

Développement d'applications mobiles

Chapitre 3 : Structure d'un projet Android(1/2)



Objectifs du cours

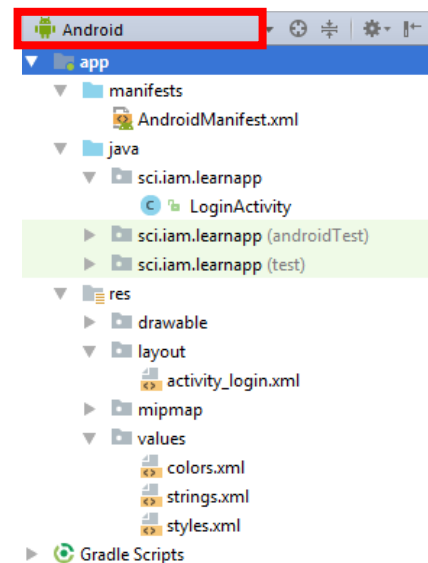
- Savoir organiser un projet Android
- Comprendre le cycle de vie d'une activité

1. Hiérarchie d'un projet Android

Situé à gauche de Android Studio, un onglet "Project" permet la visualisation de tous les fichiers du projet organisés selon la vue sélectionnée. Il en existe 12 différentes et Android Studio utilise par défaut la vue "Project". Néanmoins, la vue "Android" est la plus adaptée au développement Android en étant conçue pour améliorer la gestion des fichiers des projets Android.

En sélectionnant la vue "Android", le module **app** apparaît, et tous les scripts Gradle qui regroupent toute la configuration de votre projet et de vos modules. Dans chaque module, vous avez un accès à votre fichier **AndroidManifest.xml**, à vos sources Java (**java**) et à vos ressources (**res**).

Dans ce qui suit, nous détaillons l'arborescence du module **app** :



1.1. AndroidManifest

Le Manifest est un fichier XML se trouvant à la racine du projet Android (Sous Android Studio, dans le dossier **manifests** de la vue Android) sous le nom d'**AndroidManifest.xml**. Ce fichier est indispensable pour tous les projets Android, car il contient la configuration de l'application (titre, logo, ...). C'est pourquoi il est créé par défaut.

Le Manifest fournit au système Android des informations essentielles sur l'application, avant de pouvoir l'exécuter. Voici les principales informations que doit fournir le Manifest :

- Il définit le point d'entrée potentiel de l'application, c'est-à-dire l'activité main.
- Il décrit les composants de l'application, il en existe 4 types : les activités, les services, les récepteurs de diffusion et les fournisseurs de contenu.
- Il déclare les autorisations que l'application doit avoir pour accéder à certaines informations et aux parties protégées de l'API telles que la géolocalisation, la caméra, ...
- Il spécifie le matériel nécessaire pour les faire fonctionner l'application, tels que le Bluetooth, la caméra, le lecteur SIM, ...

Certaines informations plus ou moins triviales sont aussi définies telles que le nom de l'application, son icône, et le thème utilisé, ainsi que le package Java principal, qui sert d'identifiant unique.

Dans l'exemple suivant d'un Manifest, dans la ligne 8, **category.LAUNCHER** correspond au point d'entrée de l'application, qui signifie que **LoginActivity** s'affiche en premier lors du lancement de l'application.

manifests/AndroidManifest.xml

```
1 <manifest>
2   <application android:icon="@mipmap/ic_app"
3               android:label="@string/app_name">
4     <activity android:name=".LoginActivity"
5               android:label="@string/app_name">
6       <intentfilter>
7         <action android:name="android.intent.action.MAIN"/>
8         <category android:name="android.intent.category.LAUNCHER"/>
9       </intentfilter>
10    </activity>
11  </application>
12 </manifest>
```

1.2. Activités

Une activité est la composante principale pour une application Android. Vue souvent comme un cas d'utilisation UML, elle représente un écran graphique avec lequel l'utilisateur peut interagir. Une activité est une classe Java héritant de la classe **android.app.Activity** [1] (ligne 3) ou l'une de ses sous-classe (par exemple **AppCompatActivity**), donc stockée dans le dossier **java**. Généralement, elle est associée à une vue, appelée **layout** qui est stockée dans **res/layout/**. Voici un exemple :

java/LoginActivity.java

```
1 import android.app.Activity;
2
3 public class LoginActivity extends Activity {
4     ...
5 }
```

Cycle de vie d'une activité

Android pour des raisons de priorisation d'activités, par exemple, un appel téléphonique, peut tuer une activité quand il a besoin de ressources. Pour cette raison, chaque activité traverse plusieurs états symbolisés par un cycle de vie, qui est schématisé dans la figure suivante [2]. La transition entre les états provoque le déclenchement d'une méthode, dite callback. Par exemple, à la création d'une activité, Android appelle la méthode **onCreate(...)** et quand une autre activité s'affiche devant elle, cette dernière appelle la méthode **onPause()**. Par ailleurs, **onDestroy()** opère quand la méthode **finish()** est appelée explicitement ou quand c'est Android qui décide de tuer l'activité pour économiser la mémoire. Voici toutes les méthodes callback pouvant être appelées dans une activité :

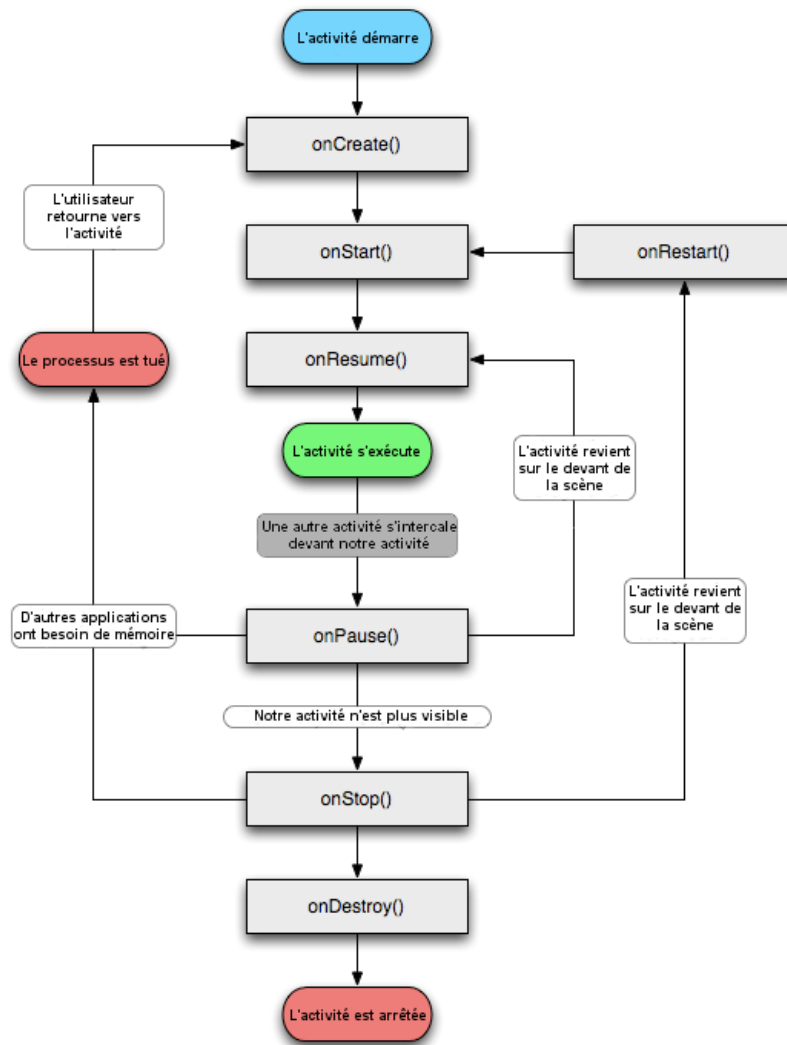
```
1 void onCreate(...) // lorsque l'activité est créée
2 void onStart()      // lorsque l'activité est démarrée (visible)
3 void onRestart()    // lorsque l'activité redémarre après un arrêt
4 void onResume()     // lorsque l'activité redémarre après une pause
5 void onPause()      // lorsque l'activité est en pause (invisible)
6 void onStop()       // lorsque l'activité s'arrête
7 void onDestroy()    // lorsque l'activité est détruite
```

Comme bonne pratique, on instancie les objets dans **onCreate(...)**, on lance les traitements dans **onStart()**, on s'abonne aux événements (par exemple le clic) et on récupère le contexte dans **onResume()**, on se désabonne aux événements et on enregistre le contexte dans **onPause()**, et enfin on arrête les traitements et on désalloue les objets dans **onStop()**. Par ailleurs, dans **onDestroy()**, le système libère automatiquement toutes les ressources qui n'ont pas été libérées dans **onStop()**.

Pour chaque méthode callback, il faut appeler la méthode de la super-classe (**Activity**). Par exemple, dans la méthode **onCreate(...)**, il faut appeler la méthode **super.onCreate(...)** (ligne 6). Le seul paramètre de la méthode **onCreate(...)** est un objet de type **Bundle**, qui permet de mémoriser l'état de l'activité lorsqu'elle passe en arrière-plan.

java/LoginActivity.java

```
1 import android.app.Activity;
2
3 public class LoginActivity extends Activity {
4     @Override
5     public void onCreate(Bundle savedInstanceState) {
6         super.onCreate(savedInstanceState);
7         ...
8     }
9
10    @Override
11    protected void onDestroy() {
12        super.onDestroy();
13    }
14 }
```



Association d'un layout

Pour associer une activité à un layout, il faut appeler la méthode `setContentView(int layout)` lorsque l'activité est créée (ligne 5).

java/LoginActivity.java

```

1 public class LoginActivity extends Activity {
2     @Override
3     public void onCreate(Bundle savedInstanceState) {
4         super.onCreate(savedInstanceState);
5         setContentView(R.layout.activity_login);
6         ...
7     }
8 }

```

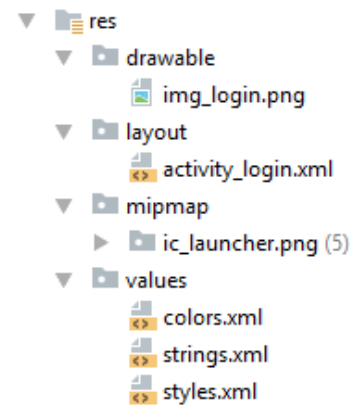
1.3. Ressources

Le dossier **res** contient toutes les ressources que l'on utilise dans l'application. La section suivante permet de décrire leur rôle et leur utilisation.

2. Ressources de l'application

Les ressources sont des éléments indispensables à l'élaboration d'un projet Android. Elles sont stockées dans des fichiers XML et organisées dans différents sous-dossiers qui sont créés de base, tels que :

- **res/drawable** : permet de stocker des images vectorielles (XML) ou matricielles (PNG et JPG),
- **res/layout** : contient les interfaces graphiques des vues,
- **res/menu** : pour stocker des menus,
- **res/mipmap** : pour stocker l'icône de l'application,
- **res/values** : regroupe les valeurs que l'on utilise dans l'application, qui sont organisées aussi dans des sous-dossiers :
 - **res/values/colors** : pour les couleurs,
 - **res/values/dimens** : pour les dimensions utilisées dans l'interface graphique (margin, padding, ...),
 - **res/values/strings** : pour les variables de type chaîne de caractères,
 - **res/values/styles** : pour les différents styles.



Il n'est malheureusement pas possible de créer de nouveaux dossiers, sous-dossiers dans le dossier **res**. Par exemple, les layouts sont tous stockés dans le même dossier **res/layout**.

2.1. Layouts

Un layout définit la structure visuelle d'une interface graphique, d'une activité par exemple. Il est possible de déclarer un layout de 2 façons :

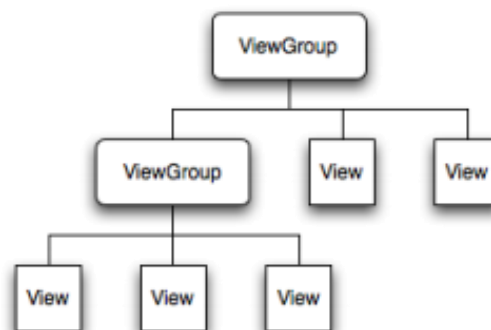
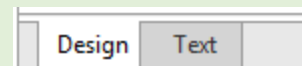
- Déclarer les éléments de l'interface graphique dans un fichier XML situé dans le dossier **res/layout/**,
- Instancier et manipuler les éléments du layout dans le code source Java au moment de l'exécution.

Déclarer un layout en XML facilite la visualisation de la structure de l'interface graphique, et ainsi résoudre les erreurs.



Remarque

Android Studio fournit un outil de manipulation des layouts de manière graphique. Pour utiliser cet outil, il faut ouvrir le layout concerné et sélectionner l'onglet "Design", se trouvant en bas du fichier.



Un **layout** est constitué d'une hiérarchie de vues (**View**) [3] et de conteneurs de vues (**ViewGroup**) [4], sachant que la racine de tout layout doit être un conteneur de vues. Les vues possèdent un nombre important d'attributs XML, permettant de définir leurs propriétés (texte, couleur, margin, ...).

Android fournit une variété de vues, distinguée en widgets et layouts :

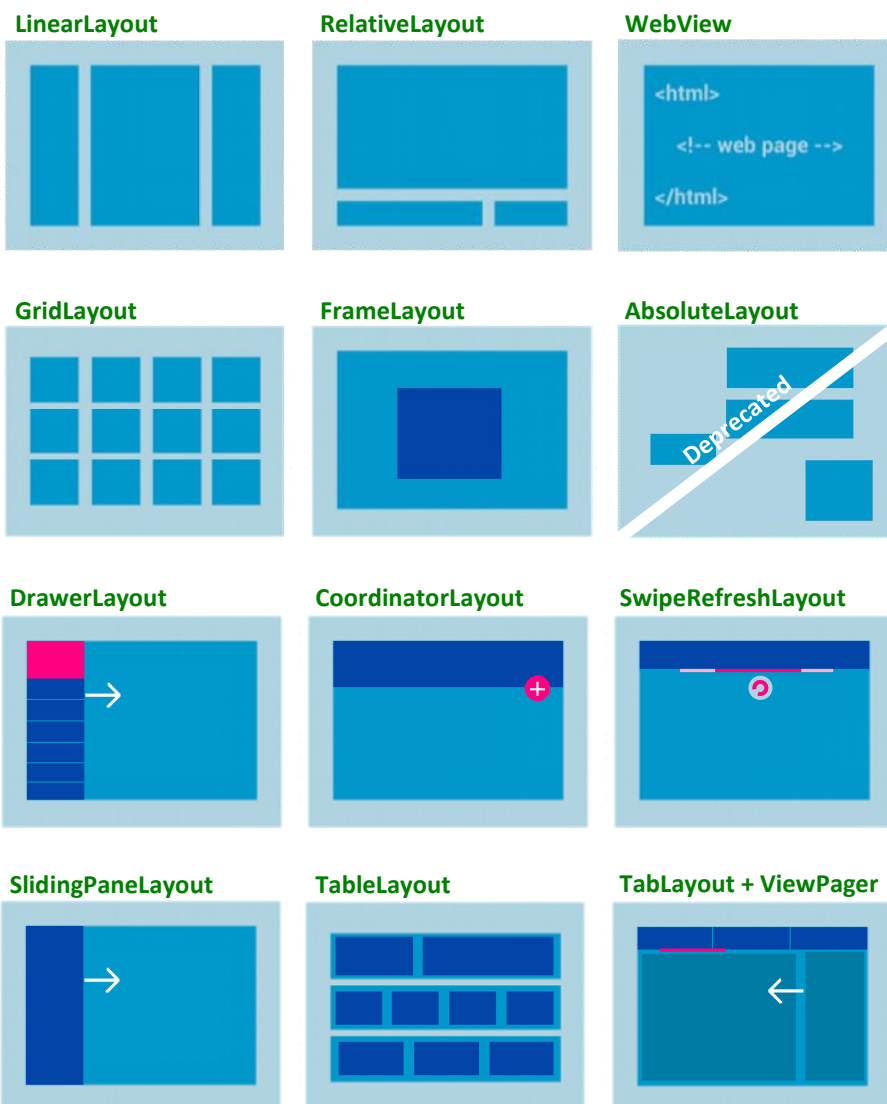
Widgets

Un widget est une vue élémentaire héritant de la classe **View**. il permet d'afficher du contenu et d'interagir avec l'application. Les widgets sont souvent sensibles à des événements (clic, frappe de clavier, ...). Voici quelques widgets : **Button**, **EditText**, **TextView**, **CheckBox** et **RadioButton**.

Layouts

Un layout est un conteneur de vues (**ViewGroup**) prédéfinis dans Android, permettant de regrouper des vues. Chaque layout propose un style de mise en page différent permettant aux vues de se positionner les unes par rapport aux autres ou par rapport au conteneur parent [5]. Les principaux layouts prédéfinis sont :

- **LinearLayout** permet d'organiser les vues horizontalement ou verticalement,
- **RelativeLayout** consiste à placer des vues les unes en fonction des autres,
- **ScrollView** est un layout qui fait défiler le contenu d'une vue qui dépasse la taille de l'écran,
- **WebView** permet d'afficher du contenu Web (HTML),
- **GridLayout** sert à ajouter des vues suivant une grille définie par un nombre de lignes et de colonnes,
- **FrameLayout** permet d'afficher une unique vue, mais il est possible d'y mettre plusieurs qui seront superposées les unes sur les autres.



- **AbsoluteLayout** permet de disposer des vues en fonction de coordonnées X et Y. Ce layout est obsolète (deprecated) depuis l'API 3, bien qu'il reste disponible, il est déconseillé de l'utiliser,
- **DrawerLayout**, appelé aussi "Burger Menu", est un menu caché qui apparaît en swipant (balayant l'écran) de gauche à droite ou en appuyant sur les 3 traits.
- **CoordinatorLayout** consiste à rajouter des dépendances entre plusieurs vues adjacentes. Par exemple, la barre de l'application (App Bar) qui diminue en défilant la vue principale,
- **SwipeRefreshLayout** permet de rafraîchir une vue juste en swipant du haut vers le bas,
- **SlidingPanelLayout** est utilisé pour swiper entre plusieurs volets de gauche à droite,
- **TableLayout** permet d'organiser des vues en tableau avec des lignes et des colonnes, comme en HTML.
- **TabLayout** fournit une disposition horizontale des onglets, tandis que **ViewPager** est utilisé pour swiper des volets en se synchronisant avec les onglets.

Tous ces layouts permettent de faciliter la gestion des vues. Ainsi, il est possible de gérer l'affichage des écrans de l'application en utilisant les conteneurs adaptés à sa situation.

2.2. LinearLayout

Un **LinearLayout** [6] est une vue de type **ViewGroup** qui contient des vues alignées horizontalement ou verticalement. Ces principaux attributs sont :

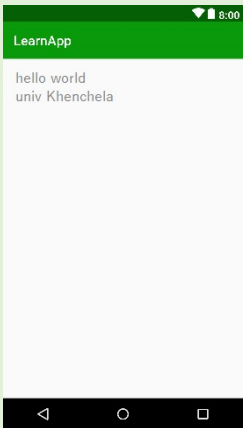
- **android:orientation : vertical | horizontal**
- **android:layout_width et android:layout_height :**
 - Valeur constante : en dp ou px
 - **wrap_content** : place minimum
 - **match_parent** : taille du parent
- **android:gravity : top, bottom, left, right, center, ...**
- **android:padding et android:margin** : en dp ou px

Voici un exemple :

```

1  <?xml version="1.0" encoding="utf8"?>
2  <LinearLayout
3      xmlns:android="http://schemas.android.com/apk/res/..."
4      android:orientation="vertical"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent">
7      <TextView
8          android:layout_width="match_parent"
9          android:layout_height="wrap_content"
10         android:text="hello world!" />
11     <TextView
12         android:layout_width="match_parent"
13         android:layout_height="wrap_content"
14         android:text="univ Khenchela" />
15 </LinearLayout>

```



2.3. Référence à une ressource (Fichier R)

Les ressources sont utilisées grâce à un identifiant. De ce fait, un fichier, appelé **R.java**, [7], est généré par Android Studio, permettant de référencer toutes les ressources définies dans **res** à partir de fichiers source Java. N'étant pas modifiable, le fichier **R.java** est mis à jour dynamiquement lorsqu'une ressource est créée ou modifiée.

Les références dans **R** sont représentées par des constantes de type entier sur 32 bits, par exemple : **0x1A34F678**.

À l'intérieur de classe R sont définies plusieurs classes internes suivant le type de ressources, telles que **string**, **drawable**, **layout**, **id**, etc. De ce fait, la syntaxe d'une référence suit l'organisation des ressources, et doit être:

- Dans le code source (java) : `R.[type_de_ressource].[nom_de_la_ressource]`
- Dans les ressources (res) : `@[type_de_ressource]/[nom_de_la_ressource]`



Emplacement du fichier R

Il est possible de visualiser le fichier R, en parcourant le projet Android :

- `\app\build\generated\source\r\debug\[package]\R.java`

Par ailleurs, il existe 2 types de ressources via R :

- **Ressources définies par le développeur** : ce sont toutes les ressources se trouvant dans le dossier **res**, par exemple `R.color.vert` (dans `res : @color/vert`)
- **Ressources prédéfinies dans Android** : Pour référencer les ressources qui sont prédéfinies dans Android, il faut utiliser `android.R.color.black` (dans `res : @android:color/black`)

2.4. Strings

Comme bonnes pratiques à appliquer dans un projet Android, il faut référencer toutes les variables (textes, messages courts, titres, ...) de l'application dans le fichier ressources **res/values/strings.xml**. Ceci permettrait de faciliter l'internationalisation de l'application.



Types de strings

Il est à savoir qu'il existe 5 types de strings : simple, avec paramètres, singulier/pluriel, tableaux et au format HTML. Nous abordons uniquement les strings simple dans ce cours [8].

Voici un exemple du fichier **string.xml** :

```
res/values/strings.xml
1 <?xml version="1.0" encoding="utf8"?>
2 <resources>
3     <string name="app_name"> LearnApp </string>
4     ...
5 </resources>
```

Les strings simples s'utilisent, soit dans **res** (layout, styles, ...) en utilisant son identifiant, par exemple `@string/app_name`, soit dans **java** (code source), l'accès au string se fait, par exemple `getString(R.string.app_name)`. La classe R sera abordée dans la section suivante.

Comme pour les strings, il est possible de réutiliser des dimensions et couleurs préalablement créés dans les fichiers **res/values/dimens.xml** et **res/values/colors.xml**.

2.5. Identification d'une vue

Quand une vue possède un identifiant, Android s'occupe de rajouter dans le fichier R une référence à cet identifiant. Dans un layout, `@+id` permet de définir une référence à une vue, par exemple un **TextView** (ligne 11).

```
res/layout/activity_login.xml
1 <?xml version="1.0" encoding="utf8"?>
2 <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:orientation="vertical"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent">
```



```

7      <TextView
8          android:layout_width="match_parent"
9          android:layout_height="wrap_content"
10         android:text="@string/app_name"
11         android:id="@+id/msgTV"/>
12 </LinearLayout>

```

Pour récupérer dans le code source Java le **TextView** identifié par **msgTV** et changer son texte, il suffit d'utiliser la méthode **findViewById(int id)**, qui renvoie un objet **View** référencé par **id**. Pour manipuler l'objet en tant que **TextView** il faut le down-caster (ligne 9).

java/LoginActivity.java

```

1  public class LoginActivity extends Activity {
2      TextView msgTextView;
3
4      @Override
5      public void onCreate(Bundle savedInstanceState) {
6          super.onCreate(savedInstanceState);
7          setContentView(R.layout.activity_login);
8          ...
9          msgTextView = (TextView) findViewById(R.id.msgTV);
10         msgTextView.setText("Hello World!");
11     }
12 }

```

2.6. ImageView

Pour afficher une image dans une vue, il suffit de :

- 1) Déposer le fichier de l'image dans **res/drawable**, par exemple **my_image.png**,
- 2) Utiliser le composant **ImageView** [9] dans le **layout**,



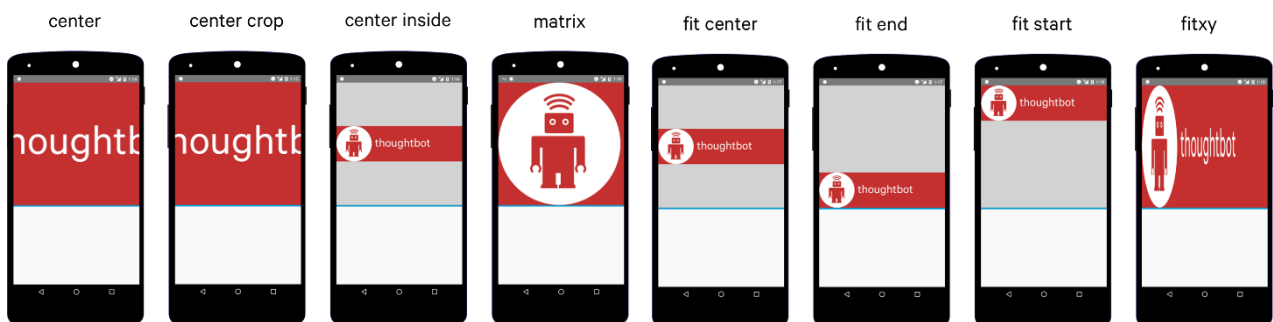
res/layout/activity_login.xml

```

1  ...
2  <ImageView
3      android:layout_width="60dp"
4      android:layout_height="60dp"
5      android:src="@drawable/myimage"
6      android:scaleType="fitCenter" />

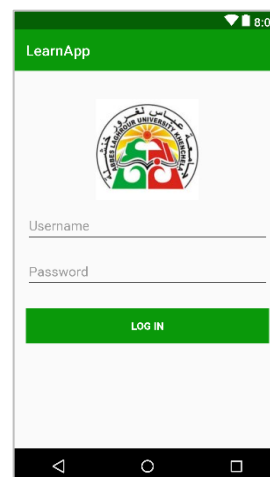
```

- 3) Référencer l'image en utilisant **@drawable/my_image** dans l'attribut **android:src** de l'**ImageView** (ligne 5),
- 4) Choisir l'échelle de l'image par rapport à l'**ImageView** parmi les options suivantes [10], en utilisant l'attribut **android:scaleType** (ligne 6).



3. Travail pratique (TP1a) : Vue d'authentification

Le TP 1 consiste à implémenter une activité d'authentification sous Android. Dans la partie 1, il faut élaborer la vue de cette activité qui doit être identique à la capture d'écran suivante (Voir l'énoncé du TP1).



Liens utiles

Les étudiants peuvent consulter ces références pour approfondir leurs connaissances dans ce cours :

- Cycle de vie d'une activité : <https://openclassrooms.com/courses/creez-des-applications-pour-android/preambule-quelques-concepts-avances#r-2032203>
- Types de strings : <http://mathias-seguy.developpez.com/tutoriels/android/utiliser-ressources#LII-A>
- Echelles d'une image : <https://robots.thoughtbot.com/android-imageview-scaletype-a-visual-guide>

Références

- [1] Android Developer, «References - Android Platform API 26 : Class Activity,» [En ligne]. Available: <https://developer.android.com/reference/android/app/Activity.html>. [Accès le 2017].
- [2] Android Developer, «The Activity Lifecycle,» [En ligne]. Available: <https://developer.android.com/guide/components/activities/activity-lifecycle.html>. [Accès le 2017].
- [3] Android Developer, «References - Android Platform API 26 : Class View,» [En ligne]. Available: <https://developer.android.com/reference/android/view/View.html>. [Accès le 2017].
- [4] Android Developer, «References - Android Platform API 26 : Class ViewGroup,» [En ligne]. Available: <https://developer.android.com/reference/android/view/ViewGroup.html>. [Accès le 2017].
- [5] V. Tallapudi, «Android Layouts and Types of Android Layouts,» 2014. [En ligne]. Available: <https://www.coderefer.com/android-layouts-types-android-layouts/>.
- [6] Android Developer, «References - Android Platform API 26 : Linear Layout,» [En ligne]. Available: <https://developer.android.com/guide/topics/ui/layout/linear.html>. [Accès le 2017].
- [7] Android Developer, «References - Android Platform API 26 : R,» [En ligne]. Available: <https://developer.android.com/reference/android/R.html>. [Accès le 2017].
- [8] Android Developer, «References - Android Platform API 26 : String Resources,» [En ligne]. Available: <https://developer.android.com/guide/topics/resources/string-resource.html>. [Accès le 2017].
- [9] Android Developer, «References - Android Platform API 26 : ImageView,» [En ligne]. Available: <https://developer.android.com/reference/android/widget/ImageView.html>. [Accès le 2017].
- [10] A. Hill, «Android ImageView ScaleType: A Visual Guide,» 25 Août 2016. [En ligne]. Available: <https://robots.thoughtbot.com/android-imageview-scaletype-a-visual-guide>.