Les notifications push et IoT

Sommaire

neopixl. A SMILE GROUP COMPANY

1

Les Notifications

De l’art d’informer l’utilisateur que quelque chose se passe dans votre application

Qu’est-ce qu’une Notification?

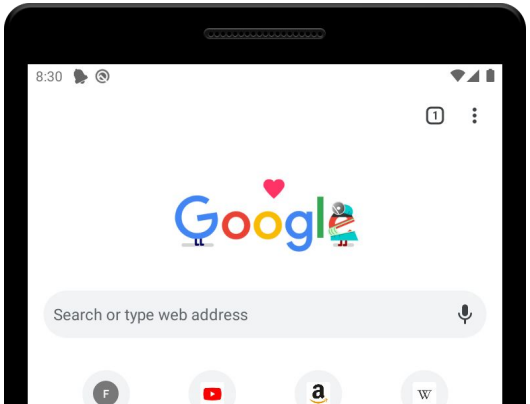
Message de l’OS pour fournir à l’utilisateur une information concernant une app (message reçu, nouvel article, document en téléchargement)

L’utilisateur peut utiliser la notification pour ouvrir l’app ou effectuer une action directe (e.g. répondre à un message ou mettre une musique en pause)

Les différents styles de notification

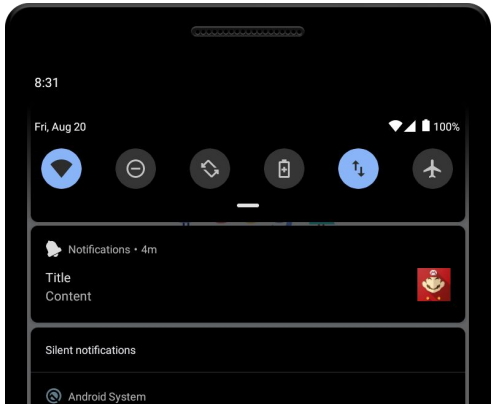
Icone dans la status bar

L'utilisateur peut swipe down pour ouvrir le notification drawer et voir ses notifications avec plus de détail



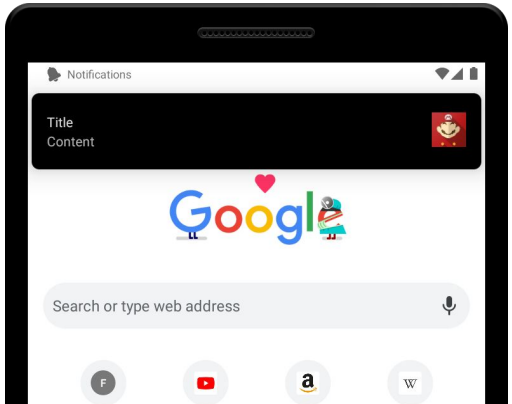
Notification étendue

Lorsque l'utilisateur ouvre le notification drawer, les notifications apparaissent de manière étendues avec plus de détail (e.g. reception d'un message : Titre de la notification, nom du contact, message, photo du contact, action répondre)



Notification d'alerte

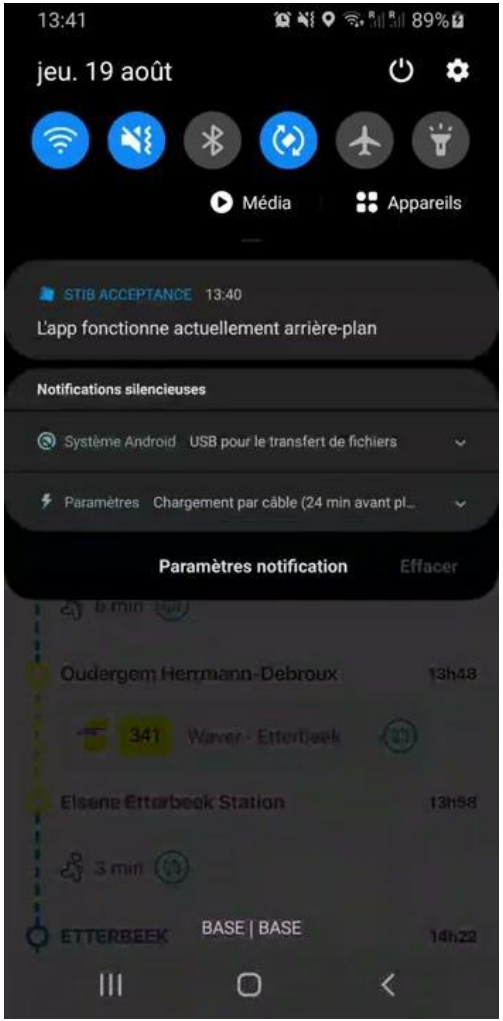
Des notifications, considérées comme importantes, peuvent apparaître par-dessus une une app en cours d'utilisation pendant un bref moment, même en cours de jeu 😡 (e.g. réception d'un message)



Dismiss une notification

Si l'utilisateur ne fait aucune action, les notifications restent visibles.
L'utilisateur peut décider de dismiss une notification en swipant sur celle-ci

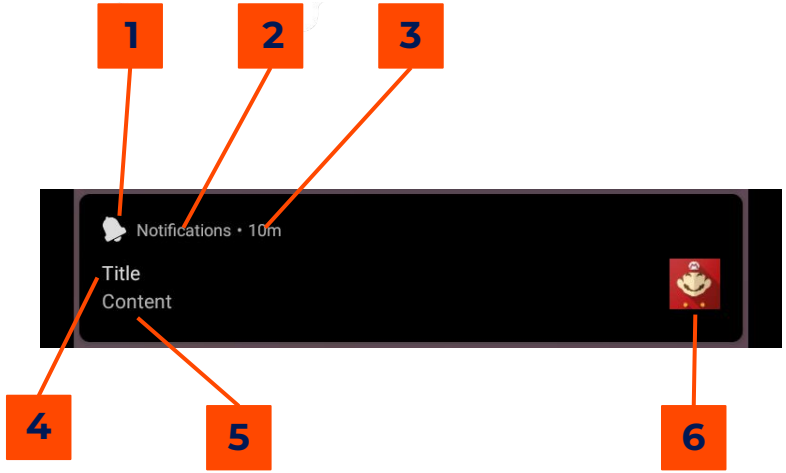
Attention : Certaines notifications ne peuvent être dismiss. Par exemple si une application effectue des tâches en background (e.g. une app localise le device en arrière-plan). Le seul moyen de dismiss la notification est d'arrêter le service qui s'exécute en background



Dissection d’une notification

Le design d’une notification est, par défaut, définie par l’OS, vous pouvez seulement changer le contenu de chaque partie de la notification. Mais il est possible d’utiliser un layout custom pour afficher une notification

1. Petite icône : icône apparaissant dans la status bar et dans une notification étendue. Obligatoire pour une notification. setSmallIcon()
2. Nom de l’app : Défini par l’OS
3. Horaire de réception : Défini par l’OS mais vous pouvez override cette information en utilisant setWhen() ou décider de la cacher avec setShowWhen(false)
4. Titre : Optionnel. setContentTitle()
5. Texte : Optionnel. setContentText()
6. Grande icône : Optionnelle. Peut-être par exemple une photo d’un contact ou l’image de l’album d’un titre en cours de lecture. setLargeIcon()



Permission

Depuis Android 13.0, une application doit demander la permission pour afficher des notifications.

Si votre application target android 12 ou inférieur, la demande de permission est faite automatiquement. **Attention**, si l'utilisateur refuse la permission, elle ne sera plus demandée sauf si vous mettez à jour votre application pour android 13, ou si l'utilisateur va lui-même dans les settings et change la permission.

Pour demander la permission, cela se fait en deux étapes :

1 - Déclarer la permission dans le manifest

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

2 - Demande de permission à l'exécution

```
when (ContextCompat.checkSelfPermission(requireContext(),
android.Manifest.permission.POST_NOTIFICATIONS)) {
    PackageManager.PERMISSION_GRANTED -> {
        //Permission granted
    }
    else -> {
        requestPermissionLauncher.launch(android.Manifest.permission.POST_NOTIFICATIONS)
    }
}
```

Permission

Récupérer le résultat de la permission :

```
private val requestPermissionLauncher =
    registerForActivityResult(ActivityResultContracts.RequestPermission()) { isGranted: Boolean ->
        if (isGranted) {
            //Permission granted
        } else {
            //Permission denied
        }
    }
```

Les Notifications Channel

Depuis Android 8.0, toutes les notifications doivent définir un channel pour être affichées.

Ce channel permet notamment de définir le niveau d'importance de votre notification

L'utilisateur peut alors choisir le niveau de détail et la façon de d'informer l'utilisateur d'une nouvelle notification (son, vibration) pour chaque "Notification channel"

Bien que vous décidiez de l'importance de votre notification, l'utilisateur peut décider d'override cette importance et la définir lui même

Sur Android 7, un seul channel est défini pour toute l'application, l'utilisateur peut donc désactiver les notifications seulement par application.

User-visible importance level	Importance (Android 8.0 and higher)	Priority (Android 7.1 and lower)
Urgent Makes a sound and appears as a heads-up notification	`IMPORTANCE_HIGH`	`PRIORITY_HIGH` or `PRIORITY_MAX`
High Makes a sound	`IMPORTANCE_DEFAULT`	`PRIORITY_DEFAULT`
Medium No sound	`IMPORTANCE_LOW`	`PRIORITY_LOW`
Low No sound and does not appear in the status bar	`IMPORTANCE_MIN`	`PRIORITY_MIN`

La naissance d'une Notification

Step 1 : Créer le Channel ID et le Notification ID

```
companion object{
    // Set a channel id only for Android 0 and above
    private val CHANNEL_ID = if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) "Channel ID" else null
    private const val NOTIFICATION_ID = 1
}
```

Step 2 : Créer le Channel

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

    val channel = NotificationChannel(CHANNEL_ID,  name: "Channel name", NotificationManager.IMPORTANCE_DEFAULT)
    val notificationManager: NotificationManager =
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.createNotificationChannel(channel)

}
```

La naissance d'une Notification

Step 3 : Créer la Notification

```
val builder = NotificationCompat.Builder( context: this, CHANNEL_ID!!)
    .setSmallIcon(android.R.drawable.ic_popup_reminder)
    .setContentTitle("Title")
    .setContentText("Content")
    .setLargeIcon(BitmapFactory.decodeResource(
        resources,
        R.drawable.mario))
    .setPriority(NotificationCompat.PRIORITY_HIGH)

NotificationManagerCompat.from( context: this).notify(NOTIFICATION_ID, builder.build())
```

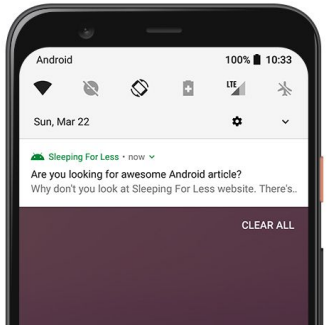
Expandception

Une notification étendue peut être étendue (encore une fois)

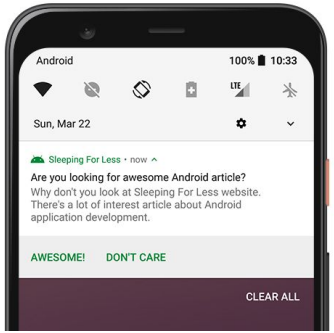
Lorsqu'une notification est étendue (affichage dans le notification drawer) celle-ci a une taille prédéfinie et les informations de votre notification peuvent donc être tronquées (content sur une seule ligne).
Vous pouvez alors activer une zone de texte plus large et l'OS affichera une flèche d'extension pour votre notification

```
    .setPriority(NotificationCompat.PRIORITY_HIGH)
    .setStyle(NotificationCompat.BigTextStyle())
```

Standard Template
Collapse



Big Text Template
Expand



Ex1

Exercice 1

Exercice 1

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo1

Package name: com.technipixl.exo1

Placez le dans un dossier TECHNIFUTUR-AND15-EXO1

Exercice 1

Dans cet exercice, vous allez devoir créer un layout contenant un bouton qui vous permettra de lancer la notification

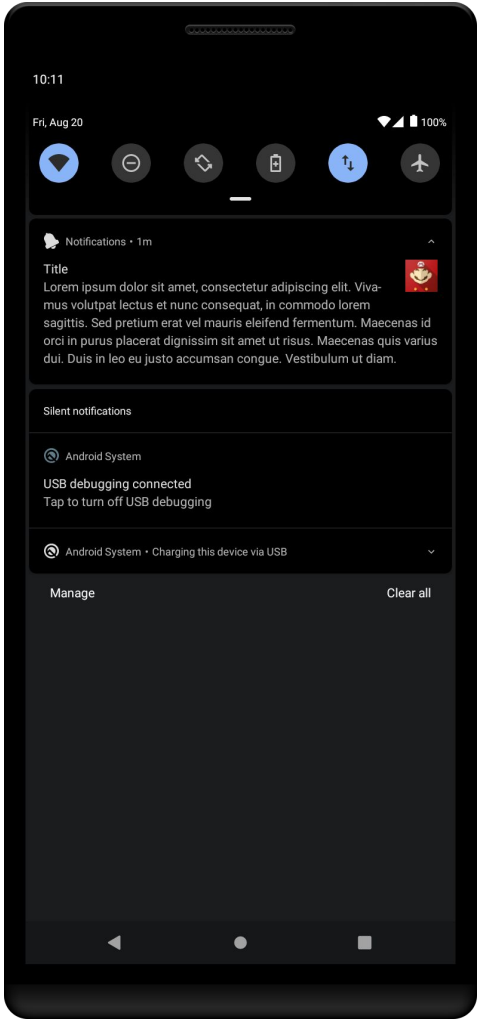
Lors du clic sur le bouton, lancez le processus de création d'une notification et vérifiez que celle-ci est bien affichée

La notification doit avoir une importance de type DEFAULT et contenir un titre, un contenu, et une large icon

Une fois la notification affichée, affichez cette même notification avec un long content et rendez la notification expandable

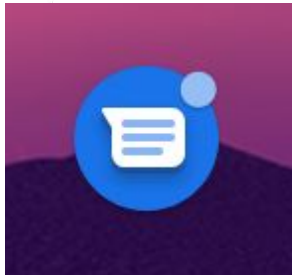
Vous pouvez, si vous le souhaitez changer l'importance pour observer les différents comportements pour chaque niveau

Attention : Si vous voulez augmenter le niveau d'importance d'une notification, il faudra, avant de lancer le build, désinstaller l'app ou changer le channel ID. L'OS ne supprime pas le Channel précédemment créé pour une nouvelle installation et réutilise celui déjà créé, et il est impossible d'augmenter, après sa création le niveau d'importance d'un channel



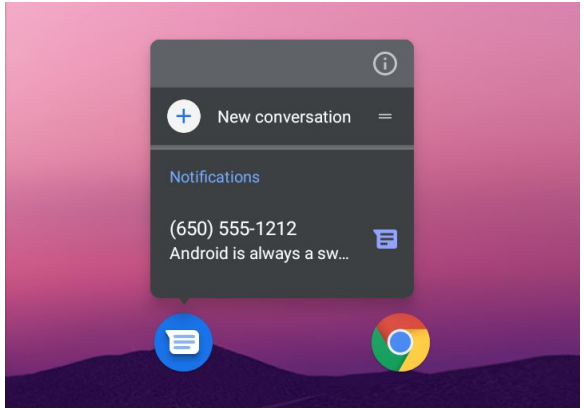
Badge de notification

Depuis Android 8.0, lorsqu'une notification est disponible pour une application, un point apparaît sur l'icône de l'application. L'utilisateur peut effectuer un long click sur l'icône et faire apparaitre les notifications disponibles pour cette app dans un menu contextuel et interagir de la même manière que dans le notification drawer.
Attention : Tous les constructeurs ne supportent pas cette feature



Vous pouvez lancer une notification avec l'app créée dans l'Exo1 et effectuer un long click sur l'icône de votre app pour observer ce qu'il se passe

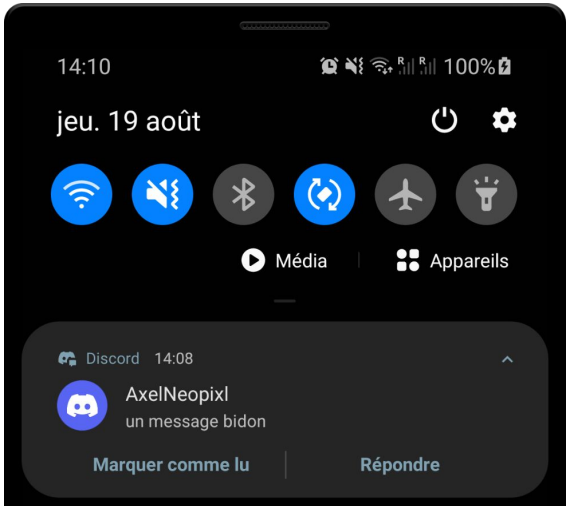
Conseil : Lancez l'application sur un émulateur ou sur un device physique utilisant l'OS sans surcouche



Actions d’une notification

Vous pouvez décider quelle activité sera appelée lorsque l’utilisateur cliquera sur la notification
Vous pouvez aussi définir des boutons d’action qui effectueront une tâche dans votre application et ce, sans même ouvrir cette application

Depuis Android 7.0, Il est possible d’ajouter une bouton répondre ou un champ texte
Depuis Android 10, l’OS peut afficher automatiquement des boutons d’action avec des suggestions basées sur des intents

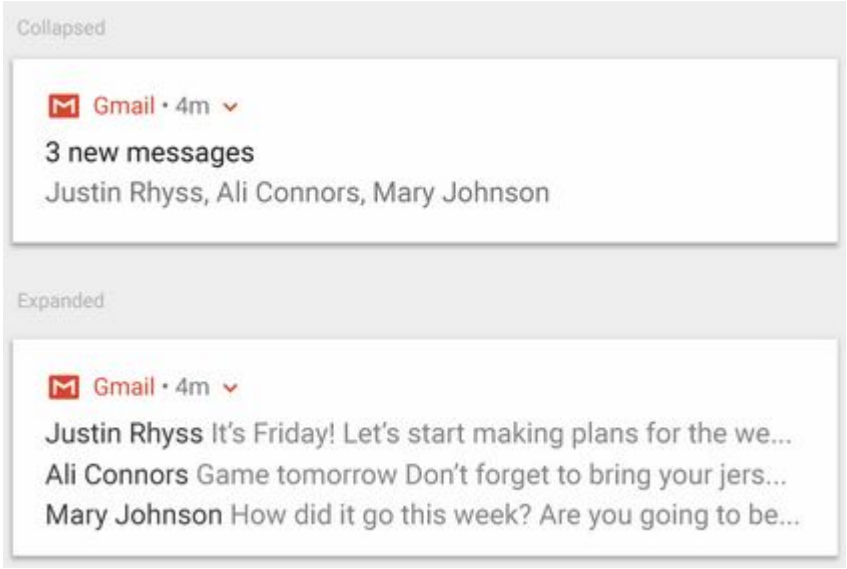


Mise à jour des notifications et groupement

Si votre application veut envoyer une notification alors que l'utilisateur n'a pas dismiss la précédente, vous pouvez mettre à jour la première notification avec la nouvelle information ou utiliser un InboxStyle afin d'afficher plusieurs lignes dans votre notification.
Ceci évite d'harceler l'utilisateur avec des dizaines de notifications

Lorsque vous définissez un InboxStyle, les lignes ajoutées à votre notification sont automatiquement gérées par l'OS qui les affiche. Sur une seule ligne de manière tronquée si l'information est trop grande

Attention : Il y a un nombre maximal de lignes affichées de 6. Si vous avez plus de 6 lignes ajoutées, seules les 6 premières seront affichées



Mise à jour des notifications et groupement

Pour update une notification, il suffit de créer une nouvelle notification et d'appeler Notificationmanager.notify() en lui passant le même notification id que celle dont vous souhaitez faire l'update

Mise à jour des notifications et groupement

Pour Créer une notification de type Inbox, vous devez tout d'abord créer un InboxStyle

```
val inboxStyle = NotificationCompat.InboxStyle()
```

Ensuite, il faut lui ajouter les différents contenues à l'aide de la méthode addLine()

```
val inboxNotificationContent = arrayListOf("Content 1", "Content 2", "Content 3", "Content 4")

inboxNotificationContent.forEach { it: String

    inboxStyle.addLine(it)

}
```

Puis, pour finir créer votre notification comme vu précédemment en lui ajoutant la InboxStyle à l'aide de la méthode setStyle() du builder

```
val builder = NotificationCompat.Builder( context: this, CHANNEL_ID!!)
    .setSmallIcon(android.R.drawable.ic_popup_reminder)
    .setContentTitle("Title")
    .setContentText("Inbox Content")
    .setLargeIcon(BitmapFactory.decodeResource(
        resources,
        R.drawable.mario))
    .setStyle(inboxStyle)
    .setPriority(NotificationCompat.PRIORITY_HIGH)

NotificationManagerCompat.from( context: this).notify(NOTIFICATION_ID, builder.build())
```

Ex2

Exercice 2

Exercice 2

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo2

Package name: com.technipixl.exo2

Placez le dans un dossier TECHNIFUTUR-AND15-EXO2

Exercice 2

Reprenez le code de votre premier exercice

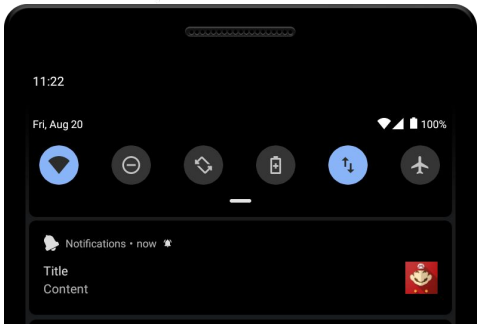
Ajoutez un bouton Update Notification dans le layout

Lors du clique sur ce bouton, vous devez recréer une nouvelle notification avec un contenu différent de la première et lancer cette nouvelle notification

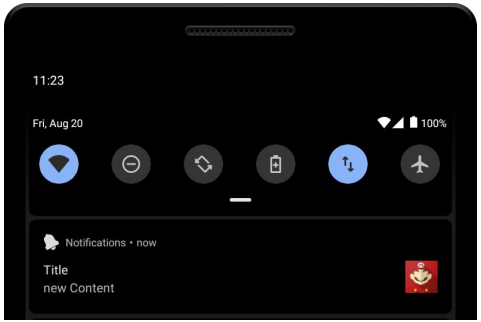
Lorsque l'app est lancée, cliquez sur le bouton "Launch Notification", la notification du premier exercice doit s'afficher, puis cliquez sur le bouton "Update", la notification précédente doit être mise à jour avec le nouveau contenu

Faites un dismiss de la notification, puis cliquez directement sur "Update", Que se passe-t-il? Quelle est la raison de ce comportement?

Before



After update



Ex3

Exercice 3

Exercice 3

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo3

Package name: com.technipixl.exo3

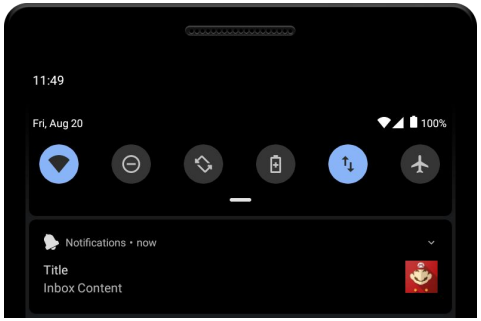
Placez le dans un dossier TECHNIFUTUR-AND15-EXO3

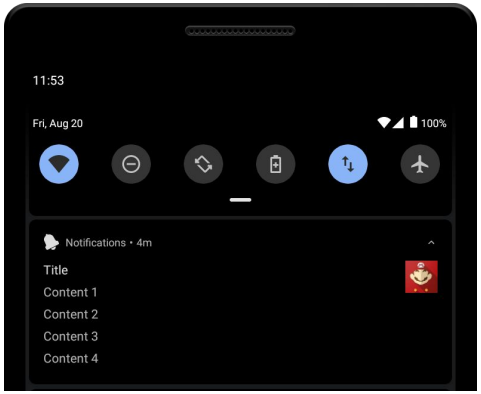
Exercice 3

Reprenez le code de votre deuxième exercice

Ajoutez un bouton Launch Inbox Notification dans le layout

Lors du clique sur ce bouton, vous devez créer une nouvelle notification avec un InboxStyle afin d'afficher plusieurs notifications groupées





Compatibilité

Il est possible d'accéder aux nouveautés concernant les notifications même sur les versions d'OS plus vieilles que celles pour lesquelles elles ont été créées en utilisant la support librairy ou AndroidX.
Pour cela, il faut utiliser NotificationCompat (ou une sous classe) au lieu de Notification.
Il est donc préférable d'utiliser NotificationCompat afin de faire bénéficier des nouveautés aux anciennes version d'OS.
Attention cependant, il n'y a aucune garantie qu'une nouvelle fonctionnalité soit disponible pour un ancien OS. Si ça fonctionne tant mieux, sinon tant pis 🤷. Mais ça ne fera pas crasher l'app pour autant.
e.g. Un bouton d'action, apparu sous Android 7.0 sera disponible pour les OS depuis 4.1 mais pas antérieur

2

Le NFC

De l’art d‘échanger des données sans contact

Présentation

Le NFC (Near Field Communication) permet d'effectuer une action lorsque le device se trouve à portée (10cm max) d'un objet supportant le NFC

Les Tags NFC sont des petits circuits, pouvant être programmé et lu par un appareil NFC.

Pas besoin de pile (énergie fournie par le lecteur/programmeur)

Le NFC se limite à 2 actions possibles avec un autre device:

- la lecture
- l'écriture

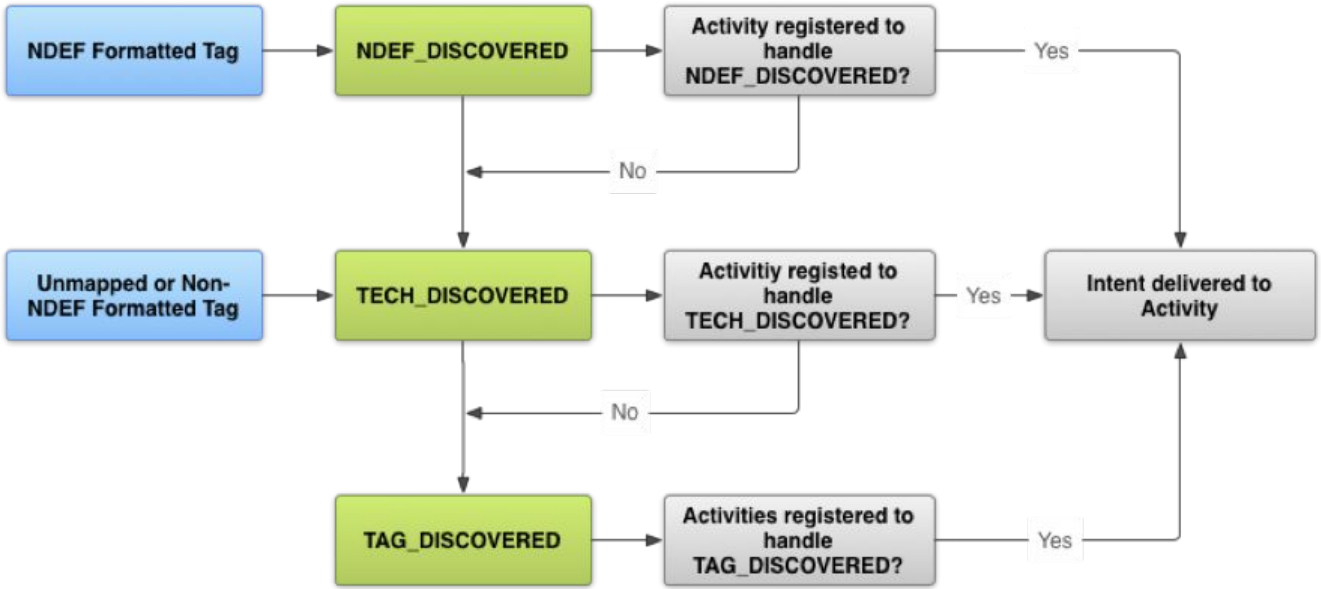
L’évolution du NFC sur Android

Disponible depuis **API 9 limité à de la lecture** simple (ACTION_TAG_DISCOVERED uniquement)

API 10 ajoute la lecture/écriture.

API 14 ajoute des méthodes pour **envoyer des données à un autre device**

À la découverte d'un TAG NFC



Préparation du Manifest

Permission nécessaire pour le NFC

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

Limiter la visibilité de l'application aux devices avec NFC sur le store

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

Filtrage d’intent

Pour pouvoir communiquer avec un device NFC, l’app doit définir un Intent Filter pour que le Tag Dispatch System (Fourni par android) sache que votre app est capable de traiter des données NFC et rediriger vers l’activité qui gère la lecture/écriture en NFC

Vous pouvez faire en sorte que votre application filtre les 3 types d’intent vus précédemment:

- NDEF_DISCOVERED
- TECH_DISCOVERED
- TAG_DISCOVERED

La plupart du temps, seul **NDEF_DISCOVERED** est nécessaire

NDEF_DISCOVERED

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>

<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="https"
        android:host="developer.android.com"
        android:pathPrefix="/index.html" />
</intent-filter>
```

TECH_DISCOVERED

Pour les Intent de type ACTION_TECH_DISCOVERED, on doit définir un filtre de technologie (dans le répertoire res/xml)

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

```xml
<intent-filter>
    <action android:name="android.nfc.action.TECH_DISCOVERED" />
</intent-filter>
<meta-data android:name="android.nfc.action.TECH_DISCOVERED"
    android:resource="@xml/nfc_tech_filter" />
```

TAG_DISCOVERED

```
<intent-filter>
    <action android:name="android.nfc.action.TAG_DISCOVERED"/>
</intent-filter>
```

Lecture d’un TAG

Lorsqu’une activité est lancée à l’aide d’un intent NFC, elle a la possibilité de récupérer les informations contenues dans le TAG via l’intent. 3 extras peuvent alors être lus :

- EXTRA_TAG (requis) : Un tag représentant le tag scanné
- EXTRA_NDEF_MESSAGES (optionnel mais obligatoire sur les intents de type ACTION_NDEF_DISCOVERED) : un tableau de message NDEF récupérée sur le tag
- EXTRA_ID (optionnel) : Un identifiant bas-niveau pour le tag

```kotlin
private fun readTag(intent: Intent) {

    if (NfcAdapter.ACTION_TAG_DISCOVERED == intent.action
        || NfcAdapter.ACTION_TECH_DISCOVERED == intent.action
        || NfcAdapter.ACTION_NDEF_DISCOVERED == intent.action) {

        val rawMessages = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES)

        if (rawMessages != null) {

            val messages: Array<NdefMessage?> = arrayOfNulls(rawMessages.size)

            for (i in rawMessages.indices) {
                messages[i] = rawMessages[i] as NdefMessage
            }

            fillTagContentView(messages)

        }

    }
}
```

Écriture sur un TAG

```
public override fun onCreate(savedInstanceState: Bundle?) {

    super.onCreate(savedInstanceState)

    setContentView(R.layout.activity_main)

    nfcAdapter = NfcAdapter.getDefaultAdapter( context: this)

    readTag(intent)

    pendingIntent = PendingIntent.getActivity(
        context: this,
        requestCode: 0,
        Intent( packageContext: this, javaClass).addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP),
        flags: 0
    )

}
```

Écriture sur un TAG

```
private fun enableWriteMode() {

    writeMode = true

    val tagDetected = IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED)

    writeTagFilters = arrayOf(tagDetected)

    nfcAdapter.enableForegroundDispatch( activity: this, pendingIntent, writeTagFilters,  techLists: null)

}
```

Écriture sur un TAG

```
private fun write(text: String, tag: Tag) {

    val records = arrayOf(NdefRecord.createMime( mimeType: "text/plain", text.toByteArray()))

    val message = NdefMessage(records)

    val ndef = Ndef.get(tag)

    ndef.connect()

    ndef.writeNdefMessage(message)

    ndef.close()

}
```

3

Les beacons

De l’art de localiser un utilisateur en intérieur?

Qu’est-ce qu’un beacon ?

Un beacon (balise) est un petit équipement électronique contenant une batterie et un émetteur BLE (Bluetooth Low Energy)
Pour utiliser des beacons, un device sous Android 4.3 minimum avec antenne BLE est nécessaire
Aucun appairement n’est nécessaire
Il suffit de le placer à un endroit précis et il emmétra son UUID (Universally Unique Identifier) en continu
Il permet de savoir si un device est entré dans la région de diffusion du beacon et d’effectuer une action lorsque celui-ci entre ou sort de la zone voire même de connaître la distance à laquelle se trouve le beacon
Il est même possible de "réveiller" le device à l’aide du beacon

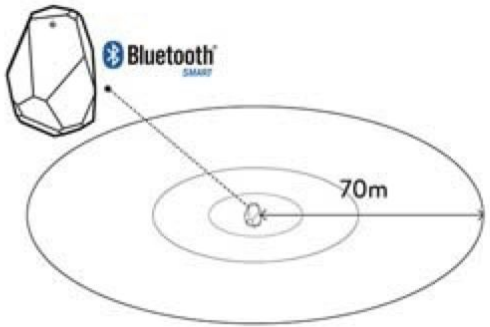
Les avantages du Beacon

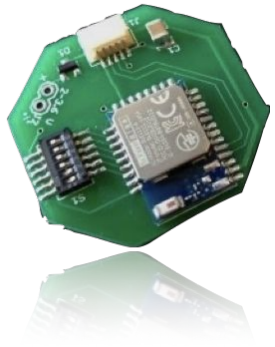
Un beacon est auto alimenté

Il utilise la technologie BLE pour communiquer avec un device, donc faible consommation de batterie

Il émet de façon continue sur une zone pouvant aller jusqu'à 70m

Puissance variable pour limiter la zone d'émission





Les cas d’utilisations

Réveiller son device à l’approche d’un Beacon

Afficher une information lorsqu’un utilisateur entre dans le rayon d’action d’un beacon

Effectuer un paiement

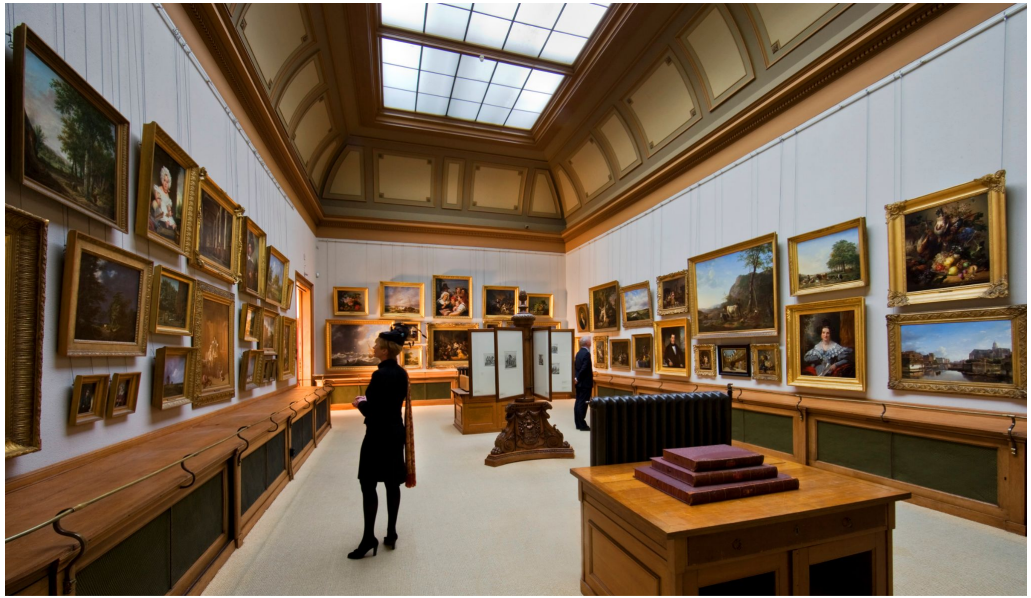
Traquer l’utilisateur de manière silencieuse (nécessite 3 beacons minimum pour effectuer une triangulation)

Exemple d’utilisation : Le musée

Un musée installe un beacon devant chaque œuvre d’art

L’app du musée permet de détecter les beacons

Lorsque l’utilisateur est proche d’une œuvre, l’app affiche un texte explique la concernant



Le Flop des Beacons

Depuis sa création par Apple et son intégration dans les Apple stores pour souhaiter la bienvenue aux clients lorsqu'ils rentraient, une hype autour des beacons s'est très vite installée mais est malheureusement retombé aussi vite.

Mais pourquoi ?

La grande promesse des beacons était de permettre une localisation indoor. Chose impossible avec un GPS et qui aurait permis de fournir tous les services liés à la localisation par GPS mais en intérieur. Quoi de mieux qu'une app qui vous dirige direction vers votre porte d'embarquement dans un aéroport qui vous est totalement inconnu ?

Cependant, une limitation technique à très vite vu le jour, le BLE est très sensible à son environnement, le moindre mur, meuble ou élément quelconque fait chuter drastiquement la portée du beacon ou sa précision de distance.

Le résultat, si l'on veut localiser un utilisateur avec des beacon, il faut non seulement les installer au plafond pour éviter un maximum d'obstacle mais leur faible précision fait que l'on doit en plus littéralement tapisser le plafond de beacons et effectuer un savant calcul afin d'exclure les données erronées

Il s'est avéré que le meilleur moyen de localiser un utilisateur en intérieur était simplement le wifi, bien que n'étant pas optimum et a donc limiter l'utilisation des beacons un l'entrée/sortie d'une zone

Le Flop des Beacons

Cependant, une limitation technique à très vite vu le jour, le BLE est très sensible à son environnement, le moindre mur, meuble ou élément quelconque fait chuter drastiquement la portée du beacon ou sa précision de distance.

Le résultat, si l'on veut localiser un utilisateur avec des beacon, il faut non seulement les installer au plafond pour éviter un maximum d'obstacle mais leur faible précision fait que l'on doit en plus littéralement tapisser le plafond de beacons et effectuer un savant calcul afin d'exclure les données erronées.

Il s'est avéré que le meilleur moyen de localiser un utilisateur en intérieur était simplement le wifi, bien que n'étant pas optimum et a donc limité l'utilisation des beacons un l'entrée/sortie d'une zone.

Le Flop des Beacons

D’autres techniques de localisation indoor basées sur l’intelligence artificielle et la réalité augmentée sont en développement.

Quand elles seront au point, elles permettront de localiser une personne sur base de la reconnaissance du lieu.

Exemple: la reconnaissance d’une couleur de mur combinée avec une décoration spécifique présente dans une pièce permettra de vous positionner dans cette pièce.

Evitez de changer trop souvent la déco…

4

AWS IoT Core

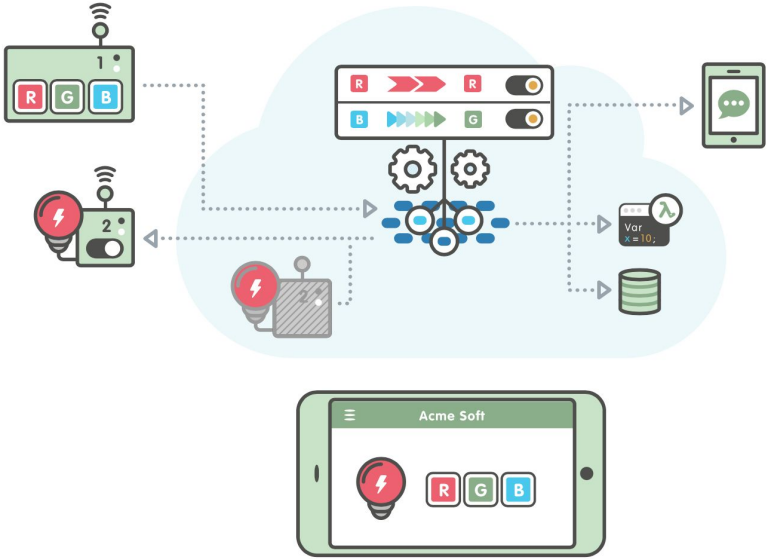
De l’art de l’interaction des services

A quoi ça sert?

La plateforme IoT Core d'Amazon est un service serverless AWS qui permet aux objets connectés d'interagir entre eux.

Il permet aussi d'interagir avec des apps qui peuvent en lire l'état et les contrôler à distance, via une connexion sécurisée (certificats).

Les données sont transmises entre objets, vers des applications, ou encore redirigés vers des services tiers.



Communication

En plus des protocoles classiques tels que HTTP et Websocket, le protocole recommandé (et le plus adapté pour IoT Core) est MQTT.

C'est un protocole très simple, consistant en de petits messages ciblés (JSON), publiés sur des topics.

Les différents objets et apps peuvent publier des messages dans des topics (Publish), ainsi que s'abonner à des topics (Subscribe).

Par ex., un capteur peut transmettre le relevé des données sur un topic représentant le lieu où il est placé (salon, terrasse, ...), et l'application.

Exemple de message MQTT

```
{
  "temperature": 21,
  "humidity": 60
}
```

Traitement des données

Des règles peuvent être appliquées par topic, afin de:

- Transformer les données
- Filtrer les messages reçus
- Stocker (en DB, sur DynamoDB par ex., ou en fichier, sur S3 par ex.)
- Rediriger vers différents services AWS (envoi d'une notification Push via SNS, invocation d'une fonction Lambda, analytics, machine learning, etc.)

Ces règles sont créées côté AWS, en utilisant un langage identique à SQL pour ce qui est de la sélection des messages à traiter.

Traitement des données

Exemple de règle enregistrant les messages reçus sur le *topic* “iot/test” en DynamoDB

```
{
  "sql": "SELECT * FROM 'iot/test'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [{
      "dynamoDB": {
          "tableName": "my-dynamodb-table",
          "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
          "hashKeyField": "topic",
          "hashKeyValue": "${topic(2)}",
          "rangeKeyField": "timestamp",
          "rangeKeyValue": "${timestamp()}"
      }
  }]
}
```

Gestion du mode déconnecté

Afin de gérer efficacement le mode hors-connexion d'un objet, on peut créer un "double" virtuel, appelé Device Shadow.

Cela permet de lire l'état d'un objet, même lorsqu'il n'est plus connecté, mais aussi de définir cet état (dans une app par exemple, via MQTT ou REST) pour les transmettre à l'objet lorsqu'il est de nouveau connecté.

Les apps ne communiquent donc jamais en direct avec l'objet pour changer son état, mais avec son shadow.

L'objet met à jour son état lorsqu'il se connecte et est en ligne, en écoutant les topics de son shadow.

Enfin, si l'app est déconnectée de son côté, l'objet continue à mettre à jour l'état de son shadow pour que l'app puisse y avoir accès lors de sa re-connexion.

Gestion du mode déconnecté

Exemple d'état d'un shadow

```
{
  "state": {
    "reported": {
      "color": "blue"
    }
  },
  "version": 9,
  "timestamp": 123456776
}
```

Exemple de demande de modification d'état

```
{
  "state": {
    "desired": {
      "color": "RED"
    }
  },
  "version": 10,
  "timestamp": 123456777
}
```

Intégration dans une app

Amazon propose des SDK haut niveau destinés à l'intégration dans une application mobile Android ou iOS

Ces SDK simplifient l'implémentation pour:

- L'authentification
- Les autorisations
- Le protocole de transfert (ex: MQTT)
- L'émission et la souscription d'états

Lien du SDK Android:
https://github.com/aws-amplify/aws-sdk-android

Plus de détails par ici:
https://docs.amplify.aws/start/getting-started/integrate/q/integration/android/#configure-amplify-and-datastore

Restons en contact

neopixl.

A SMILE GROUP COMPANY

115A, Rue Emile Mark
L-4620 Differdange

(+352) 26 58 06 03
contact@neopixl.com