Université de Rennes

ESIR ECOLE SUPERIEURE D'INGENIEURS DE RENNES

android

Programmation Mobile

Rapport TP 2

Presenté par :

TAPE-WALI ZOKOU Junior Aristide
TRAORE Zie Amara

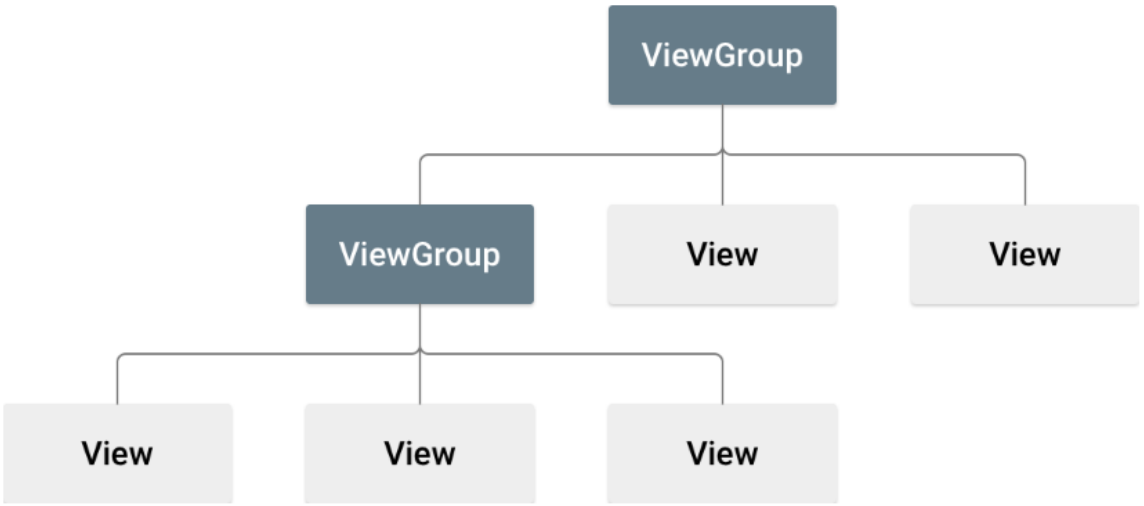
Encadré par :

Mme **Feuillatre**

SOMMAIRE

Layouts

Une **Layout** (mise en page) définit la structure d'une interface utilisateur dans notre application, par exemple dans une activité. Tous les éléments de la layout sont construits à l'aide d'une hiérarchie d'objets **View** et **ViewGroup**. Une **View** dessine généralement quelque chose que l'utilisateur peut voir et interagir avec. Alors qu'un **ViewGroup** est un conteneur invisible qui définit la structure de mise en page pour View et d'autres objets ViewGroup, comme le montre la figure suivante :



La figure précédente illustre une hiérarchie de vues, qui définit une mise en page de l'interface utilisateur.
Les objets **View** sont généralement appelés "widgets" et peuvent être l'une des nombreuses sous-classes, telles que ***Button*** ou ***TextView***. Les objets **ViewGroup** sont généralement appelés "layouts" peuvent être l'un des nombreux types qui fournissent une structure de mise en page différente, comme ***LinearLayout*** ou ***ConstraintLayout***.
Nous pouvons déclarer une layouts de deux façons :

- ★ Déclarer les éléments de l'interface utilisateur en XML. Android fournit un vocabulaire XML simple qui correspond aux classes et sous-classes View, telles que celles pour les widgets et les mises en page.
  Vous pouvez également utiliser l'éditeur de layout d'Android Studio pour créer votre mise en page XML à l'aide d'une interface glisser-déposer.
- ★ Instancier les éléments de mise en page au moment de l'exécution. Votre application peut créer des objets View et ViewGroup (et manipuler leurs propriétés) par programmation.

Nous allons déclarer notre interface utilisateur en XML. Ceci nous permettra de séparer la présentation de notre application du code qui contrôle son comportement.

L'utilisation de fichiers XML facilite également la fourniture de différents layout pour différentes tailles et orientations d'écran.
Le framework Android vous donne la flexibilité d'utiliser l'une ou l'autre de ces méthodes ou les deux pour créer l'interface utilisateur de votre application. Par exemple, nous pouvons déclarer les mises en page par défaut de notre application en XML, puis modifier la mise en page au moment de l'exécution.

1. Création d'une interface Linearlayout avec un un texte à gauche, puis un bouton à droite

En utilisant le vocabulaire XML d'Android, vous pouvez rapidement concevoir des mises en page d'interface utilisateur et les éléments d'écran qu'elles contiennent, de la même manière que vous créez des pages Web en HTML - avec une série d'éléments imbriqués.
Chaque fichier de layout doit contenir exactement un élément racine, qui doit être un objet **View** ou **ViewGroup**. Une fois que vous avez défini l'élément racine, vous pouvez ajouter des objets de mise en page ou des widgets supplémentaires en tant qu'éléments enfants pour construire progressivement une hiérarchie de vues qui définit votre mise en page. Par exemple, voici une mise en page XML qui utilise un ***LinearLayout*** vertical pour maintenir un ***TextView*** et un bouton :

TP2 > Manipulationdeslayouts > app > src > main > res > layout > main_layout.xml

```xml
<?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="horizontal" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Texte à gauche" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Bouton à droite" />
</LinearLayout>
```

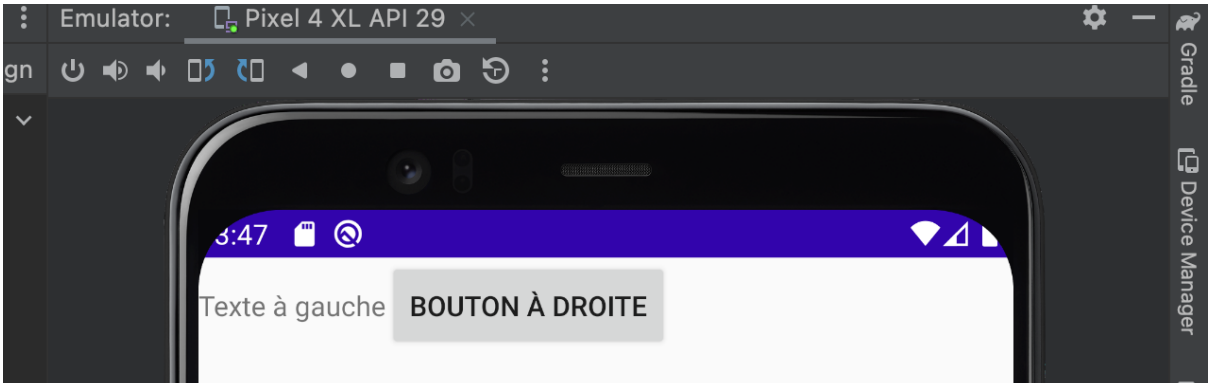
Lorsque vous compilez votre application, chaque fichier de mise en page XML est compilé dans une ressource View. Vous devez charger la ressource de mise en page à partir de votre code d'application, dans votre implémentation de rappel Activity.onCreate(). Faites-le en appelant setContentView(), en lui passant la référence à votre ressource de mise en page sous la forme de : R.layout.layout_file_name.
Par exemple, si votre mise en page XML est enregistrée en tant que main_layout.xml, vous la chargerez pour votre activité comme ceci en kotlin:

TP2 > Manipulationdeslayouts > app > src > main > java > com > example > manipulationdeslayouts > MainActivity.kt

You, 5 minutes ago | 1 author (You)

```kotlin
package com.example.manipulationdeslayouts

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent


class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main_layout)
    }
}

```

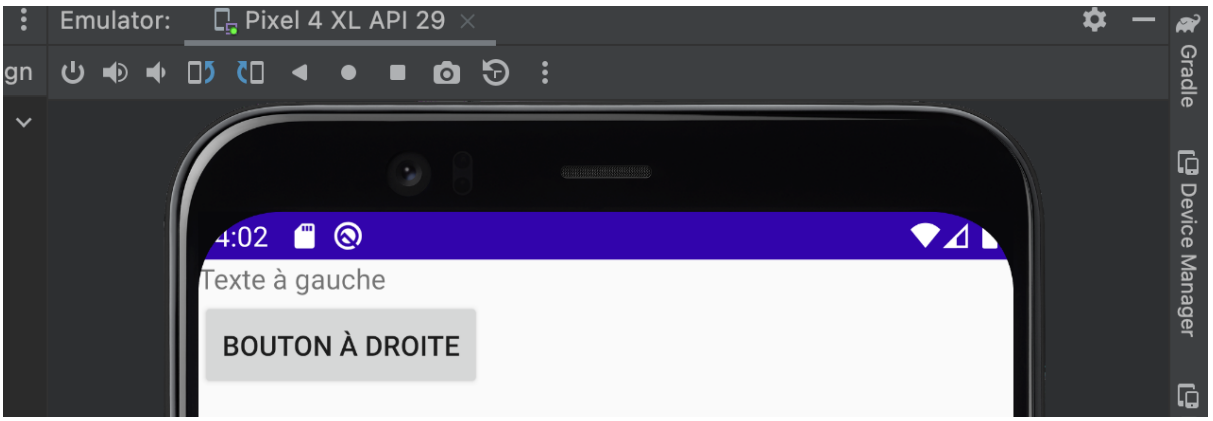
La figure suivante montre le résultat de la compilation:



LinearLayout est un type de layout dans le système d'interface utilisateur Android qui organise les éléments de la vue en une seule ligne ou une seule colonne, selon l'orientation spécifiée. Les éléments sont positionnés de manière linéaire, les uns après les autres, soit horizontalement (dans le cas d'une orientation horizontale), soit verticalement (dans le cas d'une orientation verticale).

2. Création d'une interface Linearlayout avec un texte au dessus, puis un bouton en dessous

Pour aboutir à nos fins, il nous suffit de mettre l'attribut *orientation* de LinearLayout à la valeur "*vertical*" dans le fichier de main_layout.xml. Nous avons ce resultat :



3. Création d'une interface à base de LinearLayout avec un label au dessus, puis en dessous un texte remplissable à gauche et un bouton à droite

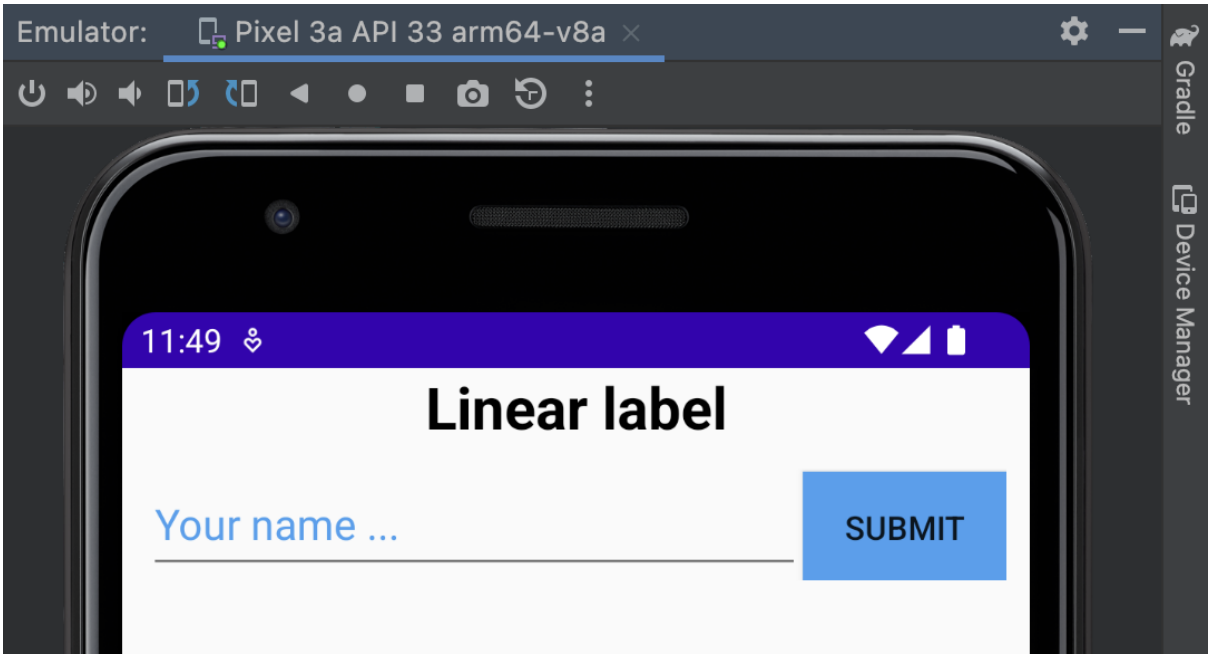
Pour ce faire nous allons d'abord créer un LinearLayout vertical comme dans la question 2. Ce dernier va contenir le **texte label** au-dessus et un **LinearLayout avec une orientation horizontale en dessous**.

```
> Manipulationdeslayouts > app > src > main > res > layout > main_layout.xml
   You, 3 minutes ago | 1 author (You)
   <?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">
>          <TextView ⋯
>          <LinearLayout      You, 1 second ago • Uncommitted changes ⋯
           </LinearLayout>
   </LinearLayout>
```

Cependant vu que le premier LinearLayout contient le second LinearLayout et le TextLabel, on sera donc obligé de modifier la taille du second LinearLayout, pour qu'il puisse mieux s'adapter à l'espace restant en fonction du Text Label avec lequel il partage l'espace. Ce que nous faisons en utilisant la valeur "*wrap_content*" pour indiquer à notre vue de se dimensionner aux dimensions requises par son contenu. Et "*Match_parent*" pour indiquer à notre vue de devenir aussi grande que son groupe de vues parent le permettra. Pour centre les éléments de nos Layout, nous avons utiliser l'attribut "*gravity*"

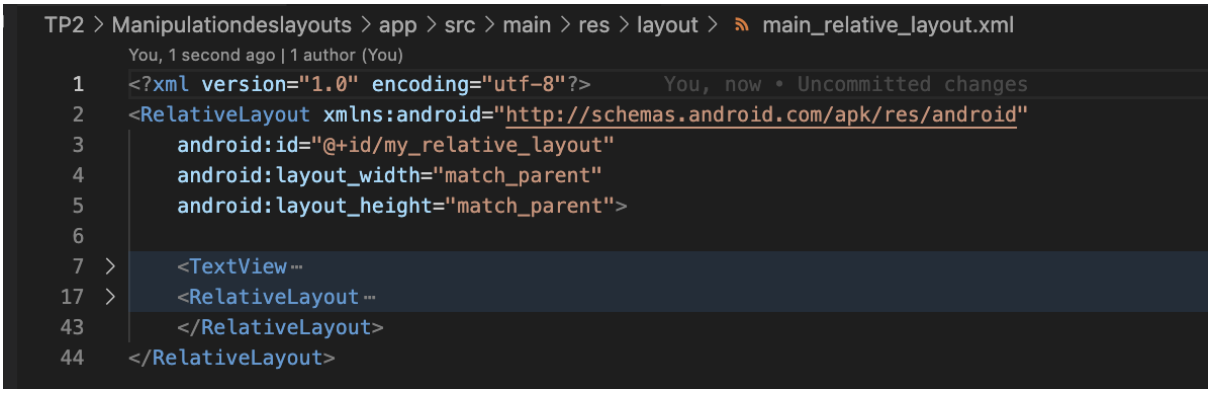
```
 8  >      <TextView ⋯
17          <LinearLayout
18              android:layout_width="match_parent"
19              android:layout_height="wrap_content"
20              android:orientation="horizontal"
21              android:layout_margin="10sp">
22
23                  <EditText
24                      android:layout_width="match_parent"
25                      android:layout_height="match_parent"
26                      android:ems="10"
27                      android:layout_weight="1"
28                      android:inputType="textPersonName"
29                      android:minHeight="48sp"
30                      android:hint="Your name ..."
31                      android:textColorHint="#41A0F0"/>
32                  <Button
33                      android:layout_width="wrap_content"
34                      android:layout_height="wrap_content"
35                      android:background="#41A0F0"
36                      android:text="submit" />
37          </LinearLayout>
38  </LinearLayout>
```

Le rendu du code est le suivant :



4. Faire la même chose qu'à la question précédente avec un Relative Layout

RelativeLayout est un type de layout dans le système d'interface utilisateur Android qui permet de positionner les éléments de la vue relativement les uns aux autres, plutôt que de les organiser en fonction d'une grille ou d'une liste fixe.

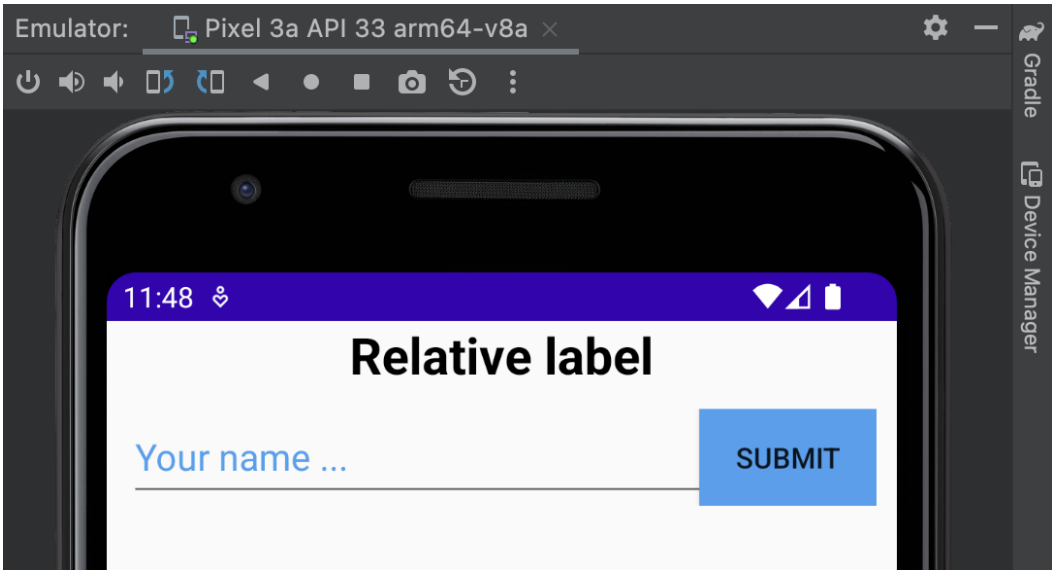


Nous avons centré horizontalement le Label (“Relative label") au-dessus, en mettant l’attribut layout_centerHorizontal à true. le positionnement des éléments les uns par rapport aux autres du RelativeLayout est possible grâce à la définition d’un id pour chacun de ces éléments. Tout objet View peut avoir un ID entier qui lui est associé, pour identifier de manière unique la vue dans l'arborescence. Lorsque l'application est compilée, cet ID est référencé en tant qu'entier, mais l'ID est généralement attribué dans le fichier XML de mise en page sous forme de chaîne, dans l'attribut id. Il s'agit d'un attribut XML commun à tous les objets View (définis par la classe View). La syntaxe d'un ID, à l'intérieur d'une balise XML est : ***android:id="@+id/my_button"***

La suite code qui répondra à la question 4 :

```
    <TextView
        android:id="@+id/my_title_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:textColor="#000"
        android:textStyle="bold"
        android:textSize="25sp"
        android:text="Relative label" />

    <RelativeLayout
        android:id="@+id/my_relative_layout_0"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@id/my_title_text"
        android:layout_margin="10sp">

        <EditText
            android:id="@+id/my_edit_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="14"
            android:hint="Your name ..."
            android:inputType="textPersonName"      You, 2 minutes ago • Uncommitted chan
            android:minHeight="48sp"
            android:textColorHint="#41A0F0" />

        <Button
            android:id="@+id/my_submit_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:background="#41A0F0"
            android:text="submit" />
    </RelativeLayout>
</RelativeLayout>
```

Le rendu du code est le suivant :



5. Création d'une interface avec une listView qui affichera le contenu d'une arraylist

Il s'agit ici de créer une interface avec une **listView** qui affichera le contenu d'une arraylist que nous aurons rempli auparavant avec le nom des 4 départements bretons. Dans notre fichier xml, nous avons procédé comme suite: création d'une

TP2 > Manipulationdeslayouts > app > src > main > res > layout > main_departements_breton_layout.xml

You, 1 second ago | 1 author (You)

```xml
<?xml version="1.0" encoding="utf-8"?>      You, 1 second ago • Uncommitted changes
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ListView
        android:id="@+id/my_listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

LinearLayout en orientation verticale, dans laquelle nous avons inséré une ListView avec l'id : ***id="@+id/my_listView"*** .

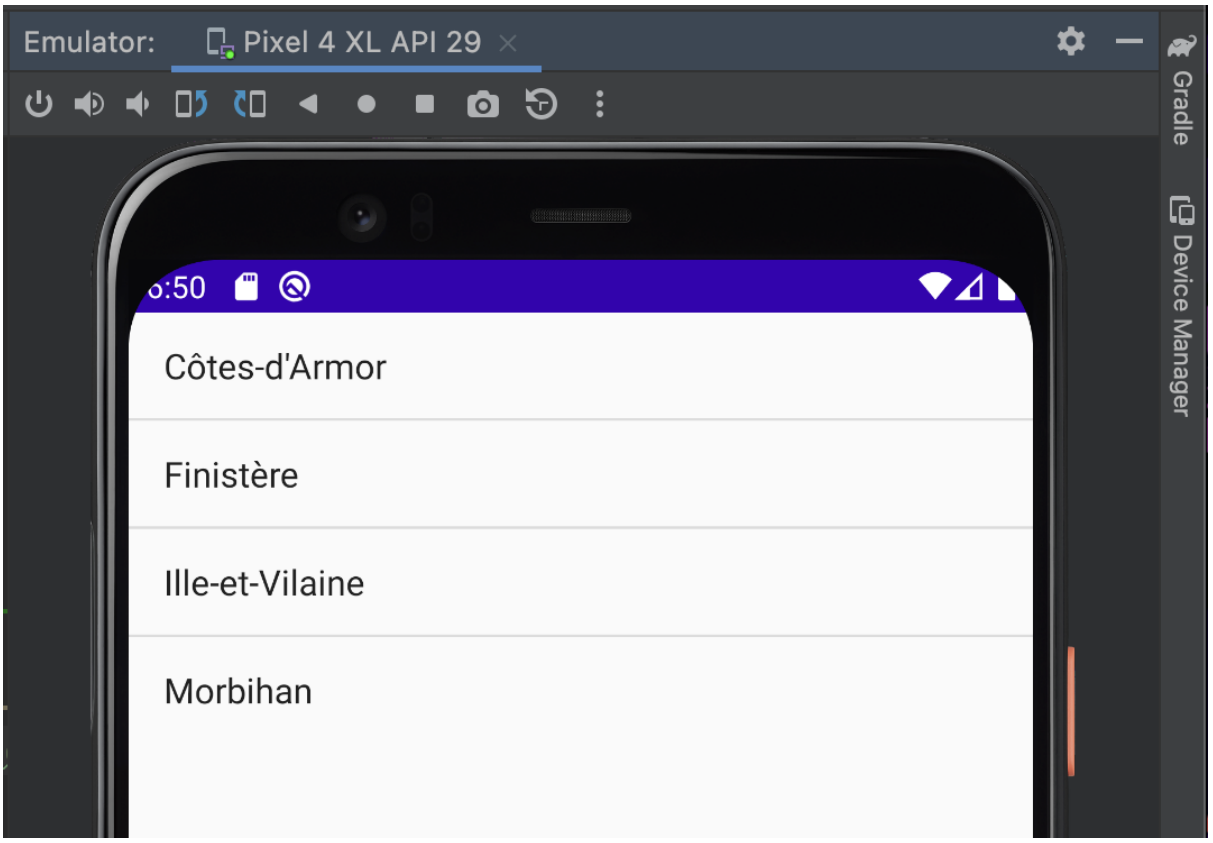
Ensuite dans le fichier MainActivity.kt grâce à **arrayListOf**, nous avons créé un tableau de chaînes que nous souhaitons afficher dans la ListView. Puis à l'aide de la classe **ArrayAdapter** et de sa méthode , Nous avons fait une mise en page dynamique avec tous les éléments de tableau. ArrayAdapter est une classe qui permet de lier une liste de données à une vue ListView dans une application Android. Elle prend en entrée un contexte, une vue de mise en forme prédéfinie pour chaque élément de la liste, et une liste de données à afficher. L'ArrayAdapter s'occupe de la gestion de la liste de données et de leur affichage dans la vue ListView.

TP2 > Manipulationdeslayouts > app > src > main > java > com > example > manipulationdeslayouts > MainActivity.kt

You, 1 second ago | 1 author (You)

```kotlin
package com.example.manipulationdeslayouts      You, 6 days ago • initialisation du TP2 ...

import android.os.Bundle
import android.widget.ArrayAdapter
import android.widget.ListView
import androidx.activity.ComponentActivity

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main_departements_breton_layout)

        // Création de la liste des départements bretons
        val departementsBretons = arrayListOf("Côtes-d'Armor", "Finistère", "Ille-et-Vila

        // Création de l'adaptateur pour la ListView
        val adapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, dep
        // Liaison de l'adaptateur avec la ListView
        val listView: ListView = findViewById(R.id.my_listView)
        listView.adapter = adapter

    }
}
```

Le rendu final de cette question donne ceci :

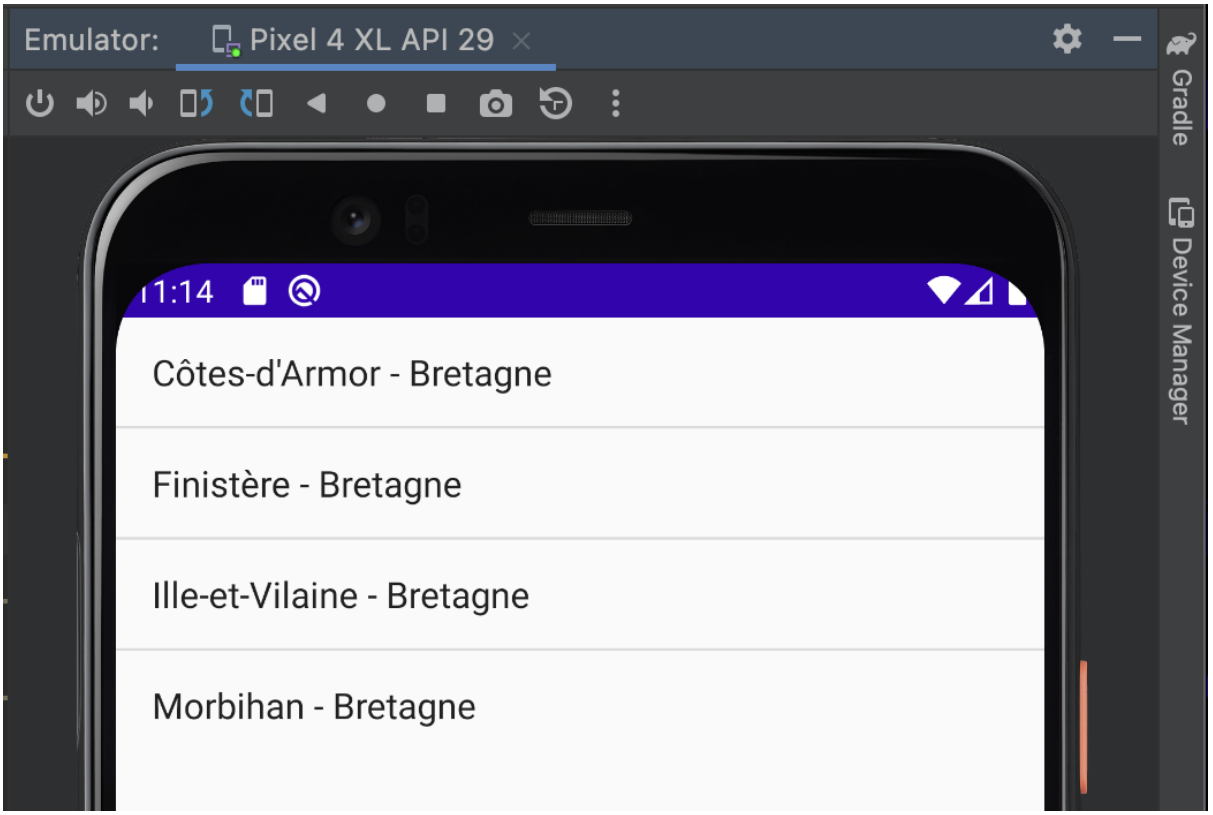


6. Ajout du suffixe “ - Bretagne” aux noms des départements

En nous appuyant sur ce que nous avons fait à la question 5, nous allons ajouter à tous les éléments de notre ListView, le suffixe “ - Bretagne”. A l’aide d’une boucle for, nous avons fait la modification par index de chaque département en y ajoutant le suffixe “ - Bretagne”. Cette opération a écrasé le contenu du tableau à l’index correspondant par son ancien contenu concaténé à la valeur fixe “ - Bretagne”.

```
24        // Création de l'adaptateur pour la ListView
25        val departements = ArrayList<String>()
26        for (departement in departementsBretons) {
27            departements.add("$departement - Bretagne")
28        }
29        val adapter = ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, departements)
30        // Liaison de l'adaptateur avec la ListView
```

Le rendu final donne :

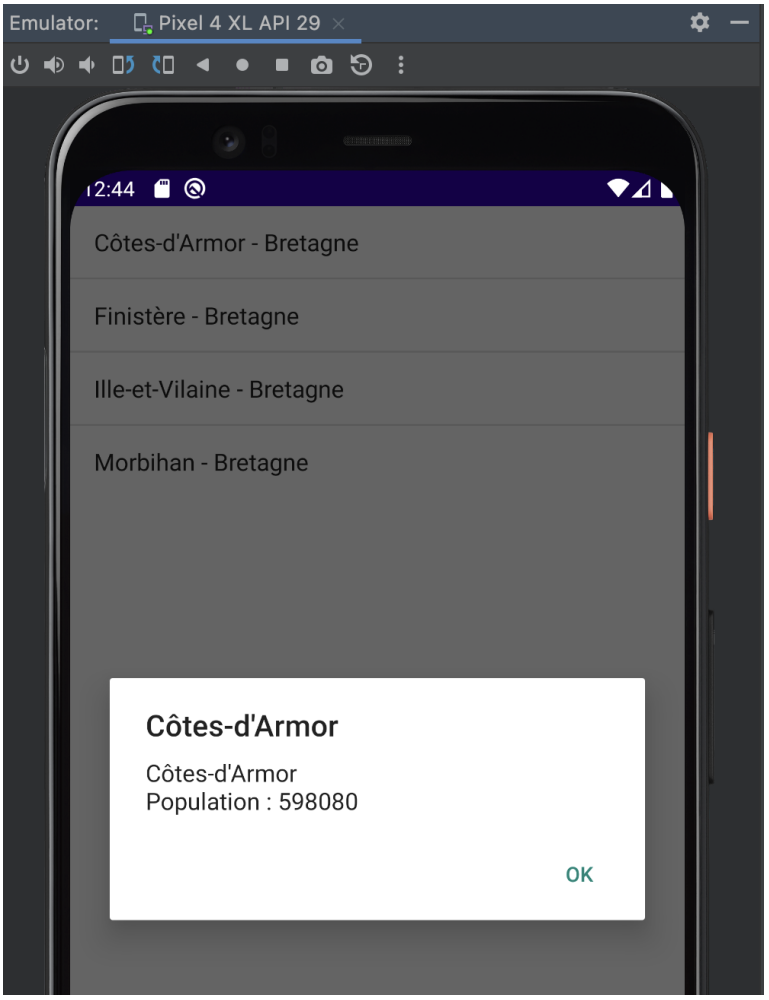


7. popup avec le Nom du département et sa population

Ici il s'agit de faire en sorte que l'on puisse cliquer sur chacun des éléments de la liste et afficher une popup avec le Nom du département et sa population qui sera stockée dans une autre arrayList ou une map.
Pour aboutir nous modifions que le **MainActivity.kt :**

```
33
34          // Gestion des clics sur les éléments de la liste
35          listView.setOnItemClickListener { parent, view, position, id ->
36              // Récupération du département sélectionné et de sa population
37              val departement = departementsBretons[position]
38              val population = populationDepartements[departement]
39              // Affichage d'une popup avec le nom et la population du département
40              val popup = AlertDialog.Builder(this)
41                  .setTitle(departement)
42                  .setMessage("$departement\nPopulation : $population")
43                  .setPositiveButton(android.R.string.ok, null)
44                  .create()
45              popup.show()
46          }
47      }
48  }
```

Le rendu de l'émulation est le suivant :



Conclusion

Au terme de ce TP, nous retenons que les Layout sont des éléments clés de l'interface utilisateur d'une application Android. Les Layout sont des éléments incontournables de la conception d'applications Android et leur maîtrise est essentielle pour les développeurs Android. Ils permettent de définir la disposition des différents composants graphiques tels que les boutons, les champs de texte, les images, etc. Il existe plusieurs types de Layout tels que LinearLayout, RelativeLayout, GridLayout, etc., chacun avec des fonctionnalités et des avantages différents.
Ce TP nous a permis de comprendre comment utiliser les Layout pour créer une interface utilisateur bien conçue, ergonomique et fonctionnelle.