Material design

* RecyclerView: mejora del ListView, para soportar mas tipos de diseños y mejoras en rendimiento ya que su función es mostrar únicamente los elementos (o view holder objects que van a contener cada ítem de la lista) que entran en la porción de pantalla visible y un par mas. A medida que el usuario scrollea, estos nuevos elementos son bindeados en los lugares que son liberados por los que no están ya visibles en pantalla.
  + Utiliza como adapter el RecyclerView.Adapter<RecyclerView.ViewHolder> que va a ir creando y bindeando a la lista los viewHolder que sean necesarios. Se sobreescriben dos métodos:
    - onCreateViewHolder: donde vamos a instanciar e inflar el layout de cada elemento o viewHolder
    - onBindViewHolder: usa la posición de un viewHolder para mostrar otros elementos en una especificada posición. Basicamente administra la posición del viewHolder object en comparación con los otros y una posición dada
* CardView es una API que permite mostrar una lista de objetos en forma de Carta de presentación que, generalmente, presentan imágenes con texto y sombras.
* Material Theming: Dark Material y Light Material son los temas base de Material Design. Es la forma de utilizar temas para aplicar a la aplicación, temas de forma general a toda la app y/o personalizar objetos con:
  + FamilyFonts: Roboto / Rubik
  + Icon Family: Filled / Outlined
  + Button Shapes: circle / rectangle / oval
  + Button shapes corners: Rounded / Cut
  + Colors
* Sombras: representan la coordenada z de los elementos, además de las ya existentes x e y. Generalmente se usan para darles efectos de elevación a los elementos.
* Animaciones: API’s de animaciones que vienen con Android sirven para:
  + Ocultar y mostrar información con efectos circulares
  + SharedActivityTransition: transición de actividades con animaciones personalizadas
  + Etc..
* AppCompatActivity: provee funcionalidades al ActionBar
* FragmentActivity: provee funcionalidades para usar fragments
* Activity: Es la activity básica de la cual derivan el resto de las activities.

Tener en cuenta, no se pueden hacer conexiones a internet desde el main activity. Tira error. Utilizar un asynctask.

Optimizacion de código

Proguard

Usado para la reducción del tamaño del apk. Se usa principalmente para que, a momento de generar el apk, el compilador lea, interprete el código de la aplicación, elimine el código sin usar y los recursos sin usar.

Como hace esto:

* + Lee nombres de métodos, clases y variables largos y los optimiza reemplazandolos por otros mas cortos, como números.
  + Genera un código equivalente mucho mas simplificado para evitar el tamaño excesivo de la aplicación y lo hace ilegible y protegido contra la ingeniería inversa. Esto le agrega una capa extra de seguridad.
  + minifyEnabled -> true para aplicar reducción de código a la aplicación
  + podemos usar @Keep para conservar una clase por completo en el. Estado tal cual esta para evitar que se incluya en la reducción de código de ProGuard

Activity lifecycle

EL ciclo de vida de un activity pasa por distintas instancias a medida que el usuario navega entre una y otra:

* **onCreate():** debe sobreescribirse si o si (siendo un callback) ya que se ejecuta una sola vez al momento de creación de un activity y es de suma importancia porque es cuando se asocian vistas al activity, se carga data necesaria para una lista antes de mostrarse, etc. Tiene un parámetro “Bundle savedInstanceState” que va a contener el estado anterior del activity al momento de querer irse de la misma y volver. Si la activity recién se ha creado, este Bundle es null
* **onStart():** el activity pasa a estar en estado listo para pasar a la interaccion con el usuario y visible para el mismo. No es un estado permanente, apenas termina de ejecutar lo necesario par aestar visible para el usuario, el activity entra en estado onResume().
* **onResume():** el activity pasa a estar en foreground.En este estado la app ya interactua con el usuario y la app permanece en este estado hasta que algo ocurra que haga que la misma pierda foco. Cuando esto pasa, el activity pasa a estado onPaused().
* **onPaused():** este callback se llama cuando el usuario esta yéndose del activity actual a otro evento. Es el lugar donde se tienen que detener todos los procesos que estén corriendo en la activity actual y no necesitan hacerlo cuando la misma pierde foco. Se detienen y se reanudan en onResume() al volver. Razones que ocurra esto:
  + **Modal dialogs sobre el activity, hacen que esta pierda el foco**
  + **Evento externo sobre el activity en momento del onResume(). Como una llamada**
  + **Multiple-windows mode en Android 7.0 en adelante. Solamente una activity puede estar con el foco a la vez. Las otras se detienen**
* **onStop():** Si otra activity cubre toda la pantalla, el activity que quedo debajo deja de ser visible para el usuario y pasa a estar en estado onStop().. En este estado, la aplicación debería liberar los recursos que han sido ocupados por el activity o minimamente analizar y ajustar los recursos no necesarios.
* **onDestroy():** se ejecuta cuando la aplicación esta por ser completamente destruida. Esto pasa cuando:
  + **El usuario llama activity.finish() u otro modo en que el activity este finalizando completamente**
  + **El sistma esta finalizando el activity debido a cambios de configuración o similar**

SQLITE

SQLite es la API proviste por Android para ser una base de datos dentro de la aplicación, a modo de SQL. Necesitamos:

* Definir un esquema de como va a ser la base de datos, junto a la/s tabla/s que va a contener.
* Definir la forma en que vamos a mantener la base de datos, es decir: incluir nuevos registros, borrar existentes, etc.
  + SQLite ofrece un SQLiteOpenHelper que provee un set de API’s útiles que nos ayudan a manejar todos estos eventos de manera muy sencilla. Al usar esta clase, el sistema realiza operaciones potencialmente grandes de creación y actualización de la base de datos cuando se la necesita, y no durante la iniciación de la app. Solo debemos realizar llamadas a getWritabeDatabase() o getReadableDatabase() y listo. Conviene realizar estas operaciones en un asynctask ya que esta tarea puede ser muy grande.

Proceso de creación y utilización de un SQLite:

* Crear un string con:
  + Nombre de base de datos
  + Columnas y entradas de la base de datos
* Crear strings para:
  + Crear registros en la base de datos
  + Borrar registros de la base de datos
* Implementar SQLiteOpenHelper:
  + Crear un database versión
  + Crear un database name
  + Implementar:
    - onCreate(new SQLiteDatabase)
    - onUpgrade() -> generalmente se borra la tabla y se crea una nueva llamando a onCreate().
    - onDowngrade()
* Para insertar registros en la base de datos:
  + SQLiteDatabase db = databaseOpenHelper.getWritableDatabase()
  + Creamos un ContentValues object.
  + Por cada ítem, hacemos un values.put(TableColumn, value)
  + .db.insert(TableName, null, values)
* Para leer registros de la base de datos:
  + SQLiteDatabase db = databaseOpenHelper.getReadablaDatabase()

Se va a generar la base de datos en apps private folder y será seguro ya que no es visible por otras apps o por el usuario.