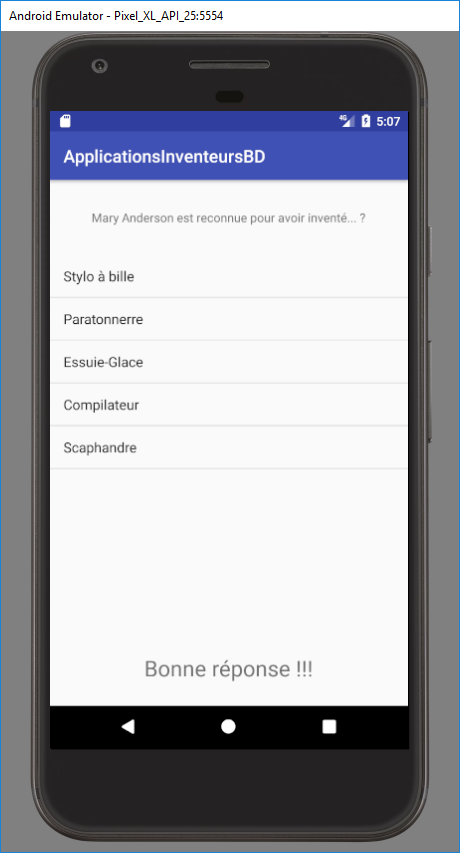
# Annexe 12B – exercice SQLite ( suite )

1. ( 20 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

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

## EXTRA : Gardez la sélection dans le ListView d’une couleur différente

( 30 min ) Présentement, un joueur distrait pourrait obtenir une bonne réponse sans se rappeler quel choix il a fait ou, dans le cas d’une mauvaise réponse, on pourrait indiquer la bonne réponse d’une autre couleur.

1. Faites-en sorte, dans le cas d’une mauvaise réponse, que la bonne réponse soit affichée avec un fond vert et la réponse originale en rouge. Sinon, la bonne réponse et la réponse donnée sont le même item à afficher en vert
2. On pourrait utiliser une méthode similaire pour afficher la réponse choisie par l’utilisateur. Dans une optique de séparer la logique du design le plus possible, suivez les opérations suivantes utilisant un **sélecteur** :
3. Ouvrez le fichier colors.xml déjà présent dans le dossier res/values. Sur le modèle des couleurs déjà définies, rajoutez celles-ci pour avoir les couleurs suivantes :

* clique : bleu // couleur que la personne va sélectionner

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/clique"** />  
*<!-- 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**>

## 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. Cliquez ici pour entrer du texte.
2. Cliquez ici pour entrer du texte.
3. Cliquez ici pour entrer du texte.
4. Cliquez ici pour entrer du texte.