriser.

Exemples d'utilisations :

- Paiement sans contact.
- Gestion des coupons de réduction ou des points de fidélité dans un magasin.

1.3 Qu'est-ce qu'un tag NFC ?

Les tags NFC peuvent être des étiquettes, des autocollants ou même des bracelets contenant de petites micros puce pouvant stocker une petite quantité d'informations afin de le transférer à un autre dispositif utilisant le NFC. Ainsi, un téléphone portable peut également être un tag NFC dans le cas d'un fonctionnement en pair-à-pair. Un tag est composé de trois parties principales: la puce, l'antenne et l'autocollant de papier ou de vinyle. L'antenne sert à capter l'énergie radio émise par le téléphone au passage du tag NFC sur ce dernier et cette énergie permet le démarrage de la puce. Nous allons voir dans cette partie les différentes catégories de tag ainsi que les informations qu'ils contiennent.

1.3.1 Les types de tag

Le NFC Forum est un organisme mondial qui promeut le NFC et qui a développé le *NFC Forum Tag Type Specification* pour assurer l'interopérabilité entre les tags NFC, les dispositifs et les services. Il existe donc plusieurs types de tag possédants différentes caractéristiques.

1.3.1.1 Les tags NFC Forum Type 1

Ce type de tag est basé sur la norme ISO/EIC14443A (norme définissant un mécanisme de communication et impose la taille des objets qui communiquent). Il est possible de le lire, de réécrire dessus et de le verrouiller de manière permanente de façon à ce qu'il ne soit plus réinscriptible mais uniquement lu. La mémoire disponible est comprise entre 96 bytes et 2kbytes.

1.3.1.2 Les tags NFC Forum Type 2

Les tags de type 2 sont les plus communs et se procurent facilement sur internet sous différents noms (NTAG203, Mifare ULTRALIGHT, NTAG, BCM 512 etc.). Ce type de tag est également basé sur la norme ISO/IEC14443A et possède les mêmes caractéristiques que les type 1 avec une taille comprises entre 48 Bytes et 2kBytes.

Les tags Mifare Classic possèdent les mêmes caractéristiques que les tags de type 2 mais n'appartiennent à aucun NFC Forum Type.

1.3.1.3 Les tags NFC Forum Type 3

Ce type de tag est rarement utilisé et repose sur la norme Japanese Industrial Standard (JIS) X6319-4 également connue sous le nom de Felica (norme dérivée de l'ISO 18092, pour la communication en mode passif.). Les tags sont préconfigurés lors de leur fabrication pour être soit réinscriptibles soit lus uniquement. La mémoire disponible est limitée à 1MByte.

1.3.1.4 Les tags NFC Forum Type 4

Les tags de type 4 sont le plus souvent utilisés dans le paiement sans contact et les cartes de voyage et sont basés sur la norme ISO 14443[1]. Ils sont configurés à leur fabrication pour être réinscriptibles ou lus uniquement et sont souvent dotés d'une capacité de chiffrement.

1.3.2 Les technologies des tags

Les tags implémentent différentes technologies indépendamment développées en fonction de leur type. Il existe dix technologies :

- IsoDep : fournit l'accès aux propriétés et opérations de la norme ISO 14443-4[2].
- MifareClassic : fournit l'accès aux propriétés et opérations du MIFARE Classic[3].
- MifareUltralight : fournit l'accès aux propriétés et opérations du MIFARE Ultralight.
- Ndef : fournit l'accès au contenu du NDEF ainsi qu'à ses opérations.
- NdedFormatable : fournit l'accès au formatage d'un tag, c'est-à-dire qu'il permet de formater un tag pour qu'il contienne du NDEF.
- NfcA : fournit l'accès aux propriétés et opérations de la norme ISO 14443A.
- NfcB : fournit l'accès aux propriétés et opérations de la norme ISO 14443B.
- NfcBarcode : fournit l'accès aux tags contenant uniquement un code barre.
- NfcF : fournit l'accès aux propriétés et opérations de la norme JIS 6319-4[4].
- NfcV : fournit l'accès aux propriétés et opérations de la norme ISO 15693[5].

1.3.3 Le contenu des tags

Le NDEF est le nom donné aux messages contenus dans les tags. C'est un format binaire structuré en *messages* contenants différents *record* comme le montre la figure ci-dessous.

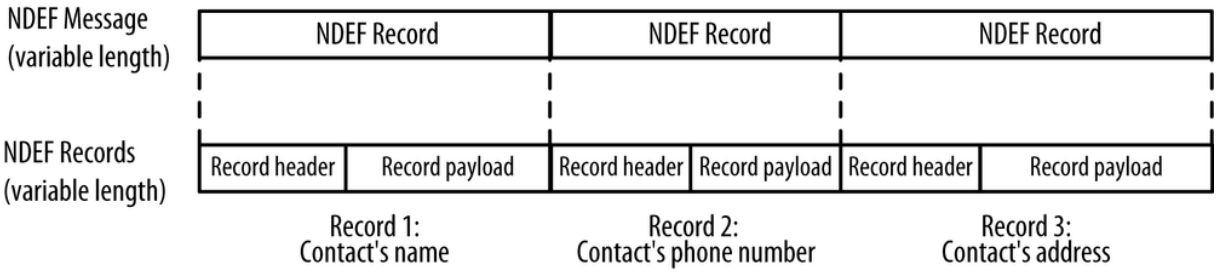


Figure 4 : Structure du NDEF message

Un record est caractérisé par quatre valeurs : son Type Name Format (TNF[1]), son type, son identifiant et ses données (Payload). Il possède par ailleurs des métadonnées décrivant la manière dont les données doivent être interprétées. L'une d'entre elles est le TNF. Ce dernier est une valeur numérique indiquant comment interpréter le champ *Type*. Il existe huit valeurs possibles pour le TNF :

- 0 : *Empty*
  Correspond à un *record* vide qui ne contient aucune donnée
- 1 : *Well-known*
  Correspond aux types existants dans les spécifications du NFC Forum RTD (Record Type Definition).
- 2 : MIME media-type
  Correspond à un type de média définie dans le RFC[2] 2046.
- 3 :Absolute URI[3]
  Correspond à un URI définie dans le RFC 3986.
- 4 :External
  Indique que le champ *type* contient un type externe à ceux définies par le NFC Forum.
- 5 : Unknown
  Indique que le *payload* est inconnu.
- 6 : Unchanged
  Utilisé pour les enregistrements tronqués. Le *type length* doit être null.
- 7 : Reserved
  Réservé par le NFC Forum pour un usage futur.

Le *Payload Type* décrit plus précisément le contenu du *Payload*. Il peut être de type *MIME type*, URI ou *External type*. La figure 5 présente la structure d'un *NdefMessage* plus en détail.

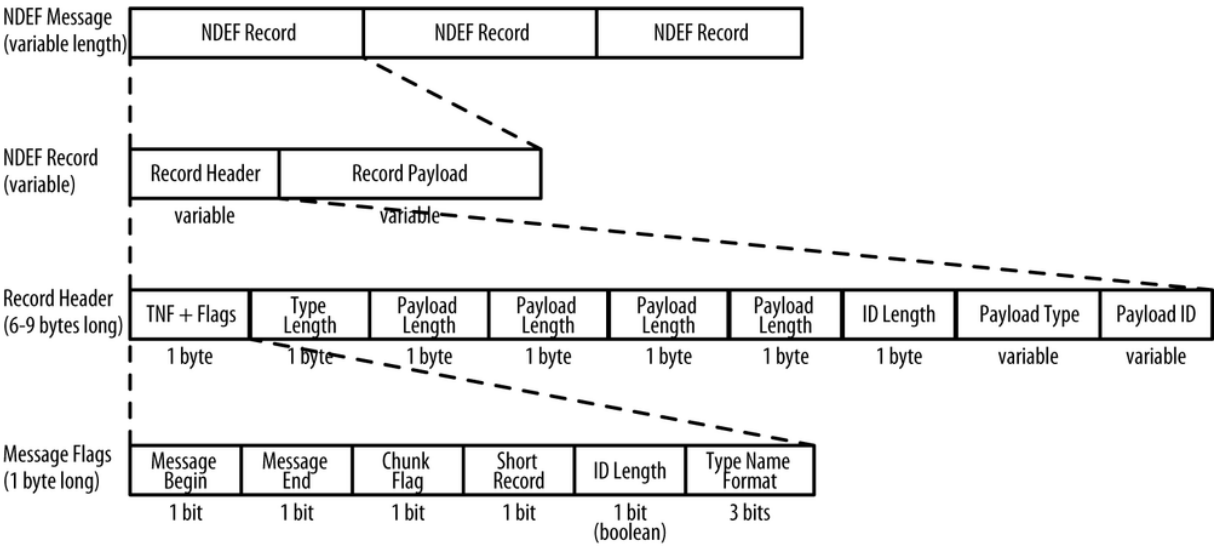


Figure 5 : Structure détaillée du NDEF message

1.3.4 Détection d'un tag

La figure 6, extraite du site http://developer.android.com, montre comment le système réagit lorsqu'il détecte un tag.

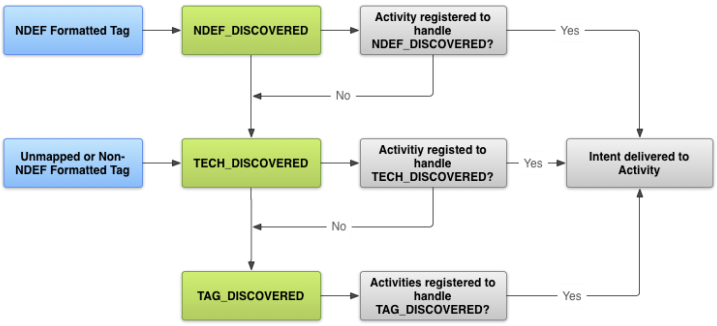


Figure 6 : Mécanisme de traitement d'un tag NFC

Il existe trois grandes étapes pour le traitement d'un tag:

- Etape 1: Android essaye de lancer l'activité filtrant l'ACTION_NDEF_DISCOVERED. Les activités qui filtrent cette action ont la plus haute priorité. S'il existe plusieurs activités disponibles sur le mobile alors l'utilisateur a la possibilité de choisir une des activités grâce à une fenêtre de sélection. Si le système détecte un tag avec le support NDEF, cela déclenche un *Intent* ACTION_NDEF_DISCOVERED. Un *Intent* est un message permettant la communication entre les composants applicatifs. Il permet notamment de transmettre des informations d'une activité à une autre.

- Etape 2: Si le système ne trouve aucune activité pour l'*Intent* précédent, alors il essaie de démarrer des activités avec la deuxième plus haute priorité. Ces activités filtrent l'ACTION_TECH_DISCOVERED. Cet *Intent* se lance quand un tag est détecté et que des activités sont enregistrées pour la technologie spécifique présente dans le tag. Ces différentes technologies ont été explicitées dans une partie antérieure.

- Etape 3: Si aucune activité n'est trouvée dans le processus ci-dessus, l'*Intent* ACTION_TAG_DISCOVERED est lancée.

1.4 Les objectifs du projet

La finalité du projet était de réaliser une application Android capable de lire les informations contenues dans le plus grand nombre de tag NFC ainsi que d'écrire sur ce dernier. L'application devait ainsi pouvoir écrire et lire le contenu des tags programmables (NFC Forum type 1 et 2) et des tags non –programmables (NFC Forum type 3 et 4). L'interface de l'application se devait d'être ergonomique et simple d'utilisation.

2. La mise en œuvre du projet

2.1 Les outils utilisés

La réalisation du projet a nécessité la prise en main d'outils tels que l'environnement de développement Android Studio ou l'API Android NFC qui sont présentés ci-après. Les tags et les smartphones utilisés au cours du projet pour les tests de l'application sont décrits par la suite.

2.1.1 Android Studio

Android Studio est l'environnement de développement intégré (EDI[1]) officiel pour développer des applications Android. Il est basé sur IntelliJ IDEA[2] et a été développé par Google.

Afin de nous aider à prendre en main l'IDE, M. Philippe Lacomme nous a proposé un tutoriel qui nous a guidés dans la construction d'une application Android codée sous Android Studio.

2.1.2 L'API Android NFC

L'API d'Android propose un accès à la fonctionnalité NFC d'un téléphone grâce au package android.nfc

Ce dernier permet aux applications de lire et écrire un message NDEF dans les tags NFC.

Le package contient 4 classes :

- *NfcManager* : Gestionnaire de haut niveau, utilisé pour obtenir le *NfcAdapter* de l'appareil
- *NfcAdapter* : Représente l'adaptateur NFC de l'appareil, qui est le point d'entrée pour effectuer des opérations NFC.
- *NdefMessage* : Représente un message de données NDEF, qui est le format standard dans lequel les "enregistrements" porteurs de données sont transmis entre les périphériques et les tags
- *NdefRecord* : Représente un enregistrement, qui est livré dans un *NdefMessage* et décrit le type de données partagées et transporte les données lui-même.

Enfin, le package android.ncf.tech propose des classes qui donnent accès aux caractéristiques de la technologie d'un tag, qui varie selon le type de tag scanné. Un tag scanné peut supporter plusieurs technologies.

Nous allons maintenant présenter les tags utilisés au cours du projet.

2.1.3 Les tags

Pour réaliser ce projet, des tags ont été commandés sur le site http://rapidnfc.com/ et mis à notre disposition, afin de pouvoir tester l'application. Ces tags sont :

- Tags de Type 1 : BCM512
- Tags de Type 2 : NTAG213, NTAG216, NTAG210, Ultralight, Ultralight C
- Mifare Classic 1K

2.1.4 Les smartphones

L'application créée au cours de ce projet peut être lancée sur un émulateur (fonctionnalité fournie par SDK Android) ou sur un téléphone réel. Le test de l'application sur l'émulateur a montré les limites de l'émulateur qui été très long à se lancer.

Aussi, nous avons utilisé 2 téléphones réels pour tester notre application, ceux-ci étant doté de la version 5.1 d'Android et bénéficiant de la fonctionnalité NFC.

Ces deux smartphones sont un Samsung Galaxy J5 appartenant à l'un des étudiants ayant réalisé ce projet et un Samsung Galaxy S4, appartenant au LIMOS[1] qui nous a été prêté pour la durée du projet.

Il s'est avéré que les tags de type Mifare Classic ne pouvaient être reconnus par le Galaxy S4 alors qu'ils l'étaient par le Galaxy J5.

Ce type de tag a été créé par NXP qui a été l'un des principaux fabricants de matériel NFC. Le NFC Forum a été créé pour concevoir des protocoles NFC afin que tout matériel et toute puce NFC (Tag) qui adhère à ce protocole soient compatibles. NXP a créé le Mifare Classic 1K spécifiquement pour être compatible avec son matériel et pas nécessairement à adhérer aux protocoles. Or le Galaxy S4 est équipé par un autre fabricant (Broadom), à cause de cela, les puces qui adhèrent aux protocoles du Forum NFC sont complètement compatibles alors que les puces Mifare Classic 1K ne le sont pas.

Figure 7 : Samsung Galaxy S4

Figure 8 : Samsung Galaxy J5

2.2 Planification du projet

Après s'être approprié le sujet et avoir fait quelques recherches sur le NFC ainsi que sur les applications Android existantes (NFC Tools, TagWritter..), nous avons entamé le projet en commençant par planifier les différentes étapes de sa réalisation.

2.2.1 Diagramme de GANTT

Nous avons tout d'abord réalisé un diagramme de GANTT prévisionnel afin de structurer l'avancement du projet. Celui-ci se trouve en annexe du rapport (cf. Annexe 1)

2.2.2 Architecture de l'application

Nous avons ensuite réfléchis à la façon dont nous voulions que l'application soit agencée. L'application devait avoir deux fonctionnalités : celle de lire et d'écrire sur un tag donc nous avons décidé de mettre deux boutons qui permettraient de rediriger les utilisateurs vers ces fonctions. Le bouton 'LIRE' redirigerait vers une page permettant de scanner le tag et d'afficher les informations qu'il contient et le bouton 'ECRIRE' redirigerait vers une page qui contient elle-même trois boutons permettant de choisir le type de données que l'on va inscrire sur le tag. De plus, sur la page d'accueil de l'application, un bouton du menu permettra d'afficher des informations sur le projet.

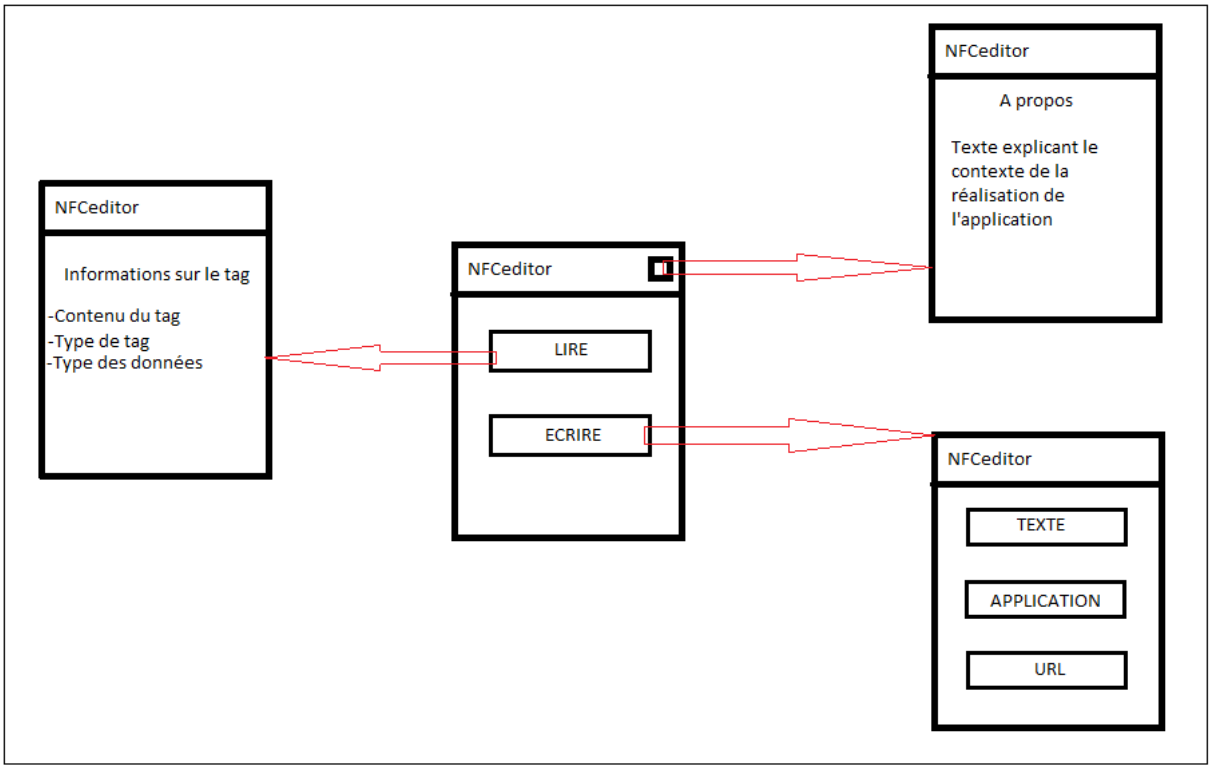


Figure 9 : Architecture de l'application

3. Réalisation

Dans cette partie nous allons présenter les réalisations effectuées sur le projet. Après avoir décrit la structure générale de l'application, nous détaillerons la classe NfcBaseActivity qui est la classe mère de gestion de la fonctionnalité NFC. Enfin, nous expliquerons comment nous avons réussi à lire, écrire et effacer des tags NFC.

3.1 Structure de l'application

L'application se compose de 8 Activités, celles-ci sont déclarées dans un fichier AndroidManifest.xml et leur structure visuelle est décrite dans des fichiers *Layout* (fichier de mise en page).

3.1.1 Layout

Chaque activité possède son propre *Layout* (fichier de mise en page). Celui-ci permet de décrire la structure visuelle de l'application.

Chaque fichier de mise en page doit contenir exactement un élément racine, qui doit être un objet *View* ou *ViewGroup*. Une fois que l'élément racine est défini, on peut ajouter des objets de mise en page supplémentaires ou des widgets (composants d'interface graphique : bouton, zone de texte, etc) comme éléments enfants pour construire progressivement une hiérarchie d'affichage qui définit la mise en page. Par exemple, la figure 10 décrit la mise en page XML de l'activité d'entrée (*MainActivity*) de notre application qui utilise un *LinearLayout* vertical pour tenir un *LinearLayout* horizontal (contenant deux boutons) et un bouton:

```
<LinearLayout
    android:id="@+id/linearLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:orientation="vertical">
    <LinearLayout
        android:id="@+id/layoutRAndW"
        android:layout_centerInParent="true">
        <Button
            android:id="@+id/readButton"
            android:drawableTop="@drawable/read_ic"
            android:text="@string/readButton"/>
        <Button
            android:id="@+id/writeButton"
            android:layout_toRightOf="@+id/readButton"
            android:drawableTop="@drawable/write_ic"
            android:text="@string/writeButton"/>
    </LinearLayout>
    <Button
        android:id="@+id/eraseButton"
        android:drawableTop="@drawable/erase_ic"
        android:text="Erase" />
</LinearLayout>
```

Figure 10 : Layout de l'activity MainActivity

3.1.2 Activities

Le point d'entrée de l'application est l'Activity *MainActivity*, elle offre la possibilité à l'utilisateur de choisir entre l'écriture, la lecture et l'effacement d'un tag.

Si on choisit de lire un tag, l'application ouvre une nouvelle activité *ReadActivity* qui permet à l'utilisateur de lire le contenu d'un tag après que celui-ci ait été scanné.

Il y a 3 types d'écriture possible : du texte, une application ou une URL, l'activité *ChoiceWriteActivity* permet à l'utilisateur de choisir ce qu'il veut écrire sur son tag au travers de 3 boutons.

L'appui sur l'un des trois boutons renvoie sur une nouvelle activité (*WriteTextActivity* ou *WriteAppActivity* ou *WriteURLActivity*) qui propose à l'utilisateur de taper du texte ou une URL ou de choisir une application parmi celles disponibles sur le téléphone.

Enfin, l'utilisateur peut choisir d'effacer le contenu du tag après validation de son choix.

3.1.3 Fichier Manifest

Pour pouvoir créer une application Android capable de gérer la fonctionnalité NFC du téléphone, le fichier AndroidManifest.xml doit être correctement préparé.

3.1.3.1 Permission d'utiliser NFC

Pour pouvoir utiliser la fonctionnalité NFC du téléphone, la permission suivante doit être déclarée :

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

3.1.3.2 Version minimale d'Android SDK

Le support de l'écriture, de la lecture et l'utilisation au premier plan du NDEF sont disponibles depuis l'API 10 d'Android (dernier niveau d'API Android : 23).

```
<uses-sdk android:minSdkVersion="10" />
```

3.1.3.3 Fonctionnalité NFC

Dans le but de de faire apparaître notre application sur le PlayStore uniquement aux appareils équipés du matériel NFC, nous avons dû déclarer cette fonctionnalité :

```
uses-feature android:name="android.hardware.nfc"
             android:required="true" />
```

3.1.3.4 Déclaration d'un Intent

Pour être sûr que l'application se lance à la detection d'un tag NDEF, l'intent suivant doit être utilisé.

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="text/plain"/>
    <data android:mimeType="application/[package name]"/>
</intent-filter>
```

3.1.4 Déclaration des technologies

Afin de pouvoir traiter les différentes technologies NFC, celles-ci doivent être déclarées dans un fichier .xml situé dans les ressources du projet :

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.Ndef</tech>
        <tech>android.nfc.tech.IsoDep</tech>
        <tech>android.nfc.tech.NfcA</tech>
        <tech>android.nfc.tech.NfcB</tech>
        <tech>android.nfc.tech.NfcF</tech>
        <tech>android.nfc.tech.NfcV</tech>
    </tech-list>
</resources>
```

3.1.5 La classe NfcBaseActivity

Cette classe est essentielle à l'application, c'est elle qui permet de gérer toutes les fonctionnalités NFC. Pour cela, les librairies les plus importantes qu'on a utilisées sont :

```
import android.nfc.NdefMessage;
import android.nfc.NdefRecord;
import android.nfc.NfcAdapter;
import android.nfc.Tag;
```

La première activité à être appelée quand un tag est détecté est celle qui est au premier plan et qui a activé le système de répartition de premier plan (foregroundDispatcher). L'activation est réalisée en récupérant l'adaptateur de dispositif NFC, et par conséquent la bibliothèque « `android.nfc.NfcAdapter;` » est nécessaire.

```
mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
```

Ensuite, on crée un *PendingIntent* (permet à une activité A de demander à une activité X de lancer l'activité B) que l'on passe à la fonction *enableForegroundDispatch()*. Ceci est fait dans une méthode *enableWriteMode()* qui est appelée dans les méthodes *onResume()*

```
protected void enableWriteMode() {
        IntentFilter mndef = new IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED);
        IntentFilter mtech = new IntentFilter(NfcAdapter.ACTION_TECH_DISCOVERED);
        IntentFilter mtag = new IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED);
        mIntentFiltersArray = new IntentFilter[]{mndef,mtech,mtag};

        mNfcAdapter.enableForegroundDispatch(this, mPendingIntent, mIntentFiltersArray, null);
    }
```

Figure 11 : Fonction enableWriteMode

A contrario, il ne faut pas oublier de désactiver le système de répartition quand l'application (ou l'activité) est en cours d'exécution en arrière-plan. C'est le rôle de la méthode *disableWriteMode()* qui est appelée dans la méthode *onPause()*.

```
protected void disableWriteMode() {
    mNfcAdapter.disableForegroundDispatch(this);
}
```

Figure 12 : Fonction disableWriteMode

3.2 La lecture d'un tag

Un clic sur le bouton 'READ' lance l'activité *ReadActivity* qui propose à l'utilisateur de scanner son tag comme le montre la figure suivante :

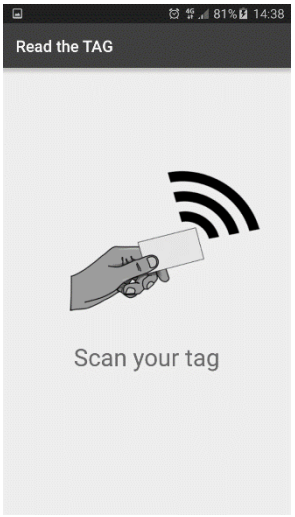


Figure 13 : Lecture d'un tag

Une fois le tag détecté, la méthode *handleIntent()* est appelée :

```
private void handleIntent(Intent intent) {
    String action = intent.getAction();
    if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
        Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
        technology = tag.getTechList();
        Ndef ndef = Ndef.get(tag);
        size = ndef.getMaxSize();
        type = intent.getType();
        ID_tag = tag.getId();

        if (MIME_TEXT_PLAIN.equals(type) || MIME_APP.equals(type)) {
            new NdefReaderTask().execute(tag);
        } else {
            Log.d(TAG, "Wrong mime type: " + type);
        }
    } else if (NfcAdapter.ACTION_TECH_DISCOVERED.equals(action)) {
        Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
        technology = tag.getTechList();
        Ndef ndef = Ndef.get(tag);
        size = ndef.getMaxSize();
        type = intent.getType();
        ID_tag = tag.getId();
        String searchedTech = Ndef.class.getName();

        for (String tech : technology) {
            if (searchedTech.equals(tech)) {
                new NdefReaderTask().execute(tag);
                break;
            }
        }
    }
    else if (intent.getType() != null &&
                    intent.getType().equals("application/" + getPackageName())) {
        Parcelable[] rawMsgs = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
        NdefRecord relayRecord = ((NdefMessage) rawMsgs[0]).getRecords()[0];
        String nfcData = new String(relayRecord.getPayload());
        Toast.makeText(this, nfcData, Toast.LENGTH_SHORT).show();
    }
}
```

Figure 14 : Fonction handleIntent

Elle permet de lire les données du tag telles que la technologie, la taille, le type, l'identifiant. Ensuite si le contenu du tag est de type texte ou s'il s'agit d'une application, on lance la lecture du contenu dans un autre Thread dans la fonction *doInBackground* qui fait appel à la méthode *readText()*.

```
protected String doInBackground(Tag... params) {
    Tag tag = params[0];
    Ndef ndef = Ndef.get(tag);
    if (ndef == null) {
        // NDEF is not supported by this Tag.
        return null;
    }
    NdefMessage ndefMessage = ndef.getCachedNdefMessage();
    NdefRecord[] records = ndefMessage.getRecords();
    for (NdefRecord ndefRecord : records) {
        if (ndefRecord.getTnf() == NdefRecord.TNF_WELL_KNOWN &&
            Arrays.equals(ndefRecord.getType(), NdefRecord.RTD_TEXT)) {
            try {
                return readText(ndefRecord);
            } catch (UnsupportedEncodingException e) {
                Log.e(TAG, "Unsupported Encoding", e);
            }
        } else if (ndefRecord.getTnf() == NdefRecord.TNF_WELL_KNOWN &&
            Arrays.equals(ndefRecord.getType(), NdefRecord.RTD_URI)) {
            try {
                return readText(ndefRecord);
            } catch (UnsupportedEncodingException e) {
                Log.e(TAG, "Unsupported Encoding", e);
            }
        }
    }
    return null;
}
```

Figure 15 : Fonction doInBackground

La méthode *readText()* prend en parameter le *NdefRecord* dont on cherche à connaître les informations. Pour cela, on utilise la méthode *getPayload()* :

```
private String readText(NdefRecord record) throws UnsupportedEncodingException {
    byte[] payload = record.getPayload();
    String textEncoding = ((payload[0] & 128) == 0) ? new String("UTF-8") : "UTF-16";
    int languageCodeLength = payload[0] & 0063;
    return new String(payload, languageCodeLength + 1,
                      payload.length - languageCodeLength - 1, textEncoding);
}
```

Figure 16 : Fonction readText

Enfin, une fois la tâche effectuée par la fonction *doInBackground*, les données sont affichées à l'écran via la fonction *onPostExecute*.

```
protected void onPostExecute(String result) {

    StringBuilder sb = new StringBuilder();
    if (result != null) {
        for (int i = 0; i < technology.length; i++) {
            sb.append(", " + technology[i].toString().split("\\.")[3]);
        }
        mContentTV.append("Read TAG Content: " + result);
        mTypeTV.append("Type of data: " + type);
        mTechTV.append("Technology: " + sb.toString());
        mSizeTV.append("Size : " + size + " Bytes");
        mIdTV.append("ID : " + bytesToHex(ID_tag));


    } else {
        mContentTV.append("TAG vide");
    }
    mScanIV.setVisibility(View.GONE);
    mScanTV.setVisibility(View.GONE);
    mContentLayout.setVisibility(View.VISIBLE);
}
```

Figure 17 : Fonction onPostExecute

3.3 L'écriture d'un tag

3.3.1 Choix du type d'écriture

Lorsque l'utilisateur appuie sur le bouton 'WRITE', l'Activity *ChoiceWriteActivity* se lance pour permettre à l'utilisateur de choisir le type de donnée qu'il veut inscrire sur le tag. L'Activity se présente comme suit :

```
public class ChoiceWriteActivity extends AppCompatActivity {

    private Button mWtriteTextButton;
    private Button mWriteAppButton;
    private Button mWriteURLButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_choice_write);

        mWtriteTextButton = (Button)findViewById(R.id.writeTextButton);
        mWtriteTextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (v.getId() == R.id.writeTextButton) {
                    Intent intentWriteText = new Intent(getApplicationContext(), WriteTextActivity.class);
                    startActivity(intentWriteText);
                }
            }
        });
        mWriteAppButton = (Button)findViewById(R.id.writeAppButton);
        mWriteAppButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (v.getId() == R.id.writeAppButton) {
                    Intent intentWriteApp = new Intent(getApplicationContext(), WriteAppActivity.class);
                    startActivity(intentWriteApp);
                }
            }
        });
        mWriteURLButton = (Button)findViewById(R.id.writeURLButton);
        mWriteURLButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (v.getId() == R.id.writeURLButton) {
                    Intent intentWriteURL = new Intent(getApplicationContext(), WriteURLActivity.class);
                    startActivity(intentWriteURL);
                }
            }
        });
    }
}
```

Figure 18 : Activity ChoiceWriteActivity

Le choix se fait à l'aide de trois boutons :

- Si l'utilisateur appuie sur le bouton 'TEXT', l'Activity WriteTextActivity se lance et permet d'écrire des données de type TEXT/PLAIN.
- Si l'utilisateur appuie sur le bouton 'APPLICATION', l'Activity WriteAppActivity se lance et permet d'insérer les données d'une application de type MIME/MEDIA.
- Si l'utilisateur appuie sur le bouton 'URL', l'Activity WriteURLActivity se lance et permet d'écrire une adresse internet de type URI.

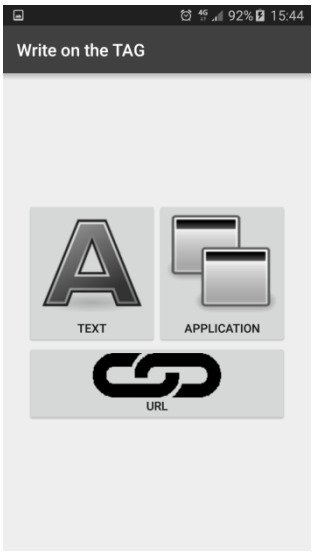


Figure 19 : Choix du type d'écriture dans l'application NFCeditor

3.3.2 Ecriture d'un texte et d'un URI

L'affichage de l'Activity *WriteTextActivity* se présente comme suit :

Figure 20 : Affichage de l'Activity WriteTextActivity

Dans la fonction *onCreate* de l'Activity on met en place le bouton 'WRITE' qui permet à l'utilisateur d'écrire le message sur le tag. La fonction e*nableWriteMode* (héritée de la classe mère NfcBaseActivity – cf. 3.1.5) crée un *IntentFilter* permettant d'informer Android que notre application est activée pour fonctionner avec les tags NFC (*NfcAdapter.ACTION_TAG_DISCOVERED*).

A chaque fois qu'un tag est détecté la méthode *onNewIntent* est appelée. Cette fonction appelle elle-même la fonction *handleIntent* qui permet de récupérer le tag contenu dans l'*Intent*.

```
private void handleIntent(Intent intent) {
    if(NfcAdapter.ACTION_TAG_DISCOVERED.equals(intent.getAction())){
        Tag tag;
        tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
        if (tag == null)
            displayMessage("TAG NULL");
        else
            writeTag(this, tag);
    }
}
```

Figure 21 : Fonction handleIntent

Si le tag a été correctement récupéré, on peut créer le NDEF message et l'écrire sur le tag grâce à la fonction *writeTag*. Dans cette méthode on commence tout d'abord par récupérer le message à écrire sur le tag et convertir ce message sous le format standard d'un *NdefMessage*, c'est-à-dire en le mettant dans un *NdefRecord*.

```
EditText tonEdit = (EditText)findViewById(R.id.edit_message);
String NFCMessage = tonEdit.getText().toString();
NdefRecord textRecord = NdefRecord.createTextRecord("", NFCMessage);

NdefMessage message = new NdefMessage(new NdefRecord[] { textRecord });
```

Figure 22 : Transformation d'un texte en NdefMessage

Avant de pouvoir écrire le message sur le tag, on vérifie s'il peut contenir du NDEF, si ce n'est pas le cas, il faut vérifier si le tag possède la technologie *NdefFormatable* permettant de formater un tag de façon à ce qu'il contienne du NDEF. S'il est *NdefFormatable*, on le formate et on écrit les données sur le tag. Dans le cas où le tag est déjà NdefFormatable, on vérifie s'il est réinscriptible et s'il contient assez de place pour stocker les données que l'on veut lui inscrire. Des messages d'erreur s'affichent si ces caractéristiques ne sont pas respectées.

```
try {
    // see if tag is already NDEF formatted
    Ndef ndef = Ndef.get(tag);
    if (ndef != null) {
        ndef.connect();
        if (!ndef.isWritable()) {
            displayMessage("Read-only tag.");
            return false;
        }
        // work out how much space we need for the data
        int size = message.toByteArray().length;
        if (ndef.getMaxSize() < size) {
            displayMessage("Tag doesn't have enough free space.");
            return false;
```

Figure 23 : Vérification de la taille du message ainsi que de sa capacité à être réinscriptible

L'étape suivante permet de vérifier si le tag contient déjà des données et affiche un message si c'est le cas. On récupère tout d'abord le *NdefMessage* contenu dans le tag et on y extirpe son *NdefRecord*. Si la métadonnée TNF est égale à 0, cela signifie que le tag est vide (ne contient aucune données), on peut donc écrire directement le message sur le tag grâce à la fonction *writeNdefMessage* de la librairie NFC. Si le TNF est différent de 0, cela signifie que des données sont déjà présentes dans le tag, un message d'avertissement est donc lancé pour prévenir l'utilisateur et lui indiquer qu'il faut effacer le tag avant de pouvoir réécrire dessus. On utilise pour cela un *alertDialog* comme indiqué sur la figure 10. Si l'utilisateur appuie sur le bouton 'YES' cela signifie qu'il accepte d'effacer le contenu actuel du tag et il est redirigé vers l'Activity *EraseActivity*, sinon, il retourne sur l'Activity *WriteTextActivity*.

```
//Initialisation de l'AlertDialog
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
//Affichage du message d'erreur
alertDialogBuilder.setMessage("Tag is already written. Do you want to erase?")
        .setCancelable(false)
        .setPositiveButton("YES",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        Intent erase_intent = new Intent(getApplicationContext(), EraseActivity.class);
                        startActivity(erase_intent);
                    }
                });
alertDialogBuilder.setNegativeButton("Cancel",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int id) { //Si clique
                dialog.cancel();//Annulation, retour sur l'Activity MainActivity
            }
        });
AlertDialog alert = alertDialogBuilder.create();
alert.show();
```

Figure 24 : Mise en place de l'alertDialog

Nous avons mis en place un *progressBar* afin que l'utilisateur visualise la progression de l'écriture du tag. Une fois le bouton 'WRITE' appuyé, le *progressBar* affiche le message « Touch and hold tag against phone to write. Wait please » pour indiquer à l'utilisateur de rapprocher le tag du téléphone afin que ce dernier le détecte. On initialise le *progressBar* comme suit :

```
progress.setMessage("Touch and hold tag against phone to write. \n Wait please ");
progress.setProgressStyle(ProgressDialog.STYLE_SPINNER);
progress.setIndeterminate(true);
progress.show();
```

Figure 25 : Initialisation du progressBar

Une fois que le tag a été correctement écrit, on désactive le *progressBar* grâce à la fonction *dismiss*.

L'écriture d'un message du type URI se fait de façon similaire mis à part le fait que lorsque l'on récupère le message à écrire, on transforme ce dernier en un URI comme suit :

```
EditText tonEdit = (EditText) findViewById(R.id.edit_message);
String NFC_URL = tonEdit.getText().toString();
Uri uri = Uri.parse(NFC_URL);
NdefRecord recordNFC = NdefRecord.createUri(uri);
NdefMessage message = new NdefMessage(recordNFC);
```

3.3.3 Ecriture d'une application

Au lancement de l'Activity *WriteAppActivity*, la liste des applications installées sur le téléphone s'affiche comme le montre la figure 26 ci-après :

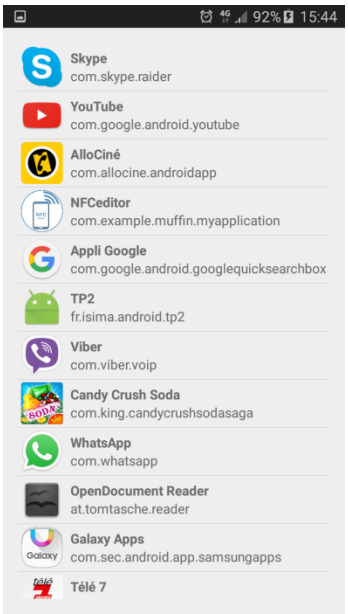


Figure 26 : Liste des applications installées

Pour ce faire, on utilise la classe *PackageManager* d'Android. La fonction *getPackageManager()* de cette classe permet de récupérer les informations des applications actuellement installées sur le mobile. Le fichier .xml de l'Activity contient une *ListView* permettant d'afficher cette liste et la mise en forme de chaque ligne est définie par un *layout* nommé *List_row.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <ImageView
        android:id="@+id/app_icon"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:padding="3dp"
        android:scaleType="centerCrop" />

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center_vertical"
        android:orientation="vertical"
        android:paddingLeft="5dp" >

        <TextView
            android:id="@+id/app_name"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center_vertical"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/app_paackage"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:gravity="center_vertical" />
    </LinearLayout>

</LinearLayout>
```

Figure 27 : Fichier List_row.xml

L'*ImageView* permet d'afficher l'icône de l'application, la *TextView* 'app_name' affiche le nom de l'application et le *TextView* 'app_package' affiche le package[1] de l'application. L'initialisation des attributs et la mise en place de la liste se fait dans la classe *LoadApplications* qui étends la classe *AsyncTask* afin de réaliser cette tâche de manière asynchrone car la récupération des données des applications est longue à exécuter. C'est la méthode *doInBackground* qui permet d'exécuter la tâche asynchrone. Dans cette méthode, représentée dans la figure 28 page suivante, nous stockons la liste des applications dans l'*ArrayList* 'applist' sous le format définit par le layout *List_row.xml*. Nous affichons ensuite cette liste grâce à la fonction *setListAdapter* de la méthode *onPostExecute*.

```
@Override
protected Void doInBackground(Void... params) {
    applist = checkForLaunchIntent(packageManager.getInstalledApplications(PackageManager.GET_META_DATA));
    listadaptor = new ApplicationAdapter(WriteAppActivity.this,
            R.layout.list_row, applist);
    return null;
}
```

Figure 28 : Méthode doInBackground

L'utilisateur choisit l'application qu'il veut écrire sur le tag en cliquant sur un item de la liste. La fonction *onListItemClick* (Figure 29), présentée sur la figure 14, permet de récupérer la position de l'item dans la liste, d'inviter l'utilisateur à approcher son tag du téléphone et de lancer la fonction *handleIntent* pour détecter le tag et lancer l'écriture du tag avec la fonction *writeTag* (Figure 30).

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {

    app = applist.get(position);
    try {
        displayMessage("Touch and hold tag against phone to write.");
        enableWriteMode();
        handleIntent(getIntent(), app);
    } catch (ActivityNotFoundException e) {
        Toast.makeText(WriteAppActivity.this, e.getMessage(),
                Toast.LENGTH_LONG).show();
    } catch (Exception e) {
        Toast.makeText(WriteAppActivity.this, e.getMessage(),
                Toast.LENGTH_LONG).show();
    }
}
```

Figure 29 : Méthode onListItemClick

Cette fonction permet de créer le *NdefMessage* avec deux *NdefRecord* :

- L'appRecord permet d'enregistrer le nom de package de l'application afin que le Play Store se lance si l'application n'est pas installée sur le mobile
- le relayRecord contient le type de donnée que l'on écrit sur le tag, ici MIME/MEDIA ainsi que l'application à lancer au passage du tag.

```
private boolean writeTag(Context context, Tag tag) {

    // record to launch Play Store if app is not installed
    Intent intent1 = packageManager.getLaunchIntentForPackage(app.packageName);
    String appName = intent1.getPackage();

    NdefRecord appRecord = NdefRecord.createApplicationRecord(intent1.getPackage());
    NdefRecord relayRecord = new NdefRecord(NdefRecord.TNF_MIME_MEDIA,
            new String("application/" + context.getPackageName())
                    .getBytes(Charset.forName("US-ASCII")),
            null, appName.getBytes());
    NdefMessage message = new NdefMessage(new NdefRecord[] {relayRecord, appRecord});
```

Figure 30 : Fonction writeTag

On effectue ensuite les mêmes vérifications que celles de l'écriture d'un texte puis on écrit les données sur le tag grâce à la fonction *writeNdefMessage* de la librairie NFC.

3.4 L'effacement d'un tag

A l'appui sur le bouton 'ERASE de l'Activity MainActivity, l'activity *EraseActivity* lance directement un *alertDialog* qui demande à l'utilisateur une confirmation pour effacer le tag. Si ce dernier appuie sur le bouton 'YES' alors un *progressBar* s'affiche de façon à ce que l'utilisateur rapproche le tag du téléphone pour être détecté et un message *null* est écrit sur le tag avec le même principe que l'écriture d'un texte sur un tag. Si l'utilisateur appuie sur le bouton 'CANCEL' de l'*alertDialog* alors il est redirigé vers l'Activity *MainActivity*. L'*alertDialog* s'affiche de la manière suivante :

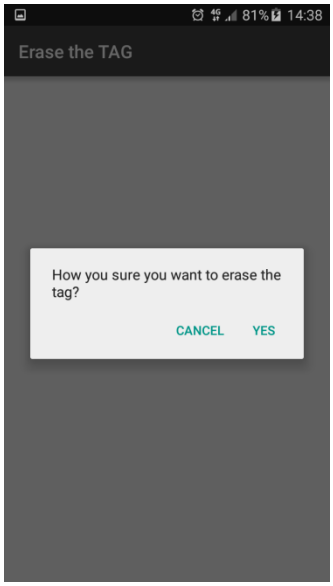


Figure 31 : Affichage de l'alertDialog

4. Les perspectives d'évolutions

Nous avons rencontré des difficultés dans l'établissement de la fonction 'Lecture' de l'application. Il reste donc des modifications à apporter à cette dernière pour un meilleur fonctionnement.

4.1 Lecture des applications

La partie 'Lecture' de NFCeditor permet actuellement de lire les informations contenues sur les tags ayant un *Payload* de type text /plain ou URI. Cela signifie que l'on peut lire le tag uniquement s'il contient du texte ou une URL en données. Or, un tag peut également avoir un *Payload* de type MIME/MEDIA, c'est-à-dire qu'il peut contenir des applications. Ainsi, pour exploiter pleinement les fonctionnalités du tag, il faudrait améliorer l'application de façon à ce qu'elle lise ces informations.

4.2 Lecture d'un tag dans l'ensemble de l'application

Dans la version actuelle de NFCeditor, la lecture d'un tag ne peut se faire que si l'on est dans l'activité de lecture. Une amélioration serait de configurer l'application de façon à ce que lorsque l'utilisateur scanne le tag, l'activité de lecture se lance, et ce, quel que soit l'endroit où l'on se situe dans l'application. Par exemple, si l'utilisateur scanne le tag en étant sur l'activité d'accueil, cela déclencherait automatiquement l'activité de lecture.

4.3 Lecture des tags non programmables

Les tags que nous avons utilisés pour le projet sont pour la plupart de type *NFC Forum type 2* donc ils sont réinscriptibles et contiennent du NDEF. Les tags de types *NFC Forum type 1* possèdent les mêmes caractéristiques donc il est également possible de lire leur contenu avec NFCeditor (par exemple le tag BCM512 que nous avons utilisé). Il serait intéressant de pouvoir lire les informations des tags de type *NFC Forum type 3 et 4 (*qui ne possèdent pas de NDEF et dont les données sont cryptés) en affichant uniquement le type du tag accompagné d'un message indiquant que le tag ne peut pas être lu.

4.4 Gestion de l'*alertDialog*

Dans les Activity *WriteTextActivity* et *WriteURLActivity,* un *alertDialog* se déclenche si le tag contient déjà des données et permet de rediriger l'utilisateur vers l'Activity *EraseActivity* pour effacer le contenu du tag s'il appuie sur le bouton 'YES'. Une amélioration serait de réécrire directement sur le tag si l'utilisateur appuie sur le bouton 'YES' sans le rediriger vers l'Activity *EraseActivity*.

5. Bilan et conclusion

Le but de ce projet était de réaliser une application Android pour la lecture et l'écriture de tags NFC. L'un des points clés était que l'application soit capable d'interagir avec le plus grand nombre de type de tags NFC possible. Par ailleurs, celle-ci devait rester simple d'utilisation et proposer une interface ergonomique.

L'objectif principal du projet a été rempli puisque l'application NFCeditor permet de lire et écrire sur des tags. La fonctionnalité d'effacement d'un tag, qui n'avait pas été spécifiée au départ, a également été implémentée pour offrir plus de possibilités aux utilisateurs. En effet, une fois la lecture/écriture maîtrisée, l'effacement a été rapidement inclus dans les fonctionnalités. De plus, l'application répond à la contrainte d'ergonomie imposée par le cahier des charges car les fonctionnalités sont accessibles via une interface claire et agréable.

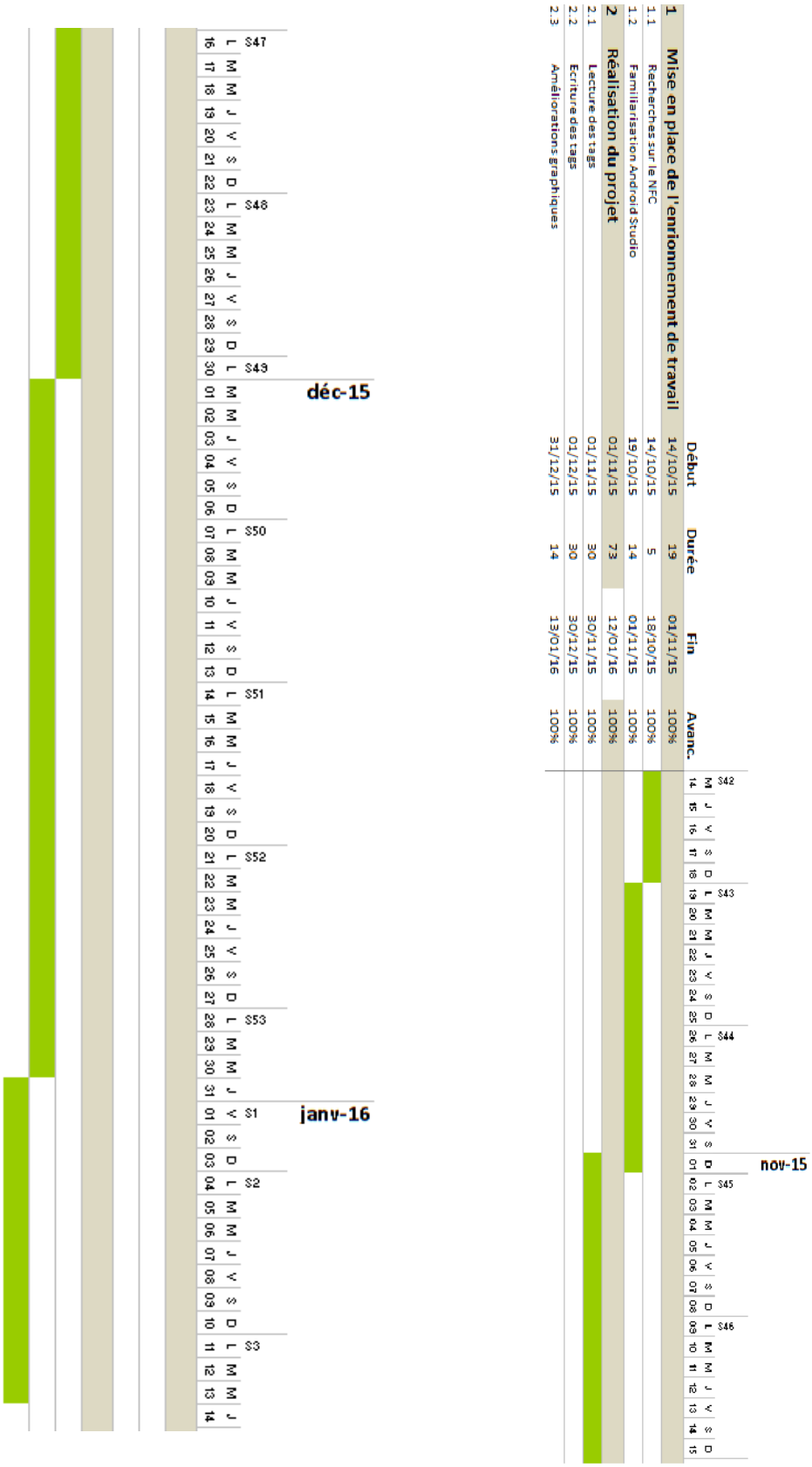
Toutefois, certaines améliorations pourraient être apportées à notre travail pour parfaire l'application. En effet, la lecture des informations de tag non programmables ainsi que la lecture des données de type MIME/MEDIA restent à implémenter. Par ailleurs, la mise en place de l'écriture sur un tag a été l'une des difficultés majeures rencontrées au cours de ce projet. Celle-ci a engendré des retards par rapport au diagramme de Gantt prévisionnel comme le montre le diagramme de Gantt réel (cf. Annexe 2).

Finalement, sur demande de notre tuteur, nous avons implanté dans notre application un message d'information destiné aux utilisateurs. Celui-ci, affiché uniquement lors du premier démarrage, informe l'utilisateur qu'il assume les risques liés à la qualité et aux effets de l'application étant donné le contexte de création de celle-ci. Ce message a pour but de nous protéger légalement quant à l'utilisation de notre application.

Ce projet a été très enrichissant tant sur le plan intellectuel que sur le plan humain. En effet, nous avons appris à maîtriser Android Studio qui est l'environnement de développement officiel pour les applications Android. Nous avons également pu nous familiariser avec la technologie NFC qui fait partie des technologies innovantes et émergentes et avons appris à la manipuler au travers des librairies mises à disposition par le SDK Android. De plus, ce projet de deuxième année nous a permis d'appréhender le travail en binôme d'une manière différente dans la mesure où le travail demandé était beaucoup plus conséquent que pour le projet de première année ou les travaux pratiques. Notre binôme a bénéficié d'une bonne cohésion et d'une bonne communication en interne et avec nos tuteurs.

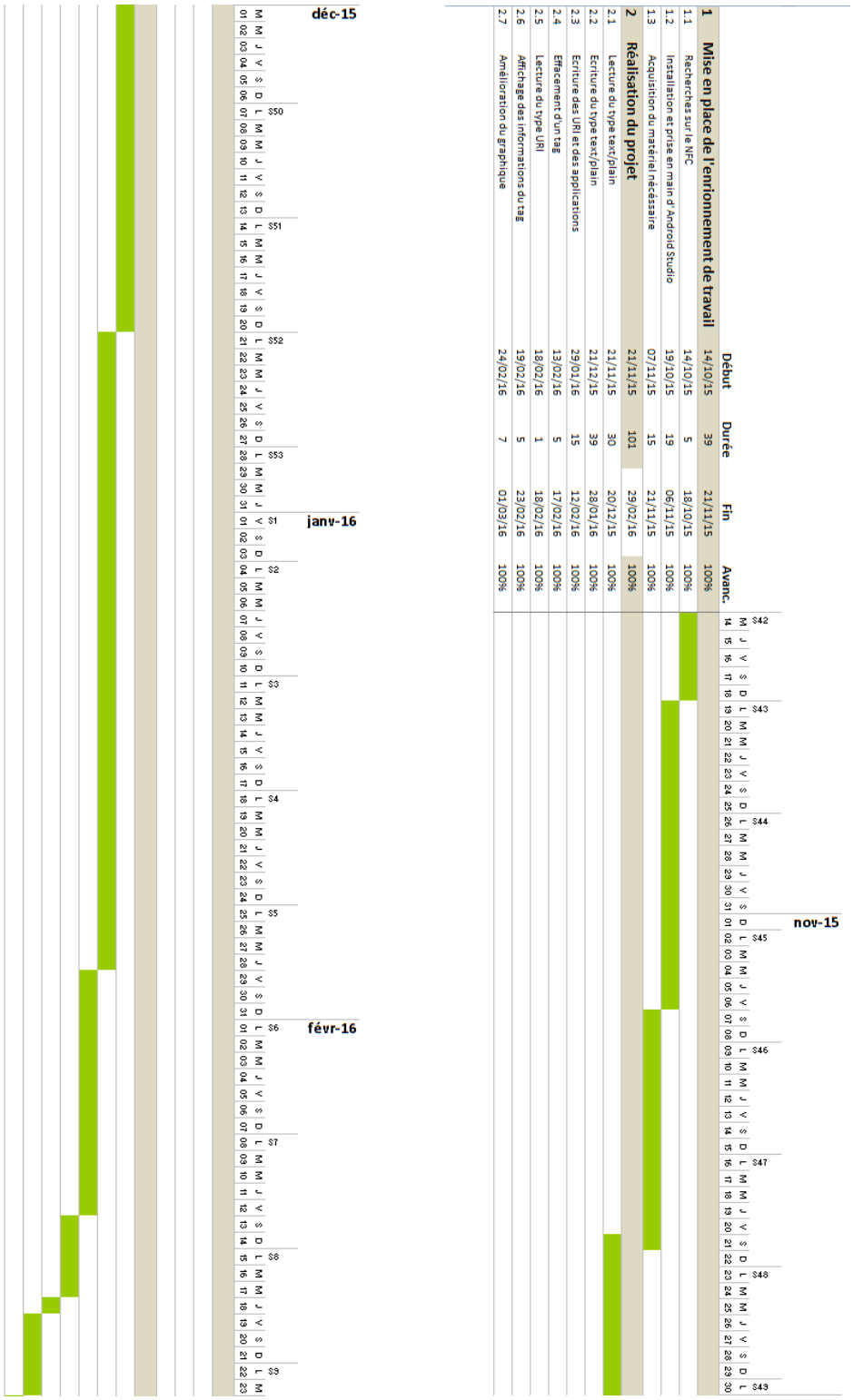
Enfin, l'application NFCeditor sera bientôt disponible sur le Google Play Store qui est la boutique en ligne de Google pour le système d'exploitation Android. Il s'agit de la seule boutique en ligne officielle pour se procurer des applications Android. Cette démarche va permettre aux utilisateurs de la technologie NFC de profiter de notre application.

Annexe 1



		Début	Durée	Fin	Avanc.
1	**Mise en place de l'environnement de travail**	14/10/15	19	01/11/15	100%
1.1	Recherches sur le NFC	14/10/15	5	18/10/15	100%
1.2	Familiarisation Android Studio	19/10/15	14	01/11/15	100%
2	**Réalisation du projet**	01/11/15	73	12/01/16	100%
2.1	Lecture des tags	01/11/15	30	30/11/15	100%
2.2	Ecriture des tags	01/12/15	30	30/12/15	100%
2.3	Améliorations graphiques	31/12/15	14	13/01/16	100%

Annexe 2


Début
Durée
Fin
Avanc.
1 Mise en place de l'environnement de travail 14/10/15 39 21/11/15 100%
1.1 Recherches sur le NFC 14/10/15 5 18/10/15 100%
1.2 Installation et prise en main d'Android Studio 19/10/15 19 06/11/15 100%
1.3 Acquisition du matériel nécéssaire 07/11/15 15 21/11/15 100%
2 Réalisation du projet 21/11/15 101 29/02/16 100%
2.1 Lecture du type text/plain 21/11/15 30 20/12/15 100%
2.2 Ecriture du type text/plain 21/12/15 39 28/01/16 100%
2.3 Ecriture des URI et des applications 29/01/16 15 12/02/16 100%
2.4 Effacement d'un tag 13/02/16 5 17/02/16 100%
2.5 Lecture du type URI 18/02/16 1 18/02/16 100%
2.6 Affichage des informations du tag 19/02/16 5 23/02/16 100%
2.7 Amélioration du graphique 24/02/16 7 01/03/16 100%
nov-15
déc-15
janv-16
févr-16