Actividad 2

Universidad Internacional de Valencia.

Maestría Oficial en Desarrollo de Aplicaciones y Servicios Web.

Programación en dispositivos móviles (wearables)

William Forero

Edgar Gamarra

2022.

Objetivos:

Para esta actividad se pretende utilizar varias de las herramientas vistas en clase que nos brinda Android para el desarrollo de aplicaciones móviles, se pretende extraer cierta información de una url en archivo json y mostrarla en pantalla en forma de lista, poder eliminar elementos de la lista y visualizar imágenes de los elementos.

Para esta actividad se instaló la última versión de Android android chipmunk, y se actualizó el Gradle a la última versión.

Ejecución de la actividad:

Librarías a agregar al build.gradle

1. Inicialmente se agregan al proyecto las librerías necesarias para la correcta ejecución de la actividad.

Recyclerview se va a utilizar para la lista de elementos, es muy recomendada cuando se tiene mucha información que representar.

```
implementation "androidx.recyclerview:recyclerview:1.2.1"
```

2. Debido a que se debe extraer la información de la url, se utiliza la librería retrofit, junto con la librería gson para usar un formato más sencillo de manejar.

```
//retrofit
implementation "com.squareup.retrofit2:retrofit:2.9.0"
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

def lifecycle_version : String = "2.1.0"
```

3. Librería Glide para representación de las imágenes que están en formato json dentro de la url dada.

```
implementation 'com.github.bumptech.glide:glide:4.13.0'
```

4. Librerías para manejar card models y para compilación de componentes optimizados en ciclos de vida.

```
// ViewModel and LiveData
implementation "androidx.lifecycle:lifecycle-extensions:$lifecycle_version"

//cardview
//implementation 'com.android.support:cardview-v7:29.0.0'
implementation "androidx.cardview:cardview:1.0.0"
```

Manifest

1. Debido a que se va a obtener información de una url publica en internet se debe configurar el manifest para autorizar el uso de internet en la aplicación.

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

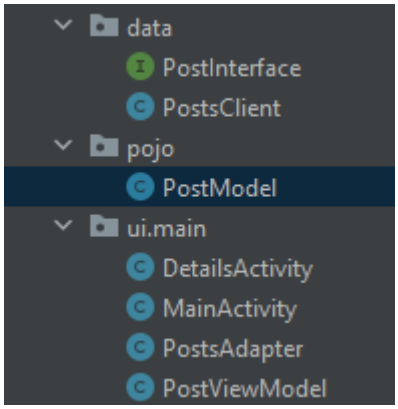
Creación de clases. Java, manejando el paradigma MVVM, con lo que buscamos separar la lógica de la aplicación de la interfaz del usuario.

Se manejarán tres carpetas para este fin:

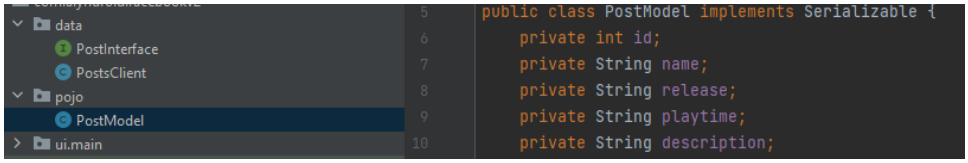
-data

-pojo

-ui



1. Empezamos creando la clase PostModel dentro de la carpeta pojo, para esta creación se tiene en cuenta los campos que nos brinda la url json. Con los métodos getters y setters se podrá acceder a cada atributo de la clase.



```java
public String getPlot() { return plot; }

public void setPlot(String plot) { this.plot = plot; }

public String getPoster() { return poster; }

public void setPoster(String poster) {

    this.poster = poster;
}

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getRelease() { return release; }

public void setRelease(String release) { this.release = release; }

public String getPlaytime() { return playtime; }

public void setPlaytime(String playtime) { this.playtime = playtime; }

public String getDescription() { return description; }

public void setDescription(String description) { this.description = description; }
```

Uso de Retrofit

2. Se realiza una petición GET para obtener la lista de películas, importando las librerías necesarias.

```java
package com.example.peliculas.data;



import com.example.peliculas.pojo.PostModel;

import java.util.List;

import retrofit2.Call;
import retrofit2.http.GET;

public interface PostInterface {
    @GET("DisneyPosters2.json")
    public Call<List<PostModel>> getPosts();
}
```

3. Creamos una clase PostClient para el acceso a la información de las películas.

```java
public class PostsClient {
    //private static final String BASE_URL = "http://jsonplaceholder.typicode.com/";



    private static final String BASE_URL = "https://gist.githubusercontent.com/skydoves/176c209dbce4a53c0ff9589e0
    private PostInterface postInterface;
    private static PostsClient INSTANCE;

    public PostsClient() {
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        postInterface = retrofit.create(PostInterface.class);
    }

    public static PostsClient getINSTANCE() {
        if (null == INSTANCE){
            INSTANCE = new PostsClient();
        }
        return INSTANCE;
    }

    public Call<List<PostModel>> getPosts() { return postInterface.getPosts(); }
}
```

4. Dentro de la carpeta ui creamos la siguiente clase la cual hereda de viewmodel la cual nos va a permitir almacenar y administrar los datos recibidos:

```java
import com.example.peliculas.data.PostsClient;
import com.example.peliculas.pojo.PostModel;

import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;



public class PostViewModel extends ViewModel {
    MutableLiveData<List<PostModel>> postsMutableLiveData = new MutableLiveData<>();
    MutableLiveData<String> posts = new MutableLiveData<>();

    public void getPosts() {
        PostsClient.getINSTANCE().getPosts().enqueue(new Callback<List<PostModel>>()
            @Override
            public void onResponse(Call<List<PostModel>> call, Response<List<PostMod
                postsMutableLiveData.setValue(response.body());
            }

            @Override
            public void onFailure(Call<List<PostModel>> call, Throwable t) {
                posts.setValue("errrr");

            }
        });
    }
}
```

Uso de REcyclerView

5. Llamamos la clase PostsModel dentro del activity y empezamos a utilizar la librería recyclerView.

```
import ...


public class MainActivity extends AppCompatActivity {

    PostViewModel postViewModel;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        postViewModel = ViewModelProviders.of( activity: this).get(PostViewModel.class);

        postViewModel.getPosts();
        RecyclerView recyclerView = findViewById(R.id.recycler);
        final PostsAdapter adapter = new PostsAdapter();
        recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
        recyclerView.setAdapter(adapter);



        postViewModel.postsMutableLiveData.observe( owner: this, new Observer<List<PostModel>>() {
            @Override
            public void onChanged(List<PostModel> postModels) { adapter.setList(postModels); }
        });



    }
}
```

6. Creamos la clase PostsAdapter que hereda de recyclerview.

```
public class PostsAdapter extends RecyclerView.Adapter<PostsAdapter.PostViewHolder>  {
    private List<PostModel> moviesList = new ArrayList<>();




    @NonNull
    @Override
    public PostViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        Context context = parent.getContext();
        return new PostViewHolder(LayoutInflater.from(parent.getContext()).inflate(R.layout.post_item, parent,
    }

    @Override
    public void onBindViewHolder(@NonNull PostViewHolder holder, int position) {


        holder.titleTV.setText(moviesList.get(position).getName());
        holder.userTV.setText(moviesList.get(position).getId()+"");
        holder.releaseTV.setText(moviesList.get(position).getRelease());
        holder.descriptionTV.setText(moviesList.get(position).getDescription());
        holder.playTimeTV.setText(moviesList.get(position).getPlaytime());
        holder.plotTV.setText(moviesList.get(position).getPlot());


        ImageView imageView = holder.bodyTv;
```

7. Empezamos a crear el layout para la visualización del contenido de las películas.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView xmlns:android="http://schema
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp">

    <androidx.constraintlayout.widget.ConstraintLayout

        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            android:id="@+id/titleTV"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="5sp"
            android:layout_marginStart="16dp"
            android:layout_marginTop="16dp"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/userIDTV"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="5sp"
            android:layout_marginTop="16dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent" />


        <ImageView
            android:id="@+id/bodyTV"
            android:layout_width="366dp"
            android:layout_height="269dp"
```

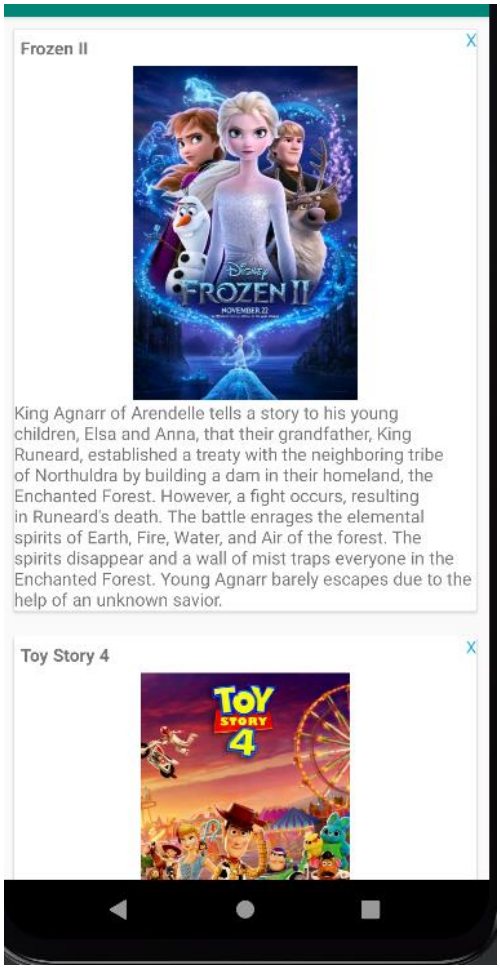
8. Con los id de los elementos creados del layout, desde la clase postAdapter empezamos a acceder a esos elementos:

```
public class PostViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
    TextView titleTV, userTV, releaseTV, descriptionTV, playTimeTV, plotTV, Elemento;
    ImageView bodyTv;
    Context context;



    public PostViewHolder(@NonNull View itemView) {
        super(itemView);
        context=itemView.getContext();
        titleTV = itemView.findViewById(R.id.titleTV);
        Elemento=itemView.findViewById(R.id.eliminarElemento);
        userTV = itemView.findViewById(R.id.userIDTV);
        userTV.setVisibility(View.GONE);
        bodyTv = itemView.findViewById(R.id.bodyTV);
        releaseTV = itemView.findViewById(R.id.release);
        releaseTV.setVisibility(View.GONE);
        plotTV = itemView.findViewById(R.id.plot);

        playTimeTV=itemView.findViewById(R.id.playTime);
        playTimeTV.setVisibility(View.GONE);
        descriptionTV = itemView.findViewById(R.id.Description);
        descriptionTV.setVisibility(View.GONE);


        Elemento.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                int pos=-1;
                for(int i=0; i<moviesList.size();i++){
```

9. Y con el método onBindViewHolder empezamos a construir la información visual de los ítems de las películas:

```
@Override
public void onBindViewHolder(@NonNull PostViewHolder holder, int position) {


    holder.titleTV.setText(moviesList.get(position).getName());
    holder.userTV.setText(moviesList.get(position).getId()+"");
    holder.releaseTV.setText(moviesList.get(position).getRelease());
    holder.descriptionTV.setText(moviesList.get(position).getDescription());
    holder.playTimeTV.setText(moviesList.get(position).getPlaytime());
    holder.plotTV.setText(moviesList.get(position).getPlot());


    ImageView imageView = holder.bodyTv;




    if (holder.itemView.getContext() != null ) {
        Glide.with(holder.itemView.getContext())
                .load(moviesList.get(position).getPoster())
                .into(imageView);
    } else {
        Log.i(TAG,  msg: "Picture loading failed,context is null");
    }
     holder.setOnclickListeners();
    // holder.bodyTv.setOnClickListener(this);
```

10. Para la representación de imágenes con Android se pueden utilizar diferentes librerías, para este caso se usó Glide teniendo en cuenta que según la página ProAndroidDev en una tabla comparativa la muestra con mayor velocidad con respecto a sus rivales, además de su fácil implementación.



Interacción con otro activity

1. **Ya contamos con el listado de películas, para la funcionalidad que se requiere al dar clic en la película visualizar más información en otra página, se debe utilizar otro activity:**

```java
public class DetailsActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_details);
        String titleTV = "";
        String releaseTV = "";
        String descriptionTV = "";
        String playTimeTV = "";

        Bundle extras = getIntent().getExtras();

        if (extras != null) {
            titleTV = extras.getString( key: "titleTV");
            releaseTV = extras.getString( key: "releaseTV");
            descriptionTV = extras.getString( key: "descriptionTV");
            playTimeTV = extras.getString( key: "playTimeTV");


        }
        TextView tvtitleTV = (TextView) findViewById(R.id.tvtitleTV);
        tvtitleTV.setText(titleTV);

        TextView tvreleaseTV = (TextView) findViewById(R.id.tvreleaseTV);
        tvreleaseTV.setText(releaseTV);

        TextView tvdescriptionTV = (TextView) findViewById(R.id.tvdescriptionTV);
        tvdescriptionTV.setText(descriptionTV);

        TextView tvplayTimeTV = (TextView) findViewById(R.id.tvplayTimeTV);
        tvplayTimeTV.setText(playTimeTV);


    }
}
```

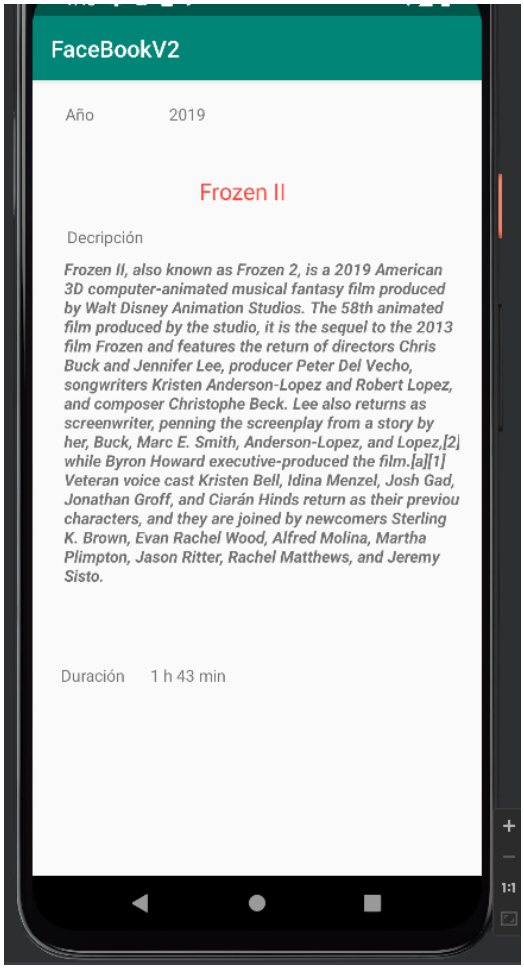
2. **Y en el postAdapter creamos el método onclick que permitirá que al dar clic sobre la imagen se comunique con el activity details y muestre información adicional de la película.**

```
public void setOnclickListeners() {
    bodyTv.setOnClickListener(this);

}

/*public void eliminar(View v){
    int pos=-1;
    for(int i=0; i<moviesList.size();i++){

        moviesList.remove(pos);


    }

}*/



@Override
public void onClick(View view) {
    Intent intent=new Intent(context, DetailsActivity.class);
    intent.putExtra( name: "titleTV", titleTV.getText());
    intent.putExtra( name: "releaseTV", releaseTV.getText());
    intent.putExtra( name: "descriptionTV", descriptionTV.getText());
    intent.putExtra( name: "playTimeTV", playTimeTV.getText());

    context.startActivity(intent);
    //Log.i(TAG, "Evento");

}
```

3. **Creamos el layout activity_details**

```
    android:layout_height="wrap_content"        ⚠ 5  ✓ 14

    android:textColor="#FA4B3E"
    android:textSize="20sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.454"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tvreleaseTV"
    app:layout_constraintVertical_bias="0.072" />

<TextView
    android:id="@+id/tvreleaseTV"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.29"
    app:layout_constraintStart_toStartOf="@+id/textView2"
    app:layout_constraintTop_toTopOf="parent" />


<TextView
    android:id="@+id/tvdescriptionTV"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="116dp"

    android:textSize="14sp"
    android:textStyle="bold|italic"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.263"
    app:layout_constraintStart_toStartOf="@+id/textView3"
    app:layout_constraintTop_toBottomOf="@+id/tvreleaseTV" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

4. **Visualización:**



5. Eliminar elementos

Al lado de cada película creamos un ícono el cual tiene el evento de eliminar películas.

Dentro de PostViewHolder buscamos el elemento:

```
public PostViewHolder(@NonNull View itemView) {
    super(itemView);
    context=itemView.getContext();
    titleTV = itemView.findViewById(R.id.titleTV);
    Elemento=itemView.findViewById(R.id.eliminarElemento);
```

Y creamos el evento para eliminar películas:

```
Elemento.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        int pos=-1;
        for(int i=0; i<moviesList.size();i++){

           if(moviesList.get(i).getName().equals(titleTV.getText().toString()))
               pos=i;

        }
        if(pos!=1){

            moviesList.remove(pos);
            titleTV.setText("");
            notifyDataSetChanged();
            Toast.makeText(v.getContext(),  text: "Se ha eliminado el elemento", Toast.LENGTH_LONG).show()

        }


    }
});
```

Creamos un condicional que nos evalúa la posición del ListArray de películas y remueve la película cliqueada, con notifyDataChanged le informamos a la aplicación que debe actualizar el nuevo listado de películas y mostramos un mensaje en la aplicación indicando que se eliminó una película.

Se intentó implementar el protoDatastore con Java, pero la información encontrada hacía en su mayoría referencia a kotlin.