HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Laboratoire :

Lecture de code-barres

Département : Technologies de l'Information et de la Communication (TIC)
Unité d'enseignement : Développement d'applications Android (DAA)

Auteurs : Harun Ouweis, Mouti Amir
Professeur : F. Dutoit
Date : 19 janvier 2025

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

1 Introduction

1.1 Contexte général

Les codes-barres sont omniprésents dans des domaines tels que le commerce, la logistique et la gestion d'inventaire. Leur adoption repose sur leur simplicité et leur efficacité à stocker et transmettre rapidement des informations. On les retrouve sur divers supports, notamment les emballages, les billets électroniques et les cartes de fidélité.

Avec l'essor des applications mobiles, la lecture de codes-barres est devenue une fonctionnalité clé sur Android, offrant une valeur ajoutée aux utilisateurs et aux entreprises, que ce soit pour scanner des tickets, vérifier des prix ou gérer un stock. Autrefois réservée aux périphériques dédiés, cette technologie s'est démocratisée grâce aux avancées en vision par ordinateur et aux bibliothèques logicielles, facilitant son intégration dans les applications mobiles.

1.2 Objectifs du projet

Ce projet vise à étudier l'intégration de la lecture de codes-barres dans une application Android, en analysant différentes bibliothèques et en mettant l'accent sur leur facilité d'utilisation, compatibilité et performances.

Les objectifs incluent :

— Présenter les fondamentaux et applications des codes-barres.
— Comparer les bibliothèques disponibles pour Android.
— Fournir un guide pratique pour intégrer un scanner de codes-barres.
— Discuter des limitations et alternatives à cette technologie.

1.3 Méthodologie

L'étude suit une approche progressive :

1. **Recherche documentaire** : Analyse des bibliothèques les plus utilisées (ML Kit, ZXing).
2. **Comparaison des solutions** : Facilité d'intégration, performances, compatibilité.
3. **Implémentation pratique** : Développement d'un prototype intégrant un scanner de codes-barres.
4. **Évaluation et retour d'expérience** : Tests, identification des limites et proposition d'améliorations.

Cette approche permettra d'offrir un guide structuré et accessible aux développeurs souhaitant intégrer cette fonctionnalité dans leurs applications.

2 Comprendre la fonctionnalité attribuée

2.1 Définir la lecture de codes-barres

La lecture de codes-barres permet d'extraire et d'interpréter les informations contenues dans un code imprimé ou affiché sur un écran. Ces codes peuvent être linéaires (1D) ou bidimensionnels (2D), chacun ayant des usages spécifiques.

2.1.1 Description technique

Le processus repose sur plusieurs étapes :

1. **Acquisition de l'image** : Capture du code via un capteur optique.
2. **Prétraitement** : Correction de l'éclairage et amélioration du contraste.
3. **Détection et analyse** : Localisation et conversion en données exploitables.
4. **Décodage** : Interprétation des données selon le format du code.

2.1.2 Types de codes supportés

Codes-barres 1D

— **EAN-13** : Identifiant produit en commerce.
— **UPC** : Utilisé en Amérique du Nord.
— **Code 128, Code 39** : Utilisés en logistique et industrie.

Codes-barres 2D

— **QR Code** : Stocke plus d'informations et est rapidement scannable.
— **Data Matrix, PDF417** : Utilisés dans l'industrie et les documents officiels.

2.2 Identifier les problématiques résolues

La lecture de codes-barres apporte des solutions dans plusieurs domaines :

2.2.1 Automatisation des tâches

— **Saisie automatique** : Récupération instantanée des informations sans saisie manuelle.
— **Gestion des stocks et suivi des produits** : Mise à jour rapide et fiable des inventaires.

2.2.2 Réduction des erreurs humaines

— **Précision accrue** : Évite les erreurs typographiques et les confusions de références.
— **Fiabilité améliorée** : Garantit un suivi précis des produits et documents.

2.2.3 Amélioration de l'expérience utilisateur

— **Gain de temps** : Accès immédiat aux informations produits.
— **Simplicité d'usage** : Lecture rapide pour diverses applications (transport, commerce).

2.3 Domaines d'application

2.3.1 Commerce (paiements, étiquetage)

— **Magasins et supermarchés** : Scan des produits en caisse.
— **E-commerce** : Suivi des colis et vérification d'authenticité.

2.3.2 Santé (suivi des patients)

— **Identification des patients** : Traçabilité des soins via bracelets scannés.
— **Gestion des médicaments** : Vérification et administration sécurisée.

2.3.3 Logistique (gestion d'entrepôts)

— **Suivi des expéditions et stocks** : Optimisation de la chaîne logistique.
— **Automatisation des inventaires** : Réduction du temps et des erreurs.

Grâce à ces applications, la lecture de codes-barres optimise les processus, réduit les erreurs et améliore l'expérience utilisateur.

3 Rechercher les outils et bibliothèques disponibles

3.1 Présentation des bibliothèques principales

L'intégration de la lecture de codes-barres sur Android repose sur des bibliothèques spécialisées. Les deux plus populaires sont **ML Kit Barcode Scanning** et **ZXing (Zebra Crossing)**, offrant des fonctionnalités adaptées aux besoins des développeurs.

3.1.1 ML Kit Barcode Scanning

ML Kit, développé par Google, fait partie de Firebase Machine Learning et utilise l'apprentissage automatique pour détecter et lire efficacement les codes-barres.

Avantages

— **Facilité d'intégration** : API simple et compatible avec Android CameraX.
— **Prise en charge multi-formats** : EAN-13, Code 128, QR Code, etc.
— **Performances optimisées** : Traitement en temps réel avec mises à jour régulières.

Limitations

— **Dépendance à Google Play Services** : Nécessite Firebase.
— **Non disponible hors ligne sans configuration avancée**.

3.1.2 ZXing (Zebra Crossing)

ZXing est une bibliothèque open-source largement utilisée pour la lecture de codes-barres et QR codes. Elle peut être intégrée directement ou via un Intent.

Avantages
- **Open-source et gratuit** : Aucune dépendance propriétaire.
- **Utilisation hors ligne** : Fonctionne sans Google Play Services.
- **Compatibilité large** : Supporte de nombreux formats de codes-barres.

Limitations
- **Interface utilisateur plus basique** : Nécessite des ajustements pour une meilleure expérience utilisateur.
- **Moins optimisé pour le temps réel** : Performances variables selon l'appareil.

3.2 Comparaison des bibliothèques

Critère	ML Kit	ZXing
Facilité d'intégration	Simple via Firebase	Implémentation manuelle ou Intent
Performance	Optimisé pour le temps réel	Variable selon l'appareil
Compatibilité	Google Play Services requis	Indépendant de Firebase
Limitations	Pas toujours offline	Interface plus basique

TABLE 1 – Comparaison des bibliothèques de lecture de codes-barres

3.3 Sélection de la solution adaptée

Le choix de la bibliothèque dépend des besoins du projet :
- **Application hors ligne** : ZXing est préférable.
- **Reconnaissance rapide et fluide** : ML Kit offre de meilleures performances.
- **Simplicité d'intégration** : ML Kit est plus intuitif à configurer.
- **Solution open-source sans dépendances** : ZXing est recommandé.

ML Kit est idéal pour les applications modernes nécessitant un traitement rapide, tandis que ZXing offre une alternative fiable et indépendante.

4 Implémentation

4.1 Préparation du projet Android

Avant d'intégrer la lecture de codes-barres avec **ML Kit Barcode Scanning**, il est nécessaire d'effectuer plusieurs préparations dans le projet Android.

4.1.1 Ajout des dépendances nécessaires (`build.gradle`)

Dans le fichier `build.gradle (Module: app)`, ajoutez la dépendance suivante pour inclure ML Kit :

```
dependencies {
    implementation (libs.barcode.scanning)
        implementation(libs.androidx.camera.view)
    implementation(libs.androidx.camera.lifecycle)
    // Dependance principale pour CameraX
    implementation(libs.androidx.camera.core)
    // Implementation par defaut pour interagir avec la cam
    implementation(libs.androidx.camera.camera2)
}
```

Listing 1 – Ajout de la dépendance ML Kit

Assurez-vous également que Google Play Services est activé dans votre projet.

4.1.2 Permissions requises pour l'utilisation de la caméra

Ajoutez les permissions nécessaires dans le fichier `AndroidManifest.xml` :

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

Listing 2 – Ajout des permissions pour la caméra

Si vous ciblez Android 10 (API 29) ou supérieur, ajoutez également cette permission dans `manifest` pour activer la caméra :

```
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Listing 3 – Activation de la caméra

Dans le **code Java/Kotlin**, demandez la permission de la caméra à l'exécution si nécessaire.

4.2 Implémentation de base

4.2.1 Configuration initiale de ML Kit

Créez un `BarcodeScanner` avec ML Kit :

```
val scanner = BarcodeScanning.getClient()
```

Listing 4 – Configuration du scanner ML Kit

4.2.2 Mise en place du scanner de code-barres

Utilisez **CameraX** pour capturer l'image et traiter le code-barres :

```
private fun startCamera() {
    val cameraProviderFuture = ProcessCameraProvider.getInstance(
    this)

    cameraProviderFuture.addListener({
        val cameraProvider: ProcessCameraProvider =
    cameraProviderFuture.get()
        val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

        val preview = Preview.Builder().build().also {
            it.setSurfaceProvider(findViewById<PreviewView>(R.id.
    viewFinder).surfaceProvider)
        }

        val imageAnalysis = ImageAnalysis.Builder()
            .setBackpressureStrategy(ImageAnalysis.
    STRATEGY_KEEP_ONLY_LATEST)
            .build()
            .also {
                it.setAnalyzer(cameraExecutor) { image ->
                    if (isScanning) {
                        processImage(image)
                    } else {
                        image.close()
                    }
                }
            }

        try {
            cameraProvider.unbindAll()
            cameraProvider.bindToLifecycle(this, cameraSelector,
    preview, imageAnalysis)
        } catch (exc: Exception) {
            Log.e("Camera", "Erreur de demarrage de la camera", exc
    )
        }
    }, ContextCompat.getMainExecutor(this))
}
```

Listing 5 – Analyse et traitement des codes-barres

4.3 Affichage des résultats

Lorsque le scanner détecte un code-barres, les données peuvent être affichées dans une `TextView` ou utilisées pour déclencher une action spécifique.

```
barcodeScanner.process(image)
    .addOnSuccessListener { barcodes ->
        if (barcodes.isNotEmpty()) {
            textView.text = "Code detecte : ${barcodes.first().
    rawValue}"
        } else {
```

```
            textView.text = "Aucun code detecte, reessayez."
        }
    }
```

Listing 6 – Affichage du code-barres détecté

4.4 Gestion des erreurs

4.4.1 Scénarios d'erreurs possibles

— **Caméra indisponible** : Assurez-vous que l'appareil dispose d'une caméra fonctionnelle.
— **Échec de la reconnaissance** : Vérifiez que le code-barres est bien visible et bien éclairé.
— **Permissions manquantes** : Demandez la permission d'accès à la caméra avant d'utiliser ML Kit.

4.4.2 Solutions proposées

Ajoutez une gestion d'erreur pour avertir l'utilisateur :

```
.addOnFailureListener {
    textView.text = "Erreur de scan, reessayez."
    Log.e("Barcode", "Erreur lors de la lecture du code-barres", it
   )
}
```

Listing 7 – Gestion des erreurs

4.5 Optimisations

4.5.1 Gestion de la batterie et performances

— **Utilisez la stratégie `STRATEGY_KEEP_ONLY_LATEST` pour éviter de surcharger le processeur**.
— **Désactivez la caméra lorsque l'application est en arrière-plan pour économiser la batterie**.

4.5.2 Réduction des temps de traitement

— **Optimisez l'éclairage et la mise au point** de la caméra pour faciliter la lecture.
— **Utilisez `CameraX` avec des paramètres adaptés** pour une meilleure fluidité.

5 Limitations et alternatives

5.1 Limitations de la lecture de codes-barres

Bien que largement adoptée, la lecture de codes-barres présente certaines limitations affectant son efficacité.

5.1.1 Contraintes matérielles

- **Qualité de la caméra** : Une faible résolution complique la détection, surtout pour les codes petits ou abîmés.
- **Luminosité insuffisante** : Un éclairage faible réduit la précision, nécessitant un flash ou des ajustements d'exposition.
- **Distance et mise au point** : Sans autofocus, la reconnaissance devient moins fiable.

5.1.2 Limites des formats

- **Compatibilité restreinte** : Certains formats ne sont pas pris en charge par ML Kit.
- **Capacité limitée** : Les codes-barres 1D stockent peu d'informations, contrairement aux QR codes.
- **Standardisation variable** : Certaines variantes propriétaires peuvent poser des problèmes d'interprétation.

5.2 Alternatives et solutions complémentaires

Face à ces limites, plusieurs solutions alternatives existent.

5.2.1 QR codes pour des données plus complexes

- **Stockage étendu** : Peut contenir des URL, identifiants ou blocs de texte.
- **Correction d'erreurs** : Un QR code reste lisible même partiellement endommagé.
- **Applications variées** : Utilisé pour l'authentification, les paiements mobiles et le marketing.

5.2.2 OCR (Reconnaissance Optique de Caractères)

- **Lecture des textes imprimés** : Utile lorsque les codes-barres sont absents ou illisibles.
- **Compatibilité universelle** : Fonctionne avec tout type de document sans format imposé.
- **Applications pratiques** : Numérisation de factures, tickets ou documents d'identité.

L'association de ces technologies avec la lecture de codes-barres améliore la fiabilité et l'expérience utilisateur, rendant les applications plus robustes et polyvalentes.

6 Cas pratiques et exemples

6.1 Exemples d'implémentation complète

6.1.1 Exemple de lecture et affichage d'un code-barres

Nous avons développé une application Android intégrant **ML Kit Barcode Scanning** pour détecter et afficher le contenu d'un code-barres. L'application utilise **CameraX** pour capturer l'image et analyse le code-barres uniquement lorsqu'un bouton "Scanner" est pressé. Si aucun code n'est détecté, un message d'erreur s'affiche. Le code source est rendu aussi si vous souhaitez le voir.

6.2 Visualisation

6.2.1 Tests réalisés et résultats obtenus

- **Test de lecture de code-barres** : Un scan réussi affiche immédiatement le contenu du code-barres dans le `TextView`.
- **Test de gestion des erreurs** : Si aucun code n'est détecté, un message `"Aucun code détecté, réessayez."` s'affiche.
- **Test de gestion de la caméra** :
  - La caméra s'ouvre correctement et se réactive après fermeture et réouverture de l'application.
  - Aucun crash n'a été observé lors de l'utilisation prolongée.

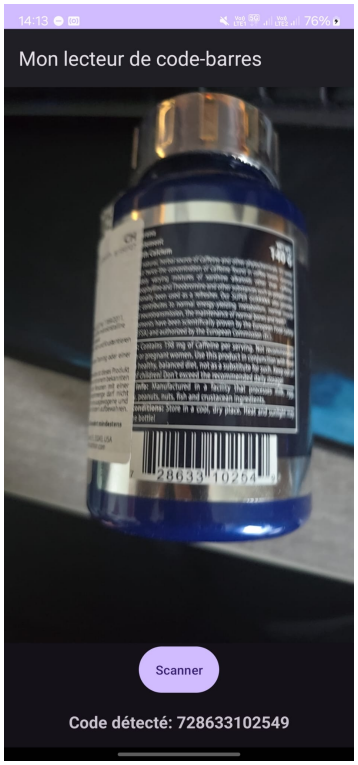

FIGURE 1 – Lecture réussie d'un code-barres

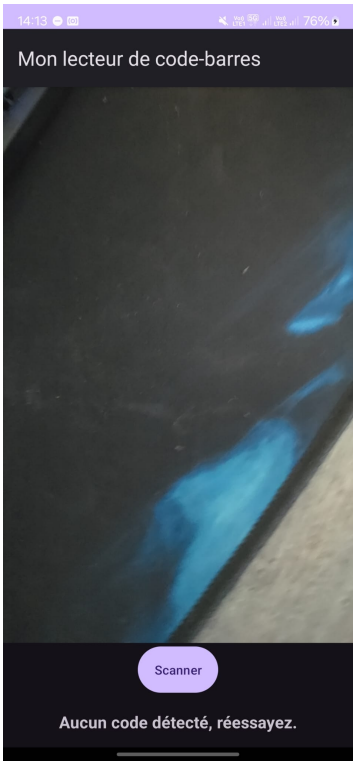

FIGURE 2 – Message d'erreur en cas d'échec

7 Conclusion

7.1 Synthèse

Ce rapport a exploré l'intégration de la lecture de codes-barres dans une application Android avec **ML Kit Barcode Scanning**. Nous avons abordé son importance, les problématiques résolues et sa mise en œuvre pratique.

Les principaux points traités sont :

— **Fonctionnement des codes-barres** : Formats et applications.
— **Choix des bibliothèques** : ML Kit comme solution optimisée.
— **Tutoriel d'intégration** : Configuration, gestion des permissions et implémentation via **CameraX**.
— **Tests et visualisation** : Lecture, gestion des erreurs et validation des performances.

L'étude a démontré que l'intégration de ML Kit est simple et efficace pour l'automatisation et la gestion des données.

7.2 Recommandations

Pour améliorer cette implémentation, plusieurs pistes peuvent être explorées :

7.2.1 Nouvelles fonctionnalités

— **Formats supplémentaires** : Intégration de **DataMatrix** ou **PDF417**.
— **Enregistrement des scans** : Stockage local ou cloud pour un suivi historique.
— **Connexion aux API** : Validation de produits ou récupération d'informations en ligne.

7.2.2 Optimisation des performances

— **Gestion avancée des erreurs** : Retour utilisateur amélioré via animations et alertes.
— **Réduction de la consommation énergétique** : Désactivation automatique de la caméra en cas d'inactivité.
— **Compatibilité étendue** : Tests sur divers appareils pour garantir la robustesse.

Ces améliorations renforceraient l'expérience utilisateur et l'efficacité du scanner, rendant l'application plus polyvalente et performante.

8 Bibliographie et références

8.1 Documentation officielle

— **ML Kit Barcode Scanning** - Documentation officielle de Google.
— **Android CameraX** - Documentation sur CameraX pour l'intégration de la caméra.

8.2 Articles et ressources en ligne

— **Barcode Scanning with ML Kit - Medium** - Tutoriel détaillé sur l'intégration de ML Kit pour la lecture de codes-barres.
— **Introduction to CameraX** - Article expliquant les bases de CameraX.

8.3 Utilisation d'outils d'assistance

ChatGPT a été utilisé pour structurer et relire le rapport, ainsi que pour fournir des explications détaillées sur des concepts nécessitant des clarifications supplémentaires.

9 Signatures

Yverdon-les-Bains le 19 janvier 2025

Harun Ouweis, Mouti Amir