Prueba - Superhéroes

- Para realizar esta prueba debes haber estudiado previamente todo el material disponibilizado correspondiente al módulo.
- Una vez terminada la prueba, comprime la carpeta que contiene el desarrollo de los requerimientos solicitados y sube el .zip en el LMS.
- Puntaje total: 10 puntos.
- Desarrollo prueba:
    - La prueba se debe desarrollar de manera Individual.

Habilidades a evaluar

- Crear y versionar proyecto Java.
- Manejar el ciclo de vida, actividades y fragmentos usando el patrón Single Activity para organizar los fragmentos.
- Utilizar lista y crear adaptador para desplegar el listado.
- Manejar interacciones de usuarios y recursos de Android utilizando ViewBinding o DataBinding para interactuar con las vistas.
- Implementar patrones de diseño (MVVM o MVP) para estructurar la app.
- Crear pruebas sobre el código para verificar el funcionamiento de trozos de código (métodos o clases).
- Usar buenas prácticas, estándares de la industria y convenciones asociadas a los nombres de variables, métodos, clases y paquetes.

Descripción

La empresa “Superheroes Unlimited” lleva años asociada al rubro de los superhéroes en distintos ámbitos. Con el afán de ampliar su mercado, ha decidido construir una aplicación nativa para teléfonos Android que permite explorar el mundo de los superhéroes de distintos universos.

Esta app permitirá probar la idea, por lo tanto, se debe construir un Producto Mínimo Viable (MVP por sus siglas en inglés) para que los interesados visualicen el listado de superhéroes y, al hacer clic en uno, vean su detalle.

Considerando que esta primera versión busca tener una gran cobertura en dispositivos manteniendo los costos de mantención bajos, se requiere que la API mínima sea nivel 23 y *target* la 30.

Se espera que cualquier usuario, sin previa autenticación, vea la lista de superhéroes y su detalle al seleccionar uno.

Alcance del proyecto

Se ha determinado construir un Producto Mínimo Viable (MVP) con 2 pantallas: una para el listado y otra para el detalle.

La información a desplegar proviene de una API que debe ser consumida siguiendo los estándares de la industria, ocupando patrones de arquitectura y diseño correctamente para construir una app escalable.

Requerimientos técnicos: arquitectura y dependencias

Aunque el alcance de la app es inicial, la arquitectura que se utilice debe permitir expandir y modificar el proyecto si los resultados iniciales son buenos, por lo que se debe construir un código legible, ordenado, indentado y que respete los siguientes requerimientos técnicos:

- La arquitectura de la app puede ser Modelo-Vista-Presentador (MVP) o Modelo-Vista-VistaModel (MVVM).

- Para la comunicación con la API utilizar la biblioteca Retrofit. Debes agregar dependencias de Retrofit y de converter para Json. Agregar permiso de internet al manifiesto. Creación de interfaz de operaciones, cliente de conexión y POJOs necesarios.
    - Los endpoints a utilizar son:
        - https://akabab.github.io/superhero-api/api/all.json
        - https://akabab.github.io/superhero-api/api/id/{id}.json

- Se deben respetar convenciones y estilos de codificación.

- Java es el lenguaje elegido para construir la app.

Requerimientos

1. Crear el proyecto base utilizando Java como lenguaje. Inicializar el proyecto con control de versiones git. (**1 Punto**)

2. La aplicación debe utilizar una actividad y al menos 2 fragmentos (Single Activity o Actividad única) (**2 Puntos**)

3. Para enlazar las vistas se debe utilizar ViewBinding o DataBinding. (**1 Punto**)

4. El listado principal debe utilizar un RecyclerView. Implementar la clase Adapter que hereda de la clase RecyclerView.Adapter para el manejo de la información, además del ViewHolder correspondiente. (**2 Puntos**)

5. Arquitectura Model-View-Presenter (MVP): Crear la clase Presenter (Presentador) y la interfaz de callbacks necesaria para la comunicación con la vista o utilizar LiveData para exponer la información. (**2 Puntos**)

6. Crear dos (2) archivos de pruebas unitarias con al menos una (1) prueba para dos (2) clases distintas. (**1 Punto**)

7. Todos los textos que no provengan como resultado de la API deben estar creados como recurso de texto (archivo strings.xml). Crear al menos 1 imagen como Image Vector. No tener código comentado. (**1 Punto**)

8. Utilizar Firebase para guardar superhéroes favoritos y su imagen asociada. (**Opcional**)

Recomendaciones

Listado de tareas

Antes de comenzar a programar, una buena forma de abordar el problema es dividiéndolo en tareas pequeñas y acotadas. Una posible división de tareas es construir primero lo relacionado con los datos para subir por cada capa hacia la vista.

Con la lista de tareas se puede abordar cada tarea por separado:

```
[X] Consumo de API
       [X] dependencias
       [X] permiso de internet
[ ] POJOs
       [X] Interfaz de operaciones
[X] cliente de retrofit
[ ] Repositorio
[ ] Presenter para listado
       [ ] Interfaz para callbacks de la vista
       [ ] Interfaz para callbacks del repositorio
[ ] Presenter para detalle
       [ ] Interfaz para callbacks de la vista
       [ ] Interfaz para callbacks del repositorio
[ ] Fragmento de listado
       [ ] Layout de item list
       [ ] Adapter + ViewHolder + RV
       [ ] Consumo de imágenes (dependencias)
       [ ] Instanciar presentador
[ ] Fragmento de detalle
       [ ] Instanciar presentador
[ ] Pruebas unitarias
       [ ] Pruebas unitarias al presentador del listado
[ ] Verificación de buenas prácticas
[ ] Textos en strings.xml
[ ] Dimensiones en dimens.xml
[ ] Colores en colors.xml
[ ] No código comentado
```

Para el consumo de API

Para el consumo de API, lo recomendado es utilizar Retrofit. Se debe agregar las dependencias de Retrofit y de *converter* para Json. Además, se debe agregar el permiso de internet al manifiesto. Creación de interfaz de operaciones, cliente de conexión y POJOs necesarios.

Agregar las dependencias al archivo build.gradle del módulo:

```
//Retrofit
def retro_version = "2.9.0"
implementation "com.squareup.retrofit2:retrofit:$retro_version"
implementation
"com.squareup.retrofit2:converter-gson:$retro_version"
```

Se debe definir el cliente de Retrofit especificando la URL base:
RetrofitCliente.java

```
public class RetrofitClient {
   private static Retrofit retrofit = null;
   private  static final String BASE_URL="https://akabab.github.io/superhero-api/api/";

   public static Retrofit getRetrofitInstance(){
       if (retrofit == null){
           retrofit = new retrofit2.Retrofit.Builder()
                   .baseUrl(BASE_URL)
                   .addConverterFactory(GsonConverterFactory.create())
                   .build();
       }

       return retrofit;
   }
}
```

Agregar permiso de internet al manifiesto AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Crear los POJOs para el superhéroe utilizado en el listado y para el detalle:
Superhero.java, SuperheroDetail.java y las clases correspondientes para el detalle e imágenes.

Crear la interfaz de operaciones que define los endpoints a utilizar **SuperheroAPI.java**

```
public interface SuperheroAPI {
   @GET("all.json")
   Call<List<Superhero>> getAll();

   @GET("id/{sid}.json")
   Call<SuperheroDetail> getSuperhero(@Path("sid") String id);
}
```

Imágenes de referencia

A continuación, se presentan imágenes de referencia del listado y el detalle.

Estas imágenes corresponden a un listado con RecyclerView y CardView para desplegar la información. En el detalle se incluye una imagen (vector) que puede ser utilizada para desplegar la información del superhéroe, por ejemplo, su fortaleza, tamaño o peso.

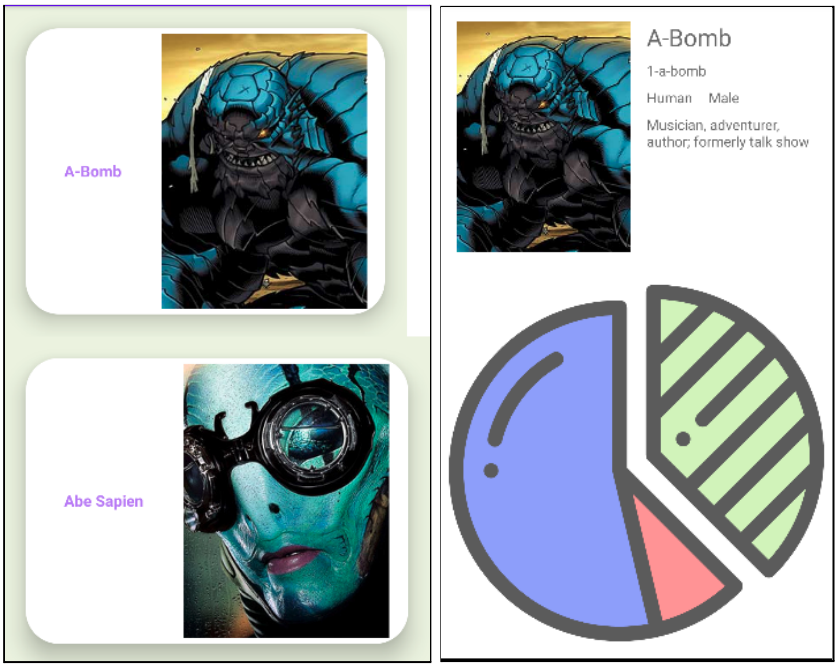

Imagen 1: Listado Detalle RecyclerView y CardView
Fuente: Desafío Latam

En el siguiente caso, se utiliza un ConstraintLayout para construir cada elemento de la lista y se alinean los textos desde la derecha. El detalle utiliza una imagen vectorial donde se pueden presentar los detalles del superhéroe.

Tip: Es posible que al utilizar LinearLayoutManager en el RecyclerView la vista se contraiga en el ancho. Una solución es utilizar un GridLayoutManager con 1 columna.

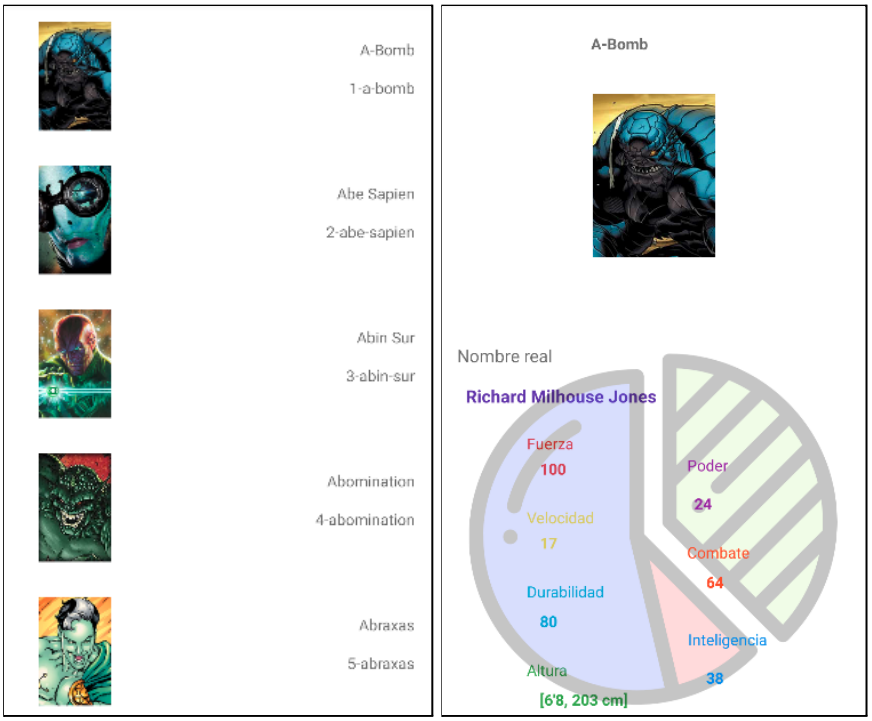

Imagen 2: LinearLayoutManager
Fuente: Desafío Latam