


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

# Modelo

## Resumen

- Crearemos una clase `Clip` que representa un objeto de la lista JSON enviada por el servidor
- Crearemos una clase `ClipsList`, que contiene varios `Clip` y representa la lista entera
- Declaremos un nuevo atributo de tipo `ClipsList` en nuestra `MainActivity`
- Lo inicializaremos con la respuesta del servidor

## Descripción

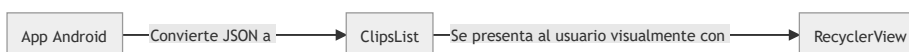
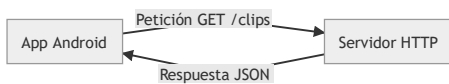
Recordemos que los datos que vienen de:

- <http://raspi:8000/clips>

...son:

```
[
  {
    "id": 1,
    "title": "Automatic Dialer (Part II)",
    "videoUrl": "https://raw.githubusercontent.com/rubenmv0/fp/main/simpsons/automatic_dialer_part_ii.mp4"
  },
  {
    "id": 2,
    "title": "Desparasitándose",
    "videoUrl": "https://raw.githubusercontent.com/rubenmv0/fp/main/simpsons/desparasitandome.mp4"
  },
  {
    "id": 3,
    "title": "Glasses",
    "videoUrl": "https://raw.githubusercontent.com/rubenmv0/fp/main/simpsons/glasses.mp4"
  },
  {
    "id": 4,
    "title": "Marcador Automático",
    "videoUrl": "https://raw.githubusercontent.com/rubenmv0/fp/main/simpsons/marcador_automatiko.mp4"
  },
  {
    "id": 5,
    "title": "Tarta",
    "videoUrl": "https://raw.githubusercontent.com/rubenmv0/fp/main/simpsons/tarta.mp4"
  }
]
```

Consisten en una lista de *clips* que queremos mostrar al usuario.



Vamos a crear clases Java que representan los datos del servidor. Serán nuestras clases *modelo*. En concreto, trabajaremos con:



## La tarea

**Crea** una nueva clase Java. **Dale** el nombre `Clip`.

Dentro de `Clip.java`, **añade** el siguiente código:

```
public class Clip {
    private int id;
    private String title;
    private String urlVideo;

    public Clip(JSONObject json) throws JSONException {
        this.id = json.getInt("id");
        this.title = json.getString("title");
        this.urlVideo = json.getString("videoUrl");
    }
}
```

(Como puedes ver, hay tres atributos que son los datos de un clip, y además, hemos añadido un método constructor especial que sirve para construir una instancia de `Clip` a partir de un `JSONObject`).

Para terminar con esta clase, **añade** los métodos **getters** de los tres atributos, con sus nombres estándar. Puedes autogenerarlos desde la barra superior de Android Studio con `Code > Generate > Getters and Setters`.

Luego, **crea** otra nueva clase Java. **Dale** el nombre `ClipsList`.

Dentro de `ClipsList.java`, **añade** el siguiente código:

```
public class ClipsList {
    private List<Clip> clips;

    public ClipsList(JSONArray array) {
        clips = new ArrayList<>();
        for (int i = 0; i < array.length(); i++) {
            try {
                JSONObject jsonElement = array.getJSONObject(i);
                Clip aClip = new Clip(jsonElement);
                clips.add(aClip);
            } catch (JSONException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

(Como puedes observar, tiene un atributo `List<Clip>` que se inicializa a partir del contenido de un `JSONArray`, con el código del método constructor).

Para terminar con esta clase, también **añade** un método **getter** del único atributo que existe, con un nombre estándar (`getClips()`).

## Usando `ClipsList`

Estas clases que hemos creado (`ClipsList.java` y `Clip.java`) son *modelos*. Representan datos que vienen del servidor, y gracias a ellas, podemos desentendernos de *parsear* el JSON del servidor y centrarnos en manejar los datos.

En `MainActivity.java`, **declara** un nuevo atributo de tipo `ClipsList`, y un **getter** y un **setter** como se indica a continuación:

```
public class MainActivity extends AppCompatActivity {
    // ...
    private ClipsList clips;

    // ...

    public void setClips(ClipsList clips) {
        this.clips = clips;
    }

    public ClipsList getClipsForTest() {
        return clips;
    }
}
```

(Como puedes ver, `getClipsForTest` no es un nombre estándar para un método getter. Este método sólo sirve para que funcione un test automático que verificarás al final del sprint).

Para terminar la tarea, en el `onResponse` de la petición a <http://raspi:8000/clips>, **añade** la siguiente línea `setClips(new ClipsList(response));` :

```
// ...
new Response.Listener<JSONArray>() {
    @Override
    public void onResponse(JSONArray response) {
        progressBar.setVisibility(View.INVISIBLE);
        Snackbar.make(mainLayout, "Clips obtained", Snackbar.LENGTH_SHORT).show();

        // Parseamos la respuesta y la asignamos a nuestro atributo
        setClips(new ClipsList(response));
    }
}
```

¡Listo! Aunque no hay cambios visuales en la aplicación, en esta tarea hemos conseguido almacenar como un atributo `ClipsList` los datos que vienen en la `response` del servidor.

## ¿Ha valido la pena?

Indudablemente. El [patrón de arquitectura Modelo-Vista-Controlador](#) es de los más populares y nosotros estamos poniéndolo en práctica.

## Por último

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



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