Desarrollo de Aplicaciones Multiplataforma

IES Melchor Gaspar de Jovellanos

SPOTFLYX

Juan Carlos Hita Sánchez

Oscar Moreno Huete

Román Pacheco Murillo

Trabajo fin de grado
Coordinador: Miguel Ángel Muñoz Herrera
Curso 2018 - 2019

TABLA DE CONTENIDOS

RESUMEN

El avance de las conexiones de internet y las tecnologías a día de hoy nos permite un gran abanico de posibilidades desde cualquier lugar y en cualquier momento gracias a nuestros dispositivos móviles. En este proyecto hemos decidido estudiar y emular una de ellas: la reproducción de contenido multimedia a través de streaming.

Gracias a plataformas de reproducción de películas y música como Netflix, Spotify, HBO, tanto como canales de TV de pago que tan en auge están en nuestros días, nos vemos obligados a investigar y explicar de manera generalizada el funcionamiento de estas.

En este proyecto, bajo una arquitectura cliente-servidor conseguiremos que desde un dispositivo móvil Android se brinde la posibilidad de acceso a multitud de recursos multimedia (Películas, series y música) sin necesidad de descargarlos en nuestro dispositivo, dando así al cliente la posibilidad consumirlos desde cualquier lugar y en cualquier momento sin necesidad de esperas y de la descarga total del contenido en sus dispositivos.

En las siguientes páginas mostraremos y detallaremos el uso de las tecnologías que hemos tenido que emplear para más adelante comprender el desarrollo y funcionamiento de nuestra aplicación. Pero antes como es obligatorio haremos hincapié en explicar con detenimiento que es el streaming, los conceptos más básicos, cuáles son sus protocolos, y el futuro de este.

TECNOLOGIAS UTILIZADAS EN EL PROYECTO

¿QUE ES EL STREAMING?

Podemos destacar principalmente dos formas de consumir contenido streaming; Live y on-demand (esta última será la utilizada por nosotros). La diferencia entre estas como bien indica su nombre es clara, live reproducirá el contenido en vivo en el momento en el que se está llevando a cabo y on-demand se hará bajo petición albergando el contenido en un servidor y dando la posibilidad de acceder a este en cualquier momento.

Podríamos distinguir tres formas de retrasmisión de contenido de manera streaming:

Streaming tradicional: Un ejemplo de streaming tradicional es el funcionamiento del protocolo RTSP (Real-Time Streaming Protocol). RTSP es un protocolo con estado; quiere decir que el servidor necesita constantemente conocer cuál es el estado del cliente ya que establece una conexión continua con el cliente hasta que se cierra la sesión. Podemos decir que una de las desventajas de este protocolo es que el envió de paquetes es de tasa fija, es decir, la tasa de bit-rate siempre es la misma (a diferencia de otros protocolos que veremos más adelante) esto supone que la codificación de calidad del video sea siempre la misma. Otra de las características de este tipo de streaming es que el servidor envía paquetes hasta que el buffer se llena, por tanto la tasa de bit o bit-rate debe ser menor al ancho de banda para no saturar el buffer si no queremos experimentar cortes.

Descarga progresiva: Esta técnica se basa en la descarga de ficheros de un servidor HTTP. La descarga es progresiva, esto significa que el video se puede reproducir mientras se efectúa la descarga. Si paramos la reproducción en una descarga progresiva y esperamos el contenido se almacena en caché posteriormente podremos visualizar el contenido sin datos. El problema que surge es que si finalmente no queremos ver el video, se habrán gastado nuestros recursos de forma innecesaria al haber descargado todo el contenido. HTTP es un protocolo sin estado y por ello no le hace falta establecer una conexión continua con el cliente puesto que la sesión se termina con cada envió de datos hasta que el cliente vuelva a solicitar más.

Streaming adaptativo: La principal diferencia respecto a la descarga progresiva es que en este caso se realizan descargas pequeñas en vez de realizar la descarga completa de todo el contenido. Para realizar esto posible troceamos el contenido de audio y video lo codificamos en la manera que lo queremos retrasmitir. Los segmentos de contenido multimedia se guardan en un servidor Web HTTP, y cuando el cliente lo solicita se realiza una descarga progresiva de cada segmento o trozo.

Lo más beneficioso de este tipo, es que el video o audio se codifica de diferentes calidades o bit-rates y el cliente puede elegir entre las diferentes calidades de reproducción aunque también podria variar dependiendo del nivel del ancho de banda de red o el dispositivo que se utilice. Este último debido ya que es el más óptimo es el que utilizaremos.

De los protocolos de Streaming tradicional (RTSP y RTMP) se ha pasado a utilizar protocolos basados HTTP por su mejor rendimiento e implementación. Uno de los pioneros fue el protocolo de Smooth Streaming, también denominado HSS (HTTP Smooth Streaming), este fue la solución de Microsoft, HLS (HTTPLive Streaming) la solución de Apple y HDS (HTTP Dynamic Streaming) la solución de Adobe. Por otro lado, existe un protocolo más reciente denominado MPEG-DASH (Moving Picture Experts Group-Dynamic Adaptive Streaming over HTTP).

Nosotros y del que hablaremos más adelante, utilizaremos HLS aunque este, siendo un protocolo diseñado por Apple, no podremos reproducirlo con la SDK nativa de Android.

El Futuro del streaming

Si hablamos del futuro del streaming no podemos pasar alto nombrar el protocolo MPEG-DASH (Dynamic Adaptive Streaming over HTTP). Este protocolo de streaming adaptativo es similar a HTTP live streaming (HLS). Es el único protocolo de estándar internacional, y que busca ganarse la confianza del mercando buscando una estabilidad evitando incompatibilidades entre los dispositivos de los usuarios. MPEG-DASH aunque no es compatible con HTML5, existen para él implementaciones en JavaScript que permiten ser utilizado.

Otras de las últimas novedades del Streaming vienen de la mano de Netflix la plataforma de reproducción de películas y series. Esta plataforma cada vez está haciendo más hincapié en la importancia del desarrollo de un nuevo algoritmo que permita la codificación plano a plano, ¿El motivo? La reducción al máximo de la tasa de datos enviados para la reducción del coste de infraestructura y satisfacción del cliente. Imagínate que estás viendo una película y en una escena aparecen pocos colores: el personaje en primer plano con un fondo blanco o de tan sólo un tono. Dicha escena no necesitará la misma tasa de bits o codificación que otra en la que aparezca un paisaje, con muchísimos colores vivos, aunque estemos hablando de la misma película. El vicepresidente de Producto e Innovación en Netflix nos hablaba de otro caso concreto: 'Barbie', la serie de dibujos, tenía una escena muy complicada al final con muchos colores, muchos tonos y mucha complejidad. Por tanto, y para que esa última escena se viera bien, todo el título requería una alta tasa de bits. Ahora, al tener cada toma su propio bit-rate, no tendrían ese problema.

¿Qué es HLS?

HLS el protocolo diseñado por Apple, utiliza una comprensión vídeo H.264 MPEG-2 TS segmentado y listas M3U8 tanto para la consumición del vídeo en directo como on-demand. Un archivo M3U8 es un índice que permite al reproductor del cliente conocer el orden de segmentos, las distintas calidades de video y la resolución que le permite reproducir. El dispositivo tiene la posibilidad de seleccionar automáticamente la secuencia más adecuada desde el archivo manifiesto debido sus características, teniendo en cuenta las limitaciones de ancho de banda y resolución del dispositivo, para más tarde descargar el segmento en buffer y reproducirlo (Siempre se recomienda colocar en primera instancia la resolución más baja y que a partir de ahí el propio dispositivo se vaya adecuando según sus características y capacidad de ancho de banda).

Como hemos citado anteriormente, HLS transmiten datos a través de HTTP, y ofrece diversas mejoras sobre los protocolos de transmisión tradicionales, como RTSP o RTMP, entre las que se cuentan:

-Reducción de los costes de infraestructura
-Reducción de las amenazas procedentes de restricciones de proxy y cortafuegos debido a la utilización de HTTP.
-Optimización de la codificación de video a tiempo real mediante la tasa de bits adaptativa en caso de experimentar cortes.
-Redundancia integrada en caso de fallos.
-Implementación sencilla de reproductores en HTML5 apoyados sobre un plug-in JavaScript o librerías externas en caso de dispositivos móviles Android.

EJEMPLO: PRUEBA DEL REPRODUCTOR HTML5 BÁSICO USANDO VIDEO.JS

```
<!DOCTYPE html>
<html>
<head>
        <title>Reproductor de pruebas de transmisión de secuencias en directo vía HTTP </title>
        <link href="http://vjs.zencdn.net/c/video-js.css" rel="stylesheet" type="test/css">
        <script src="http://vjs.zencdn.net/c/video.js"></script>
</head>
<body>
        <video id="example_video_1" class="video-js vjs-default-skin" controls autoplay
        width="640" height="360" data-setup="{}">
                <source src="http://SuServidorAquí/playlist.m3u8" type="application/x-mpegURL" />
        </video>
</body>
</html>
```

Listas M3U8

Como hemos citado anteriormente estas no son más que playlist en las que contendremos nuestro archivo multimedia de manera fragmentado. Esto nos ofrece que el cliente vaya segmento a segmento descargando y reproduciendo las particiones (o también llamados chunks) que hemos indicado dentro de la lista M3U8 sin necesidad de descargar en buffer todo el archivo multimedia, renderizando así, tanto los recursos de cara al cliente como los del servidor. En estas listas podemos indicar las diferentes codificaciones o resoluciones que tenemos de un archivo multimedia. Estas particiones normal general tendran aproximadamente una duración de 10 segundos.

```
Archivo  Edición  Formato  Ver  Ayuda
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=800000,RESOLUTION=720x404
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/axnhd/_definst_/liveevent/livestream1.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=96000,CODECS="mp4a.40.2"
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/audio-only-aac/axnhd/_definst_/liveevent/livestream1.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1200000,RESOLUTION=720x404
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/axnhd/_definst_/liveevent/livestream2.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1600000,RESOLUTION=1280x720
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/axnhd/_definst_/liveevent/livestream3.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=2300000,RESOLUTION=1280x720
http://nubeoxlivegeo-live.hls.adaptive.level3.net/hls-live/axnhd/_definst_/liveevent/livestream4.m3u8
#EXT-X-FAXS-CM:URI="/hls-live/axnhd/_definst_/liveevent/livestream1.drmmeta"
```

*Archivo manifest donde vemos las listas m3u8 de un mismo archivo multimedia con distintas resoluciones y calidades asociadas

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:380284
#EXT-X-ALLOW-CACHE:NO
#EXT-X-VERSION:2
#EXT-X-FAXS-CM:URI="livestream1.drmmeta"
#EXT-X-KEY:METHOD=AES-128,URI="https://keyserver1.adobeaccess.vualto.com/faxsks/vualto/key",IV=0x3d32059bb9268f76946de252b7701
#EXT-X-TARGETDURATION:8
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380284.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380285.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380286.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380287.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380288.ts
#EXTINF:8,
../../../../hls-live/streams/axnhd/events/_definst_/liveevent/livestream1Num380289.ts
```

*Archivo sub-manifest donde vemos los chunks o particiones del archivo multimedia

FFMPEG

Es la herramienta que utilizaremos para segmentar nuestro los archivos multimedia y cambiarlos a formato .ts, para así permitirnos reproducir nuestro contenido de manera streaming bajo el protocolo HLS. Esta es una aplicación de software libre que nos permite convertir videos y audio a cualquier otro formato mediante la línea de comandos entre otras cosas. Podemos encontrar una gran cantidad de aplicaciones similares en internet pero FFMPEG es sin duda de las más potentes y utilizadas. Esta desarrollada bajo sistema operativo GNU/Linux.

El proyecto FFMPEG contiene diversas librerías para el desarrollo de software además de aplicaciones que se pueden descargar independientemente para Windows como un reproductor multimedia o una herramienta para el análisis e información de archivos multimedia.

Las funcionalidades que nos brinda FFMPEG son las siguientes:

- Convertir entre distintos formatos de video y audio.
- Codificar video para reproducir en cualquier equipo o dispositivo.
- Capturar y procesar el video de cualquier dispositivo conectado a la computadora.
- Cambiar el tamaño y la relación de aspecto.
- Modificar el bit-rate.
- Dividir o unir videos.
- Aplicar distintos filtros como rotar videos, modificar el volumen del sonido, etc…
- Extraer el audio o la música de los videos.
- Insertar una pista de audio en unvideo.
- Crear presentaciones de video usando imágenes.
- Convertir videos en animaciones GIF.

A continuación, mostraremos una imagen con la línea de código que hemos de introducir si queremos transformar un video .mp4 albergado en el directorio donde está situado nuestro ejecutable FFPMEG, para posteriormente transformarlo en una lista M3U8 con formato .ts y con una segmentación de 10 segundos.

```
C:\Users\yonom\Desktop>cd C:\Users\yonom\Desktop\Nueva carpeta\ffmpeg-20190323-5252d59-win64-static\bin

C:\Users\yonom\Desktop\Nueva carpeta\ffmpeg-20190323-5252d59-win64-static\bin>ffmpeg -i test.mp4 -hls_list_size 0 -f  -segment_time 10 output.m3u8
```

El resultado sería el siguiente:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:2
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:2.020333,
six_days0.ts
#EXTINF:1.997111,
six_days1.ts
#EXTINF:1.997111,
six_days2.ts
#EXTINF:1.997111,
six_days3.ts
```

TOMCAT

Desarrollado por Sun MicroSystems y posteriormente donado a la fundación Apache. Tomcat es un contenedor de Java Servlets que facilita y simplifica la atención de peticiones a un servidor. Aunque puede atender cualquier tipo de peticiones es usualmente utilizado para extender aplicaciones Web como es nuestro caso. La aplicación que se ejecute en un servidor Tomcat debe ser una aplicación desarrollada en el lenguaje de programación Java.

```java
package org.pruebas;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HolaSextoInformaticaServlet extends HttpServlet {

    /**
     * Servlet de ejemplo que procesa una petición GET
     * @param request
     * @param response
     * @throws ServletException
     * @throws IOException
     */
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">");
        out.println("<html>");
        out.println("<head><title>Ejemplo HolaSextoInformatica</title></head>");
        out.println("<body>");
        out.println("<h1>¡Hola HolaSextoInformatica!</h1>");
        out.println("</body></html>");
    }
```

*Ejemplo de petición GET que gestiona una petición y devuelve una página web a través de un Servlet HTTP

¿Qué una aplicación WEB?

Una aplicación web es una estructura de ficheros con el que configuraremos y daremos instrucciones de proceder frente a las peticiones que se hagan a nuestro servidor Tomcat. Estos archivos son transferidos dentro de un archivo con extensión war (WebApplicationArchive). Los ficheros se pueden clasificar en dos tipos: Los que deben ser accesibles para los usuarios de la aplicación web y los archivos que son necesarios para el funcionamiento de la aplicación. Los archivos que no deben ser accesibles para los usuarios estarán dentro de la carpeta WEB-INF. Los ficheros contenidos en esta serán:

-El fichero web.xml, denominado el descriptor de desplegado (deployment descriptor). Le da al servidor información como por ejemplo qué servlets contiene esta aplicación, en que rutas deben ser colocados, etc…

-El directorio lib, contiene ficheros jars con las librerías necesarias para la ejecución de la aplicación.

-El directorio clases, contiene los ficheros .class que componen la aplicación.

Como ya hemos explicado, todos los demás archivos que estén fuera de este directorio estarán disponibles a través del protocolo HTTP.

REST y JAX-RS

REST es un conjunto de principios de arquitectura que utiliza el protocolo HTTP para obtener datos o indicar maneras de proceder sobre estos en diversos formatos (HTML, XML, JSON, etc.)

Una de las características de REST y gracias según expertos, la web ha disfrutado de una gran escalabilidad, es una serie de diseños fundamentales clave que nombraremos a continuacion:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para gestionar la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión.

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE.

- Una sintaxis única para identificar los recursos. En un sistema REST, cada recurso es gestionable únicamente a través de su URI.

La API que nosotros utilizaremos para trabajar sobre la arquitectura REST será JAX- RS (Del Acrónimo Java API RESTful Web Service). JAX-RS nos permite la creación de servicios web con arquitectura REST que nos serviría para simplificar el desarrollo y el despliegue de los servicios consumidos en las aplicaciones web.

JAX-RS tiene como principales anotaciones las ya conocidas etiquetas que sirven para utilizar los servicios que ofrece nuestra aplicación Web a través de las siguientes anotaciones:

-@Path: para la ruta de acceso relativa de una clase recurso o método.
-@GET, @PUT, @POST, @DELETE y @HEAD tipo de petición HTTP de un recurso a ejecutarse.
-@Produces para los tipos de medios MIME de respuesta (HTML, Json, Xml, texto plano etc...).
-@Consumes para los tipos de medios o de formatos aceptados por una petición.

Además, se nos proporciona anotaciones adicionales para enviar parámetros a nuestra petición para extraer información personalizada de la solicitud.

- @PathParam enlaza el parámetro a un segmento de ruta.
- @QueryParam enlaza el parámetro al valor de un parámetro de consulta HTTP.
- @CookieParam enlaza el parámetro a un valor de una cookie.
- @DefaultValue especifica un valor por defecto para los enlaces anteriores cuando la clave no es encontrada.

Las implementaciones que dan soporte a JAX-RS son algunas de estas herramientas: Apache CXF, un framework de servicios web de código abierto. Jersey (Oracle), RESTeasy (JBoss), Restlet, (Jerome Louve) y Apache Wink de Apache Software Foundation Incubator. Nosotros para nuestra aplicación utilizaremos Jersey.

El siguiente es un ejemplo de código en java con las anotaciones de JAX-RS:

```
package classes;

import java.util.ArrayList;

@Path ("FilmRestService")
public class FilmRestService {

    @GET
    @Path("peliculas")
    @Produces(MediaType.APPLICATION_JSON)
public List<Pelicula> getFilms()
{
        List<Pelicula> listOfFilms=new ArrayList<Pelicula>();
        listOfFilms=recdFilms();
        return listOfFilms;

}

   public static List<Pelicula> readFilms()
   {
       List<Pelicula> listaPeliculas=new ArrayList<Pelicula>();
       try {
           Type tipoListaPeliculas = new TypeToken<List<Pelicula>>(){}.getType();
             Gson gson=new Gson();

             listaPeliculas = gson.fromJson (new FileReader ("C:/Users/yonom/eclipse-workspace/WebService/listaPeliculas.json"), tipoListaPelic

           } catch (JsonIOException | JsonSyntaxException | FileNotFoundException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
           }
       return listaPeliculas;
```

Por ejemplo, si quisiéramos consumir el servicio del Path “películas” deberíamos ejecutar una petición contra el servidor con la siguiente ruta:

http://localhost:8080/WebService/rest/UsersRestService/peliculas

Donde “WebService” es el nombre de nuestra aplicación “rest” la ruta principal declarada en el fichero ApplicationConfig donde se encapsulan las diferentes clases que tendrá nuestra aplicación, “UserRestService” el nombre de nuestra clase y “películas” el nombre de la petición que queremos consumir.

A continuación pondremos un ejemplo donde enlazaremos un parámetro a nuestra petición a través de un segmento de la ruta.

```java
@GET
@Path("{artista}")
@Produces(MediaType.APPLICATION_JSON)
public List<Cancion>  getSongsByArtist(@PathParam("artista") String ar)
 {
        List<Cancion> listOfSongs=new ArrayList<Cancion>();
        List<Cancion> listOfSongsByArtist=new ArrayList<Cancion>();
        listOfSongs=readSongs();

      for (Cancion f: listOfSongs)
      {
           if(f.getGenero().equalsIgnoreCase(ar))
           {

               listOfSongsByArtist.add(f) ;

           }

      }

  return listOfSongsByArtist;
 }
```

En este caso si por ejemplo quisiéramos que la petición nos devolviera la lista de canciones de un artista en concreto tendríamos que elaborar la siguiente petición:

http://localhost:8080/WebService/rest/SongRestService/artista/DavidBisbal

FICHEROS JSON

Del acrónimo JavaScript Object Notation. Es un formato de ficheros de tipo atributo-valor utilizado para la transmisión de datos. Dentro de él podremos establecer tipos de datos como números, cadenas de caracteres, arrays, booleanos y objetos. La ventaja de JSON, al ser un formato de fichero independiente de cualquier lenguaje de programación, es que los servicios que comparten datos de esta manera, no necesitan hablar el mismo lenguaje, es decir, el emisor puede ser Java y el receptor PHP, cada lenguaje tiene su propia librería para codificar y decodificar cadenas de JSON. En nuestro caso y aunque no es la única opción utilizaremos la librería GSON para trabajar con este tipo de ficheros.

```json
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                {
                    "value": "New", "onclick": "CreateNewDoc()"
                },{
                    "value": "Open", "onclick": "OpenDoc()"
                },{
                    "value": "Close", "onclick": "CloseDoc()"
                }
            ]
        }
    }
}
```

*Definición de un objeto menú que contiene 2 atributos y un objeto popup que a su vez contiene un array llamado menuitem

GSON

Es una librería de código abierto creada por google que nos permite fácilmente serializar y deserializar objetos Java. Es decir, con la librería Gson podemos transformar un fichero Json en objetos Java y viceversa. Para conseguir esto, previamente deberíamos tener una clase Java de mapeo con los mismos atributos definidos en nuestro Json, para poder así asignar estos valores a los atributos de nuestro objeto java. En el caso contrario será similar puesto que los pares atributo-valor de nuestro objetos Java serán escritos en un fichero con formatos Json.

A continuación mostraremos uno de los ejemplos más básicos de conversión entre estos:

```java
public class Persona implements Serializable {
    private String nombre;
    private int edad;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

*Clase Java de un objeto persona

```
{"nombre":"Unai","edad":28}
```

```java
Gson gson = new Gson();

Persona persona = (Persona) gson.fromJson(json, Persona.class);
```

*Creación de un objeto persona con los valores del fichero Json

```
Persona persona = new Persona();
persona.setNombre("Unai");
persona.setEdad(28);

Gson gson = new Gson();
System.out.println(gson.toJson(persona));
```

```
{"nombre":"Unai","edad":28}
```

*Creación de un string con formato Json a partir de un objeto Java de la clase Persona.

Nosotros para nuestra aplicación utilizaremos unos de los métodos de mapeo de fichero Json que nos ofrece esta librería. Consiste en instanciar una objeto de tipo Type en el que le diremos que tipo de objeto es (En nuestro caso será una lista) y a que clase pertenece esta lista (En nuestro caso serán usuarios), para posteriormente utilizar una instancia de un objeto Gson que llamara a el método fromJson(), donde le pasaremos a este, un flujo de lectura hacia un archivo Json y nuestro objeto Type. Con esto conseguiremos que nuestra lista contenga una serie de objetos Java con el contenido de nuestro fichero Json mapeado.

```
Type tipoListaUsers = new TypeToken<List<Login>>(){}.getType();
  Gson gson=new Gson();

  listaUsuarios = gson.fromJson (new FileReader ("C:/Users/yonom/eclipse-workspace/WebService/listausuarios.json"), tipoListaUsers);
```

BRIGHTCOVE PLAYER

Construido sobre la biblioteca ExoPlayer este reproductor nos permite la reproducción de HLS de manera eficiente y solucionando los problemas que tanto MediaPlayer, el reproductor nativo de Android, como exoplayer, estabán dando a la hora de gestionar contenido bajo este protocolo. Brightcove player además de permitirnos utilizar streaming adaptativo nos permite emplear subtítulos en nuestros videos, cifrar nuestra transmisión de datos, transmitir el contenido multimedia en vivo, colocar publicidad sobre las reproducciones y utilizar análisis del comportamiento de los usuarios sobre nuestros videos con tecnologías como Adobe Analytics.

Clase HttpsURLConnection

La clase HttpsURLConnection es una clase Java que nos permitirá poder comunicarnos con un servidor mediante peticiones HTTP o HTTPS a través de una URI. Para ejecutar una petición en Android necesitaremos que nuestra clase donde instanciaremos nuestro objeto HttpsURLConnection extienda de la clase AsyncTask puesto que tendremos que realizar la petición en otro hilo independiente del que se ejecuta nuestra aplicación. Una vez instanciemos el objeto HttpsURLConnection y abramos la conexión, a través de diversos métodos de este, podemos establecer tiempos máximos de conexión, tiempos máximos de lectura, y que tipo de petición haremos contra el servidor, además de conocer cuál es código de respuesta de nuestra petición HTTP.

En caso de que la petición realizada sea una petición GET, para captar los datos enviados por el servidor necesitaremos crear un flujo de entrada y leer los datos recibidos para más tarde operar con ellos.

```
URL url = new URL(path);
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setReadTimeout(150000);
conn.setConnectTimeout(150000);
conn.setRequestMethod("GET");
int responseCode = conn.getResponseCode();
String responseMessage = conn.getResponseMessage();

if (responseCode == HttpsURLConnection.HTTP_OK) {

    BufferedReader in = new BufferedReader(
            new InputStreamReader(conn.getInputStream()));
    String output;
    StringBuffer response = new StringBuffer();

    while ((output = in.readLine()) != null) {
        response.append(output);
    }
    in.close();

    responseText = response.toString();
```

*Ejemplo de petición HTTP a un servidor RESTful y que asigna los datos de respuesta del servidor dentro de una variable de tipo String.

SPOTFLIX APP

De ahora en adelante explicaremos paso a paso cuales son los procedimientos de desarrollo que hemos tenido que llevar a cabo para dar funcionalidad a nuestra aplicación y cumplir sus objetivos, además de explicar más detalladamente su arquitectura cliente servidor y como es la interacción entre ellas.

Como es evidente detallaremos progresivamente como es la interacción con nuestra aplicación desde el inicio, que será un sistema de login, hasta su paso final que será la consumición de un archivo multimedia.

La aplicación móvil deberá ser reproducida en un dispositivo con un api 26 o mayor. La parte servidor ha sido desarrollada con la JDK 1.8.

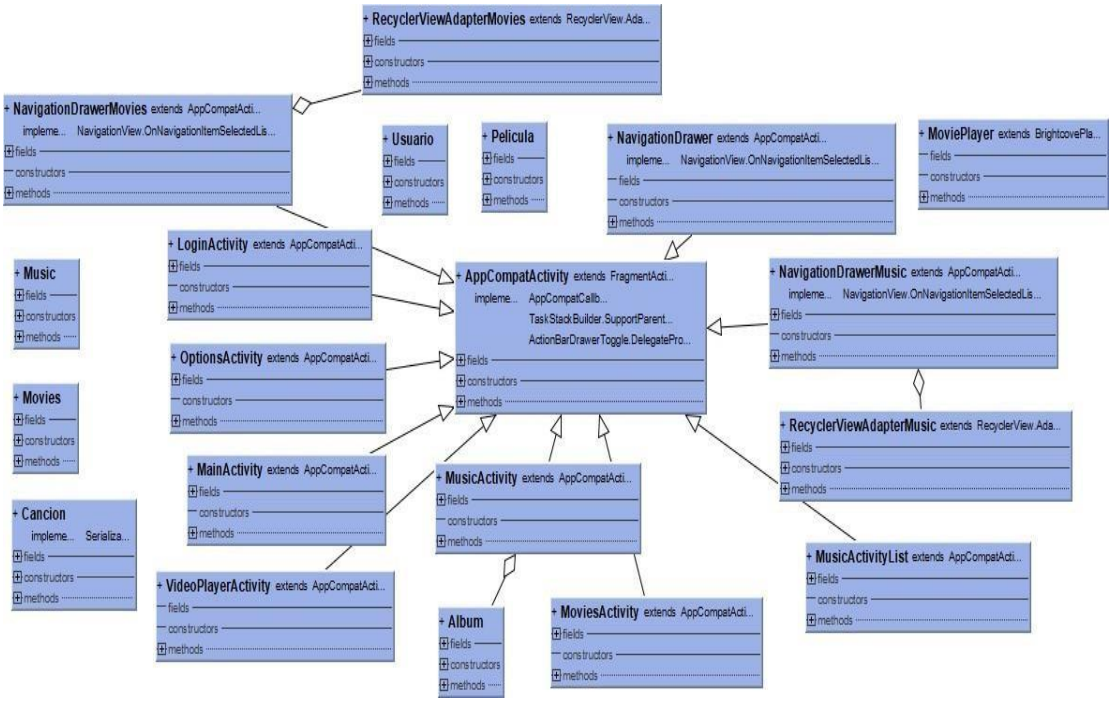


*Diagrama UML de la aplicación

Login

Nuestro sistema de logueo es sencillo, consiste básicamente en una interfaz con dos TextInputEditText. Estos mismos nos indicaran a que campo corresponde los datos que tendremos que introducir. Gracias a una de las entre muchas funcionalidades que nos ofrecen los TextInputEditText se ha habilitado en los mismos campos un contador en su parte inferior derecha que nos indicara cual es la cantidad máxima de caracteres que podremos introducir dentro de estos. En el caso de que sobrepasáramos estos caracteres el campo se pondrá de color rojo, y si aun así intentáramos loguearnos, un mensaje nos indicaría que estamos sobrepasando ese número de caracteres.

*Interfaz de logueo con el servidor.

La arquitectura del sistema de logueo es sencilla. Una vez tengamos introducidos nuestro datos e intentemos entrar, nuestra parte cliente realizara una petición contra el servidor con los datos a través de la clase HTTPURLCONNECTION para que este le devuelva un fichero Json con los usuarios ya creados anteriormente. Para conseguir esto previamente deberemos tener una clase java de mapeo con los atributos contenidos dentro del Json.

Una vez tengamos nuestro archivo Json dentro de una cadena de caracteres nos ayudaremos de la clase JSONARRAY para “setear” los valores del Json dentro del objeto de mapeo antes citado y añadirlo a una lista para posteriormente iterarla y verificar si los datos introducidos por el usuario concuerdan en nombre y contraseña.

```
[
    {"usuario":"jpako",
    "contrasena":"pako"},

    {"usuario":"pedro",
    "contrasena":"pedro123"},

    {"usuario":"joseantonio",
    "contrasena":"tony67"},

    {"usuario":"juan",
    "contrasena":"juan"},

    {"usuario":"usuario",
    "contrasena":"contrasseña"},

    {"usuario":"romanpru",
    "contrasena":"romanpru"},

    {"usuario":"admin",
    "contrasena":"admin"}

]
```

*Fichero Json con los atributos pares clave-valor que nos servirán para gestionar nuestro sistema de logueo.

```
public class Login {


    String usuario;
    String contrasena;



    public Login(String usuario, String contrasena) {
        this.usuario = usuario;
        this.contrasena = contrasena;
    }


    public String getUsuario() {
        return usuario;
    }
    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }
    public String getContrasena() {
        return contrasena;
    }
    public void setContrasena(String contrasena) {
        this.contrasena = contrasena;
    }
```

*Clase de mapeo del fichero Json.

```
BufferedReader in = new BufferedReader(
        new InputStreamReader(conn.getInputStream()));
String output;
StringBuffer response = new StringBuffer();

while ((output = in.readLine()) != null) {
    response.append(output);
}
in.close();

responseText = response.toString();

JSONArray jsonarray = new JSONArray(responseText);
for (int i = 0; i < jsonarray.length(); i++) {
    JSONObject jsonobject = jsonarray.getJSONObject(i);
    /*  et.setText(jsonobject.getString("Titulo"));*/
    String nombre = jsonobject.getString( name: "usuario");
    String contrasena = jsonobject.getString( name: "contrasena");
    Usuario obj = new Usuario(nombre, contrasena);

    lstUsers.add(obj);
}
```

*Lectura de la respuesta de la petición a el servidor y mapeo uno a uno de los valores dentro de un objeto usuario, para posteriormente añadirlos a una lista.

En el caso de que no concordaran, se mostrara un mensaje en el que se le dará la opción al usuario de crear uno nuevo, si es así, se limpiaran los campos de texto y se le indicara a través de un label que rellene los campo para la creación de un nuevo usuario. Si el usuario ya existiera, se le notificara y se le dará la oportunidad de crear otro usuario. En caso de que el usuario a crear no existiera se realizara una petición contra el servidor enlazándole al segmento de ruta los datos de usuario y contraseña para que este ultimo lo añada en el fichero Json que contiene los datos de usuario.

```
@GET
@Path("{usuario}/{contrasena}")
@Produces(MediaType.TEXT_PLAIN)
public String  crearUsuario (@PathParam("usuario") String usuario, @PathParam("contrasena") String contraseña )
 {
    String ok="ok";
    String JSON = null;
    List<Login> listaUsuarios;

    listaUsuarios=new ArrayList<Login>();
    try {
        Type tipoListaUsers = new TypeToken<List<Login>>(){}.getType();
          Gson gson=new Gson();

          listaUsuarios = gson.fromJson (new FileReader ("C:/Users/yonom/eclipse-workspace/WebService/listausuarios.json"), tipoListaUsers);

          Login usuarionuevo = new Login(usuario,contraseña);

          listaUsuarios.add(usuarionuevo);

          JSON = gson.toJson(listaUsuarios);


/         FileWriter fw=new FileWriter("C:/Users/yonom/eclipse-workspace/WebService/listausuarios.json");
/          fw.write(JSON);

          FileWriter fichero = null;
          PrintWriter pw = null;

              fichero = new FileWriter("C:/Users/yonom/eclipse-workspace/WebService/listausuarios.json");
              pw = new PrintWriter(fichero);
              pw.println(JSON);


             if (null != fichero)
                fichero.close();
```

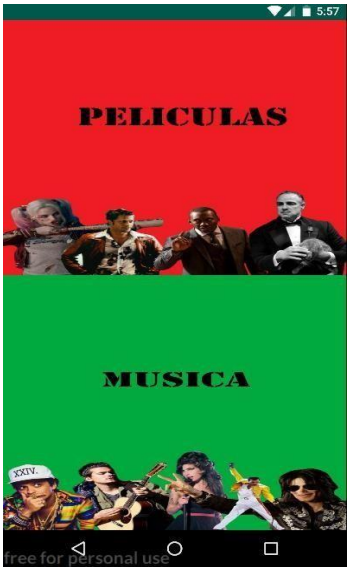
*Petición para la creación de un nuevo usuario.

```
   @GET
   @Path("usuarios")
   @Produces(MediaType.APPLICATION_JSON)
public List<Login> getUsers(){

      List<Login> listUsers=new ArrayList<Login>();
      listUsers=readUsers();
      return listUsers;

}

   public static List<Login> readUsers()
   {
      List<Login> listaUsuarios=new ArrayList<Login>();
      try {
         Type tipoListaUsers = new TypeToken<List<Login>>(){}.getType();
           Gson gson=new Gson();

           listaUsuarios = gson.fromJson (new FileReader ("C:/Users/yonom/eclipse-workspace/WebService/listausuarios.json"), tipoListaUsers);

         } catch (JsonIOException | JsonSyntaxException | FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
         }
      return listaUsuarios;
   }
```

*Petición para la obtención de un fichero Json con los usuarios creados en el servidor.

Menú de opciones

Esta interfaz podríamos considerarla de las más sencillas a nivel desarrollo y complejidad que tiene nuestra aplicación. Consiste básicamente en dos botones con una imagen de fondo ajustado al margen del layout que nos da la opción acceder al catálogo de películas o al catálogo de álbumes musicales.



Catálogos

En nuestra aplicación podemos ver dos interfaces que son similares, el catálogo de música y el de películas. A continuación nos dedicaremos a explicarlas paso a paso.

Podríamos decir que dentro de estas interfaces contenemos 3 elementos totalmente independientes entre ellos. Uno de ellos será el catálogo de películas o música, otro un barra lateral que nos permitirá realizar filtrado tanto por géneros musicales como por películas, y por ultimo una barra de búsqueda que nos permitirá realizar búsquedas por nombre de película y director o, autor, álbum y nombre de la canción. Explicaremos un poco como se construyen cada uno de ellos y cuál es su mecánica.

Catalogo

Para entender el desarrollo de la parte de catálogo de nuestra interfaz vemos conveniente explicar las clases sobre las que nos hemos tenido que apoyar para conseguir nuestro objetivo. A continuación explicaremos cuales son los conceptos claves para entenderlas un poco más.

¿Qué es un Recyclerview?

Es un contenedor de elementos (listas) similar a un ListView que recicla las vistas de elementos anteriores que no se están visualizando en la pantalla para utilizarlo en los próximos elementos.

La creación y administración de un RecyclerView es similar a la de un ListView. Se necesita una fuente de datos con la que cargar un adaptador que provea la información lógica de cada elemento para posteriormente leerlos e interpretarlos.

El LayoutManager es el encargado de añadir y reusar los Views en el recycler. Su función es calcular las posiciones fuera del foco del usuario y así remplazar el contenido de un ítem fuera del alcance visual por el contenido de otro. Esto reduce los tiempos de ejecución, ya que el número de infladas es menor.

¿Qué es un Cardview?

Con esta clase, lo que conseguimos es definir la apariencia de cada elemento del RecyclerView contenido en nuestro adaptador en forma de listado. Con ella conseguimos agrupar la información de manera personalizada y mostrarla de manera elegante y práctica.

*RecyclerView

*Elemento CardView

*ResultadoFinal

Una vez entendido esto, podemos explicar cómo estos elementos dan funcionalidad o interactúan a nuestra aplicación.

Lo primero que haremos será recargar los datos de los archivos multimedia, contenidos en el servidor para almacenarlos en una lista a través de una petición, y posteriormente añadirlos al adaptador de nuestro RecyclerView. Estos datos han sido almacenados en un fichero Json. En ellos se contendrá la información de los archivos multimedia y dos links, uno de ellos hacia una imagen y otro de ellos hacia una lista M3u8.

Todo esto los conseguiremos mediante una petición al servidor por parte del cliente, que efectuara una la lectura del fichero Json correspondiente para su posterior mapeo a objetos Java y una operación similar en la parte del cliente


```
[
{"IdPelicula":1,
"Titulo":"El Francotirador",
"Genero":"Drama",
"Descripcion":
"El francotirador y SEAL de la Armada estadounidense Chris Kyle salva muchas vidas en los campos de batalla en Irak mientras se
"Portada":"http://192.168.56.1:8080/WebServicess/peliculas/El_Francotirador/El_Francotirador_portada.jpg",
"Link":"http://192.168.56.1:8080/WebServicess/peliculas/El_Francotirador/El_Francotirador.m3u8"},

{"IdPelicula":2,
"Titulo":"Origen",
"Genero":"Suspense",
"Descripcion":"Dom Cobb es un ladron con una extraÃ±a habilidad para entrar a los sueÃ±os de la gente y robarles los secretos d
"Director":"Christopher Nolan",
"Portada":"http://192.168.56.1:8080/WebServicess/peliculas/Origen/Origen_portada.jpg",
"Link":"http://192.168.56.1:8080/WebServicess/Origen/Origen.m3u8"},

{"IdPelicula":3,
"Titulo":"El Caballero Oscuro",
"Genero":"Accion",
"Descripcion":"Batman tiene que mantener el equilibrio entre el heroismo y el vigilantismo para pelear contra un vil criminal c
"Director":"Christopher Nolan",
"Portada":"http://192.168.56.1:8080/WebServicess/peliculas/El_caballero_oscuro/El_caballero_oscuro.jpg",
"Link":"http://192.168.56.1:8080/WebServicess/peliculas/El_caballero_oscuro/El_Caballero_Oscuro.m3u8"},
```


*Fichero Json ubicado en el servidor

Una vez cargado nuestro adaptador y pasado a nuestro elemento RecyclerView este, automáticamente iterara los datos y pasara al repintado de los CardViews con la información contenida. Para agilizar el rendimiento y evitar la saturación de procesamiento que conllevaría descargar las imágenes contenidas en los CardViews en nuestro dispositivo, hemos utilizado la librería Picasso. Esta librería nos permite almacenar en cache temporalmente nuestras imágenes sin necesidad de descargarlas con tan solo dándole la dirección del servidor donde están contenidas.

A continuación mostraremos el método que carga la información dentro de un CardView además de manejar el evento Onclick que nos llevara a la interfaz siguiente donde se mostrara con más detalle el contenido multimedia que vamos a visualizar.

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

    holder.tv_book_title.setText(mData.get(position).getTitulo());

    Picasso.with(mContext).load(mData.get(position).getPortada()).into(holder.img_book_thumbnail);

    holder.cardView.setOnClickListener((v) → {

            Intent intent = new Intent(mContext, MoviesActivity.class);

            intent.putExtra( name: "Titulo", mData.get(position).getTitulo());
            intent.putExtra( name: "Descripcion", mData.get(position).getDescripcion());
            intent.putExtra( name: "Portada", mData.get(position).getPortada());
            intent.putExtra( name: "Genero", mData.get(position).getGenero());
            intent.putExtra( name: "Link", mData.get(position).getLink());
            mContext.startActivity(intent);
    });
}
```

Barra lateral

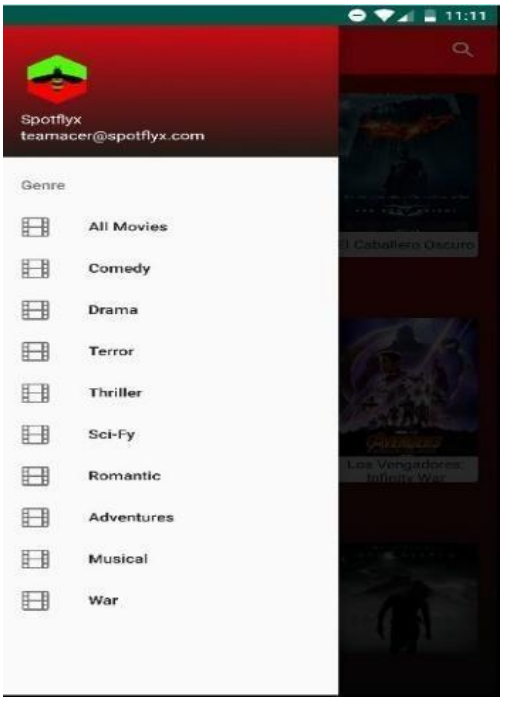
Android Studio nos da la facilidad para implementar activitys prediseñadas que nos permiten dar funcionalidad a nuestra aplicación. Nosotros para poder realizar el filtrado de categorías bien sea de música o películas nos hemos decantado por una de ellas, Navigation drawer. Esta activity que se sobrepone sobre nuestro RecyclerView al pulsar sobre un botón en la parte superior izquierda de nuestro ToolBar, nos permite realizar peticiones a nuestro servidor donde iteraremos por categoría nuestros elementos para posteriormente recargar nuestro RecyclerView con los elementos que estén catalogados como tal.

```
public boolean onNavigationItemSelected(MenuItem item) {

    int id = item.getItemId();

    if (id == R.id.nav_hip_hop) {
        String path ="http://192.168.56.1:8080/WebServicess/rest/MusicRestService/filtro/hip-hop";
        new DatosServidorMusica(path).execute();
        Toast toast = Toast.makeText(getApplicationContext(), text: "Hip-Hop", Toast.LENGTH_SHORT);
        toast.show();
    } else if (id == R.id.nav_rock) {
        String path ="http://192.168.56.1:8080/WebServicess/rest/MusicRestService/filtro/rock";
        new DatosServidorMusica(path).execute();
        Toast toast = Toast.makeText(getApplicationContext(), text: "Rock", Toast.LENGTH_SHORT);
        toast.show();
    } else if (id == R.id.nav_house) {
        String path ="http://192.168.56.1:8080/WebServicess/rest/MusicRestService/filtro/house";
        new DatosServidorMusica(path).execute();
        Toast toast = Toast.makeText(getApplicationContext(), text: "House", Toast.LENGTH_SHORT);
        toast.show();
```

*Petición cliente

```
@GET
@Path("filtro/{genero}")
@Produces(MediaType.APPLICATION_JSON)
public List<Album>  getAlbumByGender(@PathParam("genero") String genero)
 {
        List<Album> listOfAlbum=new ArrayList<Album>();
        List<Album> listOfAlbumByGender=new ArrayList<Album>();
        listOfAlbum=getAlbum();

      for (Album f: listOfAlbum)
      {

           if(f.getGenero().equalsIgnoreCase(genero))
           {

               listOfAlbumByGender.add(f) ;

          }

     }

  return listOfAlbumByGender;
 }
```

*Gestión de petición encargada de filtrar por género



*Barra lateral

Barra de Búsqueda

Este tipo de búsqueda es similar en muchas de las aplicaciones de conocemos hoy en día, consta simplemente de un botón que lanza un campo de búsqueda dentro del ToolBar. Una vez activado este se nos permitirá escribir dentro, y cada vez que introduzcamos un carácter se lanzara un evento que ejecutara una petición para comprobar si la cadena de caracteres concuerda con el título o director en caso de las películas, o con el autor y álbum en caso de que estuviéramos en la interfaz de catálogo de música.

Como veremos en la imagen siguiente en el caso de que no hubiera concordancia con ninguno de estos atributos, por defecto haría una consulta genérica donde se mostrarían todos elementos de nuestro catálogo.

```
public boolean onQueryTextSubmit(String query) {
    if (query!=null&& !query.isEmpty()) {
        String path = "http://192.168.56.1:8080/WebServicess/rest/FilmRestService/busqueda/" + query;
        new DatosServidor(path).execute();


    }else
    {
        String path = "http://192.168.56.1:8080/WebServicess/rest/FilmRestService/peliculas";
        new DatosServidor(path).execute();
    }
    return true;
}
```

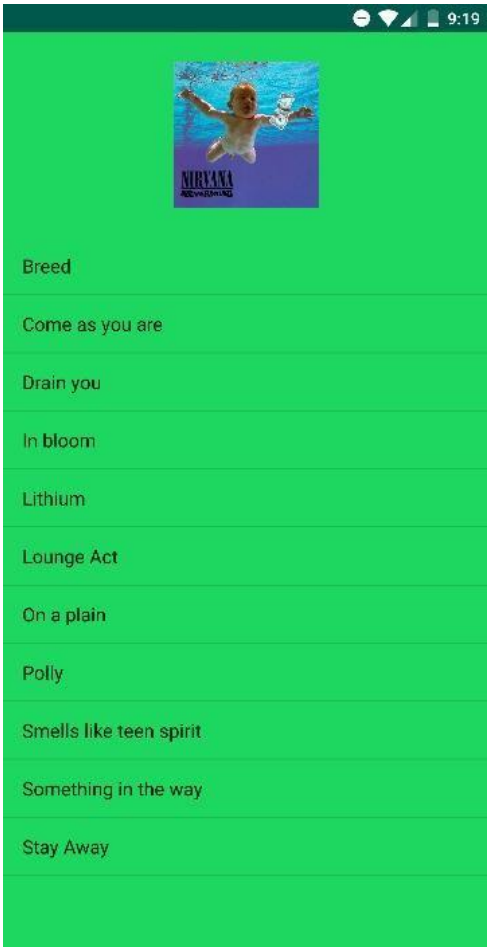
Vista preliminar películas

Como hemos visto en apartados anteriores es posible transferir datos de un intent o interfaz a otra. Para la creación de esta vista, que se lanzara al pulsar sobre la película que elijamos, hemos optado por esta opción ya que es la más eficiente para conseguir nuestros objetivos. Esta interfaz consiste en una previsualización de la película elegida en la que mostraremos más detalladamente cuál es su género, su director y una breve sinopsis de ella. Esta contendrá un botón que nos lanzara directamente a la visualización de la película.

Lista de canciones de un álbum

Esta interfaz es un simple Listview que mostrara las canciones contenidas dentro de un álbum y nos permitirá al clickear sobre ella para la reproducción de esta. El listado de estas canciones será pasado en formato lista a este intent, lo que quiere decir que la obtención de datos de esta lista se obtiene en la petición que se hace contra el servidor para cargar nuestro RecyclerView.

Para realizarlo hemos seguido procedimiento similar al de una película, a diferencia que nuestro fichero Json en vez de contener un link hacia una lista M3u8 contiene un array que para cada posición, vez contendrá un título de canción y una lista M3u8 con todas las particiones de nuestro archivo multimedia.



Reproductor multimedia

La clase encargada de reproducir el contenido multimedia no es más que un ítem BrightcoveExoPlayerVideoView sobre el que se carga un objeto Video. A este debemos indicarla que tipo formato de video es, además de pasarle el link de la lista M3u8 alojada en nuestro servidor para más tarde ejecutarlo y que se nos muestre en pantalla.

```
public class MusicPlayer extends BrightcovePlayer {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_music_player);
        Intent intent = getIntent();
        String link = intent.getExtras().getString( key: "link");

        final String Title = intent.getExtras().getString( key: "Titulo");
        final String Portada = intent.getExtras().getString( key: "Portada");
        final String Artista = intent.getExtras().getString( key: "Artista");

        brightcoveVideoView = (BrightcoveExoPlayerVideoView) findViewById(R.id.brightcove_video_view2);
       ImageView im= findViewById(R.id.imgPortada);
        TextView  tit=findViewById(R.id.txtTitulo);
        TextView ar =findViewById(R.id.txtArtista);
        tit.setText(Title);
        ar.setText(Artista);

        Picasso.with(MusicPlayer.this).load(Portada).into(im);
        Video video = Video.createVideo(link, DeliveryType.HLS);
        brightcoveVideoView.add(video);
        brightcoveVideoView.start();
        brightcoveVideoView.getEventEmitter().emit(SHOW_MEDIA_CONTROLS);
```

BIBLIOGRAFIA

Aplicaciones RESTful
https://www.youtube.com/watch?v=vJvD1-0N3AU

Brightcove:
https://support.brightcove.com/

TextInputLayout
https://www.youtube.com/watch?v=veOZTvAdzJ8

El futuro del streaming
https://blog.adigital.org/netflix-presenta-el-futuro-del-streaming-17befa1943b8

Jersey
https://anotherdayanotherbug.wordpress.com/2014/11/10/implementar-un-servicio-rest-con-jax-rs-jersey/

Mapeo de lista con Json
https://stackoverflow.com/questions/5554217/google-gson-deserialize-listclass-object-generic-type

FFMPEG
https://askubuntu.com/questions/999271/ffmpeg-command-for-mpeg-ts-encoding

AGRADECIMIENTOS

Nos quedan muchas fuentes por nombrar ya que al ser un proyecto con tecnologías e implementaciones prácticamente nuevas para nosotros hemos tenido que recopilar información de muchísimas fuentes que no hemos podido recuperar, pero desde aquí damos agradecimiento a todas esas personas que a través de internet comparten sus conocimientos de manera altruista y hacen de internet una fuente de conocimiento de acceso libre y gratuito.

También especial mención a Miguel Ángel Muñoz por su dedicación, involucración y hacernos de guía en este proyecto, que aunque para nosotros a veces ha sido duro nos llevamos una buena sensación por todo lo aprendido y las grandes satisfacciones que nos ha dado.

Queremos también dar las gracias al ayuntamiento de Las Rozas ya que gracias a su flexibilidad de horarios de apertura de sus bibliotecas nos ha facilitado el trabajo en equipo además de brindarnos buenos momentos.