SFL2 - TRAÇABILITE RFID

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dossier technique du projet

Session 2019

PARTIE PERSONNELLE

Pierre GORGE



**Table des matières**

[I. Situation dans le projet 1](#_Toc9431893)

[A. Rappel des tâches de l’étudiant 1](#_Toc9431894)

[B. Déroulement du projet 1](#_Toc9431895)

[C. Contraintes de développement 2](#_Toc9431896)

[a. Contraintes logicielles 2](#_Toc9431897)

[b. Contraintes matérielles 3](#_Toc9431898)

[II. Conception et mise en œuvre 5](#_Toc9431899)

[A. Fonctionnement du NFC sous Android Studio 5](#_Toc9431900)

[B. Détection du NFC par le téléphone 6](#_Toc9431901)

[a. Création de la variable « nfcAdapter » 6](#_Toc9431902)

[b. Vérification de l’utilisation du NFC 7](#_Toc9431903)

[c. Le téléphone ne possède pas de NFC 8](#_Toc9431904)

[C. Lecture du tag RFID par NFC 8](#_Toc9431905)

[a. Détection du tag 8](#_Toc9431906)

[b. Lecture du tag 11](#_Toc9431907)

[D. Création de l’interface graphique 14](#_Toc9431908)

[E. Détection par QR Code 19](#_Toc9431909)

[a. Interface QR Code 19](#_Toc9431910)

[b. Utilisation de la caméra par l’application 20](#_Toc9431911)

# Situation dans le projet

## Rappel des tâches de l’étudiant

Dans ce projet de traçabilité RFID, j’avais pour objectif, avec mon collègue Julio César Gomez Reyes, de créer une application Android, utilisable sur la plupart des téléphones Android, ayant pour but l’emprunt d’objets via le scan d’un tag RFID. Nous avons pour cela dû nous approprier les librairies existantes pour l’utilisation de la technologie NFC sur les smartphones Android.

## Déroulement du projet

Dans un premier temps, nous avons réfléchi aux différents moyens de mettre en œuvre le projet. Nous avons très vite déduit que de faire deux applications séparées étaient certes plus facile à mettre en place mais moins cohérent et pratique pour le service demandé. Ainsi, nous avons décidé de ne créer qu’une seule application et de travailler main dans la main avec mon collègue Julio César Gomez Reyes.

Nous nous sommes donc concentrés ensemble avec Julio, sur la conception de l’interface de l’application, pour ensuite me pencher sur l’implémentation d’une lecture d’un tag par le smartphone. J’ai aussi travaillé sur la création d’une partie QR Code pour une adaptabilité aux téléphones démunit de NFC.

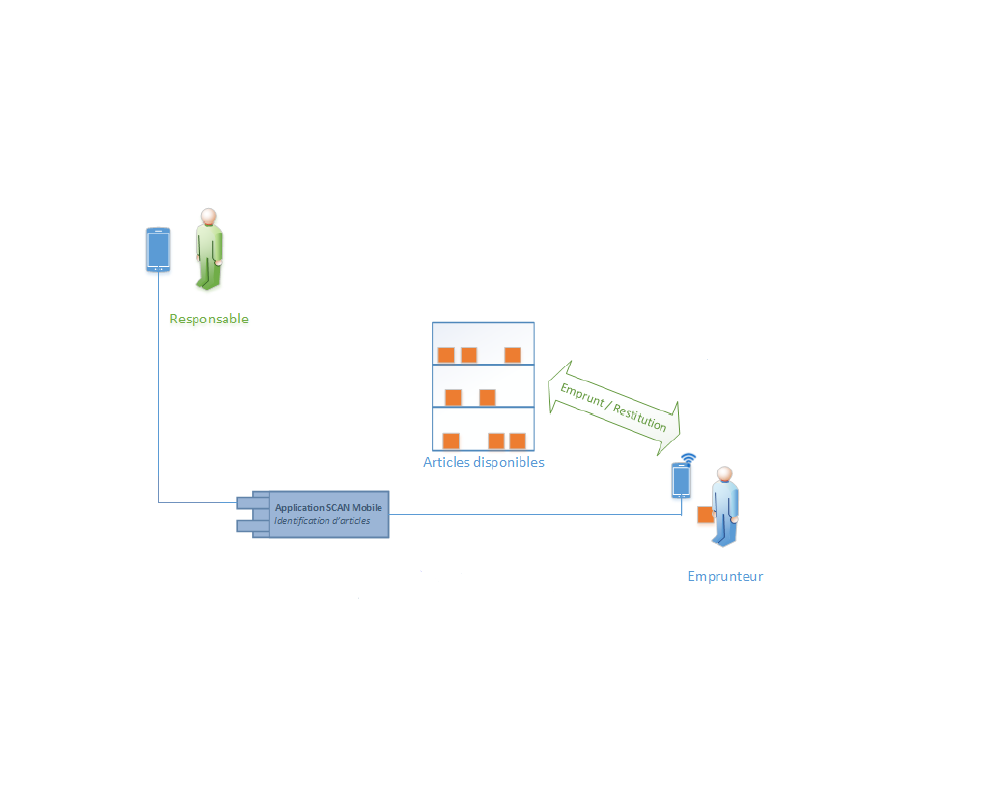


Schéma de la partie concernant l’étudiant 3

Pour cela, j’ai dû me plonger dans les librairies utilisant la technologie NFC et dans une utilisation précise et complexe du logiciel Android Studio.

|  |
| --- |
| Une image contenant clipart  Description générée automatiquementAndroid Studio est un environnement de développement pour développer des applications mobiles Android. Il est basé sur IntelliJ IDEA et utilise le moteur de production Gradle. Il peut être téléchargé sous les systèmes d'exploitation Windows, macOS et Linux3. |

La librairie NFC est très alambiquée et la documentation est étonnamment peu précise. Il m’a fallu beaucoup de temps pour comprendre comment l’utiliser et pour commencer à la maitriser. En effet, au bout de deux mois, j’ai réussi à implémenter une lecture de NFC par le téléphone. Cette tâche m’a demandé beaucoup de temps car très complexe à mettre en place. J’ai aussi travaillé sur la solution du QR Code mais qui n’a pour le moment pas encore aboutie. Une fois la lecture du tag réussi, je me suis concentré sur la création de ma partie de l’interface. Elle a évolué dans le temps avec les différents ajouts fait sur l’application au cours du projet.

La réalisation de l’application s’est déroulée en trois grandes étapes (spécifications, analyse, conception) qui suivent le diagramme de Gantt présent en annexe.

## Contraintes de développement

### Contraintes logicielles

Le logiciel Android Studio était une totale découverte et une véritable contrainte sur le projet. Nous maîtrisons le langage Java, ce qu’utilise Android Studio, cependant l’architecture du logiciel étant très particulière et propre à lui, il a fallu un temps assez important pour trouver ses marques et savoir comment l’utiliser de manière efficace.

Les librairies NFC ont été une difficulté, au même niveau que le logiciel qui les utilise. La documentation n’est pas très exhaustive, ni même explicite. Il m’a fallu beaucoup de temps, de travail et de patience pour trouver comment les utiliser correctement et les implémenter.

### Contraintes matérielles

Les téléphones ont aussi été sources de contraintes. En effet, ils sont développés sous Android mais les constructeurs mobiles l’ont tous modifié pour avoir leur propre version et certaines de ces versions créer quelques problèmes avec l’application. Par exemple, il faut impérativement désactiver le débogage USB sur le téléphone de Maxime pour que l’application fonctionne correctement.

Une image contenant texte, capture d’écran

Description générée automatiquementNotre projet nous amène à utiliser des tags RFID. Il s’agit d’une norme différente du NFC, nous devons donc utiliser un certain type de tag RFID. Il s’agit d’une légère contrainte car, il existe en effet plusieurs types de tag RFID. On peut les différencier via leur fréquence. Par exemple, une carte bancaire est une carte RFID au même titre qu’un badge pour le télépéage. La différence se fait dans la fréquence qu’ils utilisent. **La carte bancaire possède une bande de fréquence de 13.56MHz (ce que notre application va pouvoir détecter) et un badge de télépéage va plutôt utiliser une fréquence comprise entre 2450-5800 MHz.**

Il est possible de détecter un tag RFID avec du NFC si ce tag est à la même fréquence que le NFC.

|  |  |  |
| --- | --- | --- |
| **Spécificités/Technologie** | **NFC** | **RFID HF** |
| **Bande passante** | **13,56 MHz** | **3 à 30MHz** |
| **Information** | **Fréquence habituelle :**  **13.56MHz.**  **Distance de lecture :**  **1 à 10 cm.**  **Normes: NFCIP-1 (ISO/CEI 18092), ISO/CEI 14443-1 à ISO/CEI 14443-4, NDEF.** | **Fréquence habituelle :**  **13.56MHz.**  **Distance de lecture :**  **1 à 10 cm.**  **Normes : ISO 15693, ISO/IEC 18092, ISO 14443** |

*Tableau comparatif des normes NFC et RFID HF*

On utilise donc des cartes RFID de fréquence 13.56MHz dit de **« Haute Fréquence ».**

# Conception et mise en œuvre

## Fonctionnement du NFC sous Android Studio

Sur Android Studio, toutes les classes et méthodes sont répertoriés dans des librairies. Ce sont des classes qui contiennent toutes les classes et méthodes sur une fonctionnalité donnée. Il nous faut beaucoup de librairies différentes pour notre application. Notamment, pour la connexion à la base de données, pour l’API Google Maps et pour le NFC.

Exemples de librairies utilisées pour le NFC

On remarque que l’on utilise un « import » pour ajouter les librairies sur notre projet. En effet, les librairies sont stockées sur Android Studio mais elles ne sont pas toutes automatiquement implémentées sur le projet que l’on créer, cela permet d’éviter de surcharger le projet et de le rendre plus lent.

|  |
| --- |
| *Une image contenant intérieur  Description générée automatiquementNote :* Android Studio est développé par Google mais reste un logiciel libre, il est donc possible de créer de librairies soit même et de les rendre disponible sur internet. Ainsi, on peut se retrouver avec des besoins pour certaines fonctionnalités ne se trouvant pas sur le logiciel nativement (Même certaines fonctionnalités Android tel que Google Maps ne se retrouve pas nativement sur le logiciel). A ce moment-là, on se dirige vers un dossier du projet appelé « build.gradle », auquel on ajoute une ligne « Implementation » suivi d’un lien qui permet à l’IDE (Environnement de Développement Intégré) de télécharger la librairie et ainsi permettre au développeur de l’utiliser.  Ajout de librairies « externes » |

L’implémentation du NFC se déroule en plusieurs parties et se trouve dans la classe « NFCScan » créer spécialement pour gérer tous les éléments se trouvant sur la page du scan du tag RFID.

## Détection du NFC par le téléphone

### Création de la variable « nfcAdapter »

Avant toute chose, le téléphone doit détecter si le NFC est activé. Si l’utilisateur oublie de l’activer il faut que le téléphone puisse prévenir et amener l’utilisateur à l’activer.

Pour cela, après avoir initialisé les variables, on commence par créer une méthode « onCreate » où l’on va donner une valeur à tous les boutons et les fonctions du téléphone à activer.

Une image contenant capture d’écran

Description générée automatiquement

*Note :* Les méthodes « onCreate » sont celles qui sont appelés en premier au lancement de l’application. Les variables sont créées et pourront alors être utilisées par l’application. C’est très intéressant (voir nécessaire) à utiliser car cela permet au système de recréer les variables si jamais on retourne sur le menu d’accueil de notre téléphone. Si cette classe n’est pas utilisé, les variables seraient créées une seule fois, un retour au menu du téléphone sans fermer l’application reviendrait à faire disparaître les variables et y revenir ne les ferait pas réapparaitre. L’application planterai à ce moment-là.

On se concentre sur la variable « nfcAdapter » et on lui donne comme valeur la fonction NFC par défaut du téléphone :

Une fois que cette valeur est assignée, « nfcAdapter » possède la valeur du NFC sur le téléphone, c’est-à-dire que cette valeur peut savoir si le NFC est activé ou non.

### Vérification de l’utilisation du NFC

Une image contenant capture d’écran

Description générée automatiquementAinsi, on créer une méthode pour vérifier que le NFC est bien activé au moment où l’on veut scanner un tag RFID. Cette méthode s’appelle « onResume » et se présente de cette manière :

*Note :* Les méthodes « onResume » sont utiles quand les applications sont mises en pause en revenant à l’accueil par exemple. Au moment de revenir sur l’application, cette méthode sera appelée avant la méthode « onCreate » vu précédemment. Cela permet d’établir un ordre dans le relancement des méthodes et évite que certaines méthodes qui dépendent les unes des autres ne se lancent en même temps et créer des bugs.

Le « if » permet de vérifier si le NFC est activé. En arrivant sur la page de lecture du tag, si le NFC n’est pas activé (le « if ») alors on rentre dans cette boucle.

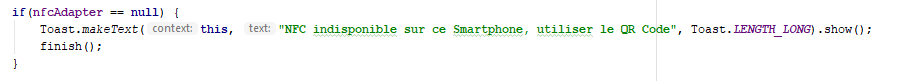
Une image contenant capture d’écran

Description générée automatiquementDans cette boucle, on créer un nouvel « AlertDialog.Builder » qui s’affichera sur la page sur laquelle on se trouve « (context : NFCScan.this) »

Ce morceau de texte nous permet de personnaliser notre message d’alerte. Ici on décide avec « builder.setTitle », « builder.setMessage » et « builder.setPositiveButton » respectivement, du titre, du message et d’un bouton « ok » sur le message d’alerte.

Une fois que l’on appuie sur le bouton « ok », on a fait en sorte de démarrer une nouvelle page qui est celle des paramètres NFC. Ce qui permet à l’utilisateur d’activer son NFC et de revenir sur l’application.

### Le téléphone ne possède pas de NFC

 Un morceau de code a aussi été rajouté pour bloquer l’accès à la page de lecture de tag via NFC si le téléphone ne possède pas de NFC. Certains téléphones un peu anciens ne possède effectivement pas cette technologie, ainsi, on a créé une condition dans laquelle « nfcAdapter » est nul :

La fonction « Toast.makeText » permet de créer une bulle de texte qui apparait pendant environ cinq seconde en bas de l’écran. Il s’agit d’une bulle de texte système utilisé pour afficher des messages qui ne nécessite pas de grande boite de dialogue ou d’alerte tel que celle utilisée pour l’activation du NFC. On ne peut pas interagir avec elles.

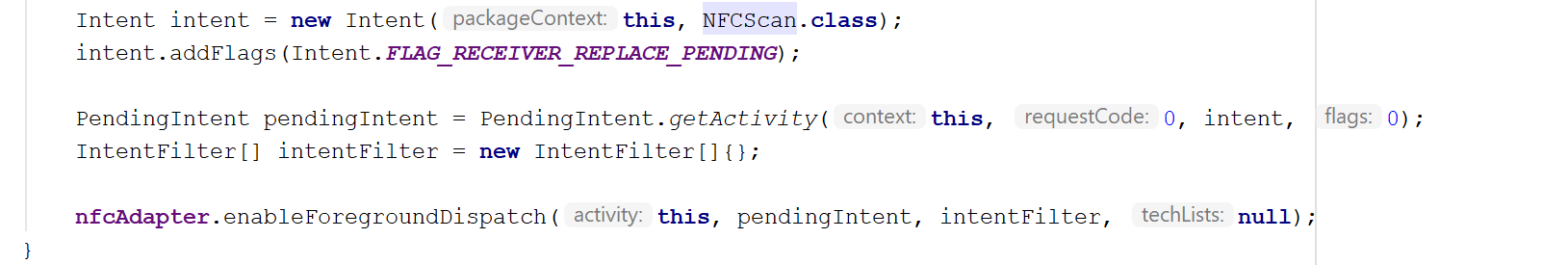
La méthode « finish() ; » quant à elle, termine la page ou « Activity » que l’on essaye d’atteindre. Ainsi, un téléphone qui ne possède pas de NFC ne pourra pas arriver sur la page de lecture du tag.

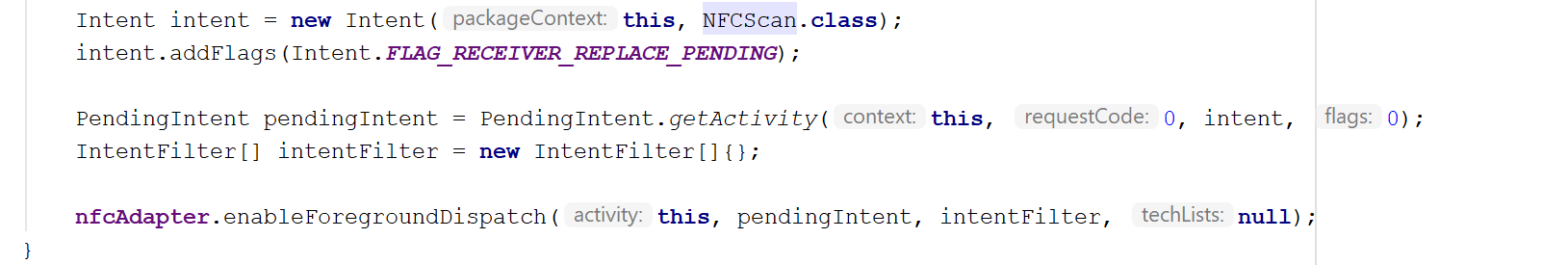
## Lecture du tag RFID par NFC

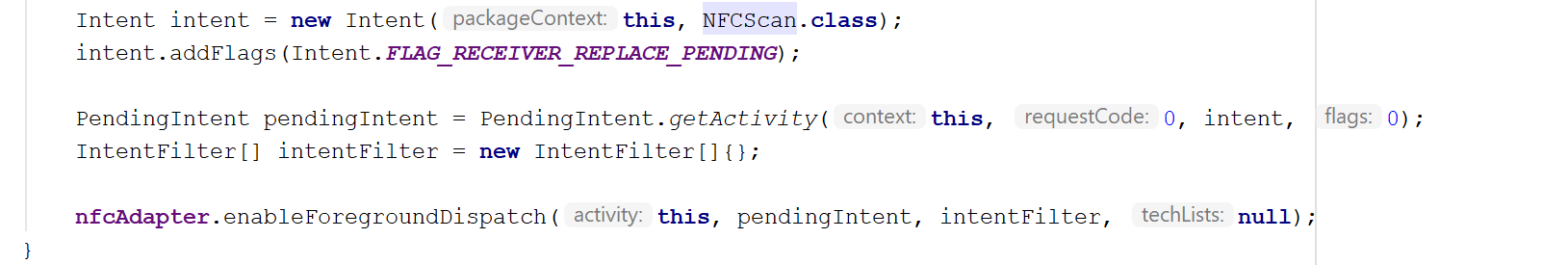
### Détection du tag

Cette partie est très complexe et fait appel à beaucoup de méthodes différentes. Nous allons détailler chaque méthode pour essayer de comprendre le fonctionnement d’une lecture de tag RFID.

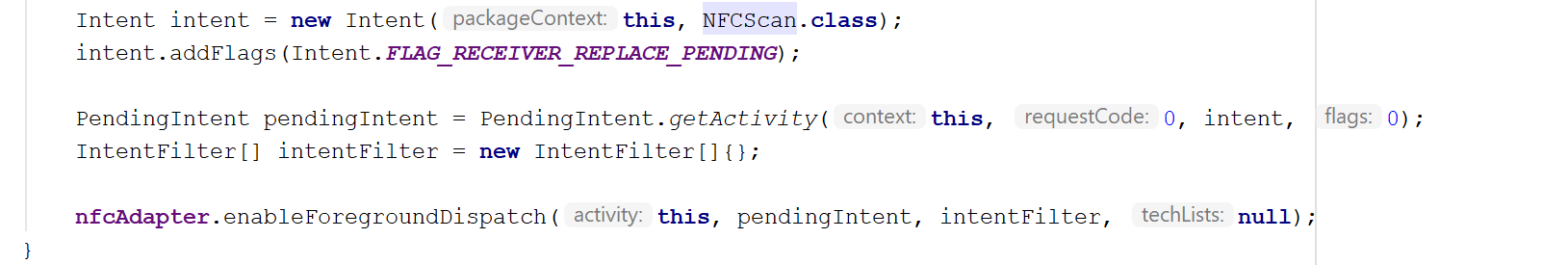
En premier lieu, il faut que l’application soit prioritaire sur la lecture du tag présenté. En effet, les différentes versions de l’OS Android des constructeurs mobiles possèdent déjà un lecteur de tag intégré. Il n’est cependant pas très poussé et ne permet que de détecter un tag, pas forcément d’en lire le contenu. Il reste malgré tout prioritaire et il faut faire en sorte que quand l’application est lancée, ce soit elle qui est la priorité sur la lecture d’un tag.

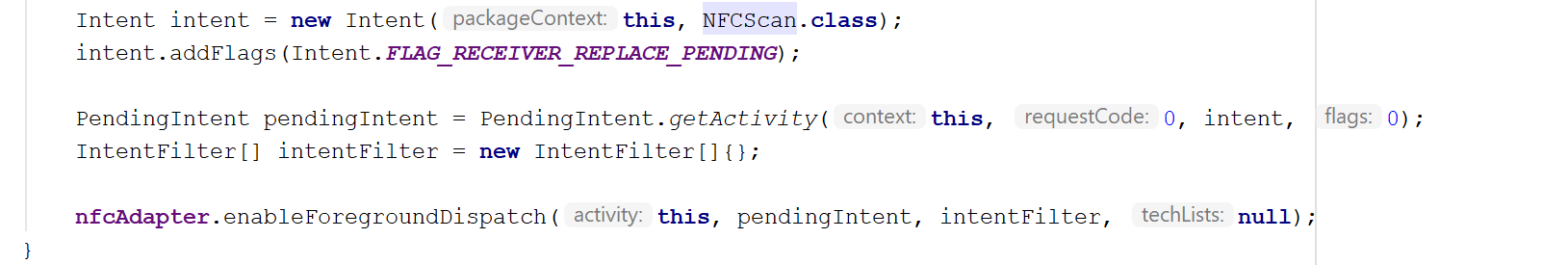
On utilise donc la méthode « enableForgroundDispatcher ». Cette méthode permet littéralement de prendre la priorité sur d’autres activités, notamment le lecteur de l’OS, et de faire en sorte que ce soit l’application dans lequel il est appelé qui récupère la lecture du tag.

La méthode en elle-même permet donc de lire le tag. C’est son seul rôle. Le reste se fait au niveau de « l’intent ». Cette méthode est fondamentale dans notre projet et notamment dans la lecture, et surtout l’affichage, de notre tag.

« L’intent » est une variable qui reçoit une information d’action et qui va réagir si on l’utilise. On initialise un « new Intent » sur cette classe précisément. Quand un tag va être scanné, l’information « un tag est détecté » est enregistré dans l’Intent.

Cette ligne permet le remplacement du tag si un autre est scanné. Cela évite de scanner deux tags à la suite et que les deux se mélangent. **Si un deuxième tag est scanné, alors il va remplacer le premier.**



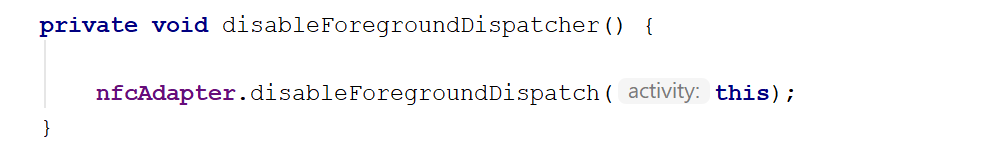
 Le « PendingIntent » n’est pas en lien direct avec l’intent en lui-même. Son but est de permettre l’exécution de l’Intent avec les permissions de notre application. En effet, notre application ne possède pas de lecteur à lui. Il utilise celui de l’OS en place, ici Android. Comme il ne fait pas parti de notre application, **il faut que ce lecteur puisse avoir accès à l’autorisation prioritaire de notre « enableForgroundDisplay ».** Cette ligne permet au lecteur du système d’y accéder.

Comme il peut exister plusieurs Intent de déclaré dans l’application, il faut les filtrer. L’ « IntentFilter » permet cela. On déclare quelles classes utilisent des Intents qui doivent être filtré.

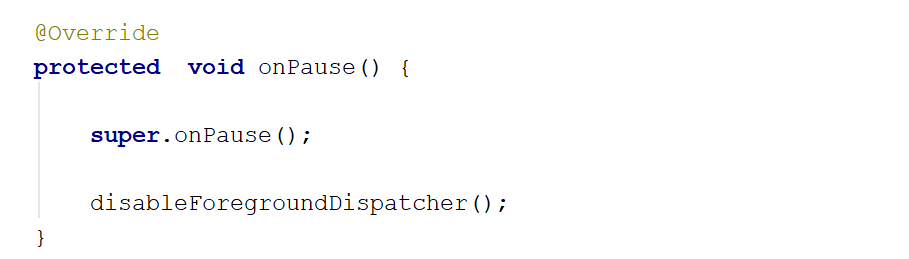
Déclaration des IntentFilter dans le Manifest

Une fois déclaré, seuls les Intent créer et appelé dans ces classes, ici « InterfaceNFCQR » et « NFCScan » pourront être récupérer avec un IntentFilter.

Finalement, on précise à la fin que la méthode dépend de « nfcAdapter » avec toutes ses caractéristiques qui lui permet de passer le « IntentFilter ».



On créer aussi une méthode pour détruire cette permission de lecture.

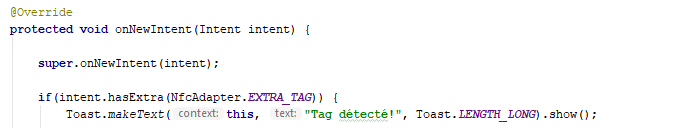


Cette méthode est appelée dans une méthode « onPause » qui se lance quand on sort de l’application pour retourner sur le menu principal de notre téléphone, sans pour autant fermer l’application. Quand c’est le cas, si on scan un tag RFID, **on ne veut pas que ce soit notre application qui prenne le relais, mais bien le lecteur du système. On désactive donc temporairement les droits prioritaires de notre application avec la méthode « disableForgroundDispatcher »**

### Lecture du tag

Maintenant que l’on a créer une méthode pour permettre l’utilisation du lecteur RFID de notre téléphone par l’application et que nous avons un Intent avec comme information « un tag est scanné », il nous reste à créer une méthode pour utiliser cet Intent.

On créer donc une nouvelle méthode appelée « onNewIntent ». L’intérêt de cette méthode est de conserver l’intent qu’elle a reçu précédemment. Si on retourne sur l’écran d’accueil, cette méthode va permettre de garder les informations du tag que l’on a scanné précédemment. Cela évite d’avoir à scanner de nouveau le tag en revenant sur l’application.



On créer une nouvelle condition. Si l’Intent reçu contient la valeur Extra, alors on rentre dans la boucle.

|  |
| --- |
| *Note :* Chaque Tag est reconnu et différencié par les lecteurs NFC. Pour Android, tout ce qui concerne la détection des tags se trouve dans cette librairie :  Une image contenant clipart  Description générée automatiquement  Elle a évoluée depuis un an, avant cette librairie s’écrivait « android.nfc.extra.Tag ». L’extra a disparu. On utilise pourtant bien la variable « hasExtra » pour vérifier qu’il s’agit bien du bon type de tag. Comme expliqué dans la partie « Contraintes matérielles », il existe différents types de tag RFID, cette librairie permet d’identifier ceux que l’on peut utiliser. |

On a aussi rajouté un Toast.makeText qui affiche un petit message indiquant que le tag à bien été détecté.



Ici, on utilise une variable « Parcelable » qui est l’équivalent du « Serialization » en Java. Le but de cette variable est de mettre dans un tableau les différentes informations du tag et sous forme de bytes. On y trouve les informations du tag, son ID ou le texte qui se trouve dedans, ainsi que le type du tag et le type de données présentes dessus.

Ce que l’on veut, c’est récupérer les données présentes sur le tag et de les mettre dans ce tableau. On précise donc avec « EXTRA\_NDEF\_MESSAGES ». En effet, parmi les tags RFID Haute Fréquence, on retrouve deux types distincts :

* Les tags NDEF
* Les tags NFCV

Les deux protocoles sont relativement similaires à la différence que le NFCV est protégé contrairement au NDEF. C’est notamment le protocole utilisé sur les cartes bancaire pour éviter toute copie des données trop facilement.

Nous n’avons pas besoin de ce type de protection, le système mise en place étant utilisé en interne et les risques très faibles. Nous avons surtout besoin qu’un smartphone puisse lire le message contenu dans le tag. L’utilisation du protocole NDEF est donc préconisé.

Maintenant que nous avons mis les bytes d’informations dans le tableau de « Parcelable », il nous faut les décrypter :

Une image contenant capture d’écran

Description générée automatiquement

On créer une nouvelle condition. Si, le tableau n’est pas vide et que sa longueur est supérieure à zéro (il s’agit d’une double vérification, le tableau avoir plusieurs cases vide, dans ce cas on reprécise que s’il y en a plusieurs mais avec des informations, on rentre dans la boucle)

On fait appel à deux nouvelles méthodes. Je vais vous présenter la première méthode « readTextFromMessage » dans ma partie personnelle. La méthode « TagID » sera traité par mon collègue Julio dans sa partie personnelle.

Une image contenant capture d’écran

Description générée automatiquement

Cette méthode est présentée sous forme de tableau. Les informations décryptées seront enregistrées dedans pour ensuite pouvoir être affichées. On utilise la variable « getRecords » pour récupérer les informations contenues dans le tableau de « Parcelable » précédemment créé.

Une image contenant capture d’écran

Description générée automatiquement

Une fois que le tableau est créé, on établit une condition, similaire à celle du « Parcelable » et on vérifie si des informations sont présentes dans le tableau.

A la ligne suivante, on précise que la **lecture du tableau doit commencer à la case zéro.**

Ensuite, on créer une nouvelle variable appelée « tagContent » qui est un string que l’on va utiliser par la suite pour afficher le contenu du tag qui aura été transformé en « UTF-8 ».

Une image contenant capture d’écran

Description générée automatiquementPour la transformation en UTF-8, on a créé une méthode à part, « getTextFromNdefRecord » avec comme paramètre la variable « ndefRecord » qui est la variable contenant le tableau avec les bytes d’informations à transformer :

Au début, on vérifie que le « tagContent » soit bien null pour éviter d’écrire par-dessus un tag déjà existant. Ensuite, c’est un bout de code qui se charge de transformer les bytes en UTF-8.

On a préféré utiliser un « Try » pour ce fragment de code car une erreur est vite arrivée sur ce type de conversion et il est préférable d’avoir un « catch » derrière pour vérifier ce qu’il se passe si jamais tel est le cas.

Finalement, on retourne la variable « tagContent » avec les informations traduites en UTF-8 d’enregistrer dedans.

Quand la variable est retournée vers la méthode « readTextFromMessage », elle est affichée sur une zone appelée « txtTag » que l’on a programmé en .xml.

Cela permet de faire la transition vers la suite de cette partie personnelle de notre rapport technique qui va traiter de la partie interface.

## Création de l’interface graphique

Pour un souci d’optimisation et d’organisation, nous avons découpé au maximum notre projet. On avait la possibilité de faire la totalité de notre code sur une seule et même page. Seulement, comme ce projet a été réaliser par mon collègue et moi-même, nous devions le rendre le plus claire possible.

Pour l’interface, nous avons créé environ une page par classe. Les voici :

* Interface Login :

Une image contenant capture d’écran

Description générée automatiquement

Nous n’aborderons pas cette interface dans cette partie du rapport technique. Pour cela, veuillez vous diriger vers la partie de Julio César Gomez Reyes.

Interface NFC/QR Code :

Une image contenant capture d’écran

Description générée automatiquement

Une image contenant capture d’écran

Description générée automatiquement

Cette interface est celle qui permet de choisir le mode de scan du tag RFID. Soit par NFC, soit par QR Code. Cette interface est composée de deux éléments interactifs. De deux boutons. Ce sont des « ImageButton », des boutons cachés en image. On peut voir que leur composition est très similaire. Il est possible de faire des modifications directement sur l’interface graphique, cela se retranscrit automatiquement en code sur le fichier .xml.

On notera la présence des **« layout\_constraint »**, ce sont des balises qui servent à verrouiller la position du bouton. On sélectionne un des côtés du bouton (cela marche pour tous les éléments, pas seulement les boutons), par exemple le côté haut, et on étire le curseur jusqu’à atteindre le haut de l’interface. Une fois cela fait, on observe une flèche qui commence du côté haut du bouton jusqu’au haut de l’interface. Cela se transcrit par **« app:layout\_constraintTop\_toTopOf= ‘parent’ »**, « parent » correspondant à l’extrémité de l’interface.

Concrètement, cela empêche le bouton de changer la distance qui lui a été attribué par rapport au haut de l’interface. En effet, il existe beaucoup de téléphones différents, notamment au niveau de la taille. Certains téléphones ont des tailles et ratios d’écrans différents. Nous prenons l’exemple très simple de nos téléphones. Mon collègue Julio possède un Samsung Galaxy S9 qui fait 5.8 pouces et possède un ratio de 18.5 :9. Tandis que je possède un Oneplus 6t qui a une taille de 6.41 pouces et un ratio de 19.5 :9.

Ainsi, si le constraint n’était pas mis, le bouton changerait de place sur chaque téléphone et changerai donc l’aspect de l’application. Risquant même de se faire superposer les boutons.

Une image contenant capture d’écran

Description générée automatiquementL’interface NFC :

Cette interface est intéressante car elle est composée de plus d’éléments que ce que l’on peut observer avec cette vue. En effet, **elle n’est pas composée de trois éléments** comme on pourrait le croire, mais bien de **cinq**.

Une image contenant capture d’écran

Description générée automatiquement

Une image contenant texte

Description générée automatiquement

Cette vision est ce que l’on pourrait comparer à une « vision infrarouge » dans les films. Elle nous permet de voir tous les éléments présents sur l’interface, même celle créer de façon à être invisible pour les utilisateurs.

On remarque donc qu’en plus des deux boutons et du texte en haut, **il y a deux autres éléments cachés.**

Cette partie ne mentionnera seulement la partie du haut avec le texte. Le reste ayant été créer par mon camarade Julio. Ce sera donc détaillé dans sa partie personnelle.

La partie en haut avec le texte à l’intérieur correspond à l’information textuelle du tag. Il s’agit du résultat de la méthode « readTextFromMessage ». La valeur a été enregistré dans la variable « txtTag » et on l’utilise ici pour afficher le contenu du tag.

## Détection par QR Code

### Interface QR Code

Après une étude approfondie du projet, il a été décidé que l’implémentation d’une option de scan par QR Code était pertinente. Leur utilisation est très répandue aujourd’hui et chaque téléphone possède une caméra et une application qui permet la lecture d’un QR Code.

Ma tâche étant quelque peu plus légère que celle de mon collègue Julio, il a été décidé que je serais en charge du développement de la partie QR Code.

Une image contenant capture d’écran

Description générée automatiquementL’interface est le premier élément qui a été produit, car très simple à créer.

Cette interface ne contient qu’un seul élément appelé « SurfaceView ». Il s’agit d’une partie dédiée pour poser une application tierce. Ici ce sera la caméra. Nous allons le détailler dans un instant.

Une image contenant intérieur

Description générée automatiquement

On remarque que le « SurfaceView » a une variable s’appelant « camerapreview » qui sert à transmettre des informations sur l’interface.

### Utilisation de la caméra par l’application

Une image contenant capture d’écran

Description générée automatiquementDe la même manière que pour le NFC, il faut commencer par créer les variables à chaque démarrage de la page. On créer donc une méthode « onCreate » :

On observe que le « SurfaceView » a bien eu « camerapreview » d’attribué ici.

On utilise une variable appelée « Builder ». Son utilisation est intéressante car elle remplace le constructor. Elle permet d’avoir les mêmes capacités complexes d’un constructeur mais de ne montrer que le plus important. Son type et son contenu. Elle permet aussi de ne pas déclarer d’arguments.

On déclare deux nouvelles variables. Seules une nous intéresse à ce niveau d’avancement de ce projet et il s’agit de « cameraSource ». On remarque que l’on utilise le « Builder » et que l’on attribue une classe, la classe dans laquelle elle est déclarée, ainsi que la variable déclaré juste avant, qui devrait servir pour détecter le QR Code.

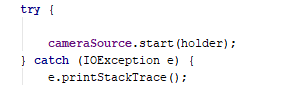
En même temps, on initie la taille de la caméra à 1920x1080. La plupart des téléphones d’aujourd’hui possède cette définition, c’est la plus standard.

Une image contenant capture d’écran

Description générée automatiquement

Cette partie du code se charge de la gestion des permissions de la caméra par l’application. On commence par créer un « SurfaceHolder » qui permet d’allouer une surface qui a été délimitée par le « SurfaceView ».

 Une fois que c’est fait, on créer un « While » qui va vérifier si la permission d’utiliser la caméra a été donné à l’application. Si ce n’est pas le cas, alors on rentre dans la boucle et on demande la permission d’utiliser la caméra dans la classe CameraActivity, « requestPermission(activity : CameraActivity.this », « Manifest.permission.CAMERA ». Dans le manifest, on a au préalable donné l’autorisation à l’application d’accéder à la caméra :

Une fois que l’application s’est vue autorisé l’accès à la caméra, elle rentre dans le « try » et lance la caméra avec un «. start » et doit se placer sur le « holder » que l’on a créé juste avant.

Une image contenant animal

Description générée automatiquement On a rajouté aussi deux méthodes qui permettent à la surface de s’adapter en fonction de l’écran et de la résolution de la caméra, mais aussi de détruire la surface créer pour la caméra ainsi que de désactiver la caméra quand on quitte la page :

# Test unitaires

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Elément testé :** | Lecture du tag | | | | | |
| **Objectif du test :** | On vérifie que l’application récupère les informations textuelles du tag | | | | | |
| **Nom du testeur :** | GÖRGE Pierre | | **Date :** | | Mercredi 22 Mai 2019 | |
| **Moyen mis en œuvre :** | Logiciel :  ApplicationAgain | | Matériel : Ordinateur | | Outils de développement :  Android Studio | |
| **Procédure de test** | | | | | | |
| **Description du vecteur de test** | | **Résultat attendu** | | **Résultat obtenu** | | **Validation (O/N)** |
| Une fois que le projet est ouvert sur Android Studio, on clique sur le bouton play vert et on appuie sur « Entré » quand une fenêtre s’est ouverte | | L’application se lance et le bouton vert a un petit rond vert à côté de lui | | Une image contenant capture d’écran  Description générée automatiquement | | O |
| Une fois le login et mot de passe rentré, appuyé sur le bouton « NFC » | | On arrive sur la page du NFC | | Une image contenant capture d’écran  Description générée automatiquement | | O |
| On scan le tag RFID | | L’information de la carte doit apparaître | |  | | O |
|  | |  | |  | | O |