


Open · Opened 3 days ago by  **Rubén Montero**

¿Quién aparece en la pantalla?

Resumen

- Añadiremos `Character.java` y `CharactersList.java`, *modelos* para una nueva petición
- Añadiremos `CharacterViewHolder.java` y `character_recycler_cell.xml`, para un nuevo tipo de celda
- Añadiremos un `CharactersAdapter.java`, nuevo *adaptador* para un nuevo `RecyclerView`
- Aparecerá sobre el vídeo cuando el usuario haga *click*, para mostrar los personajes que hay en la pantalla en ese instante

Descripción

En [Amazon Prime Video](#) y otras plataformas de *streaming*, puedes obtener información de qué personajes aparecen en la pantalla para un momento dado de un vídeo.

Nosotros no vamos a ser menos.

Por suerte, nuestro API REST cuenta con una petición HTTP exclusivamente dedicada a eso. Sólo hay que enviar el `id` del *clip* de vídeo, y el *tiempo de reproducción* (en milisegundos).

Por ejemplo:

- <http://raspi:8000/clips/5/appearances?milliseconds=16000>

(En el *clip* `5`, a los `16` segundos, aparecen los siguientes personajes):

```
[
  {
    "name": "Lisa",
    "surname": "Simpson",
    "description": "Benevolente y educada hija de la familia Simpson",
    "imageUrl": "https://raw.githubusercontent.com/rubenmv0/fp/main/simpsons/lisa.png"
  },
  {
    "name": "Bart",
    "surname": "Simpson",
    "description": "Primogénito de la familia Simpson, siempre buscando problemas",
    "imageUrl": "https://raw.githubusercontent.com/rubenmv0/fp/main/simpsons/bart.png"
  }
]
```

La tarea

Modelos: `Character.java` y `CharactersList.java`

Añade una nueva clase Java, `Character.java`. **Escribe** estos atributos:

```
private String name;
private String lastName;
private String description;
private String urlImage;
```

También, **implementa** los *getters* con su nomenclatura por defecto. Luego, **implementa** un método constructor que reciba un `JSONObject` por parámetro e inicialice los atributos de acuerdo a la respuesta de expuesta arriba (claves `"name"`, `"surname"`, `"description"` y `"imageUrl"`), de forma parecida a como has hecho en `Clip.java`.

Añade otra nueva clase Java, `CharactersList.java`. **Escribe** este atributo:

```
private List<Character> characters;
```

También, **implementa** el *getter* con su nomenclatura por defecto. Luego, **implementa** un método constructor que reciba un `JSONArray` por parámetro e inicialice el atributo de acuerdo a la respuesta de expuesta arriba (esperando una lista de `Character`), de forma parecida a como has

hecho en `ClipsList.java`.

Celda: XML y Java

Añade un nuevo archivo de interfaz `character_recycler_cell.xml`. **Haz** que el elemento raíz sea un `<ConstraintLayout>`, con `layout_width="match_parent"` y `layout_height="wrap_content"`. *Dentro* del `<ConstraintLayout>`, **añade** los dos elementos que se indican a continuación:

- Un `<ImageView>` con *id*, *anchura*, *altura* y *descripción de contenido* como se indica a continuación:

```
<ImageView
    android:id="@+id/image_view_character"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:contentDescription="Imagen de personaje"
/>
```

...y además, tendrá `scaleType="centerInside"` y *constraints* para pegarlo a *izquierda*, *derecha* y *arriba* de su contenedor (¡pero no abajo!)

- Un `<TextView>` con *id*, *anchura*, *altura*, *color de texto* y *tamaño de texto* como se indica a continuación:

```
<TextView
    android:id="@+id/text_view_character"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#fff"
    android:textSize="10sp"
/>
```

...y además, tendrá *constraints* para pegarlo a *izquierda*, *derecha* y *abajo* de su contenedor. Por *arriba* estará pegado a la parte *inferior* de la imagen¹.

Después, **añade** una nueva clase Java `CharacterViewHolder.java`. En su contenido, **incluye** el siguiente código:

```
public class CharacterViewHolder extends RecyclerView.ViewHolder {
    private TextView nameTextView;
    private ImageView characterImageView;

    public CharacterViewHolder(@NonNull View itemView) {
        super(itemView);
        nameTextView = itemView.findViewById(R.id.text_view_character);
        characterImageView = itemView.findViewById(R.id.image_view_character);
    }

    public void showData(Character character) {
        this.nameTextView.setText(character.getName());
        Util.downloadBitmapToImageView(character.getImageUrl(), this.characterImageView);
    }
}
```

...y **añade** los `import` necesarios.

(Como puedes observar, esta clase Java es análoga a `ClipViewHolder.java`).

Adaptador

Añade una nueva clase Java `CharactersAdapter.java` que comience de esta manera:

```
public class CharactersAdapter extends RecyclerView.Adapter<CharacterViewHolder> {
    private CharactersList charactersToShow;

    public CharactersAdapter(CharactersList charactersList) {
        this.charactersToShow = charactersList;
    }

    // Completa aquí Los métodos de RecyclerView.Adapter
    /* ... */
}
```

...y **completa** su implementación. Será *análoga* a `ClipsAdapter.java` y contendrá `onCreateViewHolder`, `onBindViewHolder` y `getItemCount`.

Completar la *activity*

En `activity_video.xml` **añade** una etiqueta `<RecyclerView>` a continuación del `<VideoView>` que ya existe. Así:

```
<VideoView
    android:id="@+id/video_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>

<!-- Nuevo RecyclerView -->
<!-- Solapa (por encima) al VideoView parcialmente. Es lo que queremos -->
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_view_characters"
    android:layout_width="77dp"
    android:layout_height="0dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"/>
```

Y para dar uso a lo que hemos creado hasta ahora, **abre** `VideoActivity.java`.

En `VideoActivity.java` **añade** los nuevos atributos `recyclerView`, `queue` y `charactersOnScreen` que se muestran a continuación:

```
public class VideoActivity extends AppCompatActivity {
    /* ... */
    private RecyclerView recyclerView;
    private RequestQueue queue;
    private CharactersList charactersOnScreen;
```

En el método `onCreate`, **inicializa** `recyclerView` y `queue` como cabe esperar:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_video);

    queue = Volley.newRequestQueue(this); // Nueva cola. Idealmente, en una app, se comparte una sola RequestQueue
    recyclerView = findViewById(R.id.recycler_view_characters); // Buscamos por ID el recyclerView recién añadido

    // ...
```

Ahora, vamos a hacer que cuando el usuario pulse en el elemento principal de la pantalla (`videoView`), la *app* mande una petición al API REST para consultar los personajes en pantalla.

Completa el código de `onCreate` como se muestra a continuación:

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    // ...

    videoView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            // clipId fue inicializado más arriba, a través de los 'Extras'
            // .getCurrentPosition() devuelve el milisegundo actual del video
            sendAppearancesRequest(clipId, videoView.getCurrentPosition());
        }
    });
}
```

...y **añade** el método privado `sendAppearancesRequest`:

```
private void sendAppearancesRequest(int clipId, int milliseconds) {

    /* ... */
```

```
}

```

Dentro de este método:

- (1) **Crea** una nueva instancia de `JSONArrayRequest`
- (2) Para ello, **indica** el método HTTP `Request.Method.GET`
- (3) También, **indica** la URL apropiada que fue especificada al inicio de la tarea, en función de `clipId` y `milliseconds`. (Recuerda usar `Server.name` como `host`).
- (4) Si la petición es exitosa, en `Response.Listener` **parsea** la respuesta del servidor y usa un método `setter` para almacenarla, *análogamente* a `MainActivity.java` (¡el método `setCharactersOnScreen` lo escribiremos a continuación!):

```
new Response.Listener<JSONArray>() {
    @Override
    public void onResponse(JSONArray response) {
        CharactersList parsedServerResponse = new CharactersList(response);
        setCharactersOnScreen(parsedServerResponse);
    }
},
```

- (5) Si la petición falla, en `Response.ErrorListener` **muestra** un `Toast` con el mensaje que tú quieras.
- (6) Al final, tras instanciar tu `JSONArrayRequest`, **añade** dicha petición a la cola de red con `queue.add(/*...*/);`.

Para finalizar la tarea, **escribe** el método `setter` `setCharactersOnScreen`, **así**:

```
private void setCharactersOnScreen(CharactersList charactersOnScreen) {
    this.charactersOnScreen = charactersOnScreen;
    CharactersAdapter myAdapter = new CharactersAdapter(this.charactersOnScreen);
    recyclerView.setAdapter(myAdapter);
    recyclerView.setLayoutManager(new LinearLayoutManager(this));
}
```

(Funciona como el `setClips` de `MainActivity.java`. Esta vez puede ser `private`. No hace falta crear un `getter`).

Si has seguido todos los pasos de esta importante tarea, **¡enhorabuena!** Has implementado una importante funcionalidad haciendo uso de todo lo visto hasta ahora.

Adelante, lanza la *app* y después de esperar a que cargue algún vídeo, toca la pantalla. ¿Qué sucede?

Por último

Sube tus cambios al repositorio en un nuevo *commit*.

1. La *constraint* que necesitarás será: `app:layout_constraintTop_toBottomOf="@id/image_view_character"` 



Rubén Montero @ruben.montero changed milestone to [%Sprint 2](#) 3 days ago