UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Título del Proyecto:

Proyecto de Reproductor de listas multimedia

Autor:

Ruiz Fuentes, Jesús

Director:

Carbonell Frasquet, Vicente

TESINA PARA LA OBTENCIÓN DEL TÍTULO DE:

Máster en Desarrollo de Aplicaciones sobre Dispositivos Móviles

Septiembre, 2016

Índice

Índice de Esquemas

Índice de Figuras

1. Introducción

1.1. Descripción del problema

Estamos en un momento actual en el que casi todo el mundo posee un terminal "muy a mano" para comunicarse con la gente y también como elemento de entretenimiento y diversión. Y una forma muy frecuente de "gastar tiempo" es a través de la visión de vídeos, bien que estén en posesión propia (en local, a través de una tarjeta SD, la del propio móvil, o un usb insertado) o a través de la conexión a internet, siendo esta última muy frecuente.

Es muy común que haya problemas de conexión en ciertos momentos, por eso hay aplicaciones que se preparan para preparar ciertos datos sin necesidad de haber establecido una conexión previamente, aunque luego para la reproducción sea por streaming y sí necesite "tirar de internet" para poder ver los vídeos.

Hay aplicaciones que se dedican exclusivamente a reproducir los vídeos, y otra opción es usar un navegador tipo Google Chrome para la indagación y visión de los mismos tras un recorrido más largo hasta llegar hasta ellos.

Se muestran a continuación dos ejemplos de visión de vídeos a través de las dos maneras comentadas antes.

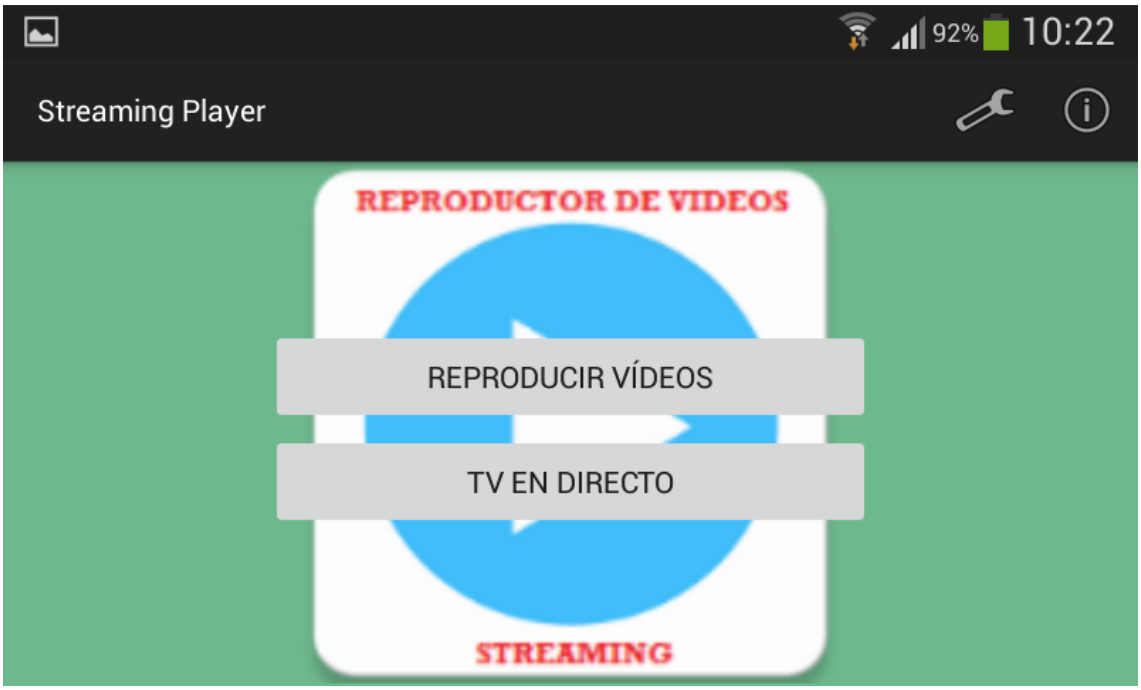


Figura nº 1: Aplicación para ver vídeos y canales de televisión.

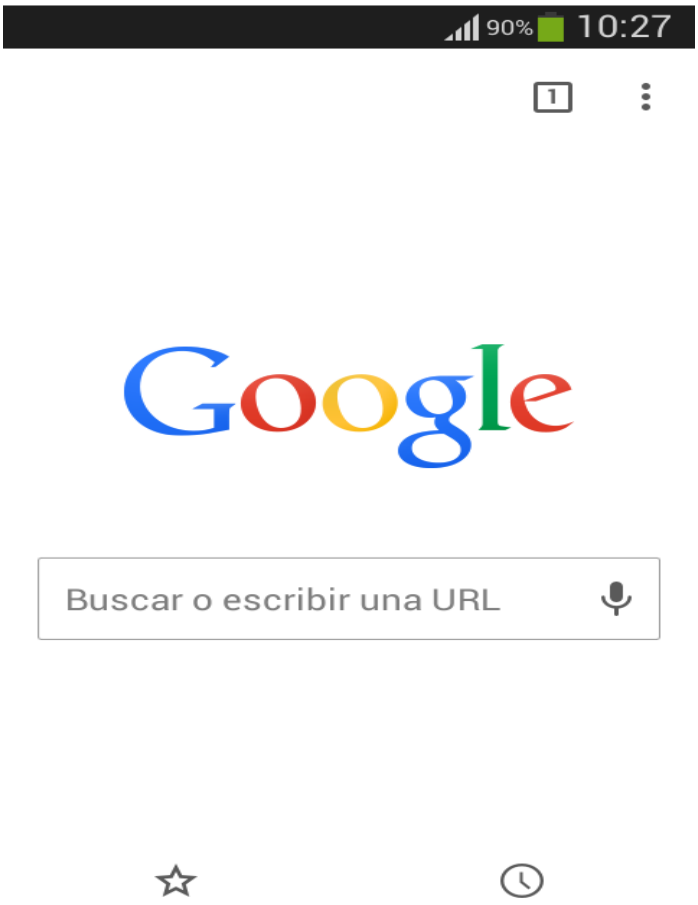


Figura nº 2: Vista de la web de Google para ir en búsqueda de vídeos.

1.2. Objetivos

El objetivo de este proyecto es crear la aplicación dedicada a la reproducción de Listas Multimedia para dispositivos Android. Existen dos partes claramente diferenciadas en este proyecto: la reproducción de vídeos y la visión de algunos canales de televisión.

En esta aplicación, se pretenderá que se pueda acceder a un fichero JSON donde se incluye los vídeos a visualizar. Este fichero podrá ser leído vía web a través de un servidor, o simplemente en local, ya sea el gusto del cliente o su posibilidad de tener o no conexión.

Con dicha lectura del archivo, se cargarán una serie de vídeos que se podrán ejecutar y ver.

Por otro lado, tenemos la opción de ver algunos canales de televisión en directo. Se visualiza una vista y por streaming se cargará aquella cadena televisiva seleccionada. Además, se podrá ver una web que al ser cargada se podrá interactuar como una página web y que su contenido se basa en una conocida dirección de internet que se encarga de ofrecer gran cantidad de canales en directo.

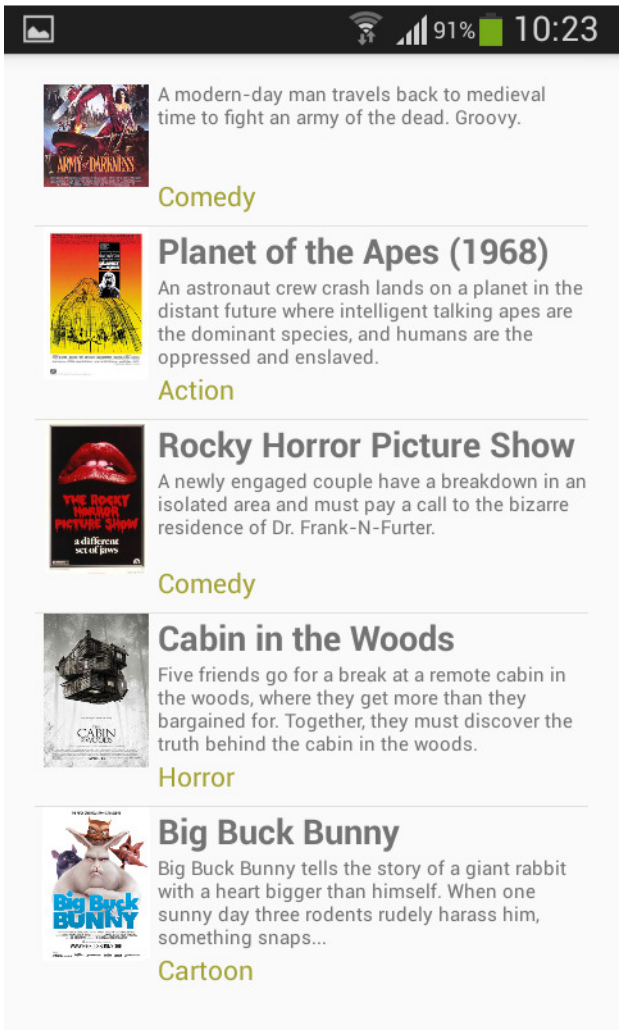


Figura nº 3: Lista de vídeos para reproducir.

1.3. Motivación

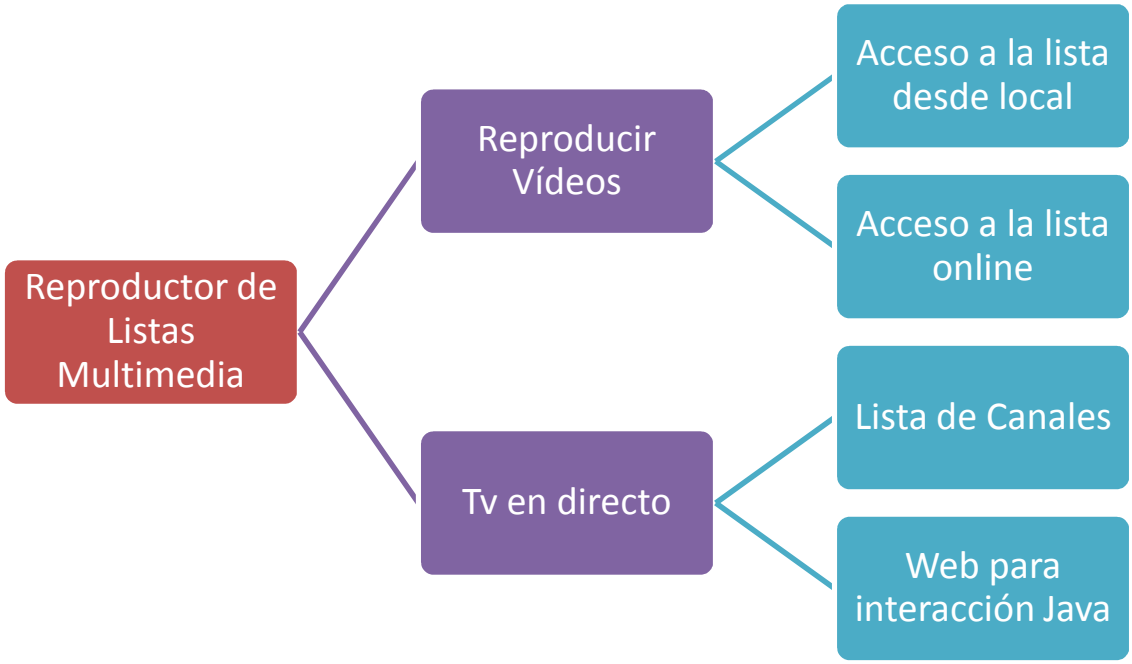
La motivación que me ha llevado al desarrollo de este proyecto de Reproductor de Listas Multimedia fue la temática del mismo a la hora de escoger un tipo de Proyecto para desarrollar la asignatura de Proyecto Fin de Máster. En concreto fue la idea de trabajar con un fichero JSON y su lectura, ya que como hemos visto los alumnos del máster, es frecuente tratar la lectura de estos archivos. En la asignatura de Programación en iOS, vimos un ejemplo claro de esta lectura de archivos, pero luego se amplió cuando vimos lo mismo en la parte del entorno Android.

Aunque a priori no estaba contemplado en el proyecto, y aunque no fuera necesario, mi tutor me comentó la idea de mejorar el presente proyecto con un reproductor en stream de canales de televisión. Y esta idea me gustó. Reproducir vídeos en streaming, canales de televisión en directo, me hizo sacarle más partida aún a la idea inicial de proyecto que teníamos mi tutor y yo.

2. Arquitectura de la aplicación

2.1. Esquema del diseño

El programa responde al siguiente esquema, en cuanto a su estructura.



Esquema nº 1: Esquema principal de la aplicación.

En cuanto a la decisión de por qué adoptar esta idea de programa fue la siguiente. Se partía de la primera de idea de elaborar unas listas de vídeos y cargarlas según la necesidad del cliente.

Conforme avanzábamos en el proyecto, cambiamos los objetivos del proyecto. Añadimos una sola lista, en la que habrá acceso local u online para contemplar ambas posibilidades. Esta idea no era básica en el proyecto, pero sí se optó por ella para ampliar la funcionalidad.

Poco más tarde, y viendo la posibilidad de hacer una aplicación más bonita desde el punto de vista funcional, optamos por meter canales de televisión que se reprodujeran vía streaming. Y poseer la opción de tener una web para interactuar con canales fue lo últimos que se ideó y se ejecutó.

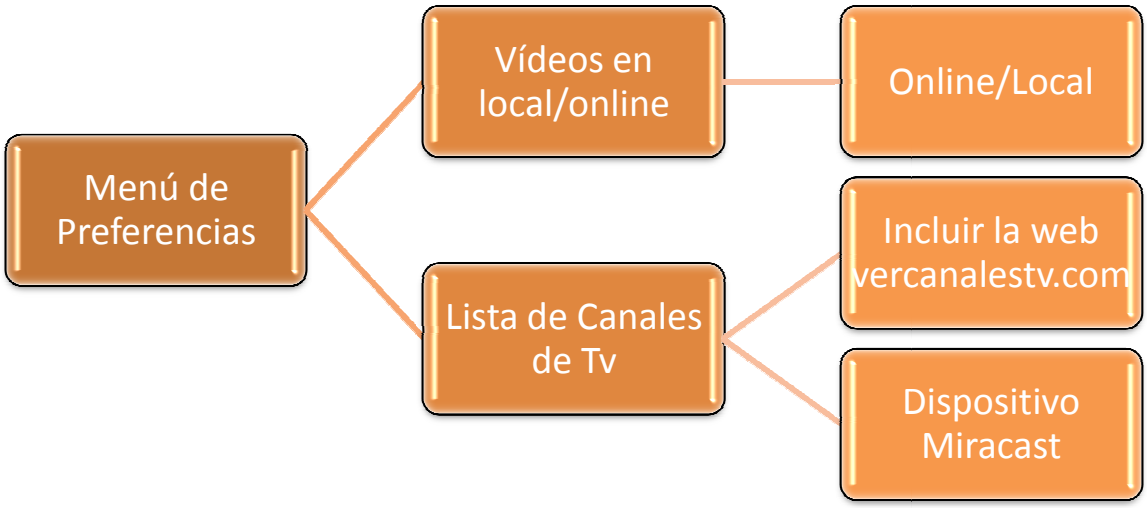
2.2. Modelo de datos

Clase principal del programa:

En la clase principal, es decir, aquella que abre la aplicación y nos permite manejarla, podemos ver tres botones tipo "Button". El primero de ellos nos permite acceder a Lista de Vídeos, el segundo a la Lista de Canales, y el tercero nos permite salir de la aplicación.

Si vemos en el apartado de Vistas, la vista correspondiente ofrece un menú con dos opciones. Un botón de "Acerca de", donde aparece un cuadro de texto en el que explica el por qué la existencia de esta aplicación. En otro botón, manejamos las Preferencias, que serán claves para el manejo del programa. Aquí controlaremos perfectamente si queremos acceder a lista de Vídeos a través de forma local (fichero JSON que existe en los ficheros del programa) o a través de un servidor web.

Como hemos comentado anteriormente en el último párrafo, existe un botón para manejar las preferencias. Pues otra opción corresponde al apartado de Lista de Canales de Televisión. En ella podemos escoger si lo que queremos es tener acceso o no a una web en la que podemos interaccionar y cargar alguno de los canales más comunes a través de un WebView. Por último, podemos ejecutar un dispositivo Miracast, por si queremos cargar en dos pantallas la aplicación, como puede ser una televisión.



Esquema nº 2: Esquema del Menú de Preferencias del programa.

Para comprobar la conexión a internet utilizamos el siguiente código:

```
// Función para comprobar la conexión
public static boolean compruebaConexion(Context context) {
    boolean connected = false;
    ConnectivityManager connec = (ConnectivityManager)
context.getSystemService(Context.CONNECTIVITY_SERVICE);
    // Recupera todas las redes (tanto móviles como wifi)
    NetworkInfo[] redes = connec.getAllNetworkInfo();
    for (int i = 0; i < redes.length; i++) {
        // Si alguna red tiene conexión, se devuelve true
        if (redes[i].getState() == NetworkInfo.State.CONNECTED) {
            connected = true;
        }
    }
    return connected;
}
```

Además, para cada botón se analizará si hay conexión y las preferencias, para según así, cargar la forma correspondiente. Como por ejemplo pasa con el botón que carga los vídeos.

```
public void ListaVideos(View view) {

    if (compruebaConexion(this)==false)
        if (prefs.getBoolean("videos", true)) {
        Toast.makeText(this, "No hay acceso a internet",
Toast.LENGTH_SHORT).show();
        }
        else {
            Toast.makeText(this, "Acceso local a la lista",
Toast.LENGTH_SHORT).show();
            Intent i = new Intent(this, ListaVideosLocal.class);
            startActivity(i);
        }
    else {
        if (prefs.getBoolean("videos", true)) {

            Toast.makeText(this, "Acceso online a la lista",
Toast.LENGTH_SHORT).show();
            Intent i = new Intent(this, ListaVideosOnline.class);
            startActivity(i);
        }
        else {
            Toast.makeText(this, "Acceso local a la lista",
Toast.LENGTH_SHORT).show();
            Intent i = new Intent(this, ListaVideosLocal.class);
            startActivity(i);
        }
    }
}
```

Clase WebRequest:

En la clase WebRequest, se hacen servicios de llamada HTTP para acceder al servicio web que usaremos para cargar la lista de vídeos Online.

Clase ListaVideosLocal:

En esta clase, usamos un fichero JSON llamado listavideos.json que tiene la siguiente estructura:

```
{"user":
[
{ "id": "0",
"title": "Gremlins",
"description": "A boy inadvertantly breaks 3 important rules
concerning his new pet and unleashes a horde of malevolently
mischievous monsters on a small town.",
"videoUrl": "http://videos.movie-list.com/classics/FLV/gremlins.mp4",
"category": "Comedy",
"imageUrl": "http://www.movie-
list.com/img/posters/big/zoom/gremlins.jpg" },

...
```

JSON (JavaScript Object Notation) es un formato para el intercambios de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. JSON nació como una alternativa a XML, el fácil uso en Javascript ha generado un gran número de seguidores de esta alternativa. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído porcualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

Este fichero tiene tiene como “llaves” para acceder a los datos que forman la lista de vídeos. Está formado por el “id” que nos sirve de identificador. Además tenemos el “title” que será el título del vídeo en cuestión. “description” nos comenta el contenido resumido del vídeo. “videoUrl” se encarga de tener la url para cargar el vídeo, y que básicamente son en formato mp4. “category” será aquella que dirá que tipo de temática tiene el vídeo. “imageUrl” dará información de dónde cargar la imagen. Pero como se comentará más adelante está en modo de información, no llegará a usarse.

Se lee el fichero, sin usar hacer llamada a la clase WebRequest, y cargamos las carátulas de los vídeos que son ficheros *jpg* en la carpeta de Drawable del proyecto. La lista se carga con la lectura del fichero comentado.

Lo peculiar de esta clase es el método para leer el fichero JSON, que será distinto al que usaremos cuando carguemos el proyecto online. Dicho código se expone a continuación.

```
public static String loadJSONFromResource(Context context, int
resource ){
    if( resource <= 0 ) return null;
    String json = null;
    InputStream is = context.getResources().openRawResource( resource
);
    try { if( is != null ) { int size = is.available();
        byte[] buffer = new byte[size];
        is.read(buffer);
        json = new String(buffer, "UTF-8");
    }
    }
    catch( IOException e ) { } finally { try { if( is != null )
is.close(); }
    catch( IOException e ) {} } return json;
}
```

Clase ListaVideosOnline:

En esta clase, usamos un fichero JSON llamado listavideos.json, pero aunque es el mismo que hay en local (comentado anteriormente su funcionamiento), se usa la clase WebRequest para hacer peticiones HTTP y obtener los datos de un servidor web, y para ello definimos lo siguiente al principio de la clase:

```
// URL de los videos JSON
private static String url =
"http://reproductorlistaspfmjrf.pe.hu/listavideos.json";
```

En esta clase usamos los ficheros *jpg* de la carpeta Drawable, y no hacer falta crear un nuevo hilo para descargar las imágenes del fichero *json*, que aunque no las necesitamos, no está mal tener esa información en el fichero. Además, así aprovechamos los recursos que ya de por sí tenemos y no tenemos que descargar las imágenes de nuevo.

El código que se ejecuta “de fondo” para interactuar con el JSON con peticiones HTTP, clave para obtener la respuesta se detalla en el siguiente fragmento.

```
@Override
protected Void doInBackground(Void... arg0) {

    // Manejador de la clase
    WebRequest webreq = new WebRequest();

    // Haciendo la llamada a la url y consiguiendo respuesta
    String jsonStr = webreq.makeWebServiceCall(url, WebRequest.GET);

    Log.d("Response: ", "> " + jsonStr);

    videosList = ParseJSON(jsonStr);

    return null;
}
```

Clases Videoplayer:

En estas clases, usamos un método que declaramos en ListaVideosLocal o ListaVideosOnline denominado "setEnlaceLocal/Online()", y así usa un VideoView para reproducir el vídeo. El código básico para ejecutar los vídeos es el que se presenta a continuación.

```
//Declaración de variables
private VideoView reproductor;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_videoplayer);
    reproductor =(VideoView)findViewById(R.id.video);
    //Para reproducir archivos en streaming, como vídeos de Youtube:
https://www.youtube.com/watch?v=iF1aSnV2U7M

//reproductor.setVideoURI(Uri.parse("http://m.youtube.com/add_favorite
?v=iF1aSnV2U7M"));
    //Formatos mp4 y 3gp: http://techslides.com/demos/sample-
videos/small.mp4

//reproductor.setVideoURI(Uri.parse("http://personales.gan.upv.es/~jto
mas/video.3gp"));
    //Para reproducir archivos almacenados en la memoria SDCard:
    //reproductor.setVideoPath("/mnt/sdcard/Ejemplo.mp4");

reproductor.setVideoURI(Uri.parse(ListaVideosOnline.setEnlaceOnline())
);
    reproductor.setMediaController(new MediaController(this));
    reproductor.start();
    reproductor.requestFocus();
}
```

Clases ListaCanalesTv:

En esta clase lo primero que hacemos es declarar la lista de canales; que no se carga con lectura JSON como se hace en el apartado de vídeos, sino que se usa una cadena de String[] cargada en la clase y por tanto privada.

```
private String [] listaCanalesGlobal = new String [] {"Canal 16
Tv","Canal Oasis Tv","Canal Magazine","Canal 26 Tv","Canal
Todonoticias TN","Canal Vale Tv","Canal HCH Tv","Canal Rural","Canal
13 Digital","Alfa Tv","vercanalestv.com","Eurosport"};
```

Además, los videos serán en streaming como el primero que se muestra en el siguiente trozo de código. Lo siguiente es una dirección web para ver los canales y por último, la lectura de un fichero html que se dispone en los recursos del proyecto para ver una cadena en concreto.

```
case 9:
    LINK = "rtsp://streamer1.streamhost.org:1935/salive/GMIalfah";
    break; //Funciona (Alfa Tv)
case 10:
    LINK = "http://www.vercanalestv.com/category/canales-de-espana/";
    break; // (Web vercanalestv.com)
case 11:
    LINK = "file:///android_asset/EjemploFicheroWeb.html";
    break; //  (Eurosport)
```

Según si cargamos un tipo de elemento de la lista, cargaremos una clase u otra para ejecutar los caneles. En el primero de ellos, una cadena de tv stream tipo rtsp utilizamos la clase Lista Streamingtv_simple, que reproduce un VideoView como en el caso anterior. Si se selecciona "Dispositivo Miracast", utilizaremos la clase Streamingtv_miracast. Dicha clase se explicará en Anexos.

Si se ejecuta el elemento de la lista que una url de tipo *http://*, utilizaremos la clase StreamingWebView que usuaremos un WebView, y utilizaremos el siguiente código para interaccionar con Java y utilizar el mismo navegador y no uno que por defecto proponga el mismo terminal que se esté usando.

```
// Asociamos
mWebView = (WebView) findViewById(R.id.webview);

mWebView.getSettings().setSavePassword(false);
mWebView.getSettings().setSaveFormData(false);
mWebView.getSettings().setJavaScriptEnabled(true);
mWebView.getSettings().setBuiltInZoomControls(true);
mWebView.setWebChromeClient(new MyWebChromeClient());
mWebView.addJavascriptInterface(new DemoJavaScriptInterface(),
"demo");
```

2.3. Vistas

Esquema nº 3: Esquema principal de la aplicación basado en imágenes.

ETAPA 1

La vista que abre el programa es la asociada a la clase principal.

Figura nº 4: Vista principal del programa.

En esta última existen dos botones para ver los vídeos/canales de tv y otro para salir de la aplicación. Además, hay una barra de menú de opciones para poder interactuar con la aplicación según necesidades de conexión o no, o de simultaneidad de pantallas.

Las vistas asociadas a los botones del menú son las siguientes, en las que se puede ver el menú de preferencias y el panel de información de la aplicación.

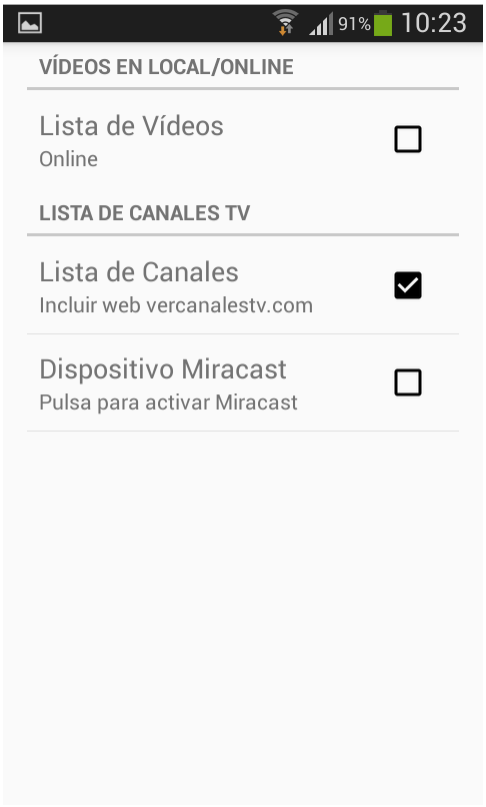


Figura nº 5: Vista asociada al menú de Preferencias.

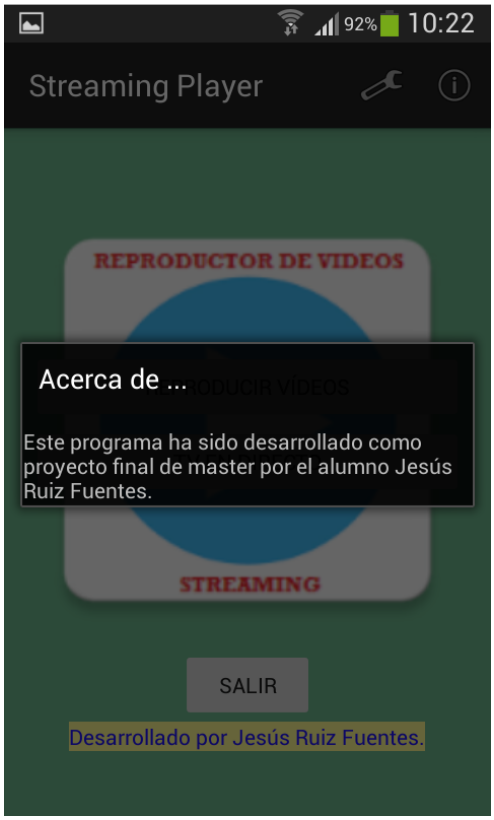


Figura nº 6: Vista asociada al Acerda de (información de la app).

ETAPA 2

La vista asociada al botón de Vídeos, nos lleva a una lista con una serie de ellos.

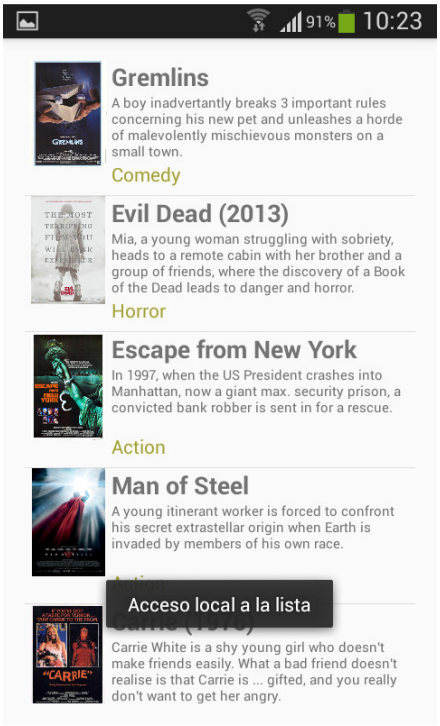


Figura nº 7: Vista asociada a la lista de vídeos para su reproducción.

ETAPA 3

Si reproducimos un vídeo, la pantalla se pone obligatoriamente horizontal y se puede ver así.

Figura nº 8: Vista del comienzo de la reproducción de un vídeo.

ETAPA 4

En la pantalla asociada al botón de ver canales de tv, podemos ver una lista de vídeos.

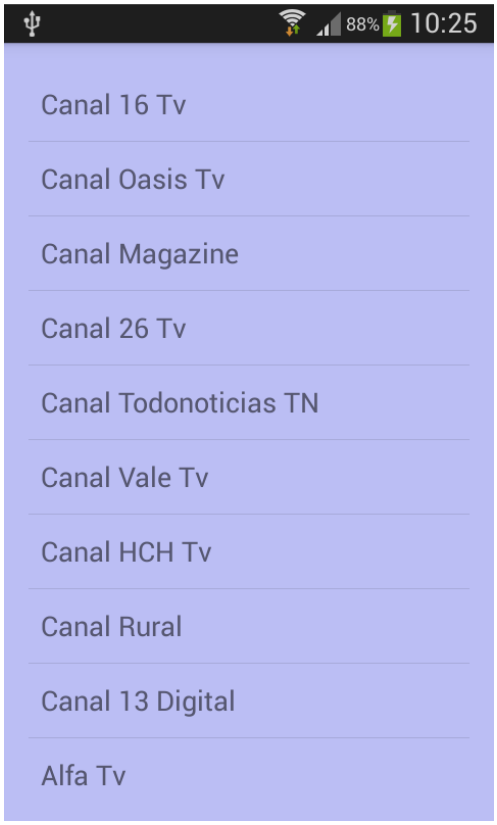


Figura nº 9: Lista de Canales de televisión.

ETAPA 5

Al ejecutar un canal de televisión, vemos perfectamente la cadena televisiva.

Figura nº 10: Vista en directo de un programa de televisión.

Figura nº 11: Otra vista en directo de un programa de televisión.

ETAPA 6

Si por el contrario escogemos la web que nos lleva a una serie de canales, podemos ver una web en la que podemos interactuar.

Figura nº 12: Web para ver canales de televisión.

ETAPA 7

Nos ofrece la posibilidad de ejecutar el programa de canales televisión con pantallas simultáneas (Dispositivo Miracast).

Figura nº 13: Ejecución del programa para simular pantallas secundarias.

3. Capítulos adicionales

3.1. Características Gradle (Module:app)

Las características del programa son:

- CompileSdkVersion: 23
- BuildToolsVersion: 23.0.2
- MinSdkVersion: 9
- TargetSdkVersion: 23
- Librerías:
    - Appcompat: 23.3.0
    - Design: 23.3.0

3.2. Pantallas simultáneas

Si anteriormente en Esquema de Diseño comentamos el por qué de las decisiones, la opción de pantallas Simultáneas, con la creación de un dispositivo Miracast, fue la última y que venía de la idea de qué mejor manera que ver vídeos además de en tu terminal, en una televisión.

La conexión Miracast está basada en una conexión Wi-Fi Direct, es decir, una conexión *peer to peer*.

Hay cuatro modos de conexión Miracast:

• Conexión directa de la fuente al visualizador. No existe un punto de acceso.

• Conexión directa de la fuente al visualizador. Existe un punto de acceso pero ni fuente ni visualizado están conectados a él.

• Fuente con acceso al punto de acceso y conexión directa al visualizador.

• La fuente y el visualizador conectados cada uno al punto de acceso. El envío de la información al visualizador se puede realizar a través del punto de acceso o directamente entre los dispositivos.

Con una conexión Miracast, puedes habilitar la conectividad entre dispositivos sin la infraestructura de un punto de acceso Wi-Fi. Puedes conectar el visualizador a través de un adaptador mientras está conectado a un punto de acceso. Este modo es muy conveniente para ver vídeos online. Si el usuario dispone de una Smart TV que admite Miracast, la televisión, el punto de acceso y el *smartphone* pueden conectarse entre sí.

De acuerdo con el estándar Miracast, la interacción entre los dispositivos fuente y el visualizador sigue el siguiente diagrama:

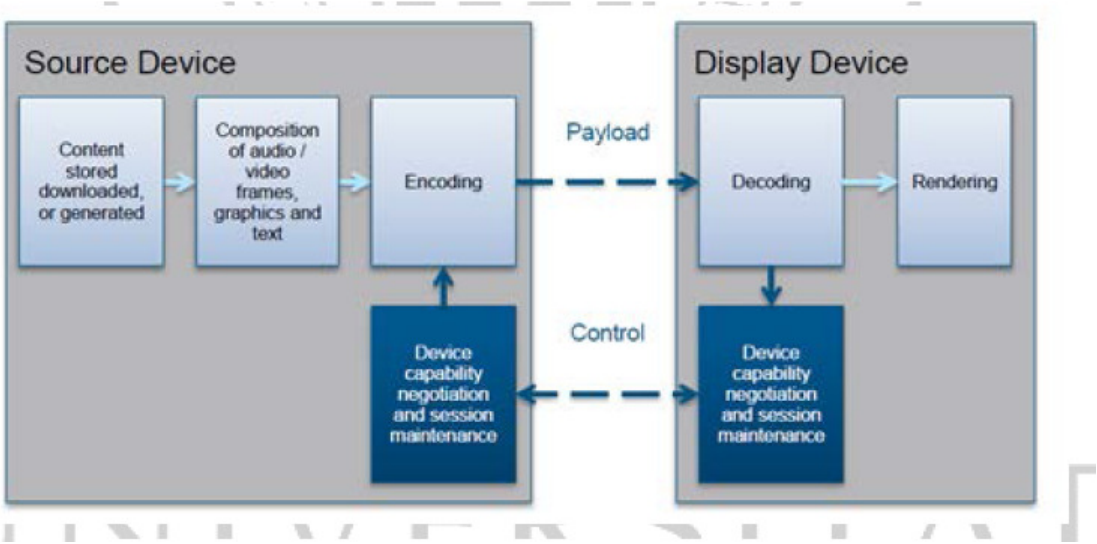


Figura nº 14: Diagrama de interacción de dispositivos Miracast.

Los dispositivos fuentes y visualizadores se descubren unos a otros mediante sus capacidades Miracast, antes de establecer la conexión. La conexión está basada en Wi-Fi Direct. La conexión usa el protocolo TCP. Los dispositivos fuente transfieren el contenido a mostrar a los visualizadores mediante vídeo con formato MPEG2-TS basado en el protocolo UDP.

Así pues, en nuestro proyecto hemos añadido esa posibilidad. Si en el menú de preferencias, se selecciona “Dispositivo Miracast”, para ver los canales de televisión, aparacerá un menú de dos pantallas y según la que se escoja se podrá ver en ambas pantallas, en el terminal solo o en el dispositivo Miracast conectado.

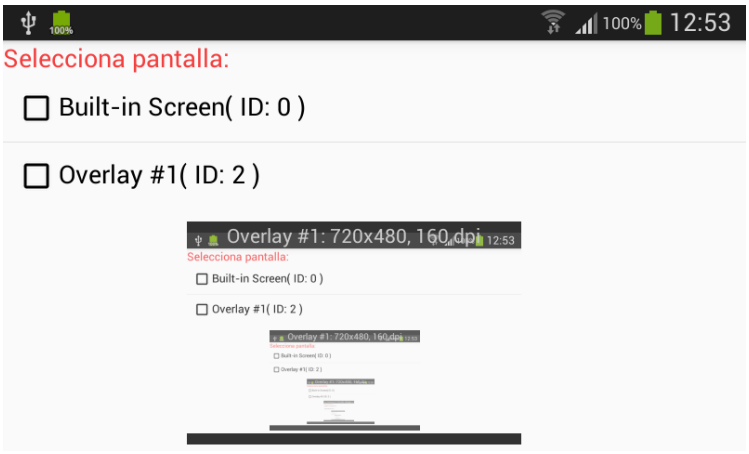


Figura nº 15: Las dos pantallas para ver los canales de televisión.

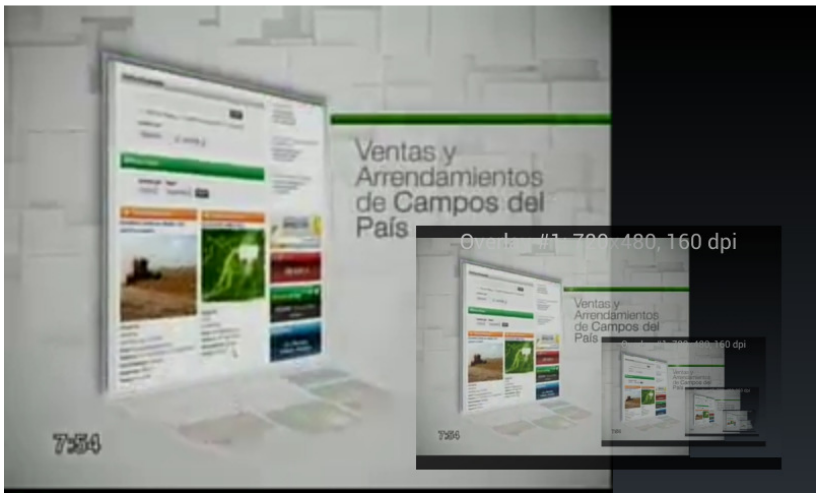


Figura nº 16: Visión solo de la pantalla de la terminal.

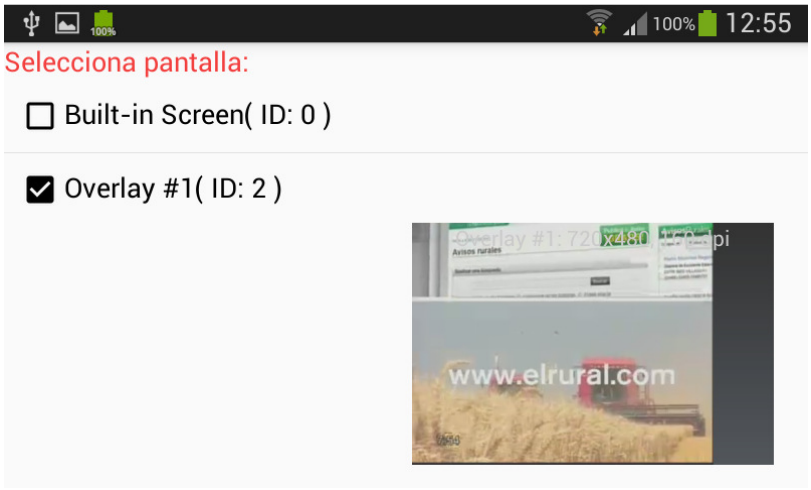


Figura nº 17: Visión de la cadena solo en el dispositivo conectado (emulado en terminal).

Figura nº 18: Visión de la cadena en ambas pantallas.

Por último, comentar que en anexo de este proyecto, se encuentran las dos clases que se utilizan para llevar a cabo esta tarea de “Simultaneidad de Pantallas”.

4. Conclusiones

Cierto es que los objetivos marcados a priori se han cumplido con creces, ya que por ejemplo, el tema de la televisión y del Miracast no estaban diseñados en un principio.

Fue con el avance en el proyecto cuando vimos que podíamos hacer una aplicación más útil y más elegante con estos dos elementos.

Además se han tenido en cuenta muchos puntos estudiados en las asignaturas del Máster, por lo que en cuanto a código, está bastante completo. Desde temas dados en la asignatura de Fundamentos de Android hasta otra de la asignatura que cerraba el Diploma de Android (Dispositivos Wearable, Visión Artificial. Google Glass y Android TV) han sido "tocados" en este proyecto.

Quizás, la "espinita" que me quede en el proyecto, es haber indagado más en la interacción con el cliente. Me refiero a que el cliente hubiera desde su terminal haber añadido más listas al programa para poder completarlo.

Aún así, nos quedamos muy satisfechos por el diseño y resultado del proyecto.

5. Anexos

5.1. Clases asociadas a Miracast

Clases Stremingtv_miracast:

```
public class Streamingtv_miracast extends Activity {

    private DisplayManager mDisplayManager;
    private DisplayListAdapter mDisplayListAdapter;
    private ListView mListView;

    private final SparseArray<RemotePresentation> mActivePresentations
= new SparseArray<RemotePresentation>();

    private final DisplayManager.DisplayListener mDisplayListener =
new DisplayManager.DisplayListener() {
        @Override public void onDisplayAdded(int displayId) {
            mDisplayListAdapter.updateContents(); }
        @Override public void onDisplayChanged(int displayId) {
            mDisplayListAdapter.updateContents(); }
        @Override public void onDisplayRemoved(int displayId) {
            mDisplayListAdapter.updateContents(); }
    };

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.miracast_canalestv);
        mDisplayManager = (DisplayManager)
getSystemService(Context.DISPLAY_SERVICE);
        mDisplayListAdapter = new DisplayListAdapter(this);
        mListView = (ListView)findViewById(R.id.display_list);
        mListView.setAdapter(mDisplayListAdapter); }

    protected void onResume() {
        super.onResume();
        mDisplayListAdapter.updateContents();
        mDisplayManager.registerDisplayListener(mDisplayListener,
null);
    }
    private void showPresentation(Display display) {
        RemotePresentation presentation = new RemotePresentation(this,
display);
        mActivePresentations.put(display.getDisplayId(),
presentation);
        presentation.show();
    }

    private void hidePresentation(Display display) {
        final int displayId = display.getDisplayId();
        RemotePresentation presentation =
mActivePresentations.get(displayId);
        if (presentation == null) { return;
        } presentation.dismiss();
mActivePresentations.delete(displayId);
```

```
    }

    private final class DisplayListAdapter extends
ArrayAdapter<Display> {
        final Context mContext;
        private CompoundButton.OnCheckedChangeListener
mCheckedRemoteDisplay = new CompoundButton.OnCheckedChangeListener() {
            @Override public void onCheckedChanged(CompoundButton
view, boolean isChecked) {
                synchronized (mCheckedRemoteDisplay) {
                    final Display display = (Display)view.getTag();
                    if (isChecked) { showPresentation(display); }
                    else { hidePresentation(display); } } } };

        public DisplayListAdapter(Context context) {
            super(context, R.layout.elementos_miracast_canalestv);
            mContext = context; }

        @Override public View getView(int position, View convertView,
ViewGroup parent) {
            final View v;
            if (convertView == null) {
                v = ((Activity) mContext).getLayoutInflater()
.inflate(R.layout.elementos_miracast_canalestv, null);
            } else { v = convertView; }
            final Display display = getItem(position); TextView tv =
(TextView)v.findViewById(R.id.display_id);
            tv.setText(display.getName() + "( ID: " +
display.getDisplayId() + " )");
            CheckBox cb = (CheckBox)v.findViewById(R.id.display_cb);
            cb.setTag(display);
cb.setOnCheckedChangeListener(mCheckedRemoteDisplay);
            return v; }

        public void updateContents() {
            clear();
            Display[] displays = mDisplayManager.getDisplays();
            addAll(displays); }
    }

    private final class RemotePresentation extends Presentation {
        public RemotePresentation(Context context, Display display) {
            super(context, display); }
        @Override protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.streamingtv);

            // LINK en formato rtsp, como
rtsp://streamer1.streamhost.org:1935/salive/GMIalfah

            VideoView videoView = (VideoView)
findViewById(R.id.video);
            MediaController mc = new
MediaController(Streamingtv_miracast.this);
            mc.setAnchorView(videoView);
            mc.setMediaPlayer(videoView);
            // Cargamos los enlaces provenientes de ListaCanalesTv
            Uri video = Uri.parse(ListaCanalesTv.setEnlaceTv());
            videoView.setMediaController(mc);
```

```
            videoView.setVideoURI(video);
            videoView.start();
        }
    }
```

Clases Opcion_miracast_canalestv:

```
public class Opcion_miracast_canalestv extends Activity {

    private DisplayManager mDisplayManager;
    private DisplayListAdapter mDisplayListAdapter;
    private ListView mListView;

    private final SparseArray<RemotePresentation> mActivePresentations
= new SparseArray<RemotePresentation>();

    private final DisplayManager.DisplayListener mDisplayListener =
new DisplayManager.DisplayListener() {
        @Override public void onDisplayAdded(int displayId) {
            mDisplayListAdapter.updateContents(); }
        @Override public void onDisplayChanged(int displayId) {
            mDisplayListAdapter.updateContents(); }
        @Override public void onDisplayRemoved(int displayId) {
            mDisplayListAdapter.updateContents(); }
    };

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.miracast_canalestv);
        mDisplayManager = (DisplayManager)
getSystemService(Context.DISPLAY_SERVICE);
        mDisplayListAdapter = new DisplayListAdapter(this);
        mListView = (ListView)findViewById(R.id.display_list);
        mListView.setAdapter(mDisplayListAdapter); }

    protected void onResume() {
        super.onResume();
        mDisplayListAdapter.updateContents();
        mDisplayManager.registerDisplayListener(mDisplayListener,
null);
    }
    private void showPresentation(Display display) {
        RemotePresentation presentation = new RemotePresentation(this,
display);
        mActivePresentations.put(display.getDisplayId(),
presentation);
        presentation.show();
    }

    private void hidePresentation(Display display) {
        final int displayId = display.getDisplayId();
        RemotePresentation presentation =
mActivePresentations.get(displayId);
        if (presentation == null) { return;
        } presentation.dismiss();
mActivePresentations.delete(displayId);
    }
```

```
    private final class DisplayListAdapter extends
ArrayAdapter<Display> {
        final Context mContext;
        private CompoundButton.OnCheckedChangeListener
mCheckedRemoteDisplay = new CompoundButton.OnCheckedChangeListener() {
            @Override public void onCheckedChanged(CompoundButton
view, boolean isChecked) {
                synchronized (mCheckedRemoteDisplay) {
                    final Display display = (Display)view.getTag();
                    if (isChecked) { showPresentation(display); }
                    else { hidePresentation(display); } } } };

        public DisplayListAdapter(Context context) {
            super(context, R.layout.elementos_miracast_canalestv);
            mContext = context; }

        @Override public View getView(int position, View convertView,
ViewGroup parent) {
            final View v;
            if (convertView == null) {
                v = ((Activity) mContext).getLayoutInflater()
.inflate(R.layout.elementos_miracast_canalestv, null);
            } else { v = convertView; }
            final Display display = getItem(position); TextView tv =
(TextView)v.findViewById(R.id.display_id);
            tv.setText(display.getName() + "( ID: " +
display.getDisplayId() + " )");
            CheckBox cb = (CheckBox)v.findViewById(R.id.display_cb);
            cb.setTag(display);
cb.setOnCheckedChangeListener(mCheckedRemoteDisplay);
            return v; }

        public void updateContents() {
            clear();
            Display[] displays = mDisplayManager.getDisplays();
            addAll(displays); }
    }

    private final class RemotePresentation extends Presentation {
        public RemotePresentation(Context context, Display display) {
            super(context, display); }
        @Override protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            //setContentView(R.layout.listacanales);
            IniciarActividad();
            }
    }

    private void IniciarActividad () {
        Intent i = new Intent(this, ListaCanalesTv.class);
        startActivity(i);
    };

}
```

5.2. Bibliografía

Los libros o referencias bibliográficas necesarias para realizar este proyecto han sido:

- El Gran libro de Android.

- El Gran libro de Android Avanzado.

- Dispositivos Wearable, Visión Artificial, Google Glass y Android TV.

- https://geekytheory.com/json-i-que-es-y-para-que-sirve-json/