

Introduction à Android

LABO 1

Table des matières

Introduction.....	2
Langue de l'interface	2
Champs textuels de saisie	3
Mode paysage	4
Vérification du format de l'email	5
Vérification du couple email / mot de passe.....	5
Création et lancement de la nouvelle activité	6
Passage de paramètres à la nouvelle activité	6
Permissions simples	7
Obtenir le résultat d'une activité	9
Affichage d'une activité.....	10
Factorisation du code	11
Cycle de vie.....	12
Sources	14

Introduction

Ce laboratoire est une initiation au développement mobile qui nous amène à effectuer plusieurs manipulations afin de nous familiariser avec le développement Android.

Ce document répond aux questions posées dans la donnée.

Langue de l'interface

Cette étape consiste à mettre en place la prise en charge du français. Actuellement l'application est en anglais.

Les applications Android peuvent demander un certain nombre de ressources. Ces dernières se trouvent dans le dossier **res/**. Les ressources, en fonction de leur type, sont stockées dans des sous-dossiers différents. Les chaînes de caractères sont généralement stockées dans le sous-dossier **values/**.

Il est possible d'ajouter des suffixes aux sous-dossiers de **res/** afin d'utiliser différentes ressources en fonction des caractéristiques de l'appareil ou de sa localisation.

Ainsi nous pouvons créer un dossier ressources Android pour notre traduction française **values-fr/**.

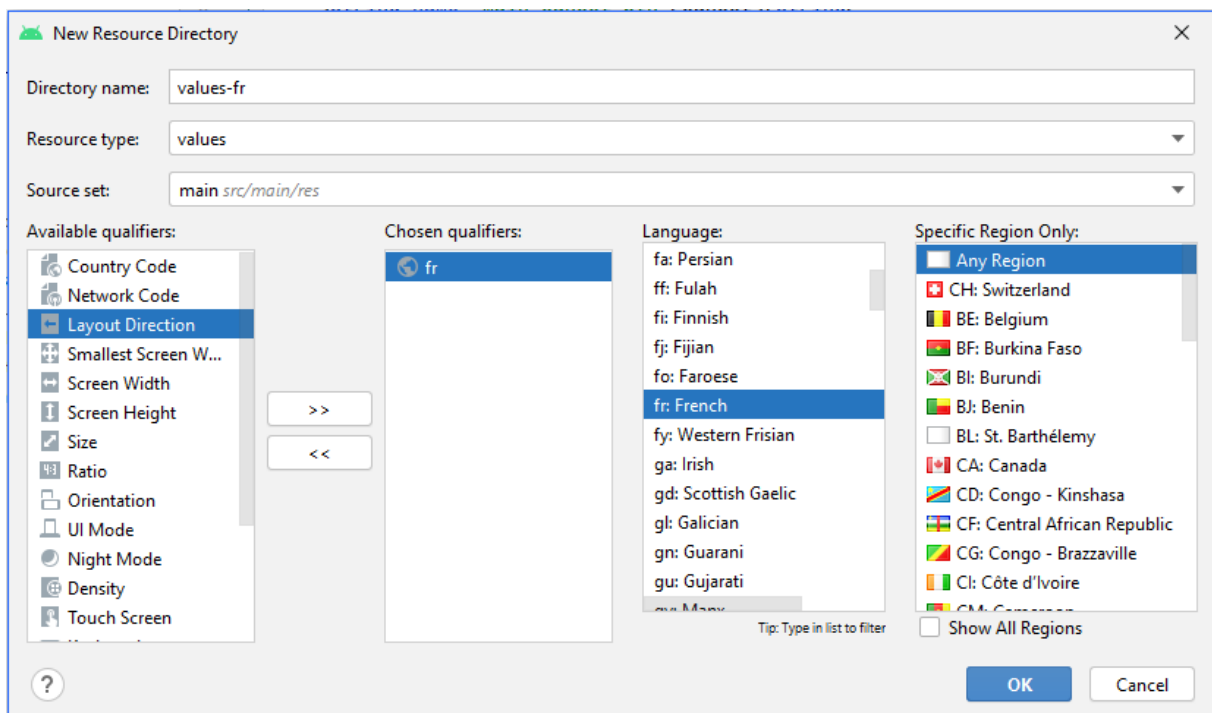


Figure 1 Création du dossier de ressources pour la traduction française

À ce point, nous comprenons que si nous souhaitons créer une application multilingues, nous aurons un dossier **values** par langue supportée.

De plus, il est possible de créer des spécifications par région en fonction des langues. Si par exemple, nous souhaitons distinguer le français suisse et le français canadien, nous aurions eu l'arborescence suivante :

res/	
values-fr-CH/ strings.xml	values-fr-CA/ strings.xml

Dans notre cas, nous avons créé un unique dossier *values* et un fichier *strings* pour ne pas distinguer le français en fonction des régions.

Si une langue demandée est non traduite, la langue affichée sera la langue par défaut, c'est-à-dire celle qui se trouve dans le dossier *values* sans spécification de langue **values/**. Ainsi en mettant l'anglais dans **res/values/string.xml** nous faisons en sorte que la langue par défaut soit l'anglais.

Pourquoi regrouper les chaînes de caractères dans un fichier XML indépendamment des layouts ?

Parce que c'est une bonne pratique.

Cette méthode permet de maintenir indépendamment les ressources et le code. Pour une application plus conséquente avec la prise en charge de toutes les langues, cela permettrait aux traducteurs et aux développeurs de travailler parallèlement sans empiéter les uns sur les autres.

De plus, cette pratique est native à l'environnement de développement. Il a suffi de nommer les répertoires et les fichiers pour que la compilation du programme gère la logique du choix des langues. Si nous avions voulu mélanger code et traduction, nous aurions dû nous charger de la détection de la langue de l'appareil, de choisir le bon tableau de string et remplir manuellement les layouts.

Que se passe-t-il si une traduction est manquante dans la langue par défaut ou dans une langue additionnelle ?

Quelle que soit la langue sélectionnée, si la traduction manquante est dans la langue additionnelle, la traduction est remplacée par la traduction de la langue par défaut. Si la traduction manquante est dans la langue par défaut, le programme ne compile pas, même si celle de la langue additionnelle (demandée ou non) est présente.

Champs textuels de saisie

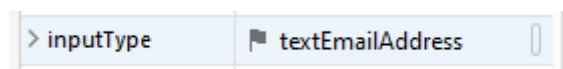
Les champs textuels de l'email et du mot de passe sont de type *EditText* sans spécification du type. Il est possible de spécialiser ces deux champs en utilisant l'attribut *InputType*.

Champ	InputType
Email	textEmailAddress
Mot de passe	textPassword

Il est possible de le faire directement dans le code.

```
android:inputType="textEmailAddress"
```

Sinon la même action peut être réalisée dans l'éditeur de layout.



Mode paysage

Comme les fichiers de traduction, les layouts sont des ressources. Ainsi, pour changer de layout en fonction de l'orientation du téléphone, nous allons spécialiser le dossier layout pour l'orientation paysage.

Le dossier **res/layout-land/** doit être créé.

Une fois encore, comme pour la traduction, nous y plaçons une copie du fichier par défaut situé dans **res/layout/**.

Finalement, nous éditons ce nouveau fichier pour obtenir l'affichage désiré.

Arborescence résultante :

res/	
layout/	layout-land/
activity_main.xml	activity_main.xml

Le layout a été réorganisé comme suit :

Vertical	Horizontal

Le passage du *ConstraintLayout* en *LinearLayout* nous permet d'afficher l'image à côté du formulaire. Pour une meilleure lisibilité, nous centrons le *main_logo* et le *relativeLayout* avec l'attribut *layout_gravity -> center*.

Nous agrandissons l'image avec les attributs *layout_width* et *layout_height* à 200.

Finalement, nous plaçons le lien *main_new_account* dans le *relativeLayout* et nous lui ordonnons de se placer sous le *LinearLayout*, identifié par *main_buttons*, grâce à la clause *layout_below*.

Vérification du format de l'email

Dans cette partie, nous vérifions si l'email est correct. Pour des raisons de simplicité, un email est correct s'il contient un '@'.

```
// Vérification de la validité de l'email
if(! emailInput!!.contains(char: '@')){
    val toast = Toast.makeText(applicationContext, getString(R.string.main_invalid_email_format), Toast.LENGTH_SHORT)
    toast.show()
}
```

À la suite du *ClickListener* du bouton « valider », si le contenu du champ de l'email n'est pas null, nous vérifions s'il contient un '@'.

Si ce n'est pas le cas, nous affichons le toast avec le message d'erreur *main_invalid_email_format* que nous avons pris soin de créer pour l'anglais et le français.

Vérification du couple email / mot de passe

Nous créons une paire *emailInput* et *passwordInput* puis, nous vérifions son existence dans la liste *credentials*.

```
// Vérification couple email / mot de passe
if(credentials.contains(Pair(emailInput, passwordInput))){
```

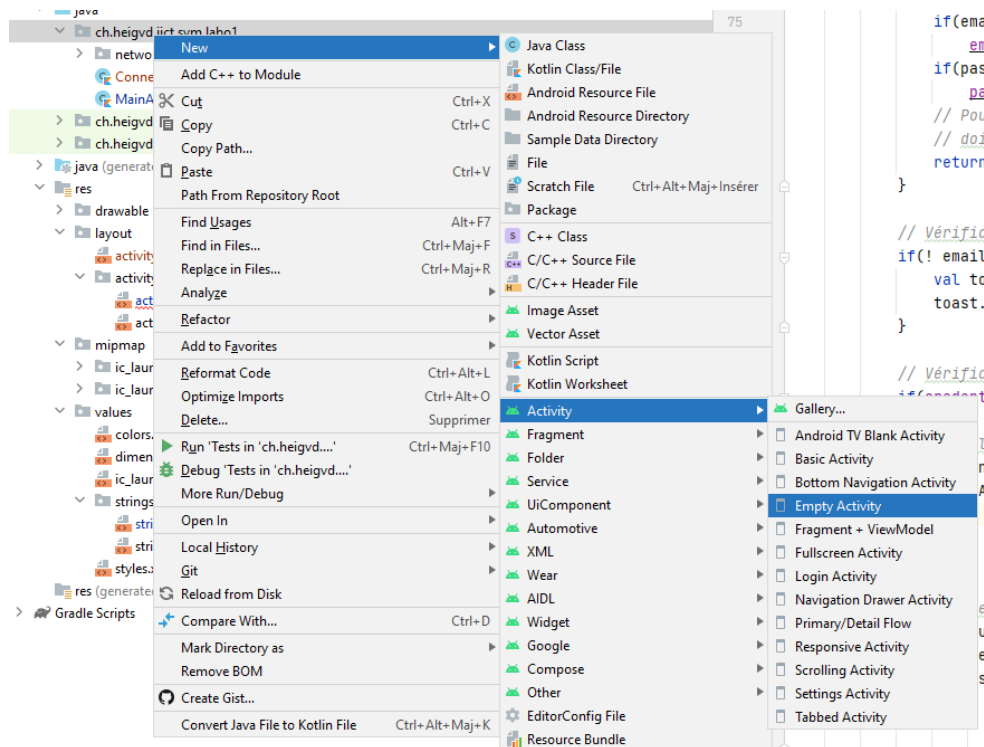
Si le couple n'est pas trouvé, nous affichons une fenêtre de dialogue qui indique l'erreur et vide les champs.

```
// Inexistant, affichage fenêtre de dialogue.
val builder = AlertDialog.Builder(context: this)
builder.setMessage(getString(R.string.main_error_email_pass))
    .setPositiveButton(getString(R.string.main_ok_button),
        DialogInterface.OnClickListener { dialog, id ->
            cancelButton.callOnClick()
        })
builder.create()
builder.show()
```

Une fois encore l'erreur *main_error_email_pass* a été traduite dans les deux langues. Toutefois, la chaîne *main_ok_button* (de valeur « OK ») a été traduite uniquement en anglais.

Création et lancement de la nouvelle activité

Nous avons créé une nouvelle activité *connectedActivity* de type *Empty Activity* avec l'utilitaire AndroidStudio.



Finalement, pour démarrer notre nouvelle activité, nous avons créé une intention que nous exécutons aussitôt.

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

Passage de paramètres à la nouvelle activité

Pour passer des paramètres à la nouvelle activité, nous avons ajouté un extra à l'intention précédemment créée.

```
val intent = Intent( packageContext: this, ConnectedActivity::class.java).apply {
    putExtra(EXTRA_EMAIL, emailInput)
}
```

Désormais, nous transmettons la valeur de l'email avec la clé *EXTRA_EMAIL* (*email.MESSAGE*) qui est une constante globale.

Concernant le layout de la nouvelle activité, nous avons ajouté deux *TextView* :

- *connected_greetings_title* : qui affiche « Bonjour » ou « Good morning »
- *connected_email_greetings* : qui affiche l'adresse de l'utilisateur.

Finalement, le code de l'activité *ConnectedActivity* se présente comme suit :

```
class ConnectedActivity : AppCompatActivity() {  
  
    private lateinit var email: TextView  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_connected)  
  
        // Récupération de l'intention et récupération de la chaîne  
        val emailInput = intent.getStringExtra(EXTRA_EMAIL)  
  
        // Récupération du champ et mise à jour.  
        email = findViewById(R.id.connected_email_greetings)  
        email.setText(emailInput)  
    }  
}
```

Nous récupérons une référence sur la *TextView* qui affichera l'email de l'utilisateur connecté. Ensuite, nous récupérons l'extra de l'intention que nous affichons dans la *TextView* précédemment récupérée.

Permissions simples

Nous avons autorisé l'accès à Internet en plaçant ces deux lignes, en avant les balises application, dans notre manifeste.

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

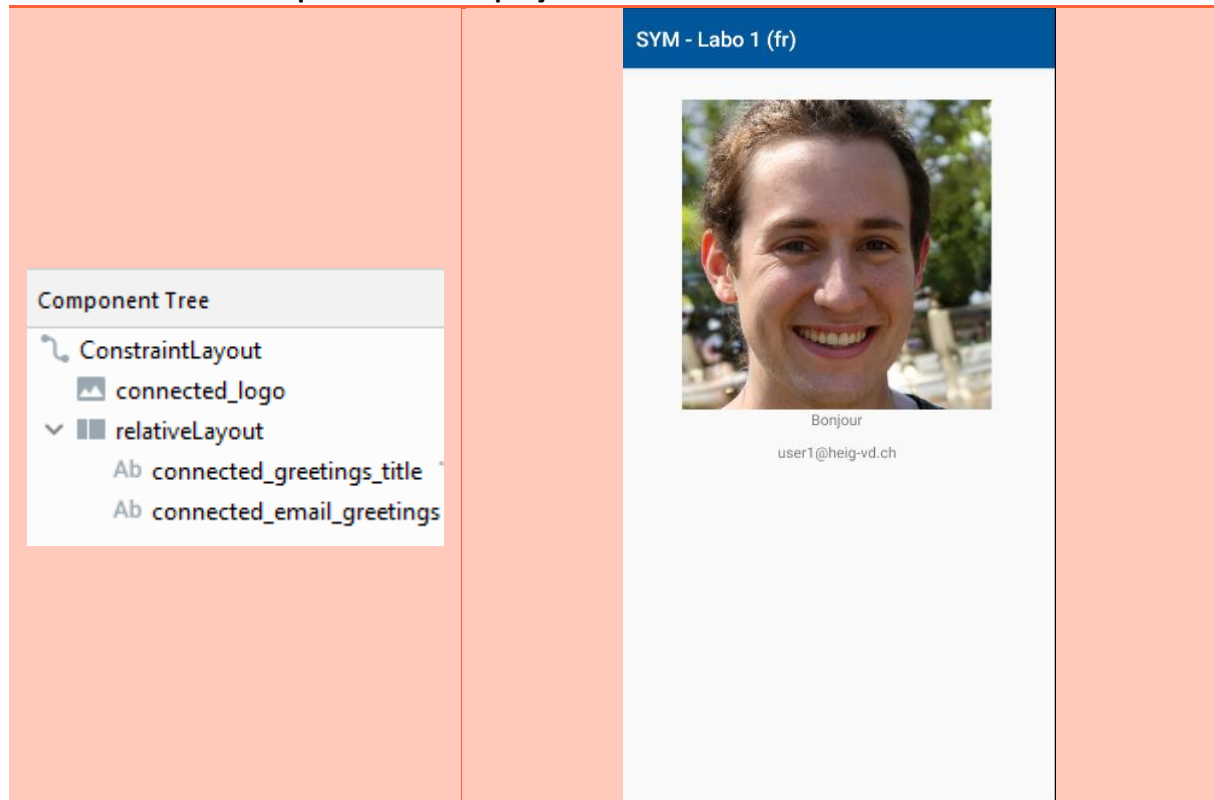
Ensuite, nous avons modifié notre activité *ConnectedActivity* en y ajoutant une *ImageView* du nom de *connected_logo*.

Le code de l'*ImageView* est le suivant :

```
<ImageView  
    android:id="@+id/connected_logo"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    app:layout_constraintBottom_toTopOf="@+id/relativeLayout"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    android:contentDescription="@string/connected_profile_picture_desc" />
```

Nous ajoutons une image vide à laquelle nous spécifions un id, le comportement pour sa taille, les contraintes de position ainsi qu'une description.

Le nouveau design de l'activité se présente comme suit :

Arborescence des composants**Aperçu**

Finalement, dans le code de l'activité nous récupérons une référence sur l'élément et nous appliquons l'image à l'aide du code fourni :

```
// Récupération de l'image ImageView et mise à jour.  
profilePicture = findViewById(R.id.connected_logo)  
ImageDownloader(profilePicture, url: "https://thispersondoesnotexist.com/image").show()
```

Obtenir le résultat d'une activité

Nous avons décidé d'utiliser la méthode qui consiste à créer différents « launcher » d'activités. De cette manière, il est alors possible de créer plusieurs launchers d'une même activité, mais qui n'effectueraient pas forcément le même code lors de la réception de la réponse.

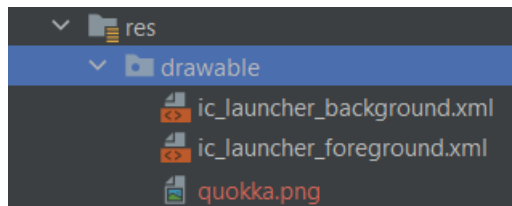
Les 2 méthodes cependant utilisent le principe de « callback ». Cependant, la plus grande différence est que la méthode d'origine d'Android possède une unique fonction pour tous les callbacks de résultat d'activités. Cela ne pose pas de problème si le nombre d'activités envoyant un résultat est faible. Mais si le nombre est grand, il faudra alors créer un switch imposant dans la fonction pour savoir quel code exécuté pour quelle activité.

	<i>Avantages</i>	<i>Inconvénients</i>
<i>Méthode d'origine Android</i>	Mise en place très simple.	Tous les traitements de callback se trouvent dans une seule et même méthode.
<i>Méthode librairie AndroidX</i>	Séparation du code. Possibilité de récupérer le résultat de l'activité dans une autre classe	On doit créer des « launchers » pour chaque activité ayant un résultat.

Affichage d'une image

Dans quel(s) dossier(s) devons-nous ajouter cette image ?




Il suffit d'ajouter à la main l'image dans le dossier « drawable », comme ceci :



Et on retrouve ces dossiers à l'aide du chemin suivant dans l'explorateur Windows du projet :

Labo1-kotlin > app > src > main > res > drawable

Voici le contenu du dossier :

Nom	Modifié le	Type	Taille
 ic_launcher_background.xml	05.10.2021 14:41	Fichier XML	6 Ko
 ic_launcher_foreground.xml	05.10.2021 14:41	Fichier XML	5 Ko
 quokka.png	05.10.2021 10:26	Fichier PNG	1 261 Ko

Veuillez décrire brièvement la logique derrière la gestion des ressources de type « image matricielle » sur Android.

Toutes les images de type « matricielle » (png ou jpeg) ne subissent aucun traitement et sont directement stockées dans le dossier « drawable ».

Quel intérêt voyez-vous donc à utiliser une image vectorielle ?

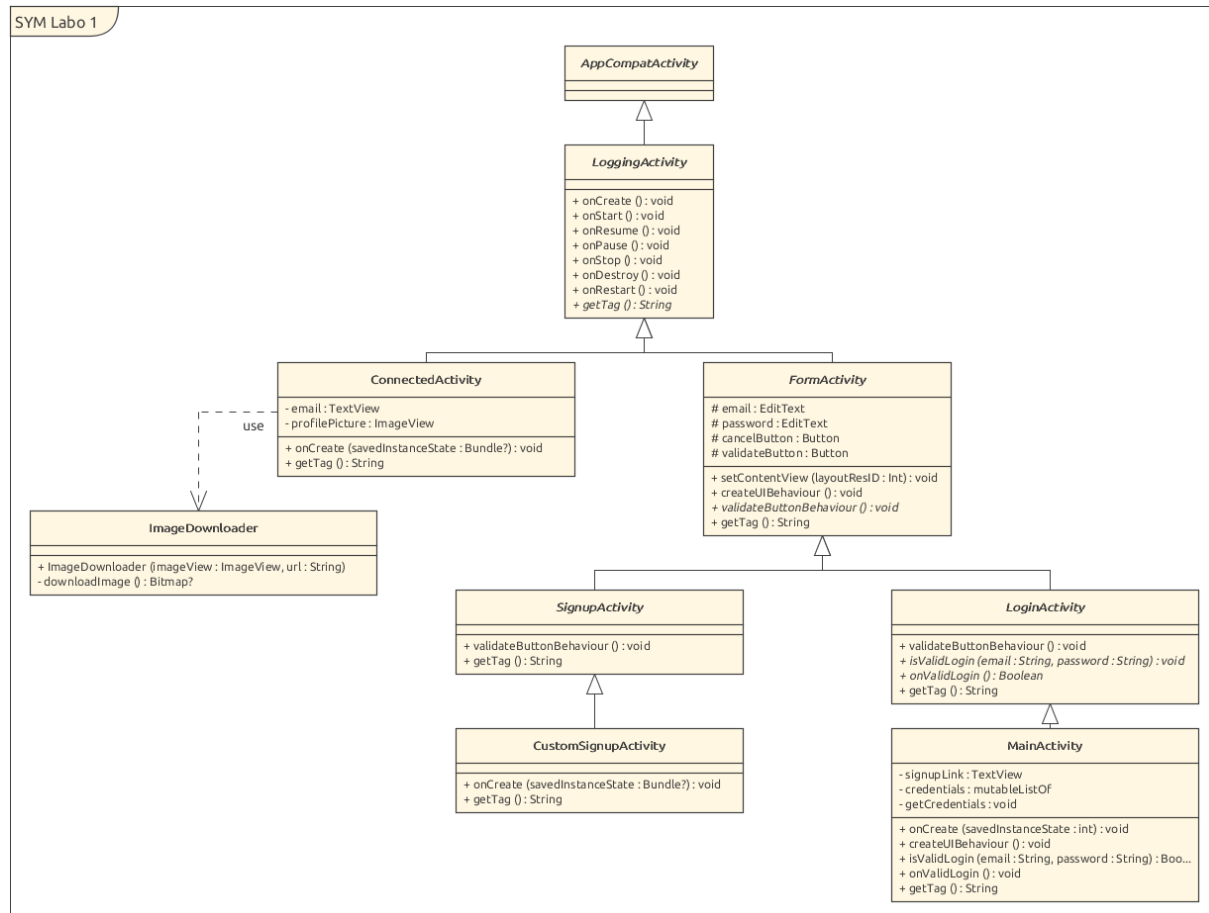
L'image vectorielle obtient différentes formes que l'on peut utiliser comme on le souhaite au sein de l'application. Une image vectorielle ne perd aucune qualité lorsque cette dernière est redimensionnée. Donc, elle est plus responsive. Elle est donc très pratique pour ne pas perdre de qualité sur les logos et icônes lors du redimensionnement de l'affichage. Elles sont donc plus flexibles.

Est-ce possible pour tout type d'images (logos, icônes, photos, etc.) ?

Les images vectorielles sont uniquement utilisables pour les images pouvant être dimensionnées à l'aide de « formules » mathématiques. On parle donc ici de logos et d'icônes. Ces images sont normalement au format SVG, CGM, etc... issu des logiciels spécialisés Illustrator, Flash, etc... Des photos prises à l'aide d'un appareil ne pourront pas être vectorielles, car elles sont basées sur des pixels.

Factorisation du code

Pour factoriser le code, nous avons décidé d'implémenter de l'héritage pour les activités.



La classe « LoginActivity » implémente tous les onCreate, onPause, onStart, etc.. Ce qui nous permet d'afficher les logs seulement dans cette classe, et que toutes les activités enfants affichent elles aussi les logs.

La classe abstraite « FormActivity » travaille en paire avec le layout « template_log ». Ce faisant, lorsqu'une nouvelle activité est créée et inclut le layout mentionné, il suffit de la faire hériter de « FormActivity » pour que toute la gestion du formulaire fonctionne. Il est ensuite possible de personnaliser le fonctionnement grâce à différentes méthodes.

La classe abstraite « SignupActivity » rajoute simplement la couche permettant de renvoyer la paire email/mot de passe en résultat de l'activité lorsque le bouton de validation est cliqué.

La classe abstraite « LoginSignup » permet de simplement redéfinir les méthodes « isValidLogin » et « onValidLogin » pour que l'implémentation du login selon les critères souhaités fonctionne.

Comme décrit auparavant, il est possible de factoriser les layouts. La factorisation de layout consiste à grouper, dans un layout, les éléments communs aux layouts pour qu'ils puissent être inclus dans différentes vues sans dupliquer le code. Ici, nous avons choisi de factoriser les champs d'email et du mot de passe ainsi que les boutons annuler et valider dans un même layout. De ce fait, il est maintenant possible d'inclure le layout « template_log » autant dans signup que dans le login et de ne pas répéter le code.

Cycle de vie

Décrivez brièvement à quelles occasions ces méthodes sont invoquées. Vous expliquerez aussi l'enchaînement de ces appels lorsque l'on passe d'une activité à une autre.

Voici un schéma résumant bien les actions réalisées sur les activités et plus particulièrement leurs changements de contexte :

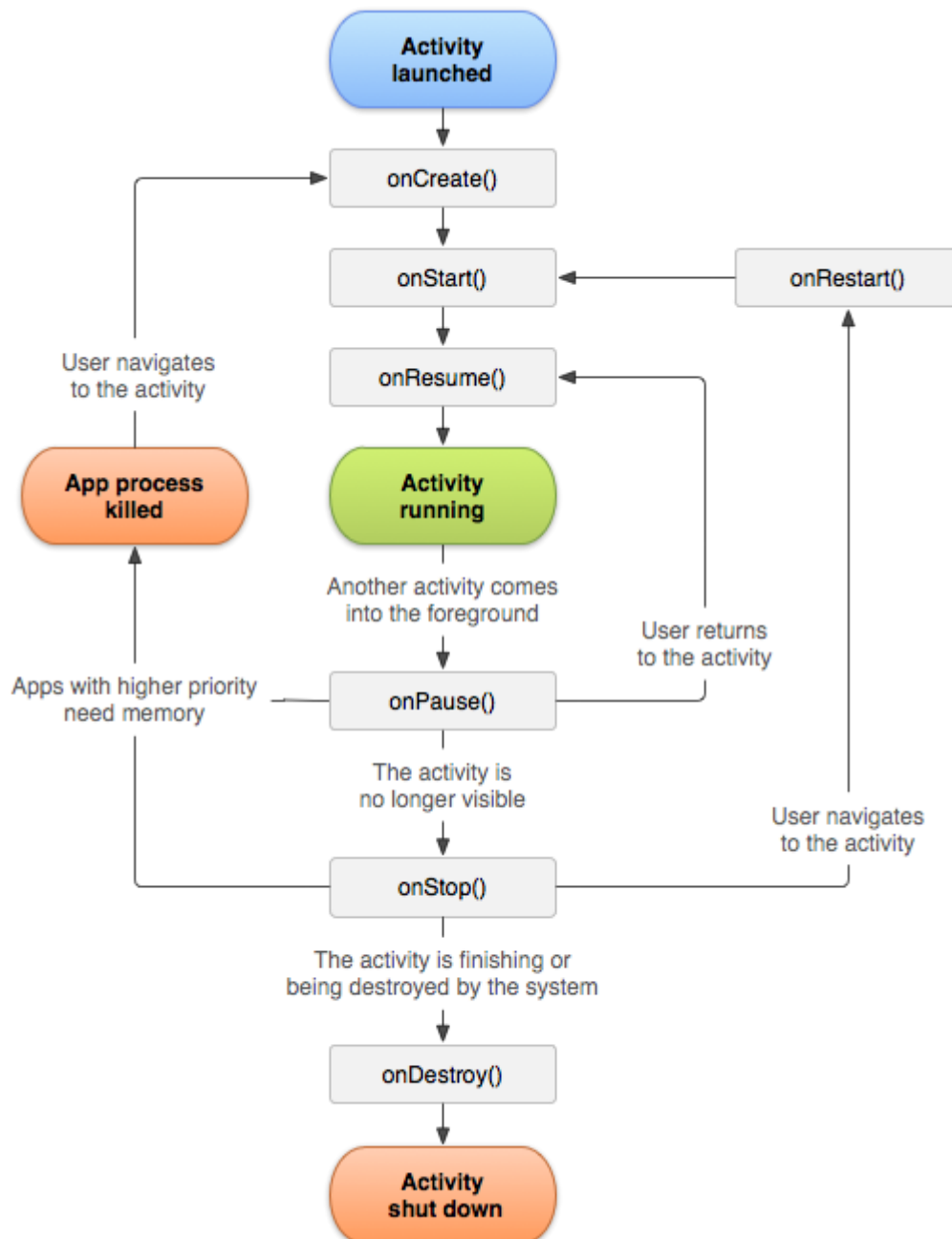


Figure 2 Diagramme d'états d'une activité

Voici un tableau récapitulant quelle méthode est invoquée et à quel moment.

Méthode	Invocation
onCreate	Lorsque l'activité est lancée ou que l'utilisateur navigue à l'activité qui avait été dans l'état « Killed » auparavant, lors de la création de l'activité.
onStart	Lorsque l'activité est lancée et qu'elle est déjà créée, lors du démarrage de l'activité.
onResume	Lorsque l'activité est lancée, lors de la reprise de l'activité. Lorsque l'utilisateur lance une nouvelle activité ou retourne à une mise en pause.
onPause	Lorsque l'activité est en cours d'exécution et qu'une autre arrive en premier plan à sa place, lors de la mise en pause de l'activité.
onStop	Lorsque l'activité est en cours d'exécution et qu'elle n'est plus visible, lors de l'arrêt de l'activité.
onDestroy	Lorsque l'activité est terminée ou détruite par le système, elle était dans l'état stoppée auparavant.
onRestart	Lorsque l'activité est relancée et qu'elle était stoppée, lorsque l'utilisateur navigue jusqu'à cette activité.

Enchaînement des appels lorsque l'on passe d'une activité à une autre :

La première activité s'arrête et rentre dans l'état Paused ou Stopped. Pendant ce temps, l'autre activité est créée. Si ces activités partagent des données, la première activité n'est pas complètement stoppée avant que la seconde soit créée afin d'assurer la bonne transmission des données.

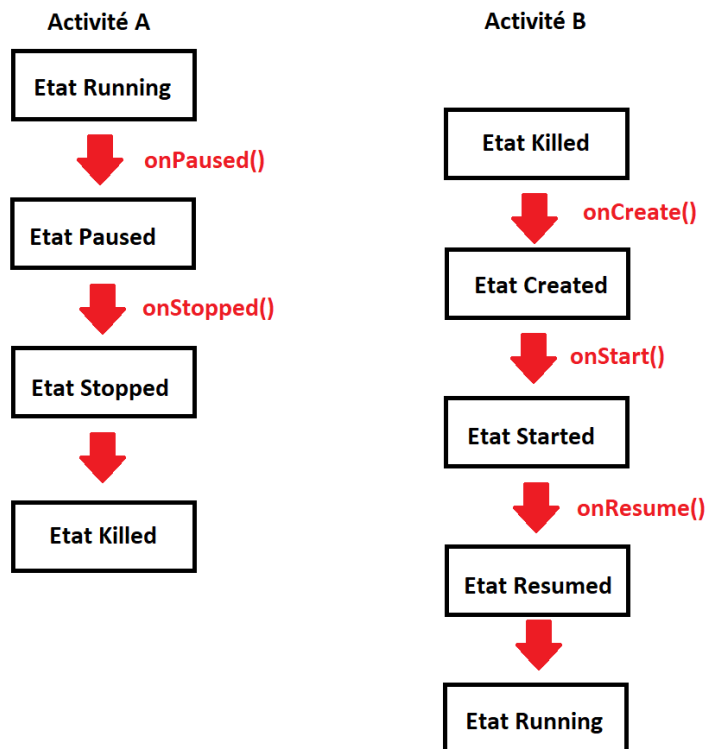
Voici le cycle de vie de 2 activités (A et B) s'exécutant dans la même application :

Activity A : la méthode onPause() est exécutée.

Activity B : la méthode onCreate(), puis la méthode onStart(), et finalement la méthode onResume() s'exécutent à la suite.

Ensuite, si l'activité A n'est plus utilisée au niveau visuel, sa méthode onStop() est exécutée.

Voici un schéma avec les états dans lequel les activités passent :



Les activités ne sont pas forcément toujours dans l'état « Killed » au moment de démarrer. Elles peuvent être simplement « Stopped ». Dans ce cas il faut sauter l'étape « onCreate() » et partir de « onRestart() », puis « onStart() ».

Sources

Android Developers, *App resources Overview* – 14/07/2021,

<https://developer.android.com/guide/topics/resources/providing-resources#ResourceTypes>,
22/09/2021

Android Developers, *Support different languages and cultures* – 17/09/2021,

<https://developer.android.com/training/basics/supporting-devices/languages>, 22/09/2021

Android Developers, *InputType* – 11/08/2021,

https://developer.android.com/reference/android/widget/TextView#attr_android:inputType,
23/09/2021

Android Developers, *Dialogs* – 30/06/2021, <https://developer.android.com/guide/topics/ui/dialogs>,
24/09/2021

Android Developers, *Start another activity* – 13/07/2021,

<https://developer.android.com/training/basics/firstapp/starting-activity>, 24/09/2021

Android Developers, *Connect to the network* – 25/09/2021,

<https://developer.android.com/training/basics/network-ops/connecting>, 25/09/2021

Xd Adobe, *Bitmap vs. Vector Images: What's the Difference?* – 28/08/2020

<https://xd.adobe.com/ideas/principles/app-design/bitmap-vs-vector-images-difference/>, 05/10/2021, 05/10/2021

Wikipedia, *Image vectorielle* – 06/05/2021

https://fr.wikipedia.org/wiki/Image_vectorielle, 05/10/2021

Wikipedia, *Image matricielle* – 25/04/2021

https://fr.wikipedia.org/wiki/Image_vectorielle, 05/10/2021

Android Developers, *The Activity Lifecycle* – 10/04/2021

<https://developer.android.com/guide/components/activities/activity-lifecycle>, 05/10/2021