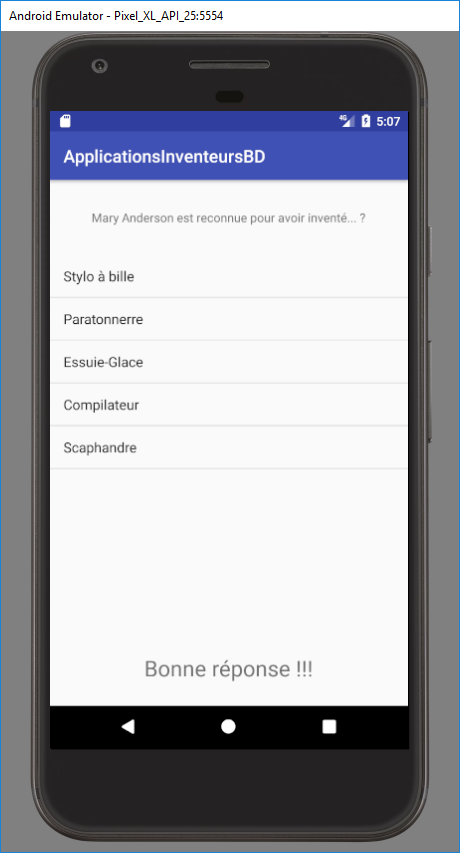
# Annexe 12B – exercice SQLite ( suite )

* Class chufle
* Class card
* Class deck
* setTag(1,2) pour un tableau a 2 dimentions

1. ( 30 min ) À l’aide du fichier de positionnement de l’activité créée par défaut dans le projet précédent, développez l’interface suivante :



1. Remplacez le ConstraintLayout ( par défaut ) par un LinearLayout ( n’oubliez pas l’orientation )
2. Assignez les layout\_height à 0dp et utilisez des poids pour partager l’espace vertical ( layout\_weight )
3. ( 30 min ) Dans le code java de l’activité…
4. Récupérez les widgets TextView et ListView
5. Connectez-vous à la BD et utilisez la méthode codée précédemment afin de remplir le ListView ( vous avez besoin d’un ArrayAdapter comme avec un Spinner)
6. Gérez les événements clic sur le ListView de manière à vérifier, à l’aide de la méthode codée dans l’annexe11, si la personne a cliqué sur la bonne réponse ( Essuie-Glace ). Ecrivez « Bonne réponse » ou « Mauvaise réponse » en fonction du résultat de la méthode faisant la recherche dans la BD.

* Écouteur à utiliser : OnItemClickListener

Faites-en sorte, dans le cas d’une mauvaise réponse, que la bonne réponse soit affichée avec un fond vert et la réponse erronée choisie par le joueur par un fond rouge.

1. Redéfinissez la méthode onStop dans la classe de l’Activité où vous fermerez la connexion à la base de données.

EXTRA – utiliser un selector pour changer la couleur suite à une action de du joueur

1. Se placer dans le navigateur, menu contextuel sur le dossier drawable et sélectionner New 🡪 drawable resource file
2. On doit y définir des états dans lequel les items du ListView seront. Rajoutez les états suivants :

*<!-- état clic-->*<**item  
 android:state\_pressed="true"  
 android:drawable="@color/green"** />  
*<!-- état sélectionné, appliqué à listSelector, sinon par défaut -->*<**item  
 android:drawable="@color/white"** />

Les valeurs d’état pouvant être utilisées dans un selector sont à :

    android:state\_pressed=["true" | "false"]  
        android:state\_focused=["true" | "false"]  
        android:state\_hovered=["true" | "false"]  
        android:state\_selected=["true" | "false"]  
        android:state\_checkable=["true" | "false"]  
        android:state\_checked=["true" | "false"]  
        android:state\_enabled=["true" | "false"]  
        android:state\_activated=["true" | "false"]  
        android:state\_window\_focused=["true" | "false"]

1. Assignez le fichier du sélecteur à votre ListView en utilisant l’attribut listSelector dans le fichier de positionnement xml  ( le coder dans le xml directement )

<**ListView  
 android:id="@+id/listView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 android:layout\_weight="4"  
 android:listSelector="@drawable/selecteur\_couleurs">**  
  
</**ListView**>

De cette façon, on verra mieux la sélection du joueur dès l’action « pressed ».

## Cycle de vie d’une Activité

On a redéfini la méthode onStop dans l’exercice précédent pour faire en sorte qu’on se déconnecte de la base de données lorsqu’on quitte l’activité…plusieurs autres méthodes peuvent être redéfinies de la même façon, chacune correspondant à un état du cycle de vie : voir présentation Prezi

Regardez le schéma Prezi sur le cycle de vie. Colligez les comportements pouvant mener à chacun de ces chemins :

1. Chemin 1, Rare une application ne couvrant pas tout l’espace arrive par-dessus
2. Clic sur le bouton home (cercle) ou bouton back
3. Tourner le telephone, changez un paramettre sur le telephone
4. Faire le bouton récents et glisser l’app vers le haut, fermer l’activité(methode finish())