ESIEE PARIS

Année 2019-2020
Semestre 2

Rapport Intermédiaire d'avancement de Projet Androïd

Avancement du 19/04/2020

Projet réalisé par :

MONTANI Maÿlis

Lien du Repos Github :

https://github.com/mayisu7798/ChuckNorrisJokesApp

Projet encadré par :

DUPONCHEL Nicolas

Partie IV - 1 : Manage screen rotation

L'objectif de cette partie est de pouvoir modifier l'orientation de l'écran sans détruire l'état actuel de notre application. En effet, pour le moment, lorsque l'on tourne l'écran et passons d'une vue verticale à une vue horizontale, l'état de l'application est reset et on recharge de nouvelles `Jokes`.

Nous allons dans un premier temps avoir besoin de passer en "attribut" de notre classe la liste des Jokes car nous allons devoir l'utiliser dans une autre fonction que `OnCreate()`. Ainsi nous avons ceci :

```
class MainActivity : AppCompatActivity ()
{
    private val disposable = CompositeDisposable()
    private val listeOfJokes: MutableList<Joke> = mutableListOf()
```

Nous allons ensuite créer une fonction `onSaveInstanceState()` qui quand elle sera appelée sauvegardera l'état courant de notre liste et nous permettra par la suite de charger cet état lors de la re-création de notre adapteur en position horizontale. Cette fonction crée un Json dans lequel notre liste de `Jokes` sera sauvegardé sous forme de `String`.

```
override fun onSaveInstanceState(outState: Bundle) {
    val json = Json(JsonConfiguration.Default)
    val listeJokesString = json.stringify(Joke.serializer().list, listeOfJokes)
    outState.putString("listeJokesString", listeJokesString)
    super.onSaveInstanceState(outState)
}
```

Enfin, dans le `OnCreate()`, nous voulons que si l'application est lancée pour la première fois (c'est à dire qu'aucun état n'est sauvegardé pour le moment, le comportement soit le même que précédemment, c'est à dire lancer l'appel de nouvelles `Jokes` puis en Scrollant vers la bas, que ça charge de nouvelles `Jokes`. Nous voulons également que s'il existe un état sauvegardé (c'est à dire que l'écran a tourné) alors celui-ci soit de nouveau affiché, puis que si on scrolle vers le bas, de nouvelles `Jokes` soient chargées. Nous écrivons alors :

```
if (savedInstanceState != null) {
    val listeJokesString = savedInstanceState.getString( key: "listeJokesString")
    if (listeJokesString != null) {
        val json = Json(JsonConfiguration.Default)
        val listeJokesSaved = json.parse(Joke.serializer().list, listeJokesString)
        listeJokesSaved.forEach{Joke -> adapteur.setJokes(Joke) }
    }
} else {
    adapteur.onBottomReached(adapteur)
}

recycler.viewTreeObserver.addOnScrollChangedListener(ViewTreeObserver.OnScrollChangedListener {
    if (!recycler.canScrollVertically( direction: 1)) {
        adapteur.onBottomReached(adapteur)
    }
}
)
```

Note : J'ai eu beaucoup de mal à faire cette question. Je ne comprenais pas comment l'état devait être sauvegardé et j'ai finalement compris que la fonction `OnSaveInstanceState()` était appelée automatiquement à l'appel de `OnDestroy` pour sauvegarder l'état courant. J'ai également eu un problème bête car j'avais oublié de retirer la création de la liste de `OnCreate` ce qui la créait de nouveau en variable locale, donc ce que je générais à la rotation de l'écran était une liste vide.
Temps : ~ 4h
Commit : Gestion de la rotation et sauvegarde de l'état

Partie IV - 2 : Custom Joke View

Dans cette partie, on nous demande d'ajouter une `CustomView` pour les `Jokes`. On doit alors créer une classe qui hérite de `ConstraintLayout`, et qui contient une `data class Model` (contenant tous les éléments de notre application, à savoir une `TextView` puis également deux `ImageButton`), une fonction `setupView()` qui est le seul à modifier ce qui s'affiche sur l'application.

```
class JokeView @JvmOverloads constructor(pContext : Context)
    : ConstraintLayout(pContext) {

    init {  inflate(pContext, R.layout.joke_layout,  root: this)   }

    data class Model(val textView: TextView, val buttonShare : ImageButton, val buttonFavori : ImageButton)

    fun setupView(model: Model)
    {
        var textView : TextView = findViewById(R.id.TextViewJoke)
        textView = model.textView

        var buttonShare : ImageButton = findViewById(R.id.ButtonPartager)
        buttonShare = model.buttonShare

        var buttonFavori : ImageButton = findViewById(R.id.ButtonFavori)
        buttonFavori = model.buttonFavori
    }
}
```

On doit également modifier le fichier `joke_layout.xml` qui doit être "une sorte de" `ConstraintLayout` et qui doit maintenant contenir non plus une seule `TextView` mais également deux `ImageButton` pour partager et mettre dans les favoris la `Joke` associée.

```
<merge xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:parentTag="androidx.constraintlayout.widget.ConstraintLayout">
```

```
<TextView
    android:id="@+id/TextViewJoke"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="0dp"
    android:background="@android:color/white"
    android:text=""
    android:textAlignment="textStart"
    app:layout_constrainedWidth="true"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintEnd_toStartOf = "@id/Barriere"
    app:layout_constraintStart_toStartOf="parent" />
```

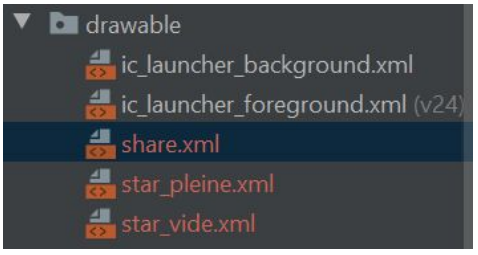
```
<ImageButton
    android:id="@+id/ButtonPartager"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/share"
    app:layout_constraintEnd_toStartOf="@id/ButtonFavori"
    app:layout_constraintStart_toEndOf="@id/TextViewJoke"
    app:layout_constraintTop_toTopOf="parent" />

<ImageButton
    android:id="@+id/ButtonFavori"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/star_vide"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@id/ButtonPartager" />
```

Afin que les différents éléments se placent correctement, il faut également ajouter un `Widget Barriere` au `ConstraintLayout` pour que la taille de chacun des éléments (en particulier le `TexteView`) soit maximisée. L'intégralité du code a changé dans ce fichier.

```
<androidx.constraintlayout.widget.Barrier
    android:id = "@+id/Barriere"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    app:barrierDirection = "left"
    app:constraint_referenced_ids = "ButtonFavori, ButtonPartager" />
```

Pour que cela fonctionne dans l'application, il faut alors créer trois nouveaux icônes dans le dossier `drawable` qui seront les icônes "`share.xml`", "`star_vide.xml`" et "`star_pleine.xml`". Ces icônes ont été générés à l'aide des indications du sujet.



Il faut également changer la classe `JokeAdapteur` qui au lieu d'utiliser une simple TextView va maintenant utiliser la `CustomView` que nous avons créée, à savoir `JokeView`. Nous avons aussi dû modifier le contenu de la fonction onBindViewHolder() afin de mettre des Listener sur les ImageBoutons et faire les actions souhaitées.

```
class JokeAdapter(private var listJokes: MutableList<Joke>, val onBottomReached: (JokeAdapter) -> Unit)
    : RecyclerView.Adapter<JokeAdapter.JokeViewHolder>()
{
    class JokeViewHolder (var jokeView : JokeView) : RecyclerView.ViewHolder(jokeView)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): JokeViewHolder {
        val jokeView = JokeView(parent.context)
        jokeView.layoutParams = RecyclerView.LayoutParams(  ViewGroup.LayoutParams.MATCH_PARENT,
                                                            ViewGroup.LayoutParams.WRAP_CONTENT)
        return JokeViewHolder(jokeView)
    }
```

```
    override fun onBindViewHolder(holder: JokeViewHolder, position: Int)  {
        var textView : TextView = holder.jokeView.findViewById(R.id.TextViewJoke)
        textView.text = listJokes[position].value

        var buttonShare : ImageButton = holder.jokeView.findViewById(R.id.ButtonPartager)
        buttonShare.setOnClickListener { Log.d( tag: "Share-IdTextView",  msg: "IdTextView = " + textView.id) }

        var nombreClic : Int = 0
        var buttonFavori : ImageButton = holder.jokeView.findViewById(R.id.ButtonFavori)
        buttonFavori.setOnClickListener { it: View!
            Log.d( tag: "Favori-IdTextView",  msg: "IdTextView = " + textView.id)
            nombreClic += 1
            if (nombreClic%2 == 1) buttonFavori.setImageResource(R.drawable.star_pleine)
            else buttonFavori.setImageResource(R.drawable.star_vide)
        }

        var model : JokeView.Model = JokeView.Model(textView = textView,
                                                    buttonShare = buttonShare,
                                                    buttonFavori = buttonFavori)
        holder.jokeView.setupView(model)

    }
```

Note : J'ai eu beaucoup de mal à aligner les `Jokes` et les deux boutons, à trouver comment ajouter les boutons au layout, car au début j'avais simplement ajouté des `Buttons` et pas des `ImageButtons`. L'`id` qui ressort dans le `LogCat` est toujours le même, quelle que soit la `Joke` que je clique. Les étoiles bougent un peu n'importe comment (le motif se répète environ une dizaine de `Jokes` plus tard), mais individuellement le comportement est celui souhaité.

Temps : ~ 6h

Commit : Ajouter une Costom Joke View non fonctionnelle