



[jeremy.avid/app/apk](https://jeremyavid.github.io/app/apk)

<https://github.com/doctorgoون/barcodescanner/>

SOMMAIRE

I. Présentation Générale (page 3)

1. Contexte
2. Objectifs
3. Solution

II. Technologies Utilisées (page 5)

1. Android - Java
2. Zxing
3. REST

III. Développement (page 7)

1. Structure de l'application
2. Structure de l'interface graphique
3. Structure de l'activité principale
4. Coté serveur

IV. Utilisation (page 11)

V. Annexe (page 12)

I. Présentation Générale

Contexte

Mon entreprise, Lamirau Technologies, est une très petite entreprise (TPE) qui revend des appareils de mesure du souffle nommés spiromètres.

L'entreprise a vu son niveau d'activité augmenter depuis quelques années. Le respiratoire étant un secteur médical en pleine expansion. Pour faire face à de nouveaux objectifs de croissance, l'entreprise doit adapter ses processus internes pour s'assurer une productivité à la hauteur des demandes des clients.

S'agissant d'une TPE, il n'existe pas d'organisation structurée et procédurale permettant de faire face à une augmentation rapide de la demande. L'entreprise se retrouve donc face à une nécessité de moderniser, c'est à dire d'automatiser et de numériser, la gestion de ses stocks.

Jusqu'à présent cette gestion était faite de manière manuelle et analogique. Il n'existe pas de base de données pour le stock, ce qui engendre des difficultés dans le suivi. La gestion du stock est effectuée manuellement ; les commerciaux se rendent dans les locaux réservés aux stocks et « regarde » les articles encore présents. Cette gestion manuelle est possible du fait des quantités relativement faibles qui transitent, mais une augmentation de l'activité va rendre cette technique compliquée et peu efficiente.

Objectifs

L'un des objectifs est donc de moderniser le processus de gestion des stocks. Il s'agit de mettre en place une solution qui va augmenter l'efficacité et la productivité des employés dans le suivi des stocks, avec les articles qui arrivent, et ceux qui sont expédiés.

Un autre objectif est de simplifier les processus. La pièce contenant le stock étant dépourvu d'ordinateur, la solution doit permettre de gérer le stock à distance, sans avoir forcément accès au site web d'administration mis en place il y a peu.

En numérisant la gestion des stocks, l'entreprise va également réduire le risque d'erreur. Une automatisation réussie est gage de diminution du risque d'erreur.

Solution

Je propose de mettre en place une solution applicative mobile permettant de gérer le stock. Elle serait connectée à la base de données du portail web d'administration (gestion du stock, des clients et des appels).

Il s'agit d'une application Android, développée en JAVA, qui permettrait aux employés de n'avoir qu'à scanner le code barre d'un article pour modifier son statut dans la base de données ; c'est à dire avoir la possibilité une fois le numéro de série récupéré lors du scan, de marquer l'article comme disponible ou expédié, ou dans le cas où l'article n'est pas présent dans la base de données de l'y ajouter.

Ce nouveau processus faciliterait grandement la gestion du stock dans la mesure où il permettrait de tenir le stock à jour et de récupérer des informations sur les articles présents dans la base de données de manière quasi instantanée.

Compétences

- **Utilisation d'un SDK** => le développement sur Android passe par l'utilisation du SDK (*Software Development Kit*), c'est un ensemble d'outils et de librairie permettant de concevoir une application Android.
- **Utilisation d'un IDE (Environnement de développement) JAVA** => j'ai utilisé Android Studio sur Mac pour développer cette application Android. Ce logiciel est spécialement conçu pour le développement sur Android grâce à des outils spécialisés comme le gestionnaire de SDK ou bien l'éditeur d'interface utilisateur.
- **Création d'un webservice REST** => architecture provenant du protocole Http, elle m'a permis de faire transiter des données entre l'application Android et le site web d'administration.
- **Appel à une seconde activité dans une Application Android grâce à l'objet Intent** => l'objet Intent m'a permis de faire appel à une autre application déjà installé sur le téléphone pour en utiliser une de ses fonctions à l'intérieur de la mienne.
- **Utilisation du langage Orienté Objet JAVA** => Android se base sur le langage JAVA pour le développement de ses applications. Même si la structure du projet est spéciale, le code reste basé sur un système de classes, d'objets et de librairies.
- **Création d'une interface graphique sur Android** => j'ai dû créer une interface graphique simple et intuitive pour utiliser l'application et ses différentes fonctionnalités. Cette tache a été effectué à travers un fichier XML et avec l'aide du Layout Editor de Android Studio.

Téléchargement .apk => jeremy.avid/app/apk

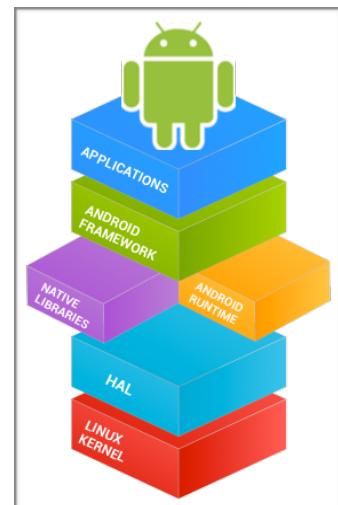
II. Technologies Utilisées

1. Android / JAVA

Android est le système d'exploitation mobile opensource le plus populaire au monde, il est basé sur le noyau Linux et est actuellement développé par Google.

Android est défini comme étant une pile de logiciels, c'est-à-dire un ensemble de logiciels destinés à fournir une solution clé en main pour les appareils mobiles. Cette pile comporte un système d'exploitation (avec un noyau Linux), les applications clés ainsi que des logiciels intermédiaires entre le système d'exploitation et les applications. L'ensemble est organisé en cinq couches distinctes :

- le noyau Linux avec les pilotes ;
- des bibliothèques logicielles telles que WebKit/Blink, OpenGL ES, SQLite ou FreeType ;
- un environnement d'exécution et des bibliothèques permettant d'exécuter des programmes prévus pour la plate-forme Java ;
- un framework, kit de développement d'applications ;
- un lot d'applications standard parmi lesquelles il y a un environnement de bureau, un carnet d'adresses, un navigateur web et un téléphone.



Jusqu'à sa version 4.4, Android comporte une machine virtuelle nommée Dalvik, qui permet d'exécuter des programmes prévus pour la plate-forme Java.

L'ensemble de la bibliothèque standard d'Android ressemble à JSE (Java Standard Edition) de la plateforme Java. La principale différence est que les bibliothèques d'interface graphique AWT et Swing sont remplacées par des bibliothèques d'Android.

À partir de la version 5.0 sortie en 2014, l'environnement d'exécution ART (Android RunTime) remplace la machine virtuelle Dalvik. Ce dernier permet des performances accrues et un gain d'autonomie pour la batterie.

Les applications tierces sont écrites en utilisant le Software Development Kit (SDK) de Android, et sont pour la plupart développées en Java. Le SDK comprend un ensemble d'outils de développement, comprenant un debugger, des librairies, un émulateur d'appareils, de la documentation, des exemples de codes etc. En décembre 2014, Google a sorti Android Studio, qui devient l'IDE officiel pour le développement d'applications Android.



2. Zxing

Zxing, prononcé de **Zebra Crossing**, est une librairie Java qui permet de lire et générer des code barres.

C'est un projet open-source qu'il est possible d'utiliser de plusieurs manières :

- soit en installant et en utilisant la dépendance complète de la librairie,
- ou bien en appelant l'application « **barcode Scanner** » préalablement installée sur l'appareil en utilisant l'intention Android qui permet de faire communiquer des applications entre elles. Dans notre cas, nous allons utiliser l'**Intent** pour déclencher dans une autre **Activity** (barcode Scanner), la méthode qui va activer l'appareil photo pour permettre de scanner, et cette dernière renverra la réponse à l'Activity principale.

La seconde solution est la plus simple et la plus malléable. Le seul inconvénient provenant du fait qu'il faille installer une autre application depuis le Plays Store, ce qui ne pose pas de problème dans notre cas.

- Documentation => <https://github.com/zxing/zxing/wiki>

3. REST

REST (representational state transfer) est un style d'architecture qui repose sur le protocole **HTTP** : on accède à une ressource, par son URI unique, pour procéder à diverses opérations (**GET** lecture / **POST** écriture / **PUT** modification / **DELETE** suppression) qui sont supportées nativement par HTTP.

Cette technologie web va me permettre de mettre en place le service web qui va faire transiter les informations sur les produits entre l'application Android et le site web d'administration.

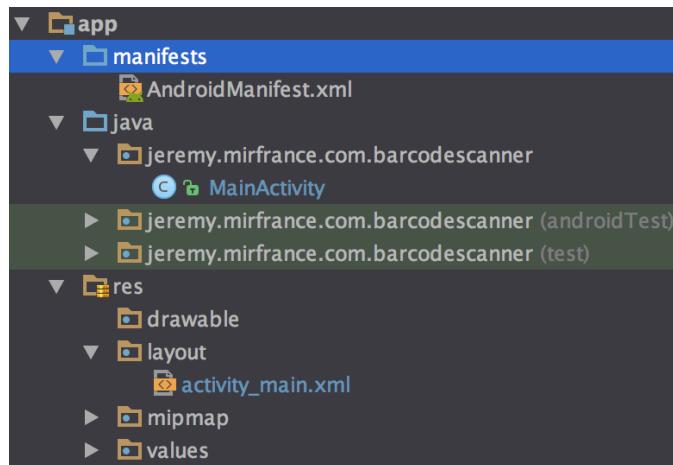
Une architecture REST doit respecter 6 contraintes qui définissent la base de cette technologie :

- *Client-serveur* : L'interface utilisateur est séparée de celle du stockage des données.
- *Sans état* : chaque requête d'un client vers un serveur doit contenir toute l'information nécessaire pour permettre au serveur de comprendre la requête.
- *Mise en cache* : toute réponse du serveur comprend des indications quant à la possibilité de mettre en cache cette réponse, comme sa fraîcheur, sa date de création, sa validité future.
- *Une interface uniforme* : une structure technique unique entre le client et le serveur.
- *Un système hiérarchisé par couches* : les ressources sont individuelles. Les ressources peuvent transporter des ressources provenant de serveurs distants ou de mise en cache par des serveurs intermédiaires.
- *Code-on-demand (facultatif)* : la possibilité pour les clients d'exécuter des scripts obtenus depuis le serveur.

III.Développement



1. Structure de l'application



L'application est structurée de manière assez classique pour une application Android développée sur Android Studio :

• res

- ▶ layout
 - **activity_main.xml** => fichier xml qui correspond à l'interface de l'activité principale de l'application. Il permet de déclarer et positionner les éléments graphiques. Android Studio dispose d'un Layout Editor qui permet de rajouter et modifier les éléments graphique à l'aide d'une interface graphique.
- ▶ mipmap => on y retrouve les .png des icônes de l'application.

• java

- ▶ jeremy.mirfrance.com.barcodescanner
 - **MainActivity.java** => fichier Java contenant le code de l'activité principale de l'application. Elle est relié à l'interface déclarée dans le fichier activity_main.xml

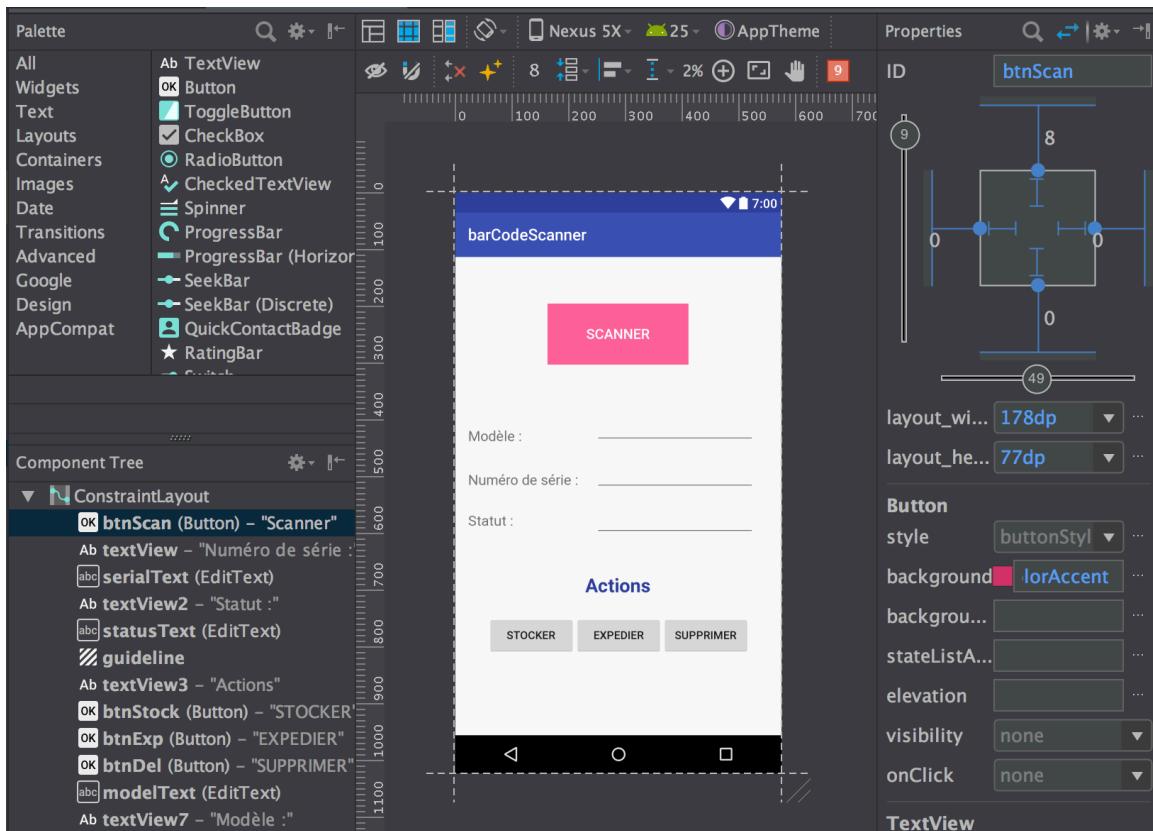
• manifests

- ▶ **AndroidManifest.xml** => fichier xml qui décrit l'application (version, thème, autorisations...) et ses fonctionnalités (Activity).

Pour faire simple, activity_main permet de définir graphiquement l'application, MainActivity de relier l'interface à des fonctionnalités et enfin AndroidManifest résume le tout.

2. Structure de l'interface graphique (activity_main.xml)

L'interface de l'application est assez simple, du fait même de sa fonction.



On y trouve :

- **5 boutons** qui permettent de lancer un scan, de marquer l'article comme en stock, expédié ou de le supprimer, ou bien encore d'ouvrir une boîte de dialogue pour régler l'adresse du serveur.
- **4 TextView**, il s'agit d'éléments textuels, des sortes de labels pour « Modèle », « Numéro de série », « Statut » et « Actions ».
- **3 EditText**, champs de texte modifiable. Ils vont permettre d'afficher les informations récupérées au près du serveur via la requête REST.

L'interface Layout Editor de Android Studio (image ci dessus) facilite grandement la création de l'interface utilisateur.

Il suffit de choisir des éléments dans le cadre en haut à gauche et de les déposer sur l'écran du téléphone. La barre de droite permet ensuite d'en définir les propriétés, telles que la couleur, l'identifiant, la taille, mais également sa relation avec les autres éléments de l'interface.

En effet, pour avoir une interface cohérente et adaptative il est important de relier les éléments entre eux. Il s'agit d'établir par exemple la distance entre deux, ou bien de faire se déplacer l'élément enfant lorsque l'élément parent change de position.

Les changements effectués sur l'interface sont ensuite retranscrits en langage XML dans le fichier **activity_main.xml** :

```
<Button  
    android:id="@+id/btnStock"  
    android:layout_width="110dp"  
    android:layout_height="50dp"  
    android:text="STOCKER"  
    tools:layout_constraintBaseline_creator="1"  
    app:layout_constraintRight_toLeftOf="@+id(btnExp)"  
    app:layout_constraintHorizontal_chainStyle="packed"  
    app:layout_constraintBaseline_toBaselineOf="@+id(btnExp)"  
    app:layout_constraintLeft_toLeftOf="parent" />
```

3. Structure de l'activité principale (MainActivity)

L'activité principale s'articule autour de 7 fonctions, 14 variables et 1 constante :

```
m ᐃ onCreate(Bundle): void ↑AppCompatActivity  
m ᐃ showSettings(): void  
m ᐃ scanBar(): void  
m ᐃ onActivityResult(int, int, Intent): void ↑FragmentActivity  
m ᐃ getInfoFromServer(String): void  
m ᐃ sendInfoToServer(String, int): void  
m ᐃ getModel(String): String  
f ᐃ ip: String = "jeremyavid.com"  
f ᐃ btnScan: Button = null  
f ᐃ btnStock: Button = null  
f ᐃ btnExp: Button = null  
f ᐃ btnDel: Button = null  
f ᐃ settings: Button = null  
f ᐃ textView: TextView = null  
f ᐃ textView2: TextView = null  
f ᐃ textView3: TextView = null  
f ᐃ textView7: TextView = null  
f ᐃ serialText: EditText = null  
f ᐃ statusText: EditText = null  
f ᐃ modelText: EditText = null  
f ᐃ ACTION_SCAN: String = "com.google.zxing.client.android.SCAN"  
f ᐃ serial_num: String = null  
f ᐃ isSerial: String = null
```

onCreate() => fonction qui initialise l'activité. Il faut donc y déclarer le layout utilisé :

setContentView(R.layout.activity_main); et y initialiser les éléments graphiques à afficher. C'est également dans cette fonction que sont positionnés les **Listeners** qui vont permettre de définir quelle fonction appeler lorsqu'un bouton est pressé.

showSettings() => fonction déclencher à la pression du bouton « Réglage ». Elle enclenche une boite de dialogue permettant de changer l'adresse du serveur.

scanBar() => fonction déclenchant la lecture de code barre. Pour ce faire, l'on déclare un nouvel objet **Intent** qui va permettre de lancer une deuxième activité en parallèle. Avec pour paramètre "com.google.zxing.client.android.SCAN", il va tenter d'ouvrir l'application de lecture de code barre et lancer le SCAN ; ou bien lancer la page de l'application dans le Play Store si cette

dernière n'est pas présente sur le téléphone, grâce à l'URI "market://search?q=pname:" + "com.google.zxing.client.android".

onActivityResult() => fonction qui écoute les résultats de l'activité lancée grâce au Intent. Si l'activité enclenchée a été fructueuse, l'on retrouve le résultat du scan dans l'objet Intent. Pour vérifier que le résultat ressemble bien au numéro de série d'un spiromètre, on fait appel à la fonction **getModel()**. S'il agit bien d'un spiromètre, alors le serveur est interrogé, avec la fonction **getInfoFromServer()**, pour récupérer les informations sur l'article scanné.

getModel() => fonction prenant en paramètre le résultat obtenu lors du scan, afin de définir s'il s'agit bien d'un spiromètre et si oui de quel modèle. Elle retourne donc le nom du modèle ou *null* si n'est pas un spiromètre. Cette fonction remplit le champ « nom » de l'application.

getInfoFromServer() => fonction prenant en paramètre le numéro de série de l'article. Elle va permettre de faire une requête POST au près du serveur qui abrite le site d'administration et la base de données. Pour se faire, un objet JSON est créé dans lequel on positionne le numéro de série de l'article. L'on y définit également l'URL qui sera appelé pour faire la requête HTTP, et nous déclarons une nouvelle connexion HTTP. Si la connexion est établie et réussie, la connexion va recevoir le code réponse 200 qui correspond à « OK ». Un nouvel objet JSON est ensuite créé comportant la réponse du serveur obtenue grâce au buffer de la connexion ; la réponse étant le statut dans la base de données de l'article préalablement scanné. Le champ statut est alors rempli en fonction.

sendInfoToServer() => fonction prenant en paramètre le numéro de série de l'article, ainsi que le type d'action souhaité (int). Elle enclenché lorsque l'utilisateur presse un bouton d'action. Cette fonction reprend une bonne partie la fonction **getInfoFromServer()** dans la mesure où elle est également en place une connexion HTTP où est envoyé un JSON comportant le type d'action et le numéro de série sous forme de requête POST. Dans le cas où la connexion est réussie, le serveur va renvoyé un code de résultat indiquant si l'action a été effectué avec succès et de quelle action il s'agissait. Le champ statut de l'application va alors changer pour se mettre à jour.

4. Côté serveur

Il m'a tout d'abord fallu créer deux nouvelles routes correspondantes aux URL appelées par l'application Android :

```
Route::post('/app/get', 'ProductsController@showStatus');
Route::post('/app/set', 'ProductsController@changeStatus');
```

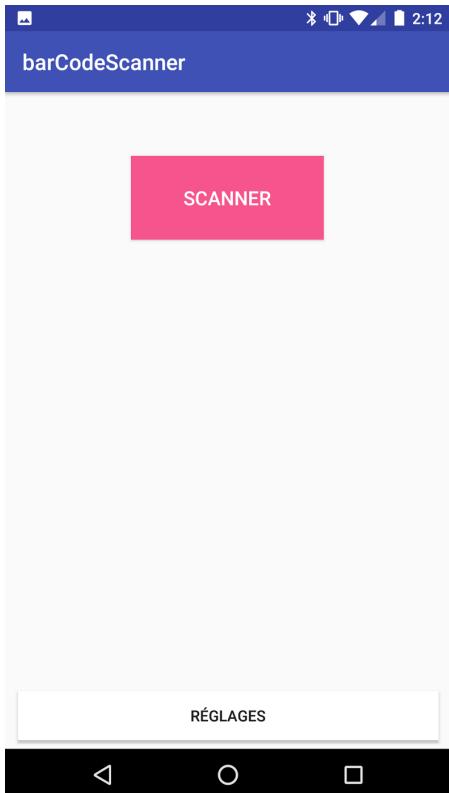
Ces routes pointant vers deux fonctions présentes dans la classe *ProductsController*.

PS : ces deux routes sont en dehors de la route englobante comportant un middleware d'authentification.

showStatus() => grâce à la classe Request, elle récupère les données (numéro de série) de la requête POST effectuée sur l'URL '/app/get'. La fonction va ensuite effectuer une requête SQL via le modèle Eloquent **Products** afin de trouver le statut de l'article à qui appartient ce numéro de série dans la base de données. Elle retourne enfin un tableau de type ['status' => 2], que Laravel va automatiquement convertir en JSON pour répondre à l'application, ou le chiffre correspond au statut de l'article (0 => absent de la base de données, 1 => en stock, 2 => expédié), ou à une erreur (3 => erreur dans le numéro de série ou -1 => dans la base de données).

changeStatus() => comme pour **showStatus()**, elle récupère les données de la requête POST effectuée par l'application Android sur l'URL '/app/set'. Cette fois-ci la requête comporte également le code de l'action souhaitée par l'utilisateur de l'application. Après avoir récupéré l'objet en question dans la base de données grâce à une requête Eloquent, la fonction va soit modifier le statut de l'article, soit créer un nouvel article avec le statut souhaité par l'utilisateur de l'application. Ici également, un code de résultat est envoyé sous forme de JSON à l'application.

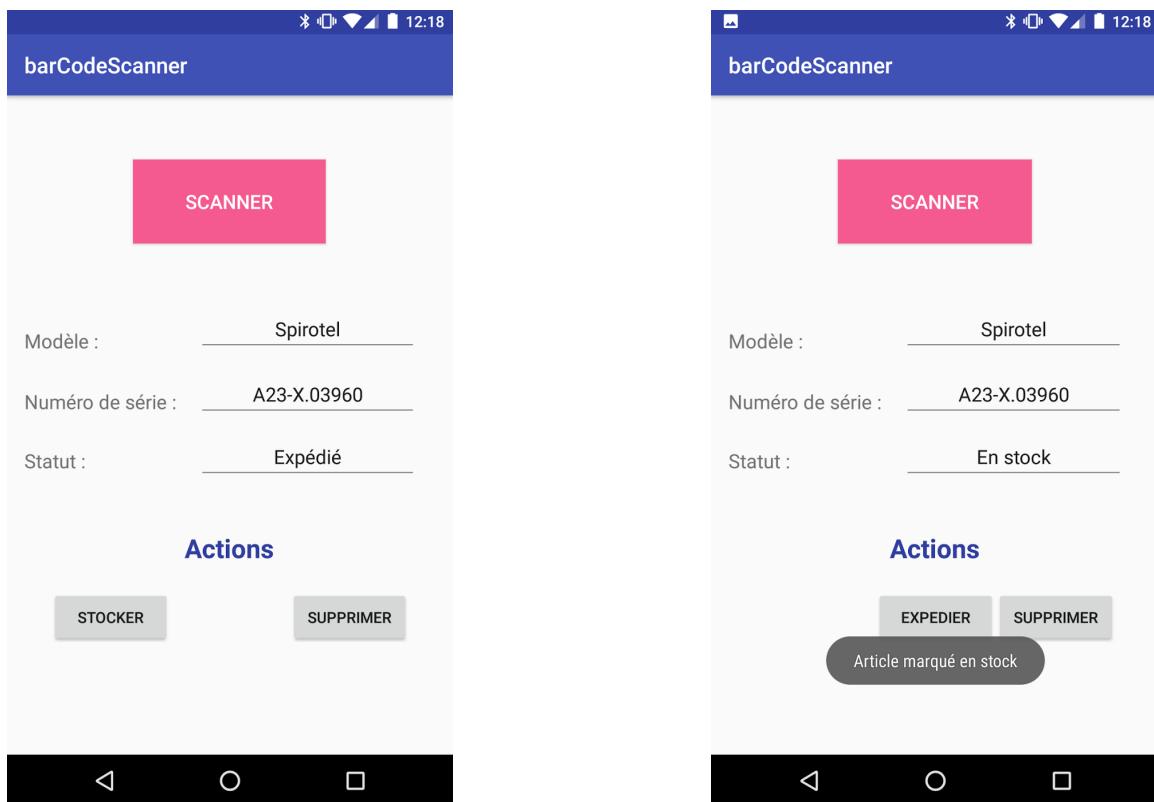
IV. Utilisation



Ecran d'accueil de l'application



Scanning d'un code barre via l'application Barcode Scanner

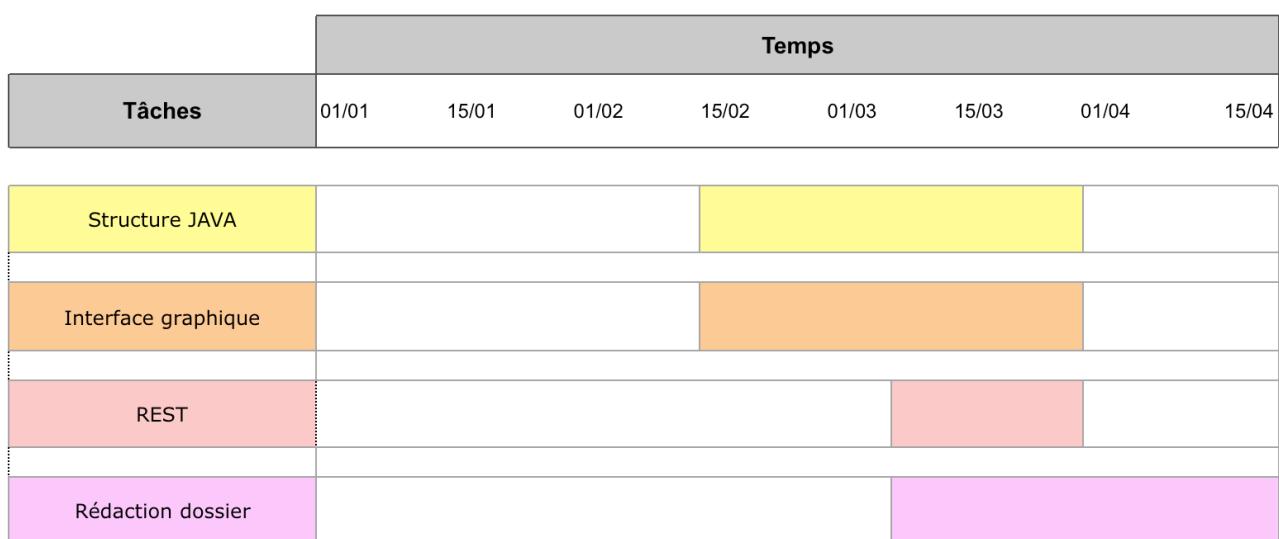


Affichage du résultat du scan et de la requête au serveur

Le bouton STOCKER a été pressé, l'utilisateur reçoit un toast lui indiquant que la demande au serveur a été réussie, le statut de l'article est passé à « en stock »

V. Annexe

Gestion du projet :



Exemples de code barres :

