App Android Ampliacion

MIS JUEGOS

Nueva actualización de la Aplicación MyApps en la cual se le agrega la funcionalidad de una Biblioteca de Videojuegos para el Usuario, todo almacenado en base de datos, por lo que no necesita conexión para hacer uso de esta nueva función.

Notas del Parche:

- Dos Fragments nuevos, uno de ellos con una lista de los juegos que agregue el usuario y un filtro para ordenar(Disponible desde el menú vertical) y otro para editar, crear o visualizar en detalle cada uno de esos juegos(No disponible a través de menu).

- Se ha implantado una Base de Datos SQLite para el almacenamiento de estos juegos .

- Nueva función multimedia para poder elegir una imagen desde la galería o cámara para asignar a cada uno de los juegos(Se pedirán permisos al inicio de la aplicación).

- Mejora en el Fragment de RSS, implementando una mejor apariencia visual cambiando el Item de la lista por un Card View.

- Mejora visual en el Fragment de Contacto y Acerca de.

Descripción Técnica (Para Desarrolladores):

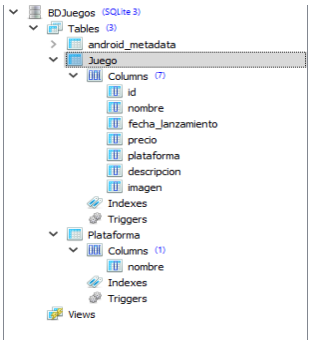
1. Base de Datos

- ### Codigo:

Para empezar a hablar sobre la nueva funcional de Biblioteca de Juegos debemos explicar antes la importación de la Base de Datos en SQLite, empezando por la clase asignada para ello **DBC.java**

```
public DBC(@Nullable Context context, @Nullable String name, @Nullable SQLiteDatabase.CursorFactory factory, int version) {
    super(context, name, factory, version);
}

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL(sqlCrear);
    db.execSQL(sqlCategorias);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

}

public void insert(Juego juego){


    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("nombre", juego.getNombre());
    values.put("fecha_lanzamiento", juego.getFecha_lanzamiento());
    juego.setPrecio((float)(Math.round(juego.getPrecio()*100.0)/100.0));
    values.put("precio", juego.getPrecio());
    values.put("plataforma", juego.getPlataforma());
    values.put("descripcion", juego.getDescripcion());
    values.put("imagen", juego.getImagen());

    db.insert( table: "Juego",  nullColumnHack: null, values);

}

public void update(Juego j){
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("nombre", j.getNombre());
    values.put("fecha_lanzamiento", j.getFecha_lanzamiento());
    j.setPrecio((float)(Math.round(j.getPrecio()*100.0)/100.0));
    values.put("precio", j.getPrecio());
    values.put("plataforma", j.getPlataforma());
    values.put("descripcion", j.getDescripcion());
    values.put("imagen", j.getImagen());
    db.update( table: "Juego", values,  whereClause: "id=" + j.getId(),  whereArgs: null);

}
```

Al iniciar la aplicación e instanciar por primera vez en el **MainActivity.java** dicha clase se creará, a partir de unos scripts, la Base de Datos en el directorio del SmartPhone: data/app/database/ con el nombre de “BDJuegos”.



La estructura de esta misma es la siguiente:

A parte de la creación de la BBDD, se ejecutará una serie de Inserts en la tabla Plataforma para que el usuario posea plataformas por defecto, las cuales podrá asignar a sus juegos. Para terminar de explicar la clase DBC, declaramos dos métodos que se referenciaran a través del Objeto instanciado que, simplemente, insertaran o actualizaran los datos en la tabla Juego

Fragment Biblioteca de Juegos(JuegoFragment):

Como incluimos otro Recycler View, primero vamos a explicar el objeto que se mostrará en dicha lista

Objeto Juego:

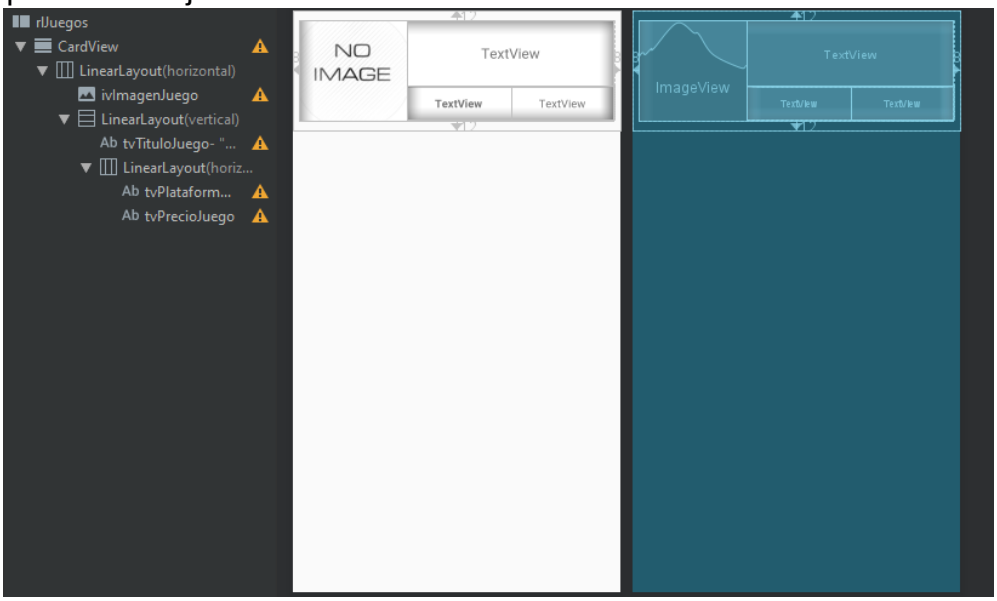
- ### Codigo:

  El nuevo objeto que instanciamos para cada ítem de Recycler llamado Juego incluye las siguientes variables(todas Strings excepto el Id y el precio): **Id**(Int), **nombre**, **fecha_lanzamiento**, **plataforma**, **precio**(Float), **imagen** y **descripción**.

Adaptador:

- ### Layout:

  El layout del ítem se compone de Relative Layout(al que le aplicamos un padding) incluyendo un Card View con 3 TextView para el titulo, plataforma y precio. Junto con una Imagen posicionada a la Izquierda para una mejor visualización:



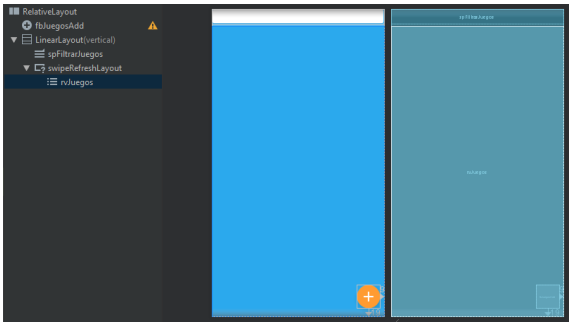
- ## Codigo:

Partiendo de la suposicion de que todo usuario que lea esta parte del documento es desarrollador y tiene unos conocimientos medios de como crear un **RecyclerView**, simplemente nos limitaremos a explicar las acciones realizadas en el **OnBindViewHolder** donde asociamos cada variable del Objeto a los ítems del Layout. Incluimos un **Listener** para el RelativeLayout el cual, accionado mediante un click, inicia otro Fragment con todos los detalles del juego:

```
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    final Juego juego = juegoslist.get(position);

    holder.titulo.setText(juego.getNombre());
    holder.precio.setText(String.format("Precio: %s€", juego.getPrecio()));
    holder.plataforma.setText(juego.getPlataforma());
    holder.imagen.setImageBitmap(MyB64.base64ToBitmap(juego.getImagen()));
    holder.relativeLayout.setOnClickListener((view) -> {

            //Cuando hacemos click en cada item, nos envia al Fragment de mas Detalles de Noticia
            Fragment JuegoDetalle = new JuegosDetalleFragment(juego, editable: false, nuevo: false);

            FragmentTransaction fragmentTransaction = mFragmentManager.beginTransaction();
            fragmentTransaction.setCustomAnimations(R.anim.fragment_effect,R.anim.fragment_effect_exit,
                    R.anim.fragment_effect,R.anim.fragment_effect_exit);
            fragmentTransaction.replace(R.id.nav_host_fragment,JuegoDetalle);
            fragmentTransaction.addToBackStack(null);

            fragmentTransaction.commit();
    });
}
```

Fragment:

- ## Layout:

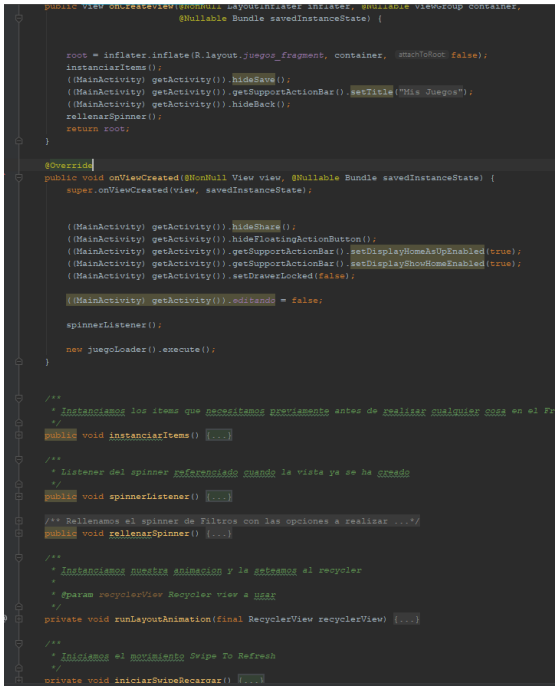


Un simple diseño basado en un Relative Layout en el que incluimos, a parte del Recycler, un spinner con las opciones de filtrado para ordenar los ítems y un Floating Action Button para agregar otro juego el cual nos enviara al Fragment de JuegosDetalle con todos los campos editables y vacios para poder crear nuestro propio ítem.

- ## Codigo:

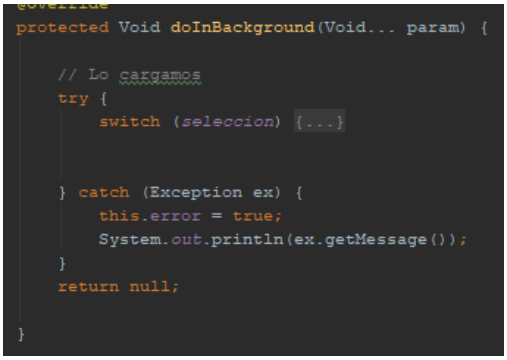
En la explicación del código, obviaremos los métodos para los **Swipers** ya que se han explicado en la versión anterior de la App. Simplemente repasaremos la acción de borrar o editar al deslizar.

Empezando por el propio fragment, en el **onCreate** y **onViewCreated** llamamos a los mismos métodos que en el Recycler del RSS para ocultar algunas opciones de la **Toolbar**, asi como la instancia de todos los ítems necesarios respecto al layout, el listener del FloatButton y del Spinner. Tambien rellenamos el Spinner con las opciones de filtrado posibles.



Por ultimo, al final ejecutamos nuestra tarea que recoge los datos de la BBDD SQLite.

Explicando la **AsyncTask**, preparando la ejecución, mostramos el swipeRefresh para indicar que esta cargando y limpiamos nuestro ArrayList con los juegos. En la misma ejecución, dependiendo del modo de filtrado seleccionado en el Spinner, ejecutamos nuestra función de recoger todos los datos de la BBDD en base al filtro.



Para ello sobrecargamos el mismo método de seleccionar datos con los parámetros necesarios:

```
private void seleccionarData() {
    //Abrimos la base de datos 'BDEjemplo' en modo lectura
    DBC bdEjemplo = new DBC(this.getContext(),  name: "BDJuegos",  factory: null,  version: 1);
    SQLiteDatabase bd = bdEjemplo.getReadableDatabase();

    ArrayList<Juego> juegos = new ArrayList<>();
    //Si hemos abierto correctamente la base de datos
    if (bd != null) {
        //Seleccionamos todos
        Cursor c = bd.rawQuery( sql: " SELECT id,nombre, fecha_lanzamiento, plataforma, descripcion, precio, imagen FROM Juego",  selectionArgs: null);
        //Nos aseguramos de que existe al menos un registro
        if (c.moveToFirst()) {
            //Recorremos el cursor hasta que no haya más registros
            do {
                Juego j = new Juego(c.getString( columnIndex: 1), c.getString( columnIndex: 2), c.getString( columnIndex: 3), c.getString( columnIndex: 4)
                        , c.getFloat( columnIndex: 5), c.getString( columnIndex: 6));
                j.setId(c.getInt( columnIndex: 0));
                juegos.add(j);

                Log.d( tag: "IMAGENES", j.getImagen());
            } while (c.moveToNext());
        }
        //Cerramos la base de datos
        c.close();

        juegoslist = juegos;
    }
    bd.close();
}
```

Metodo sobrecargado:

```
private void seleccionarData(String filtro, boolean tipo) {
    //Abrimos la base de datos 'BDEjemplo' en modo lectura
    DBC bdEjemplo = new DBC(this.getContext(),  name: "BDJuegos",  factory: null,  version: 1);
    SQLiteDatabase bd = bdEjemplo.getReadableDatabase();

    ArrayList<Juego> juegos = new ArrayList<>();
    //Si hemos abierto correctamente la base de datos
    if (bd != null) {
        juegoslist.clear();
        String modo = null;
        if (tipo) {
            modo = "DESC";

        } else {

            modo = "ASC";
        }
        //Seleccionamos todos
        Cursor c = bd.rawQuery( sql: " SELECT id,nombre, fecha_lanzamiento, plataforma, descripcion, precio, imagen FROM Juego ORDER BY " + filtro + " " + modo,  selectionArgs: null);
        //Nos aseguramos de que existe al menos un registro
        if (c.moveToFirst()) {
            //Recorremos el cursor hasta que no haya más registros
            do {
                Juego j = new Juego(c.getString( columnIndex: 1), c.getString( columnIndex: 2), c.getString( columnIndex: 3), c.getString( columnIndex: 4)
                        , c.getFloat( columnIndex: 5), c.getString( columnIndex: 6));
                j.setId(c.getInt( columnIndex: 0));
                juegos.add(j);

                Log.d( tag: "IMAGENES", j.getImagen());
            } while (c.moveToNext());
        }
        //Cerramos la base de datos
        c.close();

        juegoslist = juegos;
    }
    bd.close();
}
```

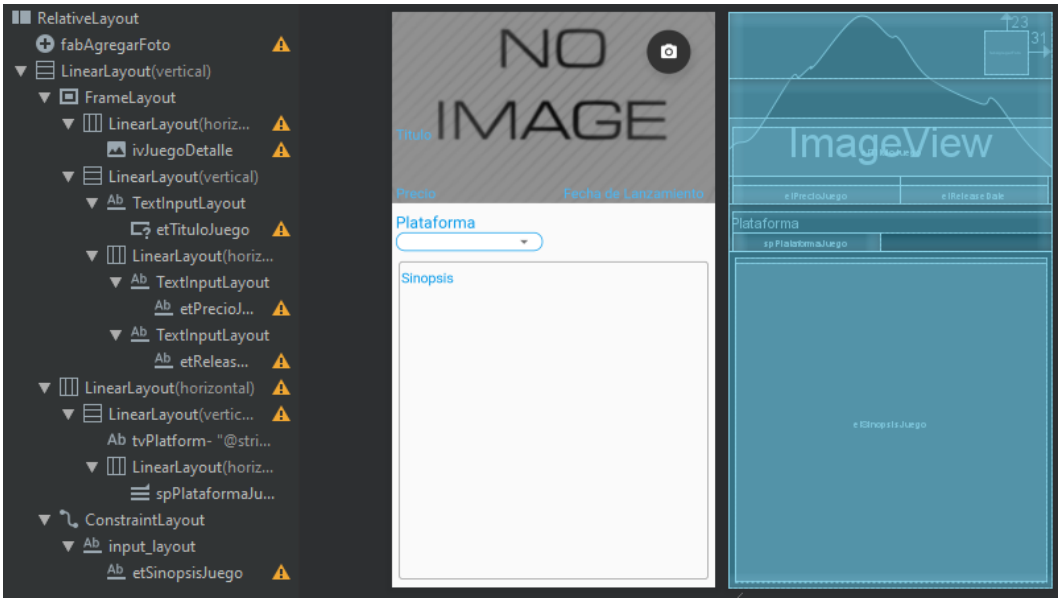
Por ultimo, después de la ejecución en segundo plano, asignamos los datos a nuestro adaptador y este los muestra en el Recycler.

Para las acciones de borrar Elemento, al deslizar un ítem hacia la izquierda, se le muestra al usuario un Dialogo de aviso de si esta seguro de dicha acción, si se confirma, simplemente se borra el ítem de la base de datos y se vuelve a ejecutar la tarea.

Fragment Detalle de Juegos(JuegosDetalle):

- ## Layout:

  Con una composición mas tediosa, tenemos varios EditTexts para los datos del Juego, un Image View para, valga la redundancia, la Imagen del juego y un Spinner para su plataforma.



- ## Codigo:

  Nada mas acceder al Fragment instanciado, dependiendo de la acción que hayamos realizado, se habilitaran los campos para editar y se mostrara información si estamos editando un elemento. Si estamos creando un elemento, los campos serán editables pero estarán vacios.

  Empezando por los métodos que usaremos, para asignar la imagen simplemente mostramos un dialogo con las opciones de escoger de la galería o usar la cámara. Para ambas, una vez realizada alguna de las acciones, gracias al método **onActivityResult**, recogemos la imagen y la acción realizada para tratar dichos datos y asignarlos al Item.
  Tambien se rellenara el Spinner con los datos extraidos de las plataformas de la base de datos

Para tratar la imagen usaremos la conversión a base64 de un bitmap y viceversa, además de hacer una redimension a 480p y compresión a JPG del 70% para que no ocupe demasiado.

Para Base64 usaremos una clase con métodos estáticos:

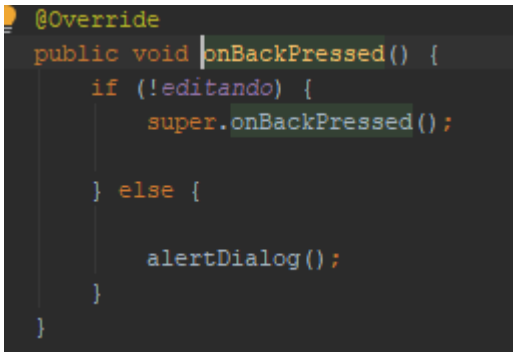
```java
public class MyB64 {

    /**
     * Metodo conversor de Base64 a Bitmap
     * @param b64 Archivo en Base64
     * @return Bitmap resultante
     */
    public static Bitmap base64ToBitmap(String b64) {
        byte[] imageAsBytes = Base64.decode(b64.getBytes(), Base64.DEFAULT);
        return BitmapFactory.decodeByteArray(imageAsBytes, offset: 0, imageAsBytes.length);

    }

    /**
     * Metodo conversor de Bitmap a Base64
     * @param bitmap Bitmap a convertir
     * @return Archivo en Base64
     */
    public static String bitmapToBase64(Bitmap bitmap) {
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        bitmap.compress(Bitmap.CompressFormat.PNG, quality: 100, byteArrayOutputStream);
        byte[] byteArray = byteArrayOutputStream .toByteArray();
        return Base64.encodeToString(byteArray, Base64.DEFAULT);
    }

}
```

En el menú se habilitaran las opciones de volver a la biblioteca de Juegos o Guardar el mismo, dicho esto, cada vez que realizamos un cambio, todos los campos se guardan en un objeto gracias a un listener de los campos. Cuando hacemos click en guardar, desde el propio MainActivity que mantiene el listener del menú, se realiza la inserción o actualizacion del ítem en la base datos.

```java
} else if (id == R.id.btSaveIt) {
    Juego j = JuegosDetalleFragment.juegoguardado;
    if (comprobarDatosJuego(j)) {
        if (j.getImagen().trim().isEmpty()) {
            Toast.makeText(getApplicationContext(), text: "Debes introducir una imagen", Toast.LENGTH_SHORT).show();

        } else {
            DBC bdJuegos = new DBC(getApplicationContext(), name: "BDJuegos", factory: null, version: 1);

            SQLiteDatabase db = bdJuegos.getWritableDatabase();

            if (JuegosDetalleFragment.nuevo) {

                bdJuegos.insert(j);


            } else {
                bdJuegos.update(j);
            }
            db.close();
            hideSoftKeyboard( activity: this);


            Toast.makeText(getApplicationContext(), text: "Cambios Guardados", Toast.LENGTH_SHORT).show();
            super.onBackPressed();
        }
    } else {
        Toast.makeText(getApplicationContext(), text: "Solo puede dejarse en blanco la sinopsis", Toast.LENGTH_SHORT).show();
    }
```

Para ello llama a los métodos que hemos explicado al principio para insertar o actualizar a través de nuestro Item DBC instanciado previamente que contiene dichos meotodos.

En cambio, en el momento que se pulsa o se acciona el método **onBackPressed**, le mostramos al usuario, gracias a que lo tenemos SobreEscrito en nuestro Main, un dialogo para advertirle de si quiere descartar los cambios.



Hay que indicar que en el caso de guardar, no se podrá realizar la acción si alguno de los campos, exceptuando la sinopsis/descripción, están vacios, asi como la propia imagen del ítem.

Aquí concluyen las explicaciones de las nuevas funcionalidades de la Aplicación y las notas del parche, espero que haya sido de utilidad!

Autor y Desarrollador Emilio Rubiales Gutierrez