Les intents

neopixl.

A SMILE GROUP COMPANY

Welcome on board

Sommaire

1

Principe de fonctionnement

Présentation

- La communication au sein d'Android est basée sur l'envoi et la réception de messages exprimant l'**intention d'une action**
- **Action** = description abstraite d'une opération à effectuer
- Chaque **message** peut être émis à destination d'un autre composant de la même application (activité, service, …) ou même d'une autre application

Mode d’envoi

Le choix d’un composant cible peut être décidé au moment de l’exécution et non à la compilation (ex : ouverture d’une page web -> navigateur par défaut ou choix du navigateur)

Deux modes d’envoi des Intents:

- **explicite** : avec un composant cible précis d’une application
- **implicite** : on laisse le système déléguer l’Intent au composant le plus approprié

Types d’utilisation

Il existe 3 types d‘utilisation principales

- Démarrer une activité dans l’application
- Solliciter d’autres applications
- Envoyer des informations

Types d’utilisation

Démarrage d’une activité au sein d’une application

Navigation entre les écrans ou affichage boîte de dialogue => mode explicite

Dans ce cas d’utilisation nous devrons spécifier explicitement le composant (ex: Activity) à afficher

Types d’utilisation

L’application peut aussi solliciter une autre application pour une action ou pour lui fournir une information

- Demander à une application de navigation GPS de guider l’utilisateur vers une destination
- Demander à une application de téléphonie d’appeler un numéro
- Demander à une application de partager un contenu

Il est possible de solliciter une application de façon explicite en précisant son identifiant (package name)

Mais aussi de façon implicite:

Le système peut trouver l’application et le composant approprié = résolution de l’Intent

Types d’utilisation

Autres types d’utilisation:

- Un intent peut aussi être utilisé pour démarrer un Service:
  Nécessaire pour les applications fonctionnant en arrière-plan.

- Possibilité aussi d’envoyer un intent à plusieurs applications (ex: Intent pour indiquer que la batterie est défaillante).
  Filtre nécessaire et déclaré dans les applications afin de ne pas recevoir tous les Intents. Dans ce cas on parlera de broadcast.

Création

Composition de l'objet Intent:

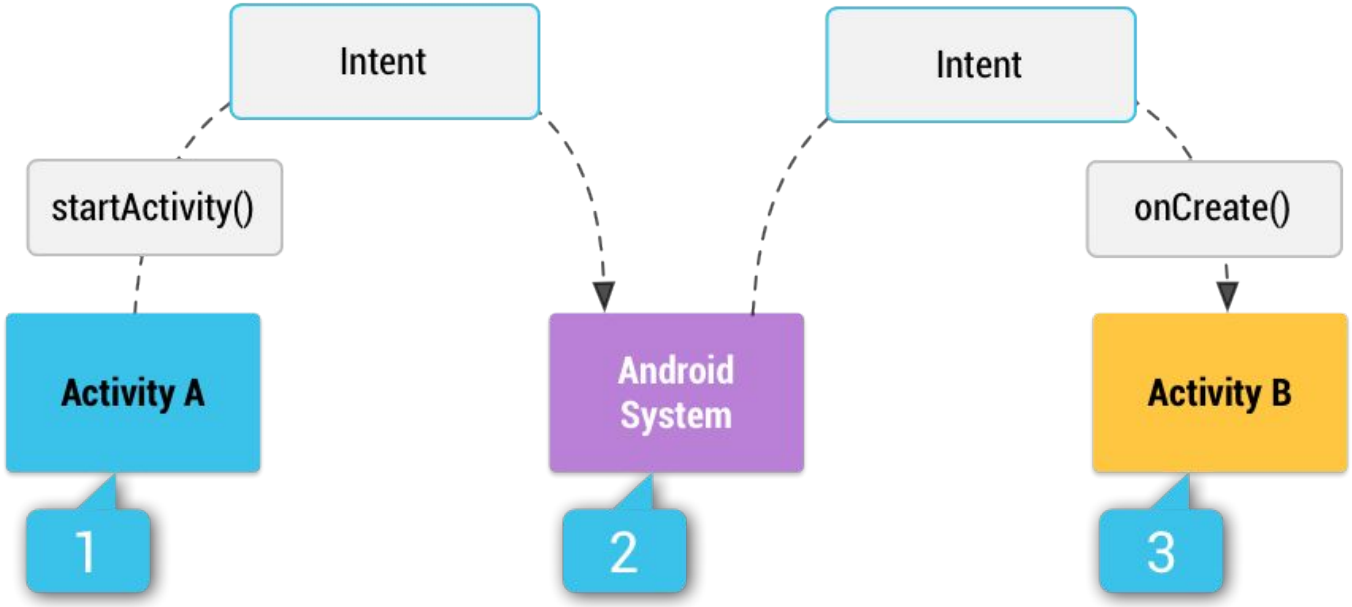
- **Nom du composant cible** : donnée facultative, si non précisé alors le système déterminera le composant le plus approprié.
- **Action** : chaîne de caractère définissant l'action à réaliser.
- **Données**: type de contenu MIME (chaîne de caractères) + contenu ciblé (URI) (ex : affichage page web = action ACTION_VIEW et URI http://www.google.com)
- **Données supplémentaires** : collection clef/valeur pour envoyer des paramètres supplémentaires.
- **Catégorie**: information complémentaire précisant qui devra gérer l'Intent émise (ex : CATEGORY_BROWSABLE : traitement par un navigateur)
- **Flags** : utilisé pour spécifier comment une activité doit être démarrée par le système (ex : FLAG_ACTIVITY_NO_ANIMATION)

2

Navigation entre écrans

Démarrage d'activité

Démarrage simple d'une activité.



Démarrage d’activité

Démarrage simple d’une activité (explicite).

Le constructeur Intent prend 2 paramètres:

- **Context** : contexte à partir duquel l’Intent sera créé et envoyé (en général l’activité courante)
- **Class<?>** : type de classe héritant de la classe Activité

```
val intent = Intent( packageContext: this, DetailActivity::class.java)
startActivity(intent)
```

Démarrage d’activité

Démarrage simple d’une activité (explicite).

- Attention: L’activité doit être définie dans le fichier manifest !
- Si vous utilisez l’outil de création d’activité, Android Studio l’ajoutera pour vous.

```
<activity android:name=".DetailActivity"/>
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Démarrage d’activité

Démarrage d’une activité avec un retour.

Parfois nécessaire de récupérer des informations de l’activité “enfant” (ex: récupérer quel bouton a été cliqué)

Utilisation de la méthode startActivityForResult

Lorsque l’activité “enfant” aura fini son cycle de vie, elle renverra à l’activité “parente” une valeur.

Démarrage d’activité

Démarrage d’une activité avec retour.

En plus du contexte et du type d’activité, nous devrons définir une constante (identifiant de notre requête)

Exemple: depuis la MainActivity, je veux ouvrir une activité de détail et attendre un résultat

```
companion object {
    const val DETAIL_ACTIVITY_REQUEST_CODE = 100
}

private fun displayDetail() {
    val intent = Intent( packageContext: this, DetailActivity::class.java)
    startActivityForResult(intent, DETAIL_ACTIVITY_REQUEST_CODE)
}
```

Démarrage d’activité

Renvoyer une valeur en retour.

Utilisation de la méthode **setResult(*int* resultCode)** depuis l’activité enfant

Il existe des constantes:

- RESULT_OK
- RESULT_CANCEL

Vous pouvez renvoyer vos propres codes tant que leur valeur est >= 1

Démarrage d'activité

Renvoyer une valeur en retour.

Vous pouvez aussi renvoyer des données en plus du code.
Pour celà vous pouvez mettre à jour votre Intent en lui ajoutant les données nécessaires.
Utilisation de la méthode **setResult(*int* resultCode, Intent data)**

```
companion object {
    const val RESULT_KEY = "result_value"
}

fun okClicked(view: View) {
    intent.putExtra(RESULT_KEY,  value: "Nice, you clicked the OK button")
    setResult(RESULT_OK, intent)
    finish()
}

fun cancelClicked(view: View) {
    intent.putExtra(RESULT_KEY,  value: "So sad, you clicked the cancel button")
    setResult(RESULT_CANCELED, intent)
    finish()
}
```

Démarrage d’activité

Traitement du résultat dans l’activité parente.

Il faudra surcharger la méthode onActivityResult

Cette méthode a trois paramètres:

- **requestCode**: identifiant de la requête
- **resultCode**: résultat de la requête (ex: OK, CANCEL)
- **data**: données supplémentaires fournies par l’activité enfant (dictionnaire clés/valeurs)

Démarrage d’activité

Traitement du résultat dans l’activité parente.

Vérification du code de la requête (cette requête est-elle bien la mienne?)

Récupération des données et traitement du résultat (ici afficher un Toast)

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    when(requestCode) {
        DETAIL_ACTIVITY_REQUEST_CODE -> {
            val message = data?.getStringExtra(DetailActivity.RESULT_KEY)

            when(resultCode) {
                RESULT_OK -> Toast.makeText( context: this,  text: "Success: $message", LENGTH_LONG).show()
                RESULT_CANCELED -> Toast.makeText( context: this,  text: "Error: $message", LENGTH_LONG).show()
                else -> Toast.makeText( context: this,  text: "Question: Have you tried turning it off and on again ?", LENGTH_LONG).show()
            }
        }
    }
}
```

Démarrage d'activité

La nouvelle façon de démarrer une activité et attendre un résultat:

- **registerForActivityResult()**
- Request code n'est plus nécessaire
- Plus nécessaire de surcharger la méthode onActivityResult()

```
// Variable de classe
var detailLauncher =
   registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {

            val message = it.data?.getStringExtra(DetailActivity.RESULT_CODE)

            when (it.resultCode) {
                RESULT_OK -> {
                    displayMessage(message, true)
                }
                RESULT_CANCELED -> {
                    displayMessage(message, false)
                }
            }
        }
```

Démarrage d'activité

La nouvelle façon de démarrer une activité et attendre un résultat:

- On n'appelle plus directement startActivityForResult
- On exécute notre launcher grâce à sa méthode launch()

```
binding.detailButton.setOnClickListener {
    val intent = Intent(this, DetailActivity::class.java)
    detailLauncher.launch(intent)
}
```

Solliciter d’autres applications

L’envoi d’un Intent permet aussi de demander à un composant d’une autre application de traiter l’action à réaliser

Le système a la charge de décider quelle application est la plus appropriée pour l’action en se basant sur l’action, les données (URI et le type de contenu MIME) et la catégorie.

Avantage : pas de dépendance vers d’autres applications, association action/application faite à l’exécution

Solliciter d’autres applications

En plus des informations utilisées pour la résolution de l’Intent : action, données (URI et type de contenu MIME) et la catégorie, le système va utiliser les filtres d’Intent pour trouver le composant le plus approprié.

Plus d’informations sur les filtres d’intent :
http://developer.android.com/guide/components/intents-filters.html#Receiving

Composer un numéro de téléphone

Notre application ne peut pas composer directement un numéro de téléphone, nous allons donc demander au système de trouver l’application la plus appropriée en utilisant un Intent

Composer un numéro de téléphone

Il faut donc créer un objet Intent en spécifiant :

- Le type d'action (ex : ACTION_DIAL)
- Une valeur complémentaire (une URI avec le numéro de téléphone)

Le système trouvera donc en fonction des filtres d'Intents déclarés par les applications, le composant le plus approprié pour répondre à cet Intent

```
private fun displayDialer(phoneNumber: String) {
    val uri =  Uri.parse( uriString: "tel:$phoneNumber")
    val intent = Intent(ACTION_DIAL, uri)
    startActivity(intent)
}
```

Afficher une page dans le navigateur web

Il faudra créer un objet Intent en spécifiant :

- Le type d'action (ex : ACTION_VIEW)
- Une valeur complémentaire (une URI avec l'adresse de la page à afficher)

```
val uri = Uri.parse( uriString: "https://www.google.com/")
val intent = Intent(Intent.ACTION_VIEW, uri)
startActivity(intent)
```

Partager du contenu avec une autre application

Il faudra créer un objet Intent en spécifiant :

- Le type d'action (ex : ACTION_SEND)
- Ajouter un extra (ex: EXTRA_TEXT)
- Définir son type (ex: text/plain)
- Utiliser un **Intent Chooser**

```
val intent = Intent(Intent.ACTION_SEND)
intent.type = "text/plain"
intent.putExtra(Intent.EXTRA_TEXT, value: "Hi mum, what's up?")
val chooser = Intent.createChooser(intent, title: "Share with...")
startActivity(chooser)
```

Les types d'actions

Action	Définition
ACTION_ANSWER	Prendre en charge un appel entrant.
ACTION_CALL	Appeler un numéro de téléphone
ACTION_DELETE	Démarre une activité permettant de supprimer une donnée
ACTION_DIAL	Interface de composition d'un numéro
ACTION_EDIT	Editer une donnée
ACTION_SEARCH	Activité de recherche (paramètre SearchManager.QUERY)

Les types d'actions

Action	Définition
ACTION_SEND	Envoyer des données (texte ou binaire) par mail ou SMS.
ACTION_SENDTO	Activité capable d'envoyer un message au contact précisé par l'URI
ACTION_VIEW	Démarre une activité permettant de visualiser un élément identifié par une URI (http: -> web, tel: -> composition numéro, geo: -> GoogleMaps...)
ACTION_WEB_SEARCH	Effectuer une recherche sur internet.

Filter les actions

Vous pouvez aussi configurer votre application afin qu'elle puisse être sollicitée par d'autres applications

Vous devrez déclarer une filtre d'Intent dans le fichier manifest pour les activités concernées

```
<activity android:name=".SomeActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="myapp"/>
    </intent-filter>
</activity>
```

Filter les actions

action : identifiant unique (String).

category : premier niveau de filtrage de l'action. La catégorie indique dans quelle circonstance l'action va s'effectuer ou non (plusieurs balises category possibles).

data : filtre directement au niveau des données (Ex : vérification du scheme ou de l'host avec android:host).

Filter les actions

Votre application sera désormais capable de réagir à une action de type View pour une Uri respectant votre Scheme (ex: myapp)

Exemple d'appel à votre application depuis une autre application

```
val uri = Uri.parse( uriString: "myapp://somevalue")
val intent = Intent(Intent.ACTION_VIEW, uri)
startActivity(intent)
```

Exploiter l'Intent

L'application sollicitée aura la possibilité de lire l'Intent ses données

Dans la méthode onCreate:

- Récupérer l'Intent
- Lire ses données
- Effectuer le traitement approprié

Dans notre exemple, notre Intent contient une Uri (data)

Cette Uri a pour valeur **myapp://somevalue**

Et son scheme est **myapp**

```
class SomeActivity : AppCompatActivity() {

    private val TAG = "SomeActivity"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_some_activity)

        intent.data?.let { uri ->
            Log.d(TAG, msg: "uri: ${uri.scheme}")
        }
    }
}
```

+ {Uri$StringUri@9621} "myapp://somevalue"

4

Aller plus loin avec les intents

Les extras

Fournir des données supplémentaires:

- Parfois lors d'une demande de réalisation d'une action à un autre composant, il est nécessaire de préciser des données supplémentaires

  ex : ouvrir une activité avec un navigateur, l'adresse du site à consulter sera requise

- Deux méthodes disponibles dans la classe Intent : putExtra et getXXXXExtra (cf : doc) lié à un objet de type Bundle

Les extras

Fournir des données supplémentaires

- Echange de données via des couples clef/valeur
- Utilisation de la méthode putExtra()

```
val intent = Intent( packageContext: this, EditActivity::class.java)
intent.putExtra( name: "some_key",  value: "some text value")
startActivity(intent)
```

Les extras

Plusieurs variantes sont disponibles permettant de gérer différents type de données

Les extras

Lire les données de l'intent dans l'activité lancée par ce dernier

En fonction du type de données à récupérer, vous devrez utiliser la bonne variante de getXXXExtra()

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityEditBinding.inflate(layoutInflater)
    setContentView(binding?.root)

    val value = intent.getStringExtra( name: "some_key")
```

ex1

Exercice

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-AND5-EXO1

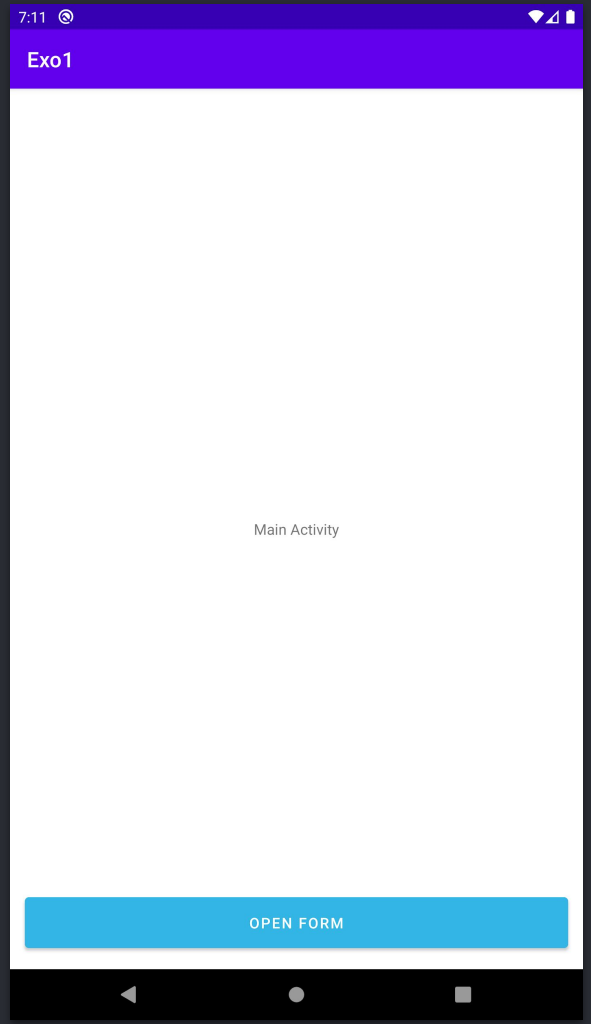
Exercice 1

Ajoutez deux nouvelles activités:

- FormActivity
- EditActivity

MainActivity:

Ajoutez un bouton au layout de votre MainActivity, il devra ouvrir la FormActivity



Exercice 1

FormActivity:

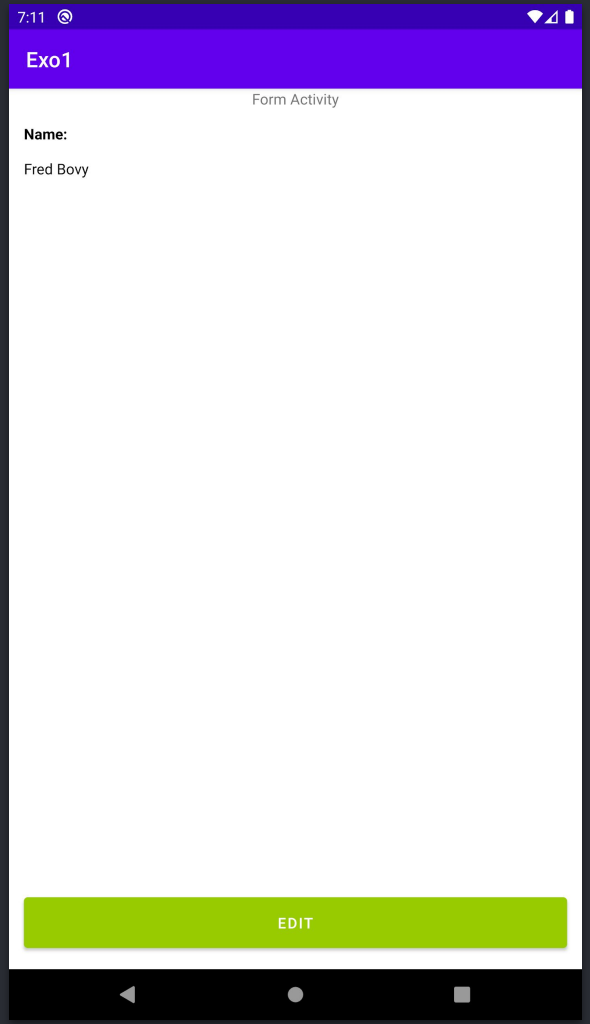
Ajouter deux TextViews, la première affichera le titre “Nom:”, le second affichera la valeur du “Nom”

Ajoutez un bouton au layout de votre FormActivity, il devra ouvrir la EditActivity avec une demande de résultat.

Il faudra passer à l’Intent la valeur initiale du “Nom” (ici: Fred Bovy)

Ajoutez aussi le code nécessaire à la récupération du résultat de EditActivity

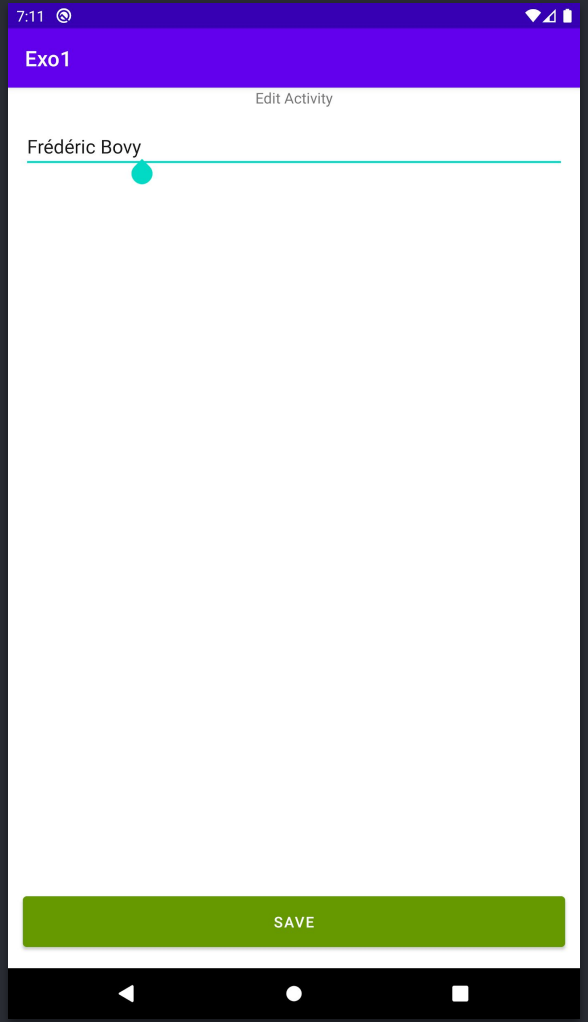
Vous devrez vérifier le requestCode et resultCode afin de mettre à jour le texte de votre TextView si nécessaire.



Exercice 1

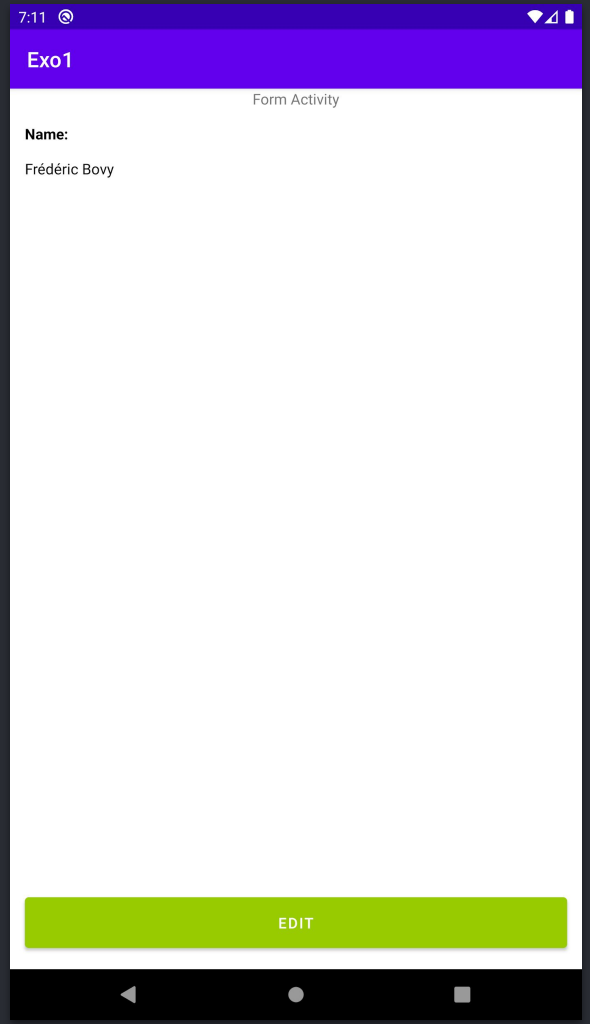
EditActivity:

Dans le layout de EditActivity ajoutez un bouton “Save”, il devra ajouter la valeur saisie pour le “Nom” dans votre Intent et fermer l’écran avec le resultCode RESULT_OK



Exercice 1

Résultat après édition lors du retour sur votre FormActivity



Transférer un Intent

Parfois à la lecture des données de l'Intent vous pouvez vous rendre compte que vous ne pouvez pas le traiter dans votre application

Exemple:
Vous développez une application de navigation qui permet de naviguer de votre position à une adresse

L'application qui vous a sollicité a demandé une navigation mais vers des coordonnées (latitude, longitude)

Dans l'état actuel de développement de votre application, vous ne gérez pas encore ce cas

Vous pouvez passer la main à une autre activité capable de le gérer grâce à la méthode **startNextMatchingActivity()**

Dans cet exemple, nous considérons que votre application de navigation est capable de gérer une Intent Uri de scheme "**geo**" qui contiendra un paramètre "**address**" dans la partie query

```
val uri = intent.data
if(uri?.query?.
        contains( other: "address",  ignoreCase: true) == true) {
    //Do what you need to do to handle the request
} else {
    startNextMatchingActivity(intent)
}
```

Transférer un Intent

Exemple d'intent appelé par une application tierce pour une demande de navigation par coordonnées latitude et longitude

Ce format est pris en charge par Google Maps, Waze...

Vous remarquerez que cette Uri contient un paramètre query (**q**) et une option permettant de lancer la navigation (**navigate**)

```
val uri = "geo:${latitude},${longitude}?q=${latitude},${longitude}&navigate=yes"
val intent = Intent(Intent.ACTION_VIEW, Uri.parse(uri))
val chooser = Intent.createChooser(intent,  title: "Naviguer vers ma destination")
startActivity(chooser)
```

ex2

Exercice

Exercice 2

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo2

Package name: com.technipixl.exo2

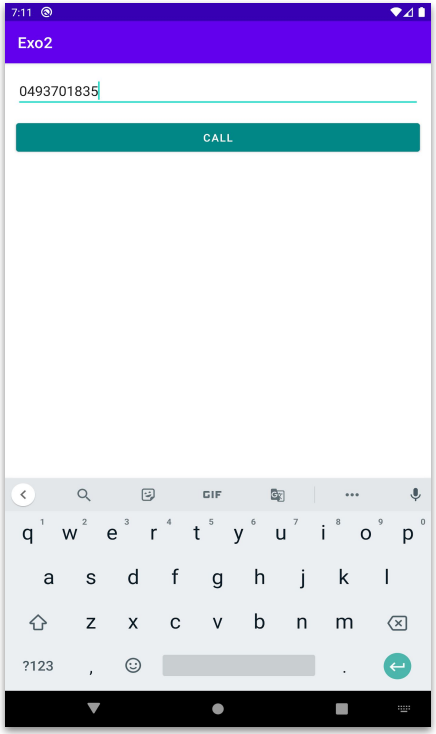
Placez le dans un dossier TECHNIFUTUR-AND5-EXO2

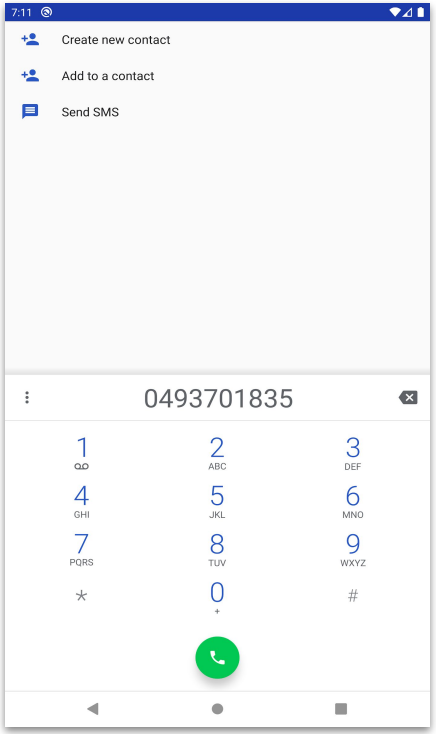
Exercice 2

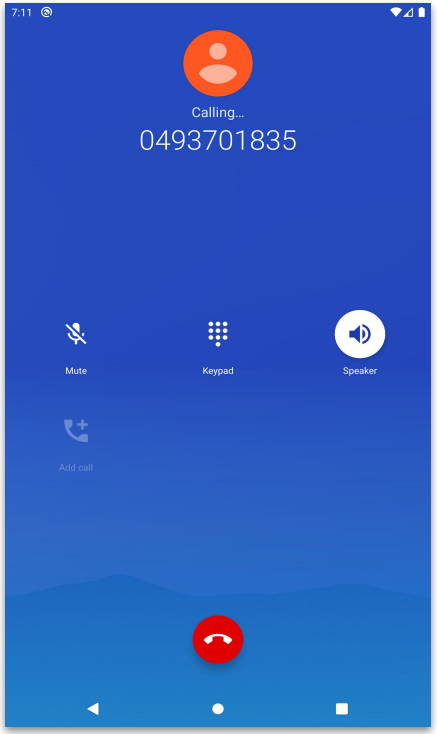
Ajoutez une zone de saisie et un bouton au layout de votre Activity

Gérez le click du bouton afin de demander à une autre application de gérer l'action DIAL pour le numéro saisi

Exercice 2


7:11
Exo2
0493701835
CALL


7:11
Create new contact
Add to a contact
Send SMS
0493701835


7:11
Calling...
0493701835
Mute
Keypad
Speaker
Add call

4

Les permissions

Les permissions

Accorder les permissions liées aux actions

- Certaines actions nécessitent des privilèges, qui devront être spécifiés dans le fichier Manifeste de votre application. (ex : appel téléphonique)
- L'action **ACTION_CALL** nécessite la permission **android.permission.CALL_PHONE** sinon une exception de type SecurityException: Permission Denial sera lancée.

Plus d'informations :
http://developer.android.com/reference/android/Manifest.permission.html

Les permissions

Il existe 2 types de permissions :

- Les permissions **normales** :
  Elles ne mettent pas en risque les données personnelles de l'utilisateur. Il suffit de les lister dans le manifeste et le système les autorise automatiquement.

- Les permissions **dangereuses** :
  L'application aura accès aux données personnelles de l'utilisateur. Il faut les lister dans le manifeste et l'utilisateur devra les autoriser à l'exécution.

Les permissions

Si vous utilisez **Android 5.1 ou moins**, ou si le SDK cible de votre app est le **22** ou moins:

- L'utilisateur doit accepter les permissions dangereuses lorsqu'il installe l'app et s'il refuse, il ne peut tout simplement pas installer l'application.

Si vous utilisez **Android 6.0 ou plus**, ou si le SDK cible de votre app est le **23** ou plus:

- L'application doit demander les permissions dangereuses quand elle en a besoin. L'utilisateur peut accepter ou refuser chaque permission et l'application continuera à fonctionner même si l'utilisateur refuse une permission.

En pratique

Création d'un intent permettant de passer une appel téléphonique

Si vous exécutez ce code, le système va lever une exception qui aura pour effet un crash de votre application

```
val uri = Uri.parse( uriString: "tel:0493701835")
val intent = Intent(Intent.ACTION_CALL, uri)
startActivity(intent)
```

Caused by: java.lang.SecurityException:

java.lang.SecurityException: Permission Denial: starting Intent

...requires android.permission.CALL_PHONE

En pratique

Accorder la permission de passer un appel téléphonique

Première étape, **indiquer la permission requise** dans le fichier manifest

Ceci permettra à un utilisateur du Google Play Store de voir que votre application permet de lancer des appels téléphoniques

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.technipixl.exo3">

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

    <application
```

En pratique

Accorder la permission de passer un appel téléphonique. Mais ce n'est pas suffisant !

En effet la permission de passer un appel est une permission dite dangereuse

Il faudra donc en plus **demander la permission** à l'utilisateur !

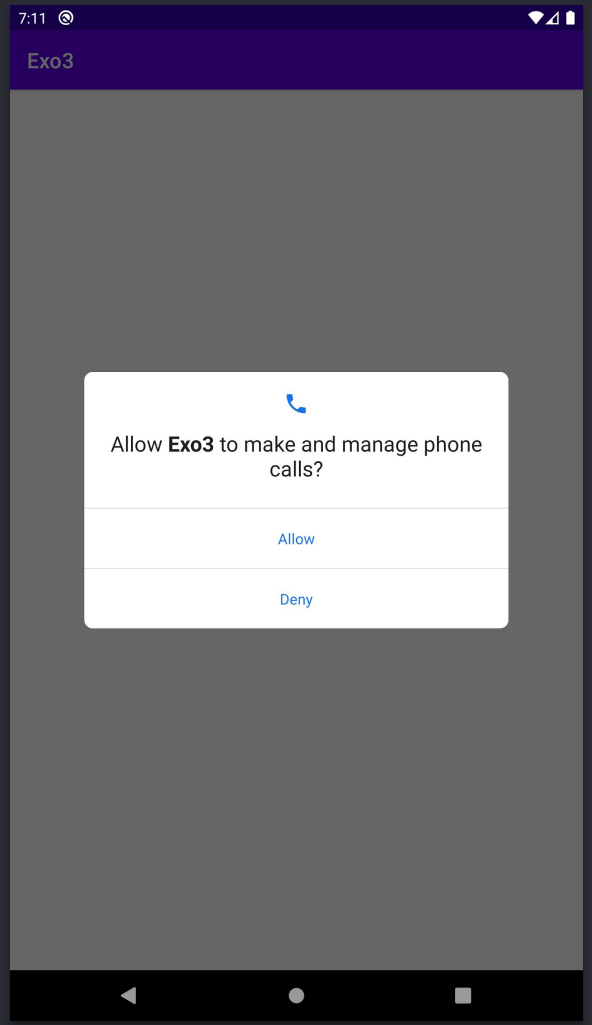
```
private val phoneCallPermissionLauncher = registerForActivityResult(
    ActivityResultContracts.RequestPermission()
){ it: Boolean!
    Log.d( tag: "PERMISSION REQUEST RESULT", msg: "$it")
}

fun requestPhoneCallPermission(){
    phoneCallPermissionLauncher.launch(Manifest.permission.CALL_PHONE)
}
```

En pratique

Accorder la permission de passer un appel téléphonique

L'appel à la méthode **launch sur le permission launcher** va déclencher l'affichage d'une popup de demande de permission et vous donner l'input utilisateur.

Notes :
-Selon la version d'Android et de la permission, les choix donnés à l'utilisateur peuvent être plus complexes que oui/non (ex : "always" ou "only when app is running" dans le cas de la localisation)
-Depuis Android 11, l'OS peut retirer de lui-même les permissions si elles n'ont plus été utilisées depuis un certain temps



ex3

Exercice

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-AND5-EXO3

Exercice 3

Créez une application capable de démarrer un appel téléphonique en sollicitant une autre application et en gérant correctement la demande de permissions

Nous allons cibler des appareils dont la version minimale est Android Marshmallow

5

Diffuser et recevoir des intents

Diffuser et recevoir des intents

Un intent peut être diffusé à l'ensemble des applications du système pour transmettre des informations afin d'effectuer une action spécifique ou pour fournir des informations sur l'environnement (ex : appel entrant, Wi-Fi connecté, ...)

Ces Intents sont alors appelés des Broadcast Intents et ses récepteurs sont appelés des Broadcast Receiver

Diffuser et recevoir des intents

Mécanisme de Broadcast Intent et de Broadcast Receiver utilisé très souvent pour communiquer des informations (niveau de la batterie, état du réseau, ...) aux applications

Système de filtrage identique aux Intents classiques

Traitement de l'information différent

Diffuser des Intents à but informatif

Dans le code de l'application qui doit diffuser un intent

Utilisation de la méthode **sendBroadcast()**

Attention à bien définir l'objet Intent afin de ne pas parasiter le système ou d'obtenir des comportements non désirés (ex : ouverture d'applications non sollicitées, ...)

```
val intent = Intent(CustomBroadcastReceiver.VIEW)
intent.putExtra( name: "myKey",  value: "extra value")
sendBroadcast(intent)
```

Recevoir et traiter des Intents diffusés

Dans le code de l'application qui va écouter ce type d'intent

Vous devrez créer une classe qui étend **BroadcastReceiver**

Définir une constante qui vous servira de valeur de filtre

Implémenter la méthode **onReceive()**

Pour chaque message émis, la méthode onReceive sera appelée

```
class CustomBroadcastReceiver: BroadcastReceiver() {

    companion object {
        const val VIEW = "technipixl.intent.action.VIEW"
    }

    override fun onReceive(context: Context?, intent: Intent?) {
        //Perform action
    }
}
```

Recevoir et traiter des Intents diffusés

Toujours dans le code de l'application qui va écouter ce type d'intent

Il faudra ensuite définir un filtre d'Intents dans le bloc application de votre fichier Manifest

```
<receiver android:name="CustomBroadcastReceiver">
    <intent-filter>
        <action android:name="technipixl.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</receiver>
```

Créer un récepteur d'Intents dynamiquement

Il est possible de créer un récepteur d'Intents pendant l'exécution d'une application.

Il faut pour cela utiliser la méthode **registerReceiver()** qui prend deux paramètres :

- un objet de type **BroadcastReceiver**
- un objet de type **IntentFilter**

```
val receiver = CustomBroadcastReceiver()
val filter = IntentFilter(CustomBroadcastReceiver.VIEW)
registerReceiver(receiver, filter)
```

Créer un récepteur d'Intents dynamiquement

Lorsque l'activité a fini son cycle de vie ou que le récepteur n'est plus utilisé

Il faudra libérer les ressources en appelant la méthode **unregisterReceiver()**

Attention : pour chaque appel à registerReceiver il faudra faire un appel à unregisterReceiver

```
private lateinit var receiver: CustomBroadcastReceiver

override fun onResume() {
    super.onResume()
    receiver = CustomBroadcastReceiver()
    val filter = IntentFilter(CustomBroadcastReceiver.VIEW)
    registerReceiver(receiver, filter)
}

override fun onPause() {
    super.onPause()
    unregisterReceiver(receiver)
}
```

Limitations Android 8.0 (API26)

Introduction de limitations des broadcast et services en arrière plan (déjà en partie sur Android 7.0).

Les applications enregistrées à un broadcast consomment des ressources chaque fois qu'un broadcast est émis.

Problèmes de performances au moment de l'émission si beaucoup d'app sont enregistrées aux mêmes broadcasts.

Conseil: S'enregistrer au runtime plutôt que par manifest

https://developer.android.com/about/versions/oreo/background.html

6

Pending intents

Pending intents

Objet encapsulant un Intent existant.

Permet à une autre application d'utiliser l'intent contenu comme si il était dans le processus de l'application d'origine.

Souvent utilisé pour :

- Effectuer une action suite à une notification push (ex : intent exécutant une activité)
- Effectuer une action depuis un widget
- Effectuer une action future via AlarmManager
- Action d'une alerte de proximité (geofencing)

Ci-contre un exemple d'envoi de notification interne à l'application.

Le pending intent contiendra les informations nécessaire à l'action qui devra être réalisée suite au clic sur la notification

```
val builder = NotificationCompat.Builder( context: this,  channelId: "DEFAULT_CHANNEL")
builder.setSmallIcon(R.drawable.ic_launcher_foreground)
val intent = Intent(Intent.ACTION_VIEW, Uri.parse( uriString: "https://www.neopixl.com/"))

val pendingIntent = PendingIntent.getActivity( context: this,  requestCode: 0, intent,  flags: 0)
builder.setContentIntent(pendingIntent)
builder.setLargeIcon(BitmapFactory.decodeResource(resources, R.mipmap.ic_launcher))
builder.setContentTitle("Notifications Title")
builder.setContentText("Your notification content here.")
builder.setSubText("Tap to view the website.")
val notificationManager = getSystemService(NOTIFICATION_SERVICE) as NotificationManager

// Will display the notification in the notification bar
notificationManager.notify( id: 1, builder.build())
```

Pending intents

Vous devez définir quel type de composant devra prendre en charge votre intent (Activity, BroadcastReceiver ou Service)

Vous devrez construire votre pending intent en utilisant une des méthodes suivantes:

PendingIntent.**getActivity()** :
- PendingIntent pour démarrer une Activity

PendingIntent.**getBroadcast()** :
- PendingIntent pour diffuser un Broadcast

PendingIntent.**getService()** :
- PendingIntent pour démarrer un Service

```
PendingIntent.getActivity( context: this, requestCode: 0, intent, flags: 0)
PendingIntent.getBroadcast( context: this, requestCode: 0, intent, flags: 0)
PendingIntent.getService( context: this, requestCode: 0, intent, flags: 0)
```

6

Messages d’information natifs

Messages d’information natifs

Vous pouvez aussi abonner votre application à des messages système

Pour cela vous devrez implémenter un Broadcast receiver et écouter les types de messages de votre choix

Quelques exemples de types de messages:

Action	Définition
ACTION_BOOT_COMPLETED	Le système a fini de démarrer. Permission RECEIVE_BOOT_COMPLETED nécessaire.
ACTION_SHUTDOWN	Le système est en train de s’éteindre
ACTION_SCREEN_ON/OFF	L’écran s’éteint / s’allume
ACTION_POWER_CONNECTED/DISCONNECTED	L’alimentation électrique a été branchée / débranchée

ex4

Exercice

Exercice 4

Créez un nouveau projet dans Android Studio

De type Empty Activity

Nom: Exo4

Package name: com.technipixl.exo4

Placez le dans un dossier TECHNIFUTUR-AND5-EXO4

Exercice 4

Vous allez devoir créer un Broadcast receiver capable d'écouter les actions d'alertes de batterie

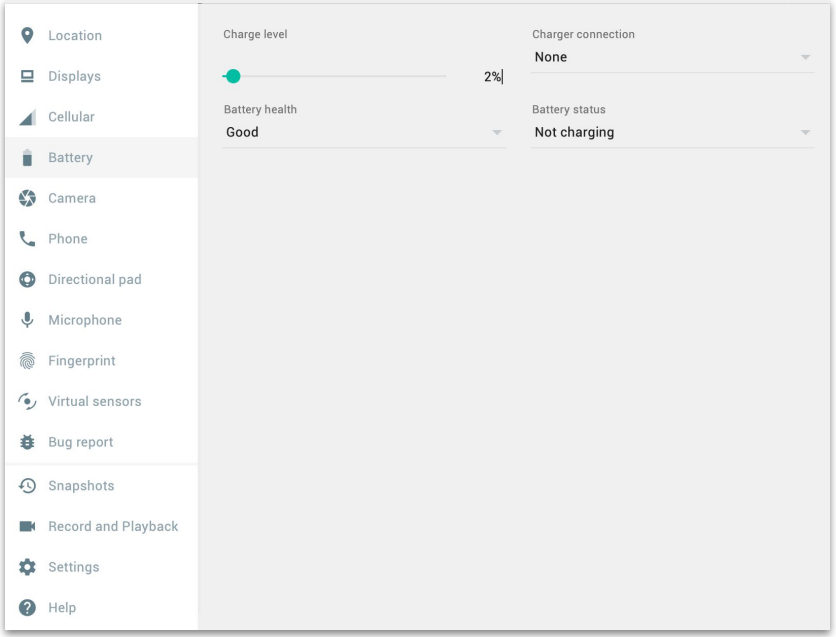
Votre application doit être capable de fonctionner au dessus de la version Oreo !

Ajouter un log lorsque vous détectez que la batterie atteint un niveau bas et quand elle atteint un niveau considéré comme OK

Utilisez les actions suivantes:

- android.intent.action.BATTERY_OKAY
- android.intent.action.BATTERY_LOW

Afin de pouvoir tester votre implémentation, utilisez l'outil de gestion de la batterie de votre simulateur



Conclusion

- Intent = mécanisme de communication de base d'Android
- Navigation entre écrans = intent + activité
- Lancer une autre application = intent
- Fournir des informations via intent = extra
- Broadcast intents = récepteurs multiples
- Broadcast intent système : très pratique

Fil rouge

Intégration fil rouge

Demandez à votre formateur de vous partager le projet de base de l'application Fil rouge Android ou utilisez votre projet si vous êtes parvenus à réaliser l'étape précédente

Depuis develop, créez une branche feature/FR002

Votre implémentation se fera sur cette branche

Une fois votre implémentation terminée, vous devrez "merge" votre branche sur **develop**

Intégration fil rouge

Nous allons reprendre le projet fil rouge pour:

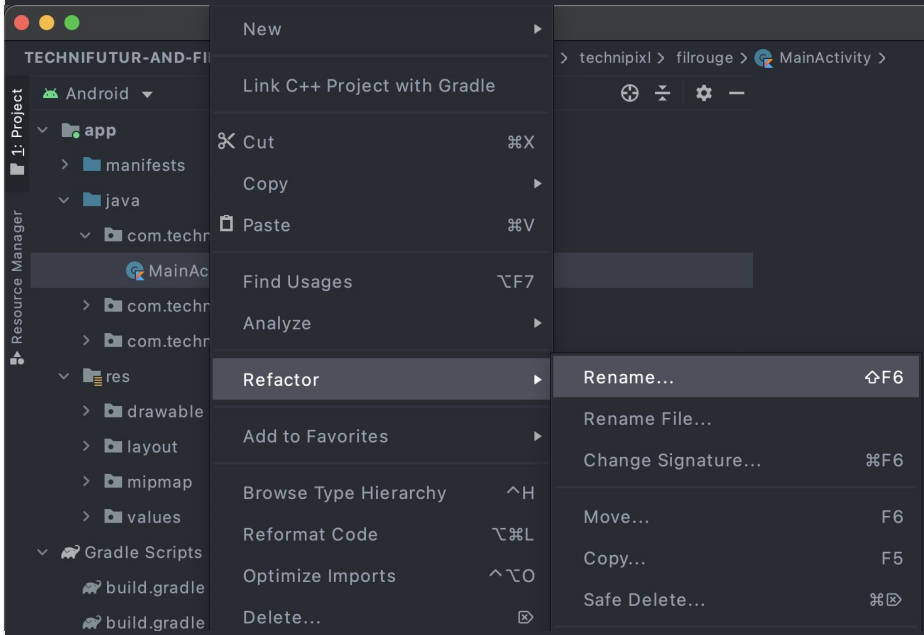
- Déplacer l'écran de login dans une nouvelle activité
- Ouvrir une activité home suite au login

Intégration fil rouge

Première étape

Renommez la MainActivity en AuthenticationActivity

Pour ce faire, sélectionnez la MainActivity dans l'explorateur de projet, refactor + rename

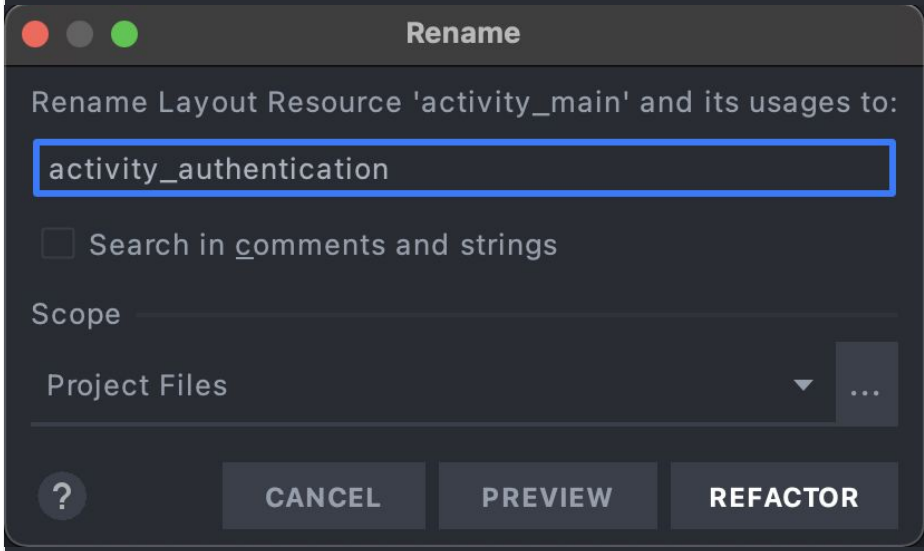


neopixl
A SMILE GROUP COMPANY

Intégration fil rouge

Votre classe MainActivity et le fichier Manifest seront automatiquement mis à jour

Répétez l'opération pour le layout activity_main.xml

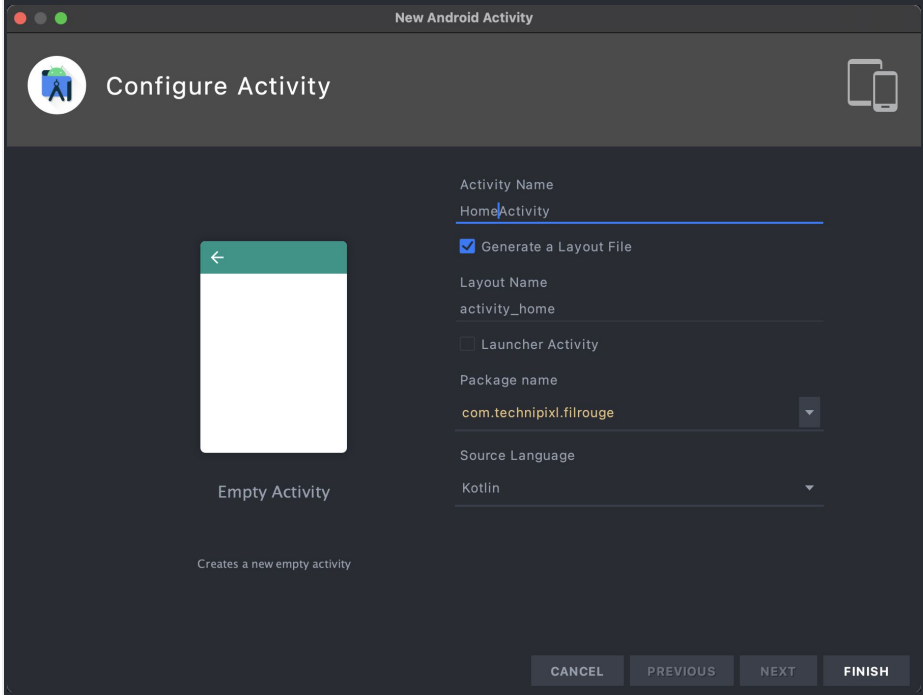


Intégration fil rouge

Ajoutez une nouvelle activité au projet et nommez là HomeActivity

À vous de jouer:

- Faites en sorte d'ouvrir la HomeActivity suite au login
- Uniquement si le login et le mot de passe ne sont pas vides



Restons en contact

neopixl.

A SMILE GROUP COMPANY

115A, Rue Emile Mark
L-4620 Differdange

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