

## Table des matières

<b>1. Contenu d'un projet Android .....</b>	<b>1</b>
1.1. Manifest .....	2
1.2. Ressources .....	3
1.2.1. Valeurs textuelles .....	4
1.2.1.1. Tableaux de valeurs .....	4
1.2.1.2. Placeholders .....	4
1.2.1.3. Gestion du pluriel .....	5
1.2.1.4. Dimensions .....	5
1.2.1.5. Couleurs .....	5
1.2.1.6. Thèmes .....	6
1.2.2. Drawables .....	6
1.2.2.1. Bitmap .....	6
1.2.2.2. Vector .....	7
1.2.2.3. Nine-Patch .....	7
1.2.2.4. State List .....	8
1.2.2.5. Level List .....	9
1.2.3. Layouts .....	9
1.2.3.1. Legacy Layouts .....	9
1.2.3.1.1. LinearLayout .....	10
1.2.3.1.2. RelativeLayout .....	10
1.2.3.2. ConstraintLayout .....	10
1.3. Code .....	11
1.4. Scripts de build .....	11
1.5. Fichiers de configuration .....	11

# 1. Contenu d'un projet Android

## 1.1. Manifest

Le fichier Manifest est obligatoire, il décrit les informations essentielles de l'application aux outils de **build**, au **système d'exploitation** et au **Google Play Store**.

Le fichier doit contenir les informations suivantes:

- Les **composants de l'application** ainsi que le nom de la classe Java/Kotlin associée ce qui inclut:
  - Activités
  - Services
  - Broadcast Receivers
  - Content Providers
- Les **permissions** requises par l'application
- Les fonctionnalités **hardware** et **software** utilisées par l'application

### ⚠ – Warning

Une partie de ces informations sont générées automatiquement par l'IDE, mais pas toutes.

```

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

  <application
    android:label="@string/app_name"
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:allowBackup="true"
    android:supportRtl="true"
    android:theme="@style/Theme.MyApplication">

    <activity
      android:name="ch.heigvd.iict.myapplication.MainActivity"
      android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

  </application>
</manifest>

```

Nom de l'application (référence) → `android:label="@string/app_name"`

Icone de l'application (référence) → `android:icon="@mipmap/ic_launcher"`

Thème de l'application (référence) → `android:theme="@style/Theme.MyApplication"`

Activité principale de l'application → `<activity>`

Package de la classe *Kotlin / Java* implémentant l'activité → `android:name="ch.heigvd.iict.myapplication.MainActivity"`

Permet au système de savoir quelle activité doit être lancée lorsque l'utilisateur clique sur l'icone de l'application → `<category android:name="android.intent.category.LAUNCHER" />`

Figure 1: Capture des slides de cours correspondant - Exemple de base

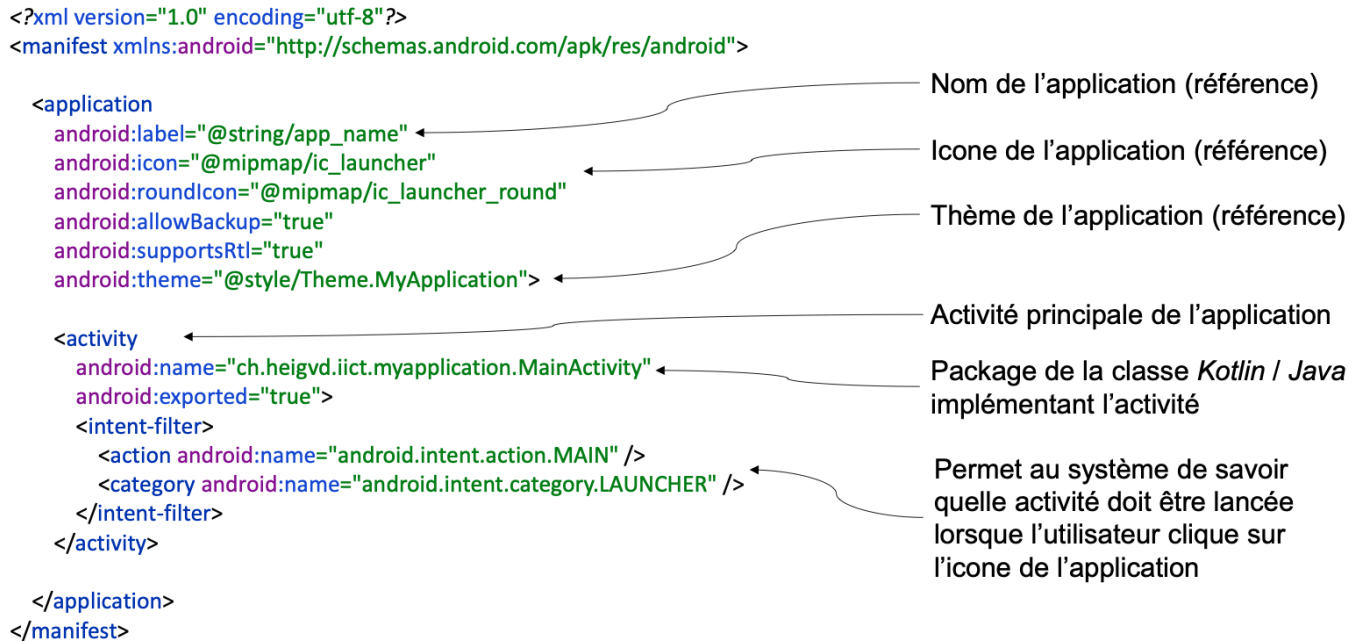


Figure 2: Capture des slides de cours correspondant - Exemple plus complet

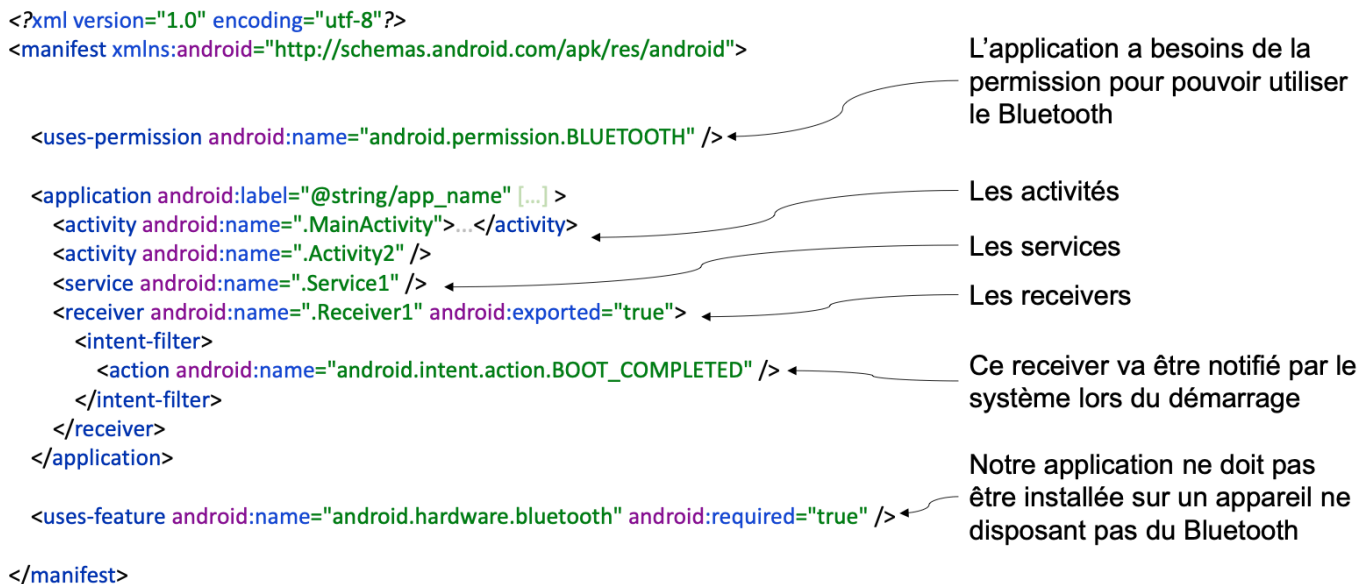


Figure 3: Capture des slides de cours correspondant - Exemple plus complet suite

## 1.2. Ressources

Le dossier des ressources contient tous les fichiers et contenus statiques utilisés par l'application. Les bonnes pratiques recommandent de séparer les ressources du code pour faciliter leur maintenance et permettre à l'application de s'adapter automatiquement au contexte d'exécution (taille d'écran, langue, etc.):

- langue et région de l'appareil
- taille / résolution / orientation de l'écran
- version d'Android
- opérateur de téléphonie
- sens de l'écriture
- etc...

Il existe plusieurs types de ressources:

- **Valeurs:** chaînes de caractères, dimensions, couleurs, styles, etc.
- **Images:** fichiers PNG, JPEG, SVG, etc.
- **Layouts:** interfaces graphiques de l'application
- **Animations**

## • Menus

La classe `R` est une classe générée automatiquement par Android qui contient des identifiants pour toutes les ressources de l'application. Elle permet d'accéder facilement aux ressources dans le code Java/Kotlin en utilisant des références typées.

### 1.2.1. Valeurs textuelles

Les valeurs textuelles sont stockées dans des fichiers XML situés dans le dossier `res/values`. Le fichier le plus courant est `strings.xml`, qui contient toutes les chaînes de caractères utilisées dans l'application. Voici un exemple de contenu pour `strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">My Application</string>
  <string name="main_title">My Activity</string>
  <string name="main_username_title">Username</string>
</resources>
```

Celle-ci sont toujours composé de la manière suivante:

```
<type name="identifiant">Valeur</type>
```

En créant d'autres dossier `values`, on peut définir des variantes pour différentes configurations. Par exemple, pour les langues, on peut créer des dossiers `values-fr` pour le français et `values-es` pour l'espagnol.

#### values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">My Application</string>
  <string name="main_title">My Activity</string>
  <string name="main_username_title">Username</string>
</resources>
```

#### values-fr/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Mon application</string>
  <string name="main_title">Mon activité</string>
  <string name="main_username_title">Nom d'utilisateur</string>
</resources>
```

Les apostrophes doivent être protégées

Figure 4: Capture des slides de cours correspondant - Gestion de plusieurs langues

#### 💡 – Hint

Les valeurs textuelles peuvent être formatées avec des balises HTML **simples** pour ajouter des styles (gras, italique, etc.).

```
<b>, <i>, <s>, <u>, <big>, <small>, <sup>, <sub>, <font>, <ul>, <li>, <br>, <div>, <p>
```

#### 1.2.1.1. Tableaux de valeurs

Les valeurs textuelles peuvent être regroupées en tableaux:

```
<string-array translatable="false"
name="countries">
  <item>@string/switzerland</item>
  <item>@string/france</item>
  <item>@string/germany</item>
  <item>@string/italy</item>
</string-array>
```

- Bien que les items du tableau puissent directement contenir du texte, nous utilisons ici des références vers d'autres valeurs textuelles
- Cela évite de devoir redéfinir le tableau pour toutes les langues (notez le paramètre `translatable` à `false`)

#### 1.2.1.2. Placeholders

Les valeurs textuelles peuvent contenir des **placeholders** pour insérer dynamiquement des valeurs dans les chaînes de caractères. Les placeholders sont définis en utilisant la syntaxe `%n$s` où `n` est l'index du paramètre (commençant à 1) et `s` indique que le paramètre est une chaîne de caractères.

```
<string name="welcome_messages">Hello, %1$s! You have %2$d new messages.</string>
```

### 1.2.1.3. Gestion du pluriel

Pour gérer les variations de texte en fonction du nombre (singulier/pluriel), Android utilise la balise `<plurals>`. Voici un exemple:

```
<plurals name="click_counter">
  <item quantity="one">%1$d click</item>
  <item quantity="other">%1$d clicks</item>
</plurals>
```

#### ⚠ – Warning

Au minimum les quantités `one` et `other` doivent être définies. D'autres quantités peuvent être ajoutées pour des langues spécifiques (`zero`, `two`, `few`, `many`).

### 1.2.1.4. Dimensions

Les dimensions sont utilisées pour définir des tailles, des marges, des espacements, etc. Elles sont définies dans un fichier `dimension.xml` situé dans le dossier `res/values`.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <!-- Unités à privilégier -->
  <dimen name="main_title_margin">150dp</dimen> <!-- Density-independent Pixels -->
  <dimen name="main_title_textsize">22sp</dimen> <!-- Scale-independent Pixels -->
  <!-- Unités à éviter -->
  <dimen name="main_logo_width">200px</dimen> <!-- Pixels - Actual pixels on the screen -->
  <dimen name="main_actionbar_height">20mm</dimen> <!-- Millimeters - Physical size of the screen -->
  <dimen name="main_title_spacer">0.5in</dimen> <!-- Inches - Physical size of the screen -->
</resources>
```

Il est recommandé de privilégier les unités `dp` et `sp` pour assurer une bonne adaptabilité de l'interface sur différents appareils.

- `dp` (density-independent pixels): utilisé pour les dimensions générales (marges, espacements, etc.). Il s'adapte à la densité de pixels de l'écran. 1dp est équivalent à 1 pixel sur un écran de densité moyenne (160 dpi).
- `sp` (scale-independent pixels): utilisé pour la taille du texte. Il s'adapte à la taille de police préférée de l'utilisateur. 1sp est équivalent à 1 pixel sur un écran de densité moyenne (160 dpi), mais il peut être redimensionné en fonction des préférences de l'utilisateur.

#### 💡 – Hint

Il est possible d'adapter les dimensions en fonction de la dimension de l'écran ou du **form factor** notamment pour les tablettes. Pour cela il suffit de créer des dossiers `values` spécifiques:

- `values-sw600dp` : pour les écrans ayant une largeur minimale de 600dp
- `values-sw720dp` : pour les écrans ayant une largeur minimale de 720dp
- etc...

### 1.2.1.5. Couleurs

Les couleurs sont définies dans un fichier `colors.xml` situé dans le dossier `res/values`.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="purple_200">#FFBB86FC</color>
  <color name="purple_500">#FF6200EE</color>
  <color name="purple_700">#FF3700B3</color>
  <color name="teal_200">#FF03DAC5</color>
  <color name="teal_700">#FF018786</color>
```

```
<color name="black">#FF000000</color>
<color name="white">#FFFFFFFF</color>
</resources>
```

### 1.2.1.6. Thèmes

Les thèmes permettent de définir un ensemble cohérent de styles pour l'application. Ils sont définis dans un fichier `themes.xml` situé dans le dossier `res/values`.

`values/themes.xml`

```
<resources xmlns:tools="http://schemas.
android.com/tools">
<!-- Base application theme -->
<style name="Theme.MyApplication"
parent="[...].NoActionBar">
<!-- Primary brand color -->
<item name="colorPrimary">@color/
purple_500</item>
<item name="colorPrimaryVariant">@color/
purple_700</item>
<item name="colorOnPrimary">@color/white</
item>
<!--
...
-->
</style>
</resources>
```

`values-night/themes.xml`

```
<resources xmlns:tools="http://schemas.
android.com/tools">
<!-- Base application theme -->
<style name="Theme.MyApplication"
parent="[...].NoActionBar">
<!-- Primary brand color -->
<item name="colorPrimary">@color/
purple_200</item>
<item name="colorPrimaryVariant">@color/
purple_700</item>
<item name="colorOnPrimary">@color/black</
item>
<!--
...
-->
</style>
</resources>
```

## 1.2.2. Drawables

### 1.2.2.1. Bitmap

Les images bitmap sont des images rasterisées composées de pixels. Elles sont généralement utilisées pour des images complexes avec de nombreux détails, comme des photographies.

Android supporte plusieurs formats d'images bitmap, les plus courants étant:

- PNG
- WEBP
- JPEG
- GIF (découragé)

Les images doivent être placées dans les dossier drawable à l'exception du logo de l'application qui doit être placé dans le dossier `mipmap`.



#### – Info

Le nom du fichier (sans l'extension) est utilisé comme identifiant de la ressource `@drawable/filename`.

Il est important de fournir des images adaptées aux différentes densités d'écran pour assurer une bonne qualité visuelle sur tous les appareils.

<code>drawable-ldpi</code>	<code>drawable-mdpi</code>	<code>drawable-hdpi</code>	<code>drawable-xhdpi</code>	<code>drawable-xxhdpi</code>	<code>drawable-xxxhdpi</code>
18x18 0.75x	24x24 1x	36x36 1.5x	48x48 2x	72x72 3x	96x96 4x

Figure 5: Capture des slides de cours correspondant - Densités d'écran et dossiers drawable

### – Info

Depuis l'API 21+ il est possible de colorer dynamiquement les images bitmap en utilisant des **tint** dans les fichiers XML de layout ou dans le code.

#### 1.2.2.2. Vector

Les images vectorielles sont des images basées sur des formes géométriques (lignes, courbes, polygones) plutôt que sur des pixels. Elles sont généralement utilisées pour des icônes et des illustrations simples.

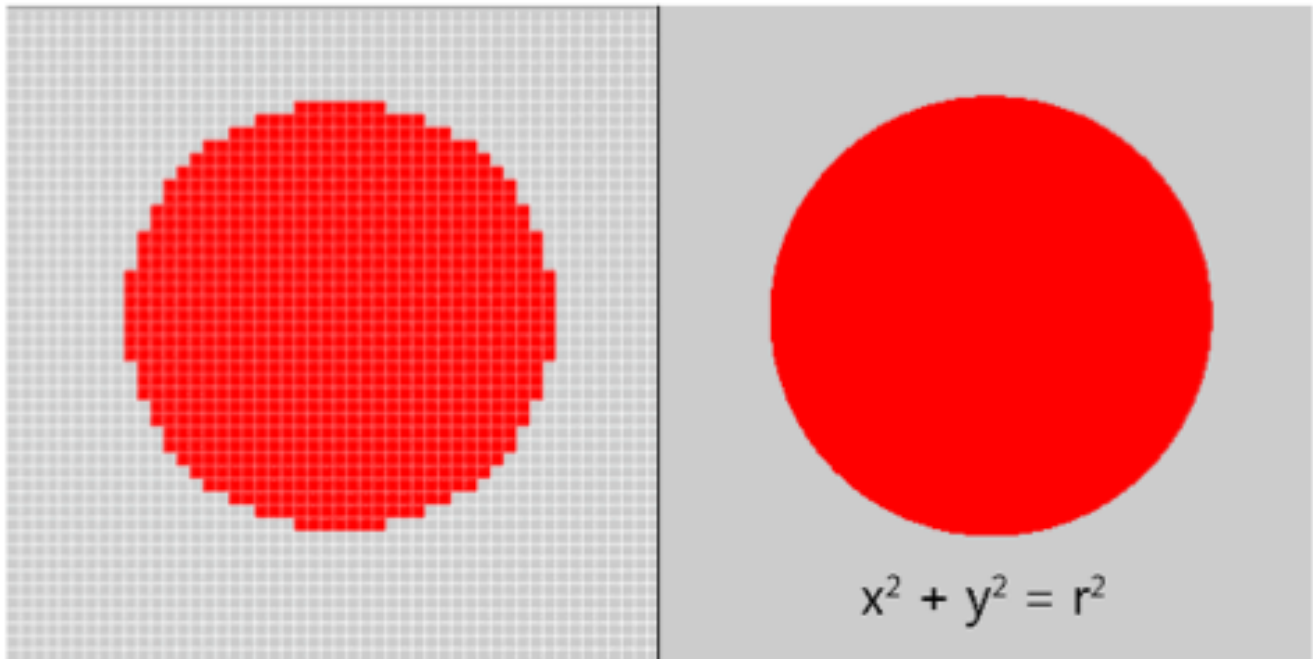


Figure 6: Capture des slides de cours correspondant - Exemple d'image vectorielle

Android utilise son propre format vectoriel. Le format `VectorDrawable` est défini dans des fichiers XML situés dans le dossier `res/drawable`.

```
<vector android:autoMirrored="true" android:height="48dp"
android:viewportHeight="24" android:viewportWidth="24"
android:width="48dp" xmlns:android="http://schemas.android.com/apk/res/android">
<path
android:fillColor="#FF000000"
android:pathData="M8,16H12V12H8V16M12,12H16V8H12V12M2,2V22H13.5C13,21.4 12.6,
20.7,12.3,20H8V16H4V12H8V8H4V12H16V8H12V4H16V8H20V12.4C20.7,12.7 21.4,13.1
22,13.6V2H2M20.1,14.5L18,16.6L15.9,14.5L14.5,15.9L16.6,18L14.5,20.1L15.9,
21.5L18,19.4L20.1,21.5L21.5,20.1L19.4,18
L21.5,15.9L20.1,14.5Z"/>
</vector>
```

#### 1.2.2.3. Nine-Patch

Les images Nine-Patch sont des images bitmap spéciales qui permettent de définir des zones extensibles et des zones de contenu. Elles sont utilisées pour créer des arrière-plans et des boutons qui peuvent s'adapter à différentes tailles tout en conservant leur apparence.

Nous souhaitons contrôler le format et éviter les distorsions.



Figure 7: Capture des slides de cours correspondant - Exemple d'image Nine-Patch

Nous voudrions plutôt avoir quelque chose comme ça:

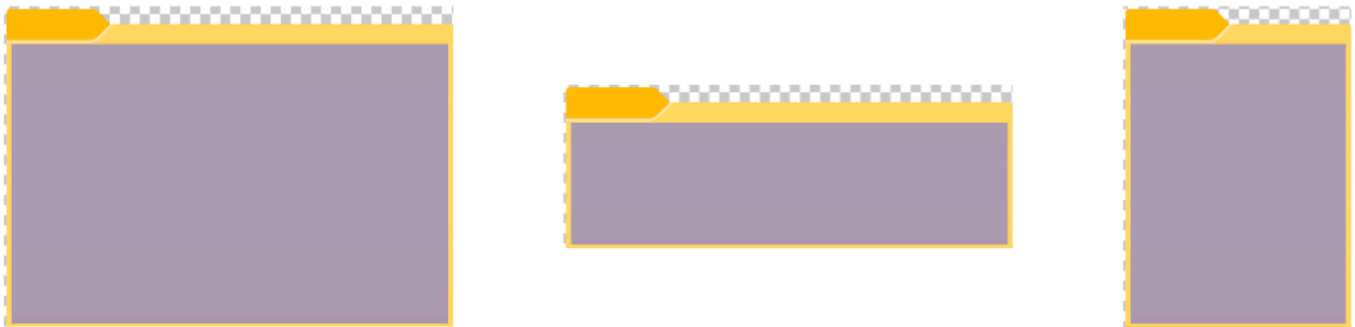



Figure 8: Capture des slides de cours correspondant - Exemple d'image Nine-Patch adaptée

Pour créer une image Nine-Patch, il faut ajouter une bordure d'un pixel autour de l'image originale. Cette bordure est utilisée pour définir les zones extensibles et les zones de contenu.

- zone redimensionnable 
- délimitée par les bandes de pixels sur la première ligne et la première colonne




- zone pour le contenu 
- délimitée par les bandes de pixels sur la dernière ligne et la dernière colonne

Figure 9: Capture des slides de cours correspondant - Exemple d'image Nine-Patch avec bordure

#### 1.2.2.4. State List

Une State List est une ressource XML qui définit un ensemble d'images ou de couleurs à utiliser en fonction de l'état d'un composant (par exemple, un bouton peut avoir des états normal, pressé, désactivé, etc.).



Figure 10: Capture des slides de cours correspondant - Exemple de State List

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true" android:drawable="@drawable/pressed" /> <!-- pressed -->
  <item android:state_focused="true" android:drawable="@drawable/focused" /> <!-- focused -->
  <item android:state_hovered="true" android:drawable="@drawable/hovered" /> <!-- hovered -->
  <item android:drawable="@drawable/normal" /> <!-- default -->
</selector>
```



### 1.2.2.5. Level List

Une Level List est une ressource XML qui définit un ensemble d'images à utiliser en fonction d'un niveau numérique. Elle est souvent utilisée pour représenter des états progressifs, comme une barre de progression ou un indicateur du niveau de wi-fi.



Figure 11: Capture des slides de cours correspondant - Exemple de Level List

```
<level-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:maxLevel="0" android:drawable="@drawable/ic_wifi_signal_1" />
  <item android:maxLevel="1" android:drawable="@drawable/ic_wifi_signal_2" />
  <item android:maxLevel="2" android:drawable="@drawable/ic_wifi_signal_3" />
  <item android:maxLevel="3" android:drawable="@drawable/ic_wifi_signal_4" />
</level-list>
```

### 1.2.3. Layouts

Les layouts définissent la structure et l'organisation des éléments d'interface utilisateur dans une activité ou un fragment. Tous les composants du layout font partie de la hiérarchie des vues. Il existe deux types principaux de layouts:

- Views: éléments d'interface utilisateur individuels, tels que des boutons, des champs de texte, des images, etc.
- ViewGroups: conteneurs qui organisent et gèrent la disposition des vues enfants, tels que LinearLayout, RelativeLayout, ConstraintLayout, etc.

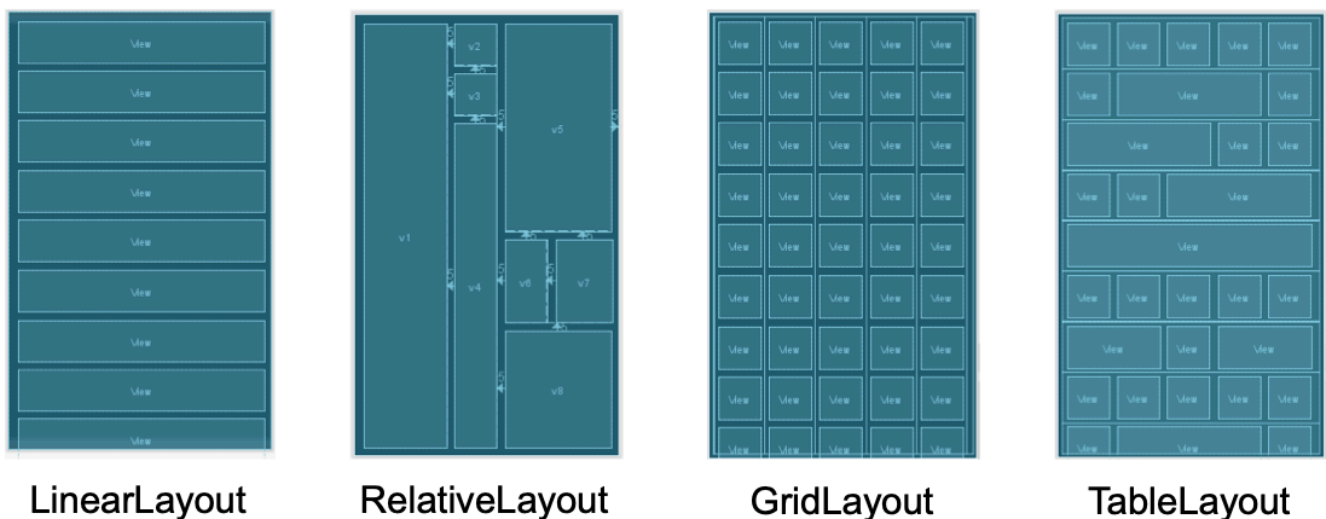


Figure 12: Capture des slides de cours correspondant - Exemple de layout XML

#### 1.2.3.1. Legacy Layouts

Avec le SDK seul, nous utilisons principalement des `LinearLayout` et des `RelativeLayout` pour organiser notre interface utilisateur.

### 1.2.3.1.1. LinearLayout

Le `LinearLayout` organise les vues enfants en une seule direction (horizontale ou verticale).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.
com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click 1"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click 2"/>
    <!-- etc -->
</LinearLayout>
```



Figure 13: Capture des slides de cours correspondant - Exemple de LinearLayout

### 1.2.3.1.2. RelativeLayout

Le `RelativeLayout` permet de positionner les vues enfants par rapport aux autres vues ou par rapport au parent.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.
com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="15dp">
    <Button
        android:id="@+id/my_button_1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Click 1"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/my_button_1"
        android:layout_alignParentEnd="true"
        android:text="Click 2"/>
</RelativeLayout>
```

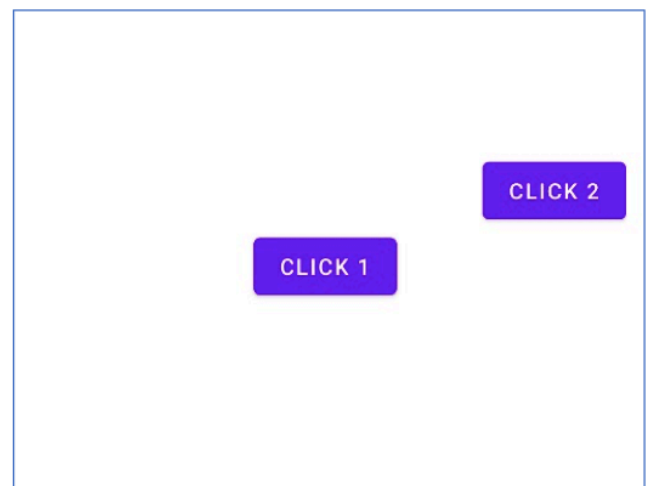


Figure 14: Capture des slides de cours correspondant - Exemple de RelativeLayout

- Le bouton 1 est centré par rapport au parent
- Le bouton 2 est positionné au-dessus du bouton 1 et aligné à droite du parent

### 1.2.3.2. ConstraintLayout

Le `ConstraintLayout` est un layout plus flexible et puissant qui permet de créer des interfaces utilisateur complexes en définissant des contraintes entre les vues.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@color/teal_200"
```

```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
android:text="Mon texte"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

- Comme il s'agit d'une librairie, les options ne font pas partie du namespace Android
- On ajoute des contraintes aux 4 bords de l'écran
- L'élément se centrera automatiquement

### 1.3. Code

### 1.4. Scripts de build

### 1.5. Fichiers de configuration