Berney Alec & Forestier Quentin & Herzig Melvyn

SYM – 18.12.2021

Codes-barres, iBeacons et NFC

Labo 3



Table des matières

[Introduction 2](#_Toc90743564)

[Démarrage 2](#_Toc90743565)

[Remarques préalables 2](#_Toc90743566)

[Manipulations 2](#_Toc90743567)

[Balises NFC 3](#_Toc90743568)

[Utilisation 3](#_Toc90743569)

[Mise en place 5](#_Toc90743570)

[Question 7](#_Toc90743571)

[Code-barre 8](#_Toc90743572)

[Utilisation 8](#_Toc90743573)

[Mise en place 9](#_Toc90743574)

[Question 10](#_Toc90743575)

[IBeacons 13](#_Toc90743576)

[Utilisation 13](#_Toc90743577)

[Mise en place 13](#_Toc90743578)

[Question 15](#_Toc90743579)

# Introduction

Ce laboratoire est une introduction à l’utilisation des données environnementales avec Android. Ces données sont :

* Les balises NFC
* Les codes-barres
* Les iBeacons

# Démarrage

L’application démarre sur la page principale :



Figure 1 Interface principale

Chaque bouton crée une nouvelle activité qui met en scène les différentes données environnementales citées précédemment.

## Remarques préalables

Ce laboratoire est une introduction aux données environnementales. En conséquence, certains aspects de la programmation Android ne sont pas développés par manque de temps. Par exemple, les activités n’utilisent pas les principes des view-models.

# Manipulations

Comme présenté préalablement, ce laboratoire consiste en 3 manipulations majeures. Nous avons réalisé une activité par manipulation. Au cours des points suivants, nous allons décrire de manière incrémentale ce que nous avons mis en place pour arriver à l’état final de ce laboratoire.

## Balises NFC

### Utilisation

L’interface de l’activité se présente comme telle :

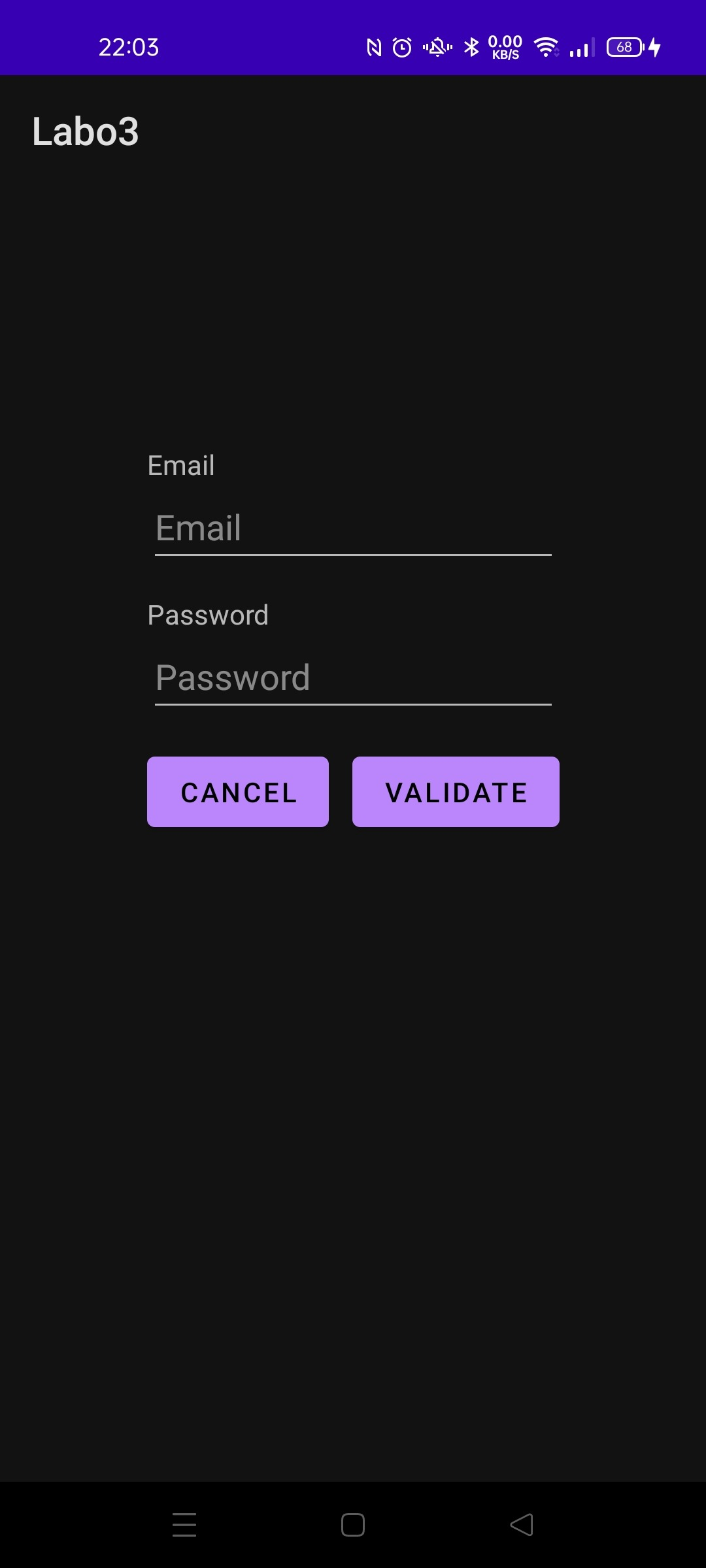


Figure 2 Formulaire de login pour l'activité NFC

Tout d’abord, il faut se logger à l’aide du formulaire, en utilisant les informations de connexion suivantes :

* Email : [labo3@sym.ch](mailto:labo3@sym.ch)
* Mot de passe : kotlin

Après avoir rentré les informations et appuyé sur le bouton de validation, l’activité demandera le scan de la balise NFC.

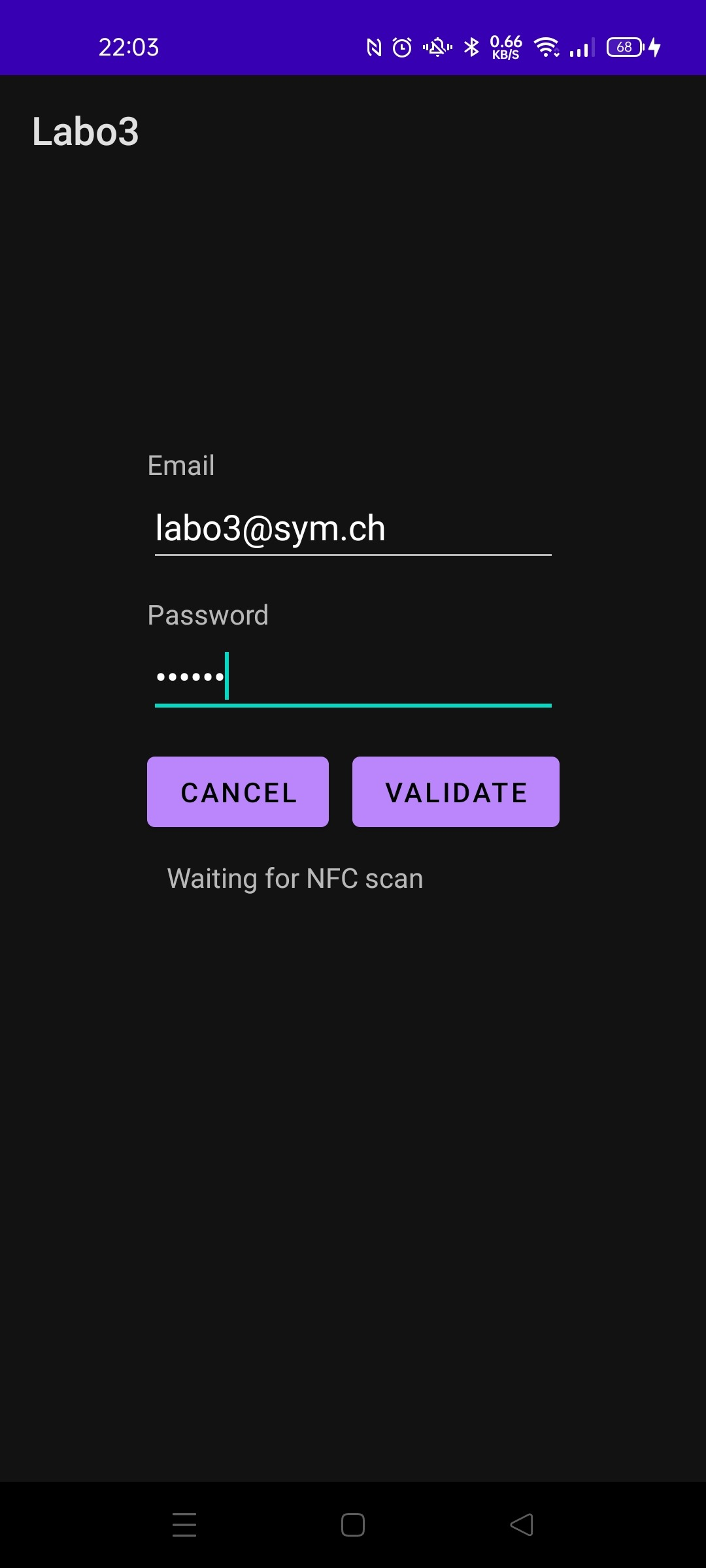


Figure 3 Attente du scan de la balise NFC

Lorsque la balise a été scannée, nous arrivons enfin à la zone sécurisée.

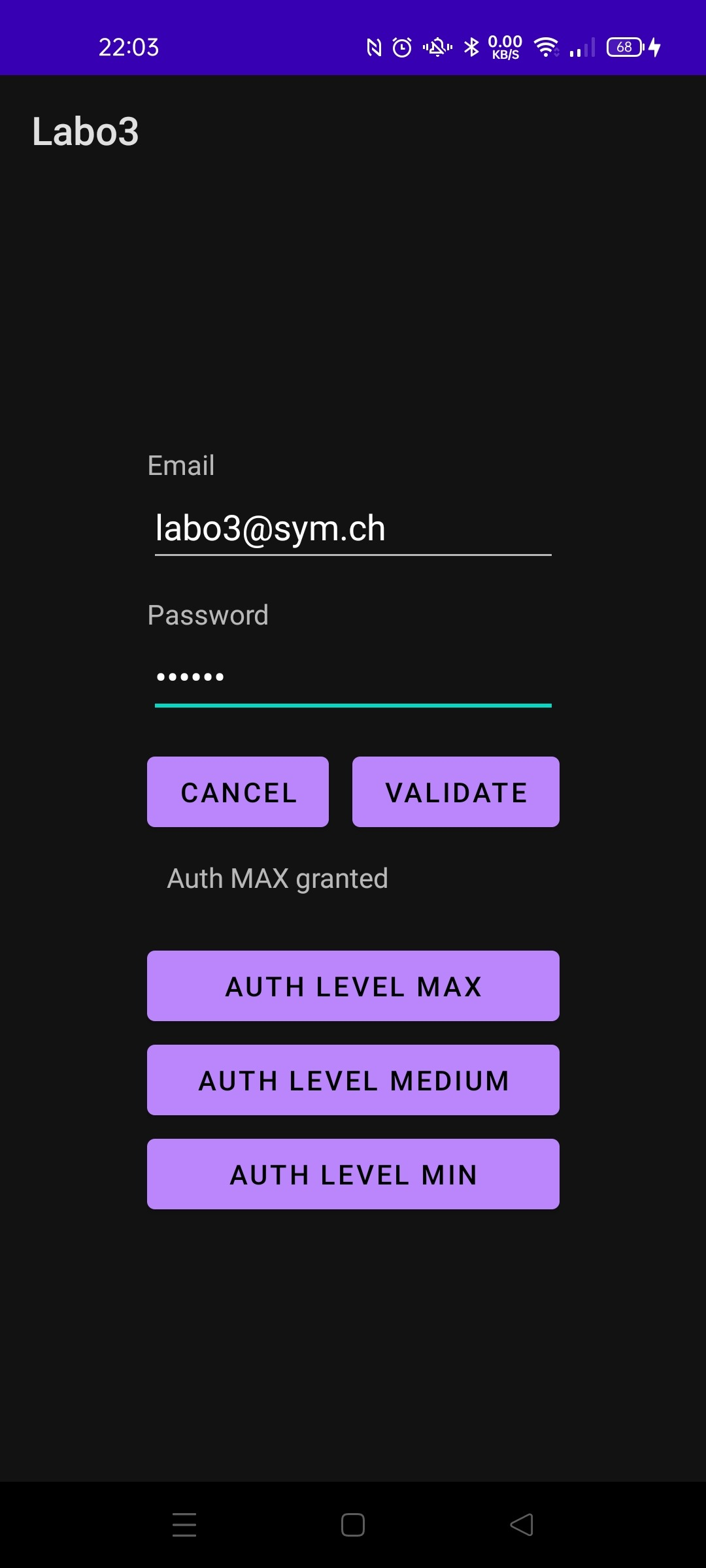
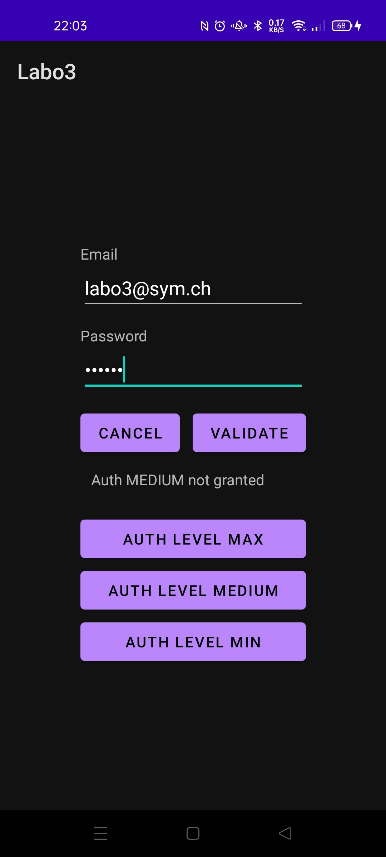


Figure 4 Affichage de la zone sécurisée

Les boutons affichent simplement sur une textview si l’accès est accordé ou pas. Les niveaux d’autorisation sont limités à :

* Level maximum : 5 secondes
* Level medium : 10 secondes
* Level minimum : 15 secondes

À chaque palier, un toast apparait pour signifier que les droits du niveau ont été perdus.

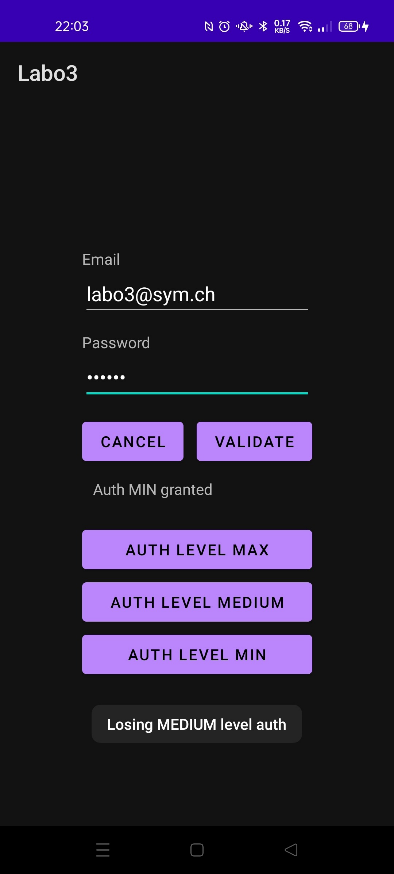


Figure 5 Accès refusé

Figure 6 Perte des droits d'un niveau d'accès

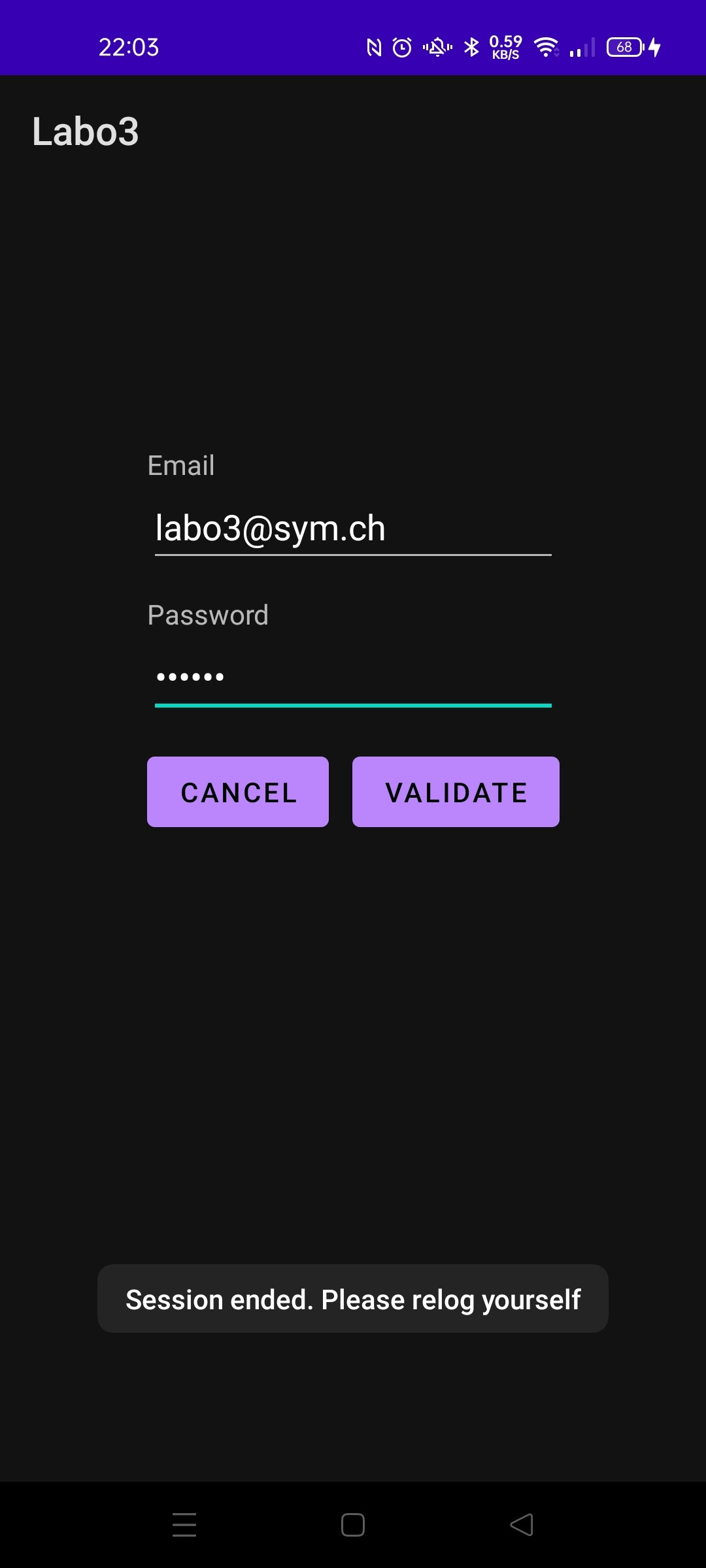


Figure 7 Demande de réauthentification

Une fois que le niveau minimal n’est plus accessible non plus, les boutons disparaissent et il est nécessaire de se relogger **complètement** avec le formulaire et la balise NFC.

### Mise en place

**Autoriser le NFC**

Dans le fichier AndroidManifest.xml, il faut rajouter les lignes suivantes :

<uses-permission android:name="android.permission.NFC" />  
  
<uses-feature  
 android:name="android.hardware.nfc"  
 android:required="true" />

**Traiter les Intent NFC**

Comme demandé dans la donnée du laboratoire, en utilisant la méthode « onNewIntent », il est possible de s’enregistrer afin d’être notifié lorsqu’un nouvel Intent arrive. De ce fait, nous pouvons gérer la lecture du Tag NFC.

override fun onNewIntent(intent: Intent?) {  
 super.onNewIntent(intent)  
 if (intent != null && logged) {  
 handleIntent(intent)  
 } else {  
 Toast.makeText(this, R.string.*nfc\_logged\_required*, Toast.*LENGTH\_LONG*).show();  
 }  
}

Ici, on détermine si l’action est de type NDEF, et si le MIME type est le bon. Si tout est valide, on lance la coroutine qui lira le tag.

private fun handleIntent(intent: Intent) {  
 val action = intent.*action* if (NfcAdapter.*ACTION\_NDEF\_DISCOVERED* == action) {  
 val type = intent.*type* if (MIME\_TEXT\_PLAIN == type) {  
 val tag: Tag = intent.getParcelableExtra(NfcAdapter.*EXTRA\_TAG*)!!  
 launchCoroutine(tag)  
 } else {  
 Log.d("NfcActivity", "Wrong mime type: $type")  
 }  
 }  
}

La coroutine retourne une liste de string, dans notre cas ce sont les 4 valeurs textuelles présentes sur le tag NFC. Si une des valeurs retournées appartient à la liste des Tags acceptés, alors on considère que l’utilisateur a accès à la zone sécurisée, et un timer se met en marche pour restreindre petit à petit l’accès à différents secteurs critiques.

private fun launchCoroutine(tag: Tag) {  
 GlobalScope.*launch* **{** val result = NfcReaderCoroutine().execute(tag).*getOrDefault*(*listOf*())  
  
 for (rTag in result) {  
  
 if (acceptedNfcTag.contains(rTag)) {  
 tagOk = true  
 runOnUiThread **{** textView.*text* = ""  
 setVisibilityAuthButtons(View.*VISIBLE*)  
 **}** countdown = MAX\_LOGGED\_TIME  
 timer.cancel()  
 timer.start()  
 return@launch;  
 }  
 }  
  
 **}**}

L’exécution de la coroutine s’effectue dans le context IO. Les valeurs sont lues dans le tag, puis retournées.

suspend fun execute(tag: Tag) : Result<List<String>>{  
  
 return withContext(Dispatchers.IO) **{** val ndef = Ndef.get(tag)?: return@withContext Result.failure(RuntimeException("NDEF is not supported by this Tag."))  
 val ndefMessage = ndef.*cachedNdefMessage* val records = ndefMessage.*records* var results = *mutableListOf*<String>()  
  
 for (ndefRecord in records) {  
 if (ndefRecord.*tnf* == NdefRecord.*TNF\_WELL\_KNOWN* && Arrays.equals(  
 ndefRecord.*type*,  
 NdefRecord.*RTD\_TEXT* )  
 ) {  
 try {  
 results.add(readText(ndefRecord))  
 } catch (e: UnsupportedEncodingException) {  
 Log.e("NdefReaderTask", "Unsupported Encoding", e)  
 }  
 }  
 }  
  
 return@withContext Result.success(results)  
 **}**}

### Question

**Dans la manipulation ci-dessus, les tags NFC utilisés contiennent 4 valeurs textuelles codées en UTF-8 dans un format de message NDEF [...] À partir de l’API Android concernant les tags NFC3 , pouvez-vous imaginer une autre approche pour rendre plus compliqué le clonage des tags NFC ? Existe-t-il des limitations ? Voyez-vous d’autres possibilités ?**

Il n’existe pas de protection contre le clonage pour le NDEF. Il existe cependant d’autres tags utilisant de la cryptographie, ou encore ayant un mécanisme anti-clonage. Une idée serait donc d’avoir un mot de passe pour pouvoir accéder aux données. Cette option n’est pas toujours possible, notamment si l’entrée du mot de passe n’est pas possible.

Une autre option serait d’utiliser un identifiant unique préprogrammé par le fabricant. Ces ID ne pourraient pas être modifiés, car le segment mémoire serait en read only. On pourrait donc garantir l’authenticité du tag, mais cela obligerait une connexion internet afin de connaitre les ID des tags valides. De plus, toutes les données pourraient toujours être extraites.

**Est-ce qu’une solution basée sur la vérification de la présence d’un iBeacon sur l’utilisateur, par exemple sous la forme d’un porte-clés serait préférable ? Veuillez en discuter.**

Le problème majeur est la portée d’un iBeacon. Si les paiements ne nécessitaient plus de contact, il serait alors très facile pour un attaquant d’effectuer une transaction en passant inaperçu.

Il est également simple de faire un spoofing avec le même ID du iBeacon, et donc de le cloner.

## Code-barre

### Utilisation

Au lancement l’interface ressemble à ceci :

Une image contenant texte

Description générée automatiquement

Figure 8 Activité code-barre - démarrage

La zone supérieure est la zone enregistrée par la caméra. Lorsqu’elle détecte un code barre ou un QR code, elle affiche la frame sur laquelle la détection a eu lieu ainsi que la valeur du scan.



Figure 9 Activité code-barre - scan réussi

### Mise en place

**Démarrage et permissions**

L’activité est démarrée à partir de la MainActivity. Avant le démarrage de l’activité, au clic sur le bouton pour l’ouvrir, nous vérifions si l’application a la permission d’utiliser la caméra.

codeBarreButton.setOnClickListener**{** *// Check for permissions, if acquired, starts CodeBarreActivity* checkPermission(Manifest.permission.*CAMERA*, CAMERA\_PERMISSION\_CODE)  
**}**

La fonction checkPermission, pour une permission donnée, effectue une demande à l’utilisateur pour valider l’accès à la permission demandée. Si la permission est déjà accordée, l’activité est démarrée.

private fun checkPermission(permission: String, requestCode: Int) {  
 if (ContextCompat.checkSelfPermission(  
 this@MainActivity, permission) == PackageManager.*PERMISSION\_DENIED*)

{  
  
 *// Requesting the permission* ActivityCompat.requestPermissions(this@MainActivity,   
 *arrayOf*(permission), requestCode)  
 } else {  
 if(requestCode == CAMERA\_PERMISSION\_CODE) {  
 startActivity(Intent(this, CodeBarreActivity::class.*java*))  
 }  
 }  
}

Dans le cas où la permission est demandée à l’utilisateur, nous avons override la méthode de callback utilisée lorsqu’une permission est accordée ou refusée. Si la permission est accordée, l’activité est démarrée.

override fun onRequestPermissionsResult(requestCode: Int,  
 permissions: Array<String>,  
 grantResults: IntArray) {  
  
 super.onRequestPermissionsResult(requestCode,

permissions,

grantResults)  
 if (requestCode == CAMERA\_PERMISSION\_CODE) {  
 if (grantResults.*isNotEmpty*() &&

grantResults[0] == PackageManager.*PERMISSION\_GRANTED*) {  
 startActivity(Intent(this, CodeBarreActivity::class.*java*))  
 }  
 }  
}

**CodeBarreActivity**

L’activité mise en place est simple. Elle fait intervenir trois composants :

* un DecoratedBarcodeView (barcodeView) pour afficher le décodeur de code barre
* une ImageView (imageView) pour afficher l’image capturée pour le décodage
* une TextView (result) pour afficher la valeur décodée.

À la création de l’activité, nous récupérons des références sur chacun des éléments puis nous initialisons le lecteur de codes-barres.

val formats: Collection<BarcodeFormat> =  
 Arrays.asList(BarcodeFormat.*QR\_CODE*, BarcodeFormat.*CODE\_39*)  
barcodeView.*barcodeView*.*decoderFactory* = DefaultDecoderFactory(formats)  
barcodeView.initializeFromIntent(*intent*)  
barcodeView.decodeContinuous(callback)

Lorsque le barcodeView décode une image, il appelle un objet callback pour gérer le résultat du décodage. C’est ce que fait la dernière ligne de code précédente.

Ainsi, nous avons défini l’objet callback de la manière suivante :

private val callback: BarcodeCallback = object : BarcodeCallback {  
 override fun barcodeResult(result: BarcodeResult) {  
 if (result.*text* == null || result.*text* == lastText.*text*) {  
 *// Prevent duplicate scans* return  
 }  
 lastText.*text* = result.*text* beepManager.playBeepSoundAndVibrate()  
  
 *//Added preview of scanned barcode* imageView.setImageBitmap(result.getBitmapWithResultPoints(Color.*YELLOW*))  
 }  
  
 override fun possibleResultPoints(resultPoints: List<ResultPoint>) {}  
}

Nous avons simplement override la méthode barcodeResult pour mettre à jour l’ImageView avec la frame qui a été capturé et pour afficher la valeur du décodage dans la TextView.

Finalement, les méthodes onResume, onPause et onKeyDown ont été override pour appeler les méthodes du même nom sur le barecodeView, typiquement pour arrêter l’enregistrement lorsque l’activité est mise en pause afin de ne pas consommer des ressources inutilement.

### Question

**Quelle est la quantité maximale de données pouvant être stockée sur un QR-code ? Veuillez expérimenter, avec le générateur conseillé de codes-barres (QR), de générer différentes tailles de QR-codes.**

En jouant avec le générateur, pour une correction d’erreur LOW, nous avons généré un QR code pour 2953 caractères

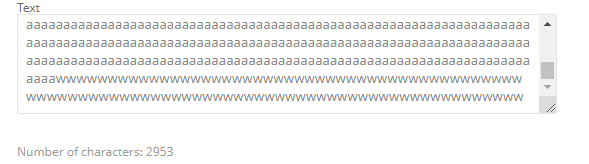
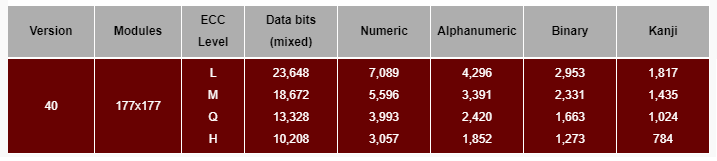


Figure 10 exemple de création de code QR - qté de caractères

Ce qui est curieux, lorsque nous regardons la [spécification](https://www.qrcode.com/en/about/version.html) des QR codes, nous apercevons que le 2953 précédent correspond à la quantité de données binaires pour la version la plus grosse des QR code.



En conclusion, en vue de la manipulation, nous définissons la taille maximale des QR code à 2953 caractères textuels. Attention, la quantité varie en fonction du taux de la correction d’erreur.

**Pensez-vous qu’il est envisageable d’utiliser confortablement des QRcodes complexes (par exemple contenant >500 caractères de texte, une vCard très complète ou encore un certificat Covid) ?**

En règle générale oui. Toutefois, faisons la nuance, car cela dépend de nombreux facteurs :

Premièrement la correction d’erreur influe. Au format vignette standard, un texte de 1000 caractères en LOW sera scannable tandis que le même texte en HIGH ne sera pas scannable, car trop détaillé.

|  |  |
| --- | --- |
| Figure 11 texte de 1000 caractères en LOW ECC | Figure 12 texte de 1000 caractères en HIGH ECC |

En admettant que la taille du QR code soit au format « vignette » traditionnelle (environ 300px sur 300px) et que l’ECC soit LOW, nous avons pu monter jusqu’à 1000 caractères. Aux alentours de 500 caractères aucun problème de scan. Cependant, aux alentours de 1000 caractères, il faut parfois s’y prendre 2 à 3 fois avant de décoder correctement le code.

Au-delà de 1000 caractères, le scanner n’arrive presque plus à décoder le code QR dans son ensemble, il en décode des fragments.

En conclusion, oui il est possible de travailler confortablement avec des QR codes avec plus de 500 caractères. Pour preuve, les certificats Covid sont déjà traités au format QR code dans l’application Covid Cert. Toutefois, il faut faire attention à ce que le nombre de modules (en largeur et hauteur) ne soit pas trop élevé pour permettre aux caméras de correctement les distinguer. Si les modules sont trop petits, il faut veiller à afficher le QR code aussi grand que possible pour faciliter l’analyse des caméras. En outre, il faut également prendre en considération l’éclairage et la qualité des caméras.

**Il existe de très nombreux services sur Internet permettant de générer des QR-codes dynamiques. Veuillez expliquer ce que sont les QR-codes dynamiques. Quels sont les avantages et respectivement les inconvénients à utiliser ceux-ci en comparaison avec des QR-codes statiques. Vous adapterez votre réponse à une utilisation depuis une plateforme mobile.**

Les QRcodes statiques encodent directement l’information dans le code QR, ainsi pour modifier le contenu, le code devra être regénéré et le pattern sera différent.

Les QRCodes dynamiques encodent une URL courte qui, une fois scannée, redirige l’utilisateur vers une URL avec son contenu. Ainsi pour modifier le contenu, on ne modifie pas l’URL et donc nous ne modifions pas le QRcode.

Avantages :

* Permet de changer le contenu sans avoir besoin de regénérer le QR code.
* Les QR codes dynamiques sont généralement plus courts que les QR codes statiques, ils sont donc plus facilement scannable.
* Possibilité de monitorer l’accès au contenu.

Inconvénients :

* Nécessitent une connexion à internet pour fonctionner. Cela implique l’utilisation de données mobiles. Parfois le réseau n’est pas bon, ainsi ce n’est pas forcément toujours optimal pour les téléphones (surtout si la page retournée est lourde).
* Augmentation de l’utilisation du réseau. Comme en général l’URL utilisée est une URL courte, cela implique une redirection. De plus les données ne sont plus décodées, mais téléchargées. Ainsi nous consommons plus de ressources du mobile pour récupérer le contenu.

En résumé, en tant qu’utilisateur mobile, les QR codes dynamiques sont plus facilement scannable. Toutefois, ils nécessitent plus de mouvement sur le réseau. Parfois, ce dernier n’est pas toujours optimal, ce qui peut rendre la récupération des informations lente, voire impossible.

## IBeacons

### Utilisation

Au lancement l’interface ressemble à ceci :

Une image contenant texte

Description générée automatiquement

Figure Activité iBeacon- démarrage

Chaque iBeacon détecté est séparé par un petit espace. Comme demandé dans la donnée, les informations suivantes sont visibles pour chaque iBeacon :

* L’UUID
* Le numéro majeur
* Le numéro mineur
* La force du signal / RSSI
* La date de dernière détection du iBeacon

### Mise en place

**Démarrage et permissions**

L’activité est démarrée à partir de la MainActivity. Avant le démarrage de l’activité, au clic sur le bouton pour l’ouvrir, nous vérifions si l’application a la permission d’utiliser la localisation lors de l’utilisation de l’application. La gestion des permissions est très similaire à celle réalisée pour l’activité des codes-barres. Nous avons juste modifié un peu la partie checkPermission pour ajouter un switch dans la partie else de la fonction. Comme ci-dessous :

private fun checkPermission(permission: String, requestCode: Int) {  
 if (ContextCompat.checkSelfPermission(  
 this@MainActivity, permission) == PackageManager.*PERMISSION\_DENIED*)

{  
  
 *// Requesting the permission* ActivityCompat.requestPermissions(this@MainActivity,   
 *arrayOf*(permission), requestCode)  
 } else {  
 when (requestCode){

CAMERA\_PERMISSION\_CODE -> {  
 startActivity(Intent(this, CodeBarreActivity::class.*java*))

}

FINE\_LOCATION\_PERMISSION\_CODE -> {  
 startActivity(Intent(this, BeaconActivity::class.*java*))

}  
 }  
 }  
}

Nous avons réitéré la même chose pour la fonction onRequestPermissionResult :

override fun onRequestPermissionsResult(requestCode: Int,  
 permissions: Array<String>,  
 grantResults: IntArray) {  
  
 super.onRequestPermissionsResult(requestCode,

permissions,

grantResults)

when (requestCode){  
 CAMERA\_PERMISSION\_CODE -> {  
 if (grantResults.*isNotEmpty*() &&

grantResults[0] == PackageManager.*PERMISSION\_GRANTED*) {  
 startActivity(Intent(this, CodeBarreActivity::class.*java*))  
 }

}

FINE\_LOCATION\_PERMISSION\_CODE -> {  
 if (grantResults.*isNotEmpty*() &&

grantResults[0] == PackageManager.*PERMISSION\_GRANTED*) {  
 startActivity(Intent(this, BeaconActivity::class.*java*))  
 }

}  
 }  
}

**BeaconAdapter**

Le *BeaconAdapter* est un adapter utilisant une *RecyclerView* et implémente donc toutes les méthodes nécessaires pour que cela fonctionne.

Nous utilisons un *LayoutInflater* qui sera lié à un second layout, ce dernier ayant pour but d’afficher les informations d’un iBeacon.

Nous avons ensuite créé une classe *ViewHolder* afin de bind les éléments de la *RecyclerView* et celle du layout.

Nous avons finalement réalisé une méthode *bindValues* afin d’attribuer les valeurs d’un iBeacon à la vue.

**BeaconActivity**

L’activité mise en place est simple. On utilise la libraire iBeacon donnée dans la donnée avec un des samples codes de la libraire un petit peu modifiée.

Nous avons commencé par créer un tableau de Beacon de la libraire fraichement ajoutée puis nous avons utilisé un sample code pour la suite.

Il s’agit du sample code « Ranging Example Code » qui nous permet d’observer en continu les iBeacons en périphérie, on met ensuite à jour la vue avec le beacon s’il était déjà présent dans cette dernière ou nous l’ajoutons si ce n’était pas le cas.

Nous avons bien sûr utilisé le format demandé et ce dernier est stocké dans une constante.

Nous avons bien entendu lié la *RecyclerView* et le *BeaconAdapter* mais aussi setup le *BeaconManager* de la librairie dans la fonction *onCreate*. Nous avons également ajouté un pop-up informatif.

### Question

**Les *iBeacons* sont très souvent présentés comme une alternative à *NFC*. Vous commenterez cette affirmation en vous basant sur 2-3 exemples de cas d’utilisations (use-cases) concrets (par exemple e-paiement, second facteur d’identification, accéder aux horaires à un arrêt de bus, etc.).**

Les iBeacons sont tout comme le NFC des transmetteurs d’informations à faible portée et c’est pour cela que ces derniers pourraient être perçus comme une alternative à NFC.

La portée des iBeacons reste cependant plus grande et offre donc plus / d’autres possibilités que le NFC.

Le coût de fabrication d’un iBeacon n’est également pas le même qu’une puce NFC. En effet, le iBeacon coûte nettement plus cher.

Les iBeacons impliquent également l’utilisation et le développement d’une application là où les puces NFC fonctionnent avec tous les téléphones de nature. Ce qui rajoute un certain coût aux iBeacons.

Il est également plus facile, pour une personne mal intentionnée de récupérer des données ou hacker un iBeacon qu’un système NFC qui possède un système de session et un chiffrement des données.

Les iBeacons et le système NFC sont plutôt complémentaires et non des rivaux ou une alternative. Le NFC sera meilleur pour certains cas et le iBeacon pour d’autres. Nous allons voir ceci grâce aux quelques exemples de cas ci-dessous.

Cas 1 : un système de paiement sans fil.

Pour un système de paiement « sans fil », il est préférable d’utiliser un système NFC qui est premièrement beaucoup plus sécurisé (comme expliqué plus haut) que les iBeacons, mais surtout, car NFC permet une authentification alors que le iBeacon pas. Une authentification pour un paiement est primordiale. NFC est donc bien meilleur est pas remplaçable par un iBeacon dans ce cas-là.

Cas 2 : musée guidé automatiquement

On parlerait ici d’une visite auto guidée sur son smartphone à la place de suivre un guide classique. Les iBeacons seraient parfaits pour ça, car il vous détectera à l’entrée de chaque nouvelle salle sans que vous ayez à vous soucier de les trouver. Ils pourraient ainsi vous donner un lien directement sur le guide audio de la salle. On pourrait essayer de faire la même chose avec un système NFC, mais l’utilisateur perdrait du temps à trouver la puce à chaque entrée de nouvelle salle, ce qui serait pénible. L’avantage de la range des iBeacons ressort sur ce point et est plus adapté.

Cas 3 : un système de guichet pour bus, train ou aéroport

On fait référence ici à un guichet classique de bus ou train où l’on se doit de présenter un billet. Une aide intéressante qui pourrait être mise en place, c’est la détection d’un téléphone et proposer un lien direct sur le billet de transport nécessaire. Dans ce cas-là, on pourrait très bien imaginer un iBeacon donnant un lien sur l’application de billets au téléphone ou un système NFC qui pourrait être utilisé pour également rediriger le téléphone sur un lien vers l’application de billets ou le billet. On a ici une sorte d’équivalence et une préférence s’impose.

Pour finir sur le sujet, nous pensons vraiment que ces deux technologies sont plus complémentaires / différentes qu’en concurrence. Il se peut cependant que certaines situations puissent envisager d’utiliser soit l’un soit l’un autre sans que l’un soit mieux adapté.