Universidad de Jaén

Escuela Politécnica Superior de Jaén

Prototipo de una herramienta para la reproducción de contenido multimedia

Autor: Juan Francisco Escudero Toribio

Grado: Ingeniería Informática

Director: Francisco Daniel Pérez Cano
Departamento del director: Departamento de informática

Subdirector: Juan José Jiménez Delgado
Departamento del subdirector: Departamento de informática

Fecha: 06/09/2024

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Francisco Daniel Pérez Cano y Juan José Jiménez Delgado, tutores del Proyecto Fin de Carrera titulado: Prototipo de una herramienta para la reproducción de contenido multimedia, que presenta Juan Francisco Escudero Toribio, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2024

El alumno: Los tutores:

Juan Francisco Escudero Toribio

Francisco Daniel Pérez Cano y Juan José Jiménez Delgado

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. Introducción

Este trabajo se centra en el desarrollo de una aplicación para dispositivos Android que permita a los usuarios disfrutar de una reproducción de canales en directo. A lo largo del documento, se detallarán aspectos técnicos, diseño e implementación necesaria para llevar a cabo el proyecto, así como la metodología usada para conseguir los fines requeridos en el mismo.

1.1 Motivación

Este proyecto surge debido al incremento masivo que se está llevando en la actualidad en el uso de dispositivos móviles o tablets. Además, gran cantidad de usuarios están dejando de usar televisiones tradicionales, siendo sustituidos los mismos por dispositivos móviles, ya que características como su tamaño mucho más manejable o coste mucho más reducido hacen de ellos unos productos muy recomendables para el consumo de contenido audiovisual por parte de los usuarios.

Otro de los muchos factores que hace que esté sucediendo este cambio tan drástico en el uso de estos aparatos es la comodidad que disfruta el usuario que puede acceder a contenido en directo usando su móvil o tablet sin estar limitado a estar en la misma ubicación que su televisor. Esto permite que el usuario pueda visualizar el contenido en el momento que desee aunque no esté en su hogar.

Aprovechando esta situación, surge la necesidad de aprovechar el auge de este tipo de dispositivos para desarrollar servicios que permitan de forma eficiente y personalizada conseguir el entretenimiento de los usuarios.

La elección de dispositivos Android [1] se debe a que es el sistema operativo móvil más utilizado a nivel mundial. Al optar por esta plataforma, se garantiza la compatibilidad con una amplia variedad de dispositivos.

Por otra parte, la plataforma Android cuenta con una comunidad de desarrolladores activa y una amplia gama de herramientas de desarrollo. Esto permitirá un ciclo de desarrollo ágil y la incorporación de nuevas tendencias, garantizando que la aplicación esté al día en los últimos avances tecnológicos.

Por todo esto, se ha pensado en la realización de una aplicación de reproducción de canales en directo que simule el comportamiento de una televisión, haciendo que el usuario use la misma de una forma fluida, cómoda e intuitiva.

1.2 Objetivos

El principal objetivo del proyecto será crear una primera instancia de aplicación destinada a dispositivos móviles y tablets Android que permita al cliente visualizar contenido en directo a través de canales M3U8 [2] ofreciendo una interfaz sencilla [3] e intuitiva que permita el control total de la aplicación por parte del usuario.

Además, se pretenderá que la interfaz sea usable por cualquier usuario independientemente de su edad. Por otra parte, se tendrá en cuenta problemas de visión como daltonismo [4] o de audición, por lo que se implementarán posibilidad de usar subtítulos. Esta interfaz tendrá controles como cambiar el volumen, mostrar la información de los programas de cada canal, cambiar el canal, añadir pistas distintas de subtítulos o de audio, entre otras.

Por otra parte se desarrollarán funcionalidades que garanticen la retransmisión de contenido en directo de forma eficiente, asegurando una transmisión estable y fluida la cual haga que la experiencia por parte del usuario sea lo mejor posible.

Para finalizar, se hará una prueba exhaustiva en dispositivos de distintos modelos para garantizar la funcionalidad de la aplicación en todos ellos.

Al cumplir con estos objetivos, se espera crear una aplicación robusta y eficiente que satisfaga las necesidades de los usuarios en cuanto a la visualización de contenido televisivo en dispositivos móviles, contribuyendo así al aprovechamiento del auge de estos dispositivos para el entretenimiento personalizado y accesible.

1.3 Metodología Scrum

La metodología de desarrollo de software que va a ser usada para la realización de la aplicación del reproductor será Scrum [5].

SCRUM es un marco de trabajo ágil utilizado comúnmente para el diseño, desarrollo y gestión de proyectos software con cierto nivel de complejidad. SCRUM se centra en la entrega incremental de productos. Esto aumentará la eficiencia del

equipo de desarrollo, ya que permitirá desde etapas tempranas del proyecto tener una retroalimentación del cliente, lo cual permitirá corregir fallos y hacer cambios según las peticiones del cliente.

Viendo esto, comprobamos que SCRUM nos va a ofrecer muchas facilidades debido a que utiliza un enfoque incremental e iterativo lo que resulta ideal para este tipo de proyectos, los cuales sus requisitos varían según avanza el proyecto.

Gracias a este marco de trabajo obtenemos una serie de ventajas muy significativas como son:

- **Transparencia:** El equipo de desarrollo y el propio cliente podrán estar al tanto de todas las fases del desarrollo y de los cambios que se van implementando.

- **Inspección:** Todas las partes podrán revisar el proyecto, procesos y resultados. La inspección busca identificar problemas y variaciones en comparación con los resultados obtenidos.

- **Adaptación:** Se podrán hacer cambios en la planificación durante el Spring para cubrir las nuevas necesidades del cliente o suprimir algunas de las que ya estaban. Con esto se pretende ajustar y mejorar continuamente el trabajo del equipo en función de las inspecciones realizadas.

Por otra parte, la metodología de SCRUM contiene una terminología específica referente a los distintos tipos de reuniones, cargos del personal o tipos de documentos. Algunos de los principales son:

- **Product Owner:** Se encargará de maximizar el valor del producto y del trabajo del equipo de desarrollo.

- **Scrum Master:** Se encarga de asegurar que se cumplan los valores de SCRUM.

- **Equipo de desarrollo:** Completan las tareas asignadas al proyecto.

- **Product Backlog:** Contiene una lista priorizada de todo el trabajo que debe hacerse para el producto, el cual es gestionado por el Product Owner.

- **Sprint:** Se refiere a un intervalo de tiempo en el cual el equipo de desarrollo trabaja y crea un incremento del producto entregable. Esto permite un enfoque iterativo e incremental.

● **Incremento:** Entregable final de un sprint.

● **Daily Scrum:** Breve reunión diaria para planificar el día.

● **Sprint Review:** Reunión al final de cada Sprint donde se presenta el incremento a los clientes esperando retroalimentación por su parte.

● **Sprint Retrospective:** Reunión al final de cada Sprint por parte del equipo para comentar el rendimiento y buscar mejoras.

● **Burn-down chart:** Gráfico que muestra la cantidad de trabajo que queda por hacer en un Sprint.

Explicada un poco la terminología usada en SCRUM, denotaremos que hay una serie de diferencias sobre esta metodología en la realización de nuestro proyecto por el hecho de ser individual.

Para comenzar, me tomaré como cliente a mi mismo, que de forma externa al desarrollo que realizo intentaré dar una serie de retroalimentaciones para mejorar la calidad del proyecto y conseguir el máximo valor posible del mismo.

Por otra parte, el Product Owner y Scrum Master pasaré a ser yo mismo. Con este cargo tendré la tarea de ir proponiendo modificaciones o anotaciones para realizar en el proyecto con el fin de incrementar el valor del mismo. De esta forma, organizaré el Product Backlog determinando qué funcionalidades se implementan en cada Sprint y en qué orden. Como papel de Scrum Master, mi misión será asegurarme de que se sigan las normas y prácticas de Scrum de manera efectiva durante todo el curso del proyecto.

Para terminar con los cargos, el equipo de desarrollo únicamente estaría integrado por mí, el cual realizaría todas las actividades necesarias para completar las tareas del Sprint. Esto incluye análisis, diseño, implementación, pruebas y documentación de todo el proyecto. Tendré que tener la capacidad de autogestionarse y tomar decisiones para abordar cada problema que surja en el desarrollo del mismo.

Por último, siguiendo con los eventos de Scrum también tenemos una serie de modificaciones respecto a la versión original de Scrum.

● La duración del Sprint se ha establecido en 2 semanas, en el que se entregará un incremento con una versión para que el cliente pueda dar

retroalimentación y poder seguir avanzando e intentando mejorar la calidad del proyecto.

Esta retroalimentación por parte del cliente se llevará a cabo en el Sprint Review.

● Con respecto al Daily Scrum se aprovechará para revisar el trabajo realizado el día anterior y para plantear los objetivos propuestos para el día actual. Además, se podrán plantear modificaciones del trabajo a realizar con el fin de incrementar valor en períodos tempranos del proyecto.

● Por último, el Sprint Retrospective no se realizará ya que al ser solamente un miembro en el equipo de proyecto no podríamos evaluar en conjunto el mismo.

En resumen, estamos intentando adaptar una metodología Scrum a un proyecto individual con el fin de gestionar de manera efectiva el proyecto y maximizar el valor entregado, siempre manteniendo los principios fundamentales de Scrum ajustando los roles y eventos a la situación individual del proyecto.

Además de las ventajas añadidas anteriormente, se ha utilizado Scrum para la realización del proyecto frente a otras metodologías ya que gracias a ella podemos adaptarnos a cambios frecuentes durante la realización del proyecto. En metodologías más clásicas como puede ser la metodología Waterfall no permite esta flexibilidad, ya que es necesario tener finalizados los procesos de análisis y diseño antes de pasar a la etapa de implementación. Por lo tanto, si surgen nuevas funcionalidades que son imprescindibles de añadir retrasaría la ejecución del proyecto. Otra opción que se podría haber usado para el proyecto sería Kanban, sin embargo, esta metodología se realiza sin tener etapas claras definidas como son el análisis, diseño e implementación, por lo que no se podría proporcionar una estructura que permita evaluar el progreso iterativo del proyecto.

Por todo esto, se ha pensado que Scrum es la mejor opción para la realización del proyecto con el objetivo de maximizar el valor del mismo y permitir cambios constantes y una mejora continua.

2. Análisis

En este apartado se describirán puntos cruciales para el desarrollo del proyecto. Para comenzar, empezaré con un análisis preliminar en el que se estudiarán diferentes aplicaciones similares a la propuesta para el proyecto. Además se propondrán una serie de bibliotecas, tecnologías y entornos de desarrollo útiles para la construcción de nuestra aplicación.

A continuación, se presentará la solución propuesta, mostrando los componentes y la idea principal que se quiere llevar a cabo. Después se llevará a cabo una enumeración de los requisitos que deberá de cumplir el sistema.

Por último, se establecerá una planificación del proyecto, asignando recursos, tiempos y costos. Este proceso es crucial para la realización del proyecto dentro del tiempo establecido.

2.1 Estado del arte

En el contexto actual donde nos encontramos, debido a la creciente demanda de contenido televisivo en dispositivos móviles y la conveniencia de poder acceder a la televisión en cualquier lugar y en cualquier instante, este tipo de aplicaciones está en auge en el mercado por lo que encontraremos mucha competencia.

2.1.1 Aplicaciones similares

A continuación, estudiaremos algunas de estas aplicaciones para ver cómo funcionan y quedarnos con aspectos positivos y negativos que nos ayuden a la realización de nuestra propia aplicación de retransmisión.

2.1.1.1 Kodi

Kodi [6] es una aplicación de código abierto que permite a los usuarios reproducir gran cantidad de contenido multimedia, incluidos canales de televisión en directo generados con listas M3U8. Los usuarios pueden instalar distintos complementos a usar dentro de la aplicación.

La ilustración 2.1 muestra la interfaz principal al entrar a la aplicación. Podemos ver que la cantidad de opciones y posibilidades que tenemos para hacer es bastante

elevada, lo que para clientes cuya habilidad tecnológica no sea muy elevada puede ser un poco abrumador.

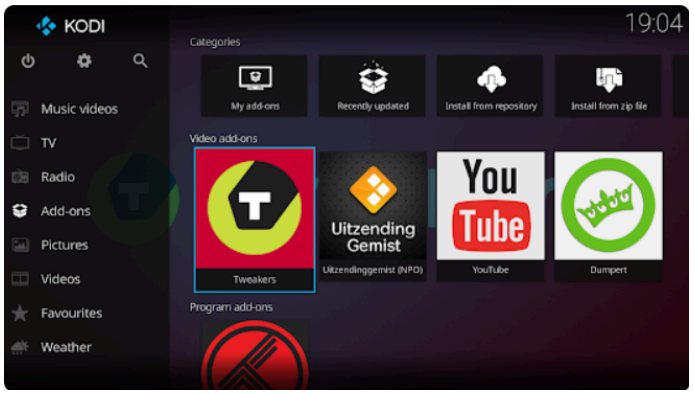

Ilustración 2.1: Interfaz principal de Kodi

En la ilustración 2.2, podemos observar la interfaz que usa la televisión donde muestra la lista de canales que tiene junto con información del canal que se está transmitiendo. La interfaz parece bastante intuitiva y muestra la información correctamente. Esta idea nos podrá ayudar para nuestro posterior diseño.

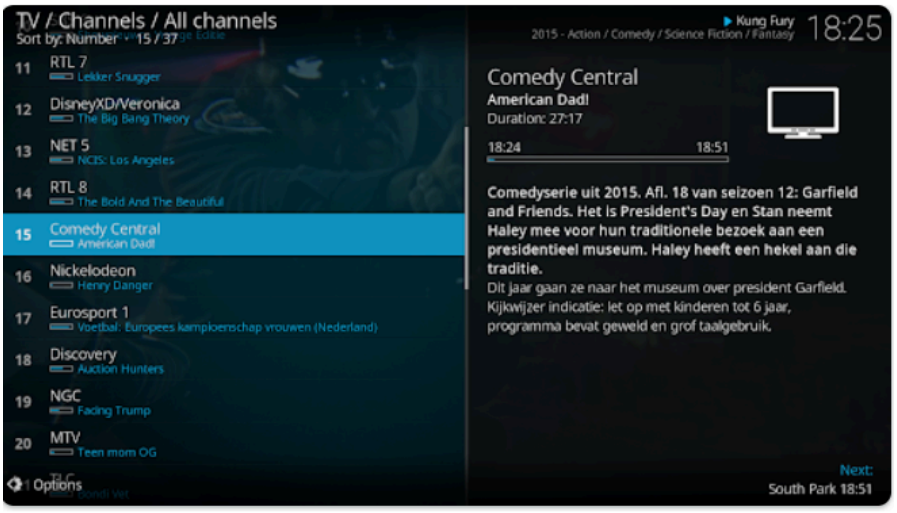

Ilustración 2.2: Interfaz canales de Kodi

2.1.1.2 VLC

VLC media player [7] es un reproductor multimedia gratuito y de código abierto multiplataforma que reproduce la mayoría de los archivos multimedia, así como discos, dispositivos y protocolos de transmisión en red.

En la ilustración 2.3 se presenta la interfaz de inicio del reproductor de VLC. Podemos observar como nuevamente tenemos muchas opciones ya que además de

transmisión en red permite reproducir videos de forma local. Además, para reproducir contenido en vivo es necesario incluir una lista de canales M3U8, ya que no viene una por defecto que se pueda usar.

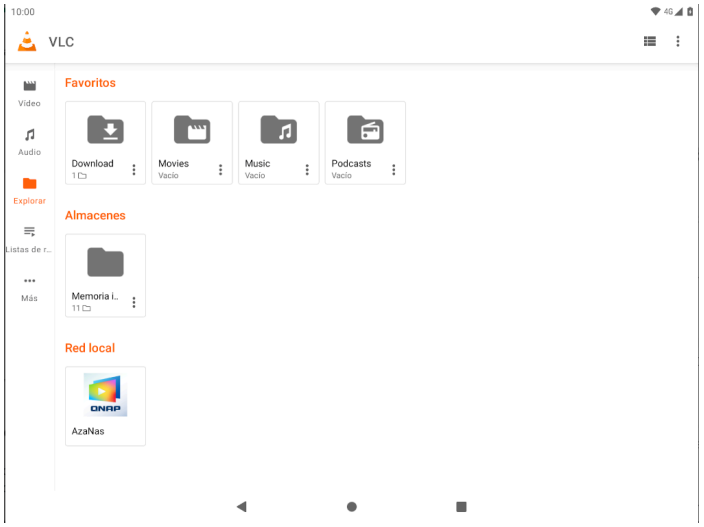

Ilustración 2.3: Interfaz inicio de VLC Player

Una vez probada la reproducción de streaming en VLC comprobamos que la transmisión es eficiente, sin cortes y fluida por lo que intuimos que el motor de reproducción que tiene es bastante potente. Además VLC ofrece una librería de código abierto que se puede incorporar a proyectos propios para la reproducción y que ofrece todas las funcionalidades que usa en la aplicación Android, por lo que la podremos integrar en nuestro proyecto para conseguir la transmisión fluida que esperamos conseguir.

En la ilustración 2.4 podemos ver la interfaz de reproducción que usa el reproductor de VLC. Podemos observar que es una interfaz sencilla lo que permite que el usuario se centre en el contenido que se está reproduciendo. No obstante, esta interfaz ofrece distintos controles que permite el control total de la reproducción como puede ser los botones de anterior y siguiente, el detenido de la reproducción.

Esta idea de interfaz nos podrá servir para hacer un tipo de interfaz sencilla que permita manejar todo el reproductor y que no tenga muchos elementos gráficos para que el usuario pueda centrarse en lo que está reproduciendo.

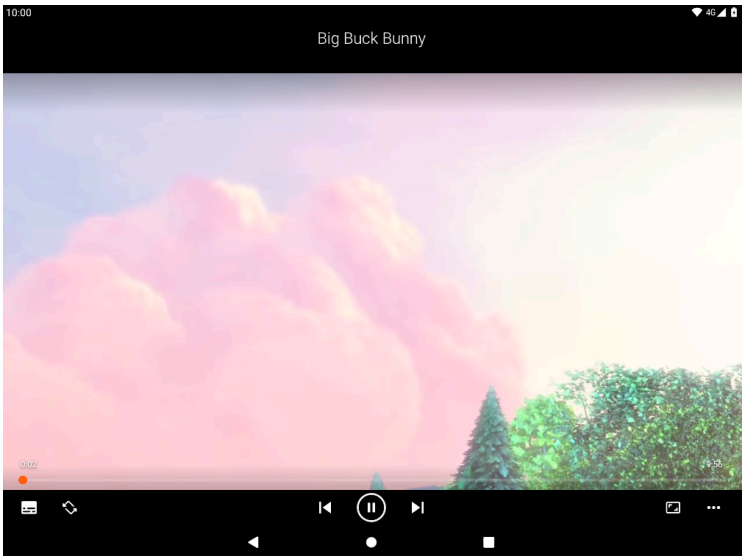

Ilustración 2.4: Interfaz reproducción VLC

2.1.1.3 OTT Navigator IPTV

OTT Navigator [8] es una aplicación de reproductor de IPTV [9] para dispositivos Android. Esta herramienta te permite disfrutar de una amplia variedad de contenidos multimedia, siempre que dispongas de los enlaces correspondientes.

La ilustración 2.5 muestra la interfaz que recoge la lista de canales y la programación actual que tiene cada uno de ellos. En ella vemos la información relevante que aparece como es el nombre del canal, el título del programa actual y el horario de emisión del mismo. Esta información es importante para el usuario ya que le permite conocer lo que se está reproduciendo en cada canal en el momento actual y sería importante añadirlo en este proyecto.

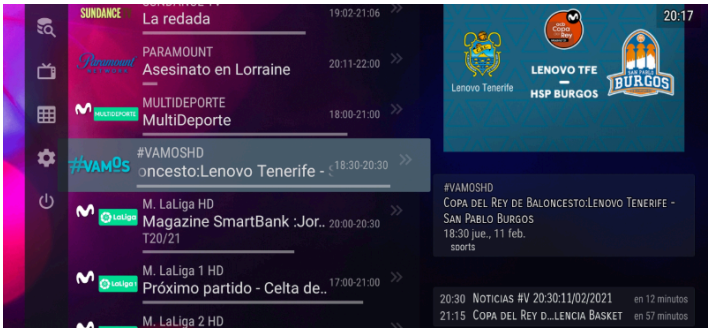

Ilustración 2.5: Interfaz canales OTT Navigator

En la ilustración 2.6 tenemos un fragmento de un canal reproduciendo, junto con la interfaz de usuario que ofrece. Podemos observar como muestra un panel informativo que recoge datos importantes, como son el nombre del canal, el título y

descripción del programa y el horario del mismo. Este tipo de panel está muy bien estructurado y contiene la información necesaria que permite al usuario conocer la información del canal y programa actual mientras está visualizando de fondo la reproducción. Además, el tamaño del panel no es invasivo, es decir, no ocupa toda la pantalla.

Ilustración 2.6: Interfaz reproducción OTT Navigator

Nuevamente, tenemos una aplicación muy robusta con muchas funcionalidades pero es necesario incluir una lista propia para poder reproducir contenido, además de tener una interfaz con muchas opciones en el inicio.

2.1.1.4 Tivimate

Tivimate[10] es un reproductor multimedia capaz de reproducir listas IPTV, permitiéndonos ver canales en directo. Es más propio de aparatos para Android TV [11], aunque también se puede usar para móviles y tablets.

Como se muestra en la ilustración 2.7, la interfaz principal de la aplicación de Tivimate permite añadir algunos tipos de lista de canales, pero tenemos el problema de que no viene asignado ninguno por defecto para poder probar directamente.

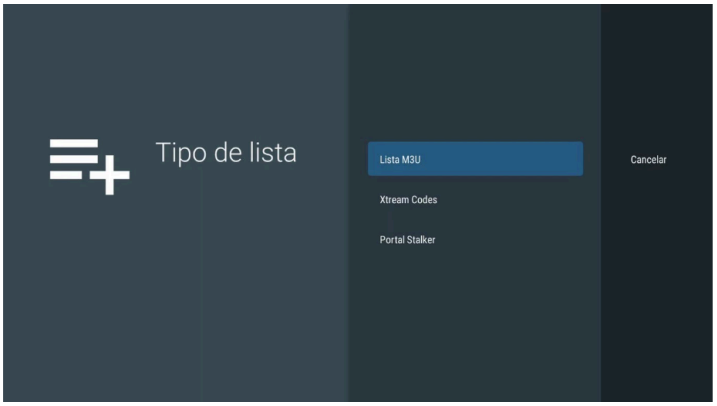

Ilustración 2.7: Interfaz inicio Tivimate

La ilustración 2.8 representa la interfaz del apartado de EPG [12] de la aplicación de Tivimate. En ella podemos comprobar que se muestra para cada canal la información de los siguientes programas dividida en una línea del tiempo. Está bien estructurado en una tabla, aunque puede sobrecargar de información al usuario al aparecer toda la información de todos los canales a la vez.

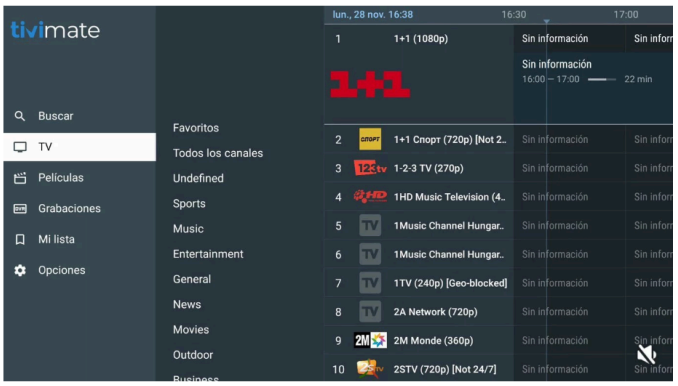

Ilustración 2.8: Interfaz EPG Tivimate

En resumen, una vez vistas varias aplicaciones de las más usadas en la actualidad de entre los cientos que existen, comprobamos que todas tienen su comportamiento de forma parecida. Tenemos una aplicación que permite ejecutar una serie de listas y ver contenido en directo, con una interfaz más o menos compleja. Sin embargo, todas acaban con el mismo inconveniente, ya que es necesario cargar una lista por nosotros mismos y no viene una predeterminada para poder usarla.

2.1.2 Bibliotecas, tecnologías y frameworks.

Como se ha comentado anteriormente, para la implementación de nuestra aplicación móvil es necesario el uso de bibliotecas, tecnologías y frameworks que permitan el tratamiento de información extraída de peticiones en la nube, además de un potente motor de reproducción que permita visualizar los directos de una forma fluida y con buena calidad.

2.1.2.1 Bibliotecas

Media3

Media3 [13] es una librería de código abierto [14] para la gestión y reproducción de contenido multimedia en aplicaciones Android. Tiene mecanismos para la gestión de la reproducción de video y audio de manera eficiente y flexible. La biblioteca de

Media3 permite reproducir contenido en diferentes formatos, tanto de video como de audio, además de la gestión de listas de reproducción.

En la ilustración 2.9, podemos observar alguna de las puntuaciones que se recogen acerca del reproductor de Media3 como son calidad, mantenimiento y documentación. Podemos observar que todas las notas están en torno al aprobado, aunque el mantenimiento cae por debajo de él. Este tipo de características nos ayudará posteriormente a decidir sobre qué reproductor elegir para nuestro proyecto.

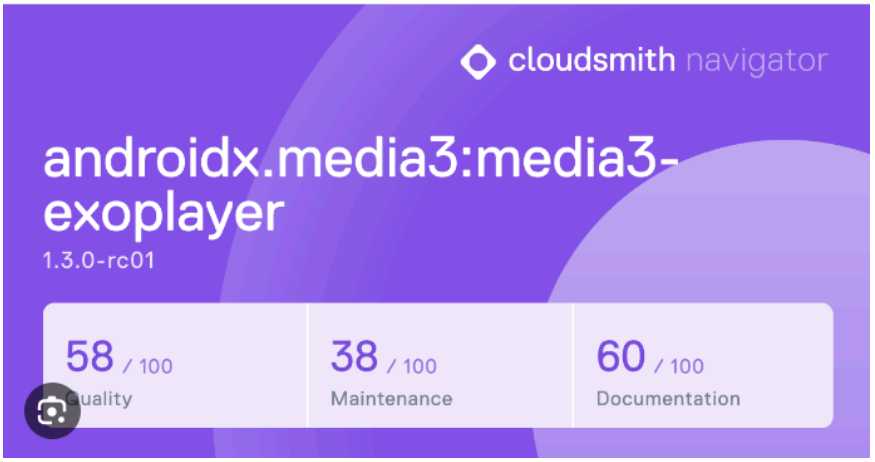

Ilustración 2.9: Reproductor Media3 Exoplayer

LibVLC

LibVLC [15] es una biblioteca de código abierto desarrollada por VideoLAN [16]. Permite la reproducción de contenido de alta calidad en un elevado número de plataformas, siendo una de ellas la plataforma de Android. LibVLC es el motor de reproducción que se usa en la aplicación multimedia de VLC, la cual es conocida por su amplio soporte para reproducir la mayoría de formatos y su capacidad de mostrar contenido de manera fluida y sin ningún tipo de problemas. Con la implementación de libVLC en el proyecto se puede aprovechar todas las ventajas de VLC en nuestra propia aplicación.

El logo que representa la librería de LibVLC se puede observar en la ilustración 2.10.

Ilustración 2.10: Reproductor LibVLC

Glide

Glide [17] es una biblioteca de carga y visualización de imágenes en Android. Esta biblioteca está desarrollada por Bumptech [18]. Se utiliza para la carga eficiente de imágenes a aplicaciones en sistemas operativos Android. Además ofrece funcionalidades avanzadas para el procesamiento de imágenes. Alguna de las características de esta librería son:

- Carga eficiente minimizando el consumo de memoria. Incluye la carga asíncrona de las imágenes en un hilo separado del principal para no interrumpir la ejecución de la aplicación. Además, almacena en la memoria caché las imágenes para un acceso rápido posterior.

- Permite cargar imágenes de distintas fuentes como fuentes de datos, archivos locales o URL externas.

- Permite el tratamiento de las imágenes. Ofrece funcionalidades como recortar, rotar, redimensionar…

- Se integra de forma muy sencilla con las vistas de Android, como por ejemplo, cargando una imagen extraída a un ImageView [19] de Android de forma casi automática.

Esta librería nos ofrecerá muchas facilidades para la carga de las imágenes de los logos que usaremos en nuestra aplicación.

El logotipo de Glide, ilustrado en la Ilustración 2.11, se ha convertido en un estándar en la gestión de imágenes para aplicaciones Android

Ilustración 2.11: Logo Glide

OkHttp

OkHttp [20] es una biblioteca de cliente HTTP para Android que facilita el manejo de solicitudes HTTP a servidores y la posterior obtención de respuestas. Ofrece distintas funcionalidades como son:

○ Facilita creación y envío de solicitudes HTTP, además de realizar de forma automática operaciones como creación de conexiones, seguimiento de cookies… También procesa el resultado devuelto por el servidor.

○ Gestiona la reutilización de conexiones TCP [21] para reducir la utilización de recursos.

○ Ofrece soporte para conexiones seguras HTTPS y manejo de certificados SSL.

○ Proporciona funcionalidades para la cancelación de solicitudes de manera eficiente.

La Ilustración 2.12 contiene el logo de OkHttp, representando su papel clave en la comunicación HTTP segura y eficiente.

Ilustración 2.12: Logo OkHttp

Gson

Gson [22] es una biblioteca Java que se usa para la conversión de objetos Java a JSON(JavaScript Object Notation) y viceversa. Facilita mucho el procesamiento de datos JSON [23], proporcionando funcionalidades para serializar y deserializar objetos. Algunas de las características de esta biblioteca son:

- **Conversión de objetos sin necesidad de escribir código manual para ello.**
- **Permite manejar distintos tipos de datos como listas, mapas, arrays…**
- **Se integra muy bien con distintas librerías o framework como Retrofit [24] o Android SDK.**

En la Ilustración 2.13 se puede ver el logotipo de Gson, que destaca en su uso extendido en la gestión de datos JSON en aplicaciones Android.

Ilustración 2.13: Logo Gson

2.1.2.2 Framework Android SDK

El framework que se va a utilizar para el desarrollo del proyecto será Android SDK. Android SDK [25](Software Development Kit) es un conjunto de herramientas, recursos y APIs que permite a los desarrolladores crear aplicaciones para el sistema operativo Android. Está formado por una serie de componentes:

- **Android Platform Tools**

Incluye herramientas de líneas de comandos que facilitan la comunicación entre en sistema Android y el ordenador para conseguir objetivos cómo depurar o gestionar aplicaciones.

- **Android Debugging Bridge(ADB)**

Permite la comunicación entre ordenador y dispositivo Android para la gestión de aplicaciones, depuración, instalación y otras funciones a través de línea de comandos.

- **Android SDK Tools**

Permite compilar aplicaciones, depurar o generar APKs. También podemos descargar diferentes versiones de SDK gracias al SDK Manager para poder probar nuestras aplicaciones en diferentes versiones y dispositivos.

- **Android SDK Platform**

Contiene las API de Android. Cada versión de SDK contiene unas funcionalidades específicas que cambian dependiendo del sistema operativo Android.

- **Android Emulator**

Nos permite ejecutar aplicaciones en entornos virtuales en distintos dispositivos virtuales con diferentes versiones y hardware dentro de nuestro propio ordenador.

- **Android Support Library**

Proporciona librerías que ayudan a mantener la compatibilidad entre distintas versiones de Android.

Concluimos que este framework es esencial en la realización de nuestro proyecto, ya que nos dará muchas facilidades en la implementación del código

además de ofrecernos recursos y herramientas para poder depurar y ejecutar aplicaciones.

La Ilustración 2.14 muestra el logotipo del framework Android SDK, que enfatiza su importancia para el desarrollo eficiente de aplicaciones en la plataforma Android.

Ilustración 2.14: Logo Android SDK

2.1.2.3 Entorno de desarrollo

Android Studio

Android Studio [26] es el entorno de desarrollo integrado oficial para la plataforma Android. Está disponible en las plataformas de Linux, MacOS, Windows y Chrome OS. Algunas de sus características son:

- Está basado en IntelliJ IDEA [27], del cual hereda muchas funcionalidades como el editor de código, herramientas de factorización o compatibilidad de un elevado número de lenguajes.

- Permite diseñar interfaces de aplicaciones de manera visual, además de visualizar layouts en diferentes dispositivos virtuales.

- Permite depurar código además de ejecutarlo en diferentes emuladores virtuales o conectar dispositivos físicos a través de mecanismos como ADB [28].

- Permite la integración de distintas librerías, paquetes y componentes para darle más valor a nuestra aplicación.

- Ofrece la posibilidad de utilizar varios lenguajes de programación como son Kotlin o Java

○ Está integrado con servicios de Google lo que permite realizar análisis de aplicaciones, autenticación de usuarios o distribución de aplicaciones a través de Google Play Store.

El logotipo de Android Studio se encuentra en la Ilustración 2.15, destacando su papel central como herramienta de desarrollo en la plataforma Android.

Ilustración 2.15: Logo Android Studio

Apache Cordova

Apache Cordova [29] es un entorno de desarrollo de aplicaciones móviles que está también disponible en plataformas como Android, iOS o Windows y permite desarrollar aplicaciones nativas usando tecnologías web como HTML, CSS o JavaScript. Algunas de sus características son:

○ Permite utilizar código de tecnologías web lo que reduce la complejidad de la programación.

○ Proporciona acceso y funcionalidades a las APIs nativas del dispositivo a través de un conjunto de plugins.

○ Es compatible con distintos entornos de desarrollo, como pueden ser Visual Studio Code o Android Studio.

○ Permite distribuir fácilmente las aplicaciones en tiendas como Google Play Store.

La Ilustración 2.16 incluye el logotipo de Apache Cordova, que resalta su capacidad para el desarrollo híbrido de aplicaciones móviles.

Ilustración 2.16: Logo Apache Cordova.

2.2 Tecnologías seleccionadas

Una vez realizado el análisis preliminar y habiéndonos informado de distintas bibliotecas, frameworks y entornos de desarrollo, he decidido realizar una aplicación que conste de un reproductor de contenido en línea, el cual barajando las distintas opciones que teníamos, hemos visto que sería más factible usar el motor de reproducción de libVLC. Esto se debe a la serie de ventajas que ofrece con respecto a Media3 como la diferencia de documentación que hay entre ambos, la comunidad de desarrolladores que llevan a las espaldas cada uno de ellos o la cantidad de decodificadores que manejan.

Por otra parte, usaremos OkHttp para gestionar la comunicación entre el servidor y la aplicación cliente y así poder controlar las peticiones por las que obtendremos las información de los canales o de los distintos EPGs de cada uno de ellos. Esta librería nos facilitará mucho el trabajo, debido a las ventajas comentadas anteriormente.

Seguimos con la utilización de Glide, el cual nos permitirá de una forma muy sencilla a través de URLs remotas en servidores externos poder capturar información que guardan y convertirlos en imágenes en nuestra aplicación, además de asegurarnos distintas funcionalidades para tratar estas imágenes. Esto nos servirá de gran ayuda para obtener los logos de los canales y poder mostrarlos en nuestra aplicación directamente extraídos de fuentes externas. Gracias a esto, podremos tener una interfaz muy atractiva para el usuario.

Para la conversión de la información extraída de los servidores tendremos que usar la biblioteca Gson. Esta librería nos servirá de gran ayuda para convertir la información original en formato JSON en objetos Java, con los que haremos toda la

operativa de la aplicación. Esta librería nos facilitará enormemente el trabajo y en pocas líneas de código podremos conseguir nuestro objetivo.

Por último, como entorno de desarrollo tendremos Android Studio junto con el framework de Android SDK ya predeterminado en el entorno. Esto nos proporcionará mucha ayuda como hemos visto antes en tareas como la depuración o ejecución de código en dispositivos virtuales de distintos tipos directamente en nuestro ordenador. Además, nos permitirá diseñar interfaces de diseño directamente colocando elementos en una pantalla. Para terminar usaremos el lenguaje Java para la programación de nuestra aplicación, para así aprovechar los conocimientos que han sido adquiridos durante la carrera y poder invertirlos en otra rama distinta como es la realización de una aplicación móvil.

2.3 Requisitos

Un requisito [30] se puede definir como una condición o capacidad que debe tener o cumplir un producto o servicio para satisfacer una necesidad u objetivo formalmente impuesto. Son muy importantes en la realización de un proyecto ya que definen lo que se espera del producto y cómo se medirá el valor del mismo. Dentro de este término pueden surgir muchos tipos, aunque nosotros haremos una clasificación dividida en dos criterios:

- **Requisitos funcionales.**

Describe la funcionalidad del sistema, es decir, las acciones que el sistema es capaz de hacer.

- **Requisitos no funcionales.**

Este tipo de requisitos especifican características que debe tener el sistema que pueden ser del ámbito de calidad, rendimiento, usabilidad…

No está ligado a la funcionalidad del sistema como los anteriores.

2.3.1 Requisitos funcionales

A continuación se muestra una lista de los requisitos de tipo funcional que debería de cumplir la aplicación.

1. El sistema debe ser capaz de reproducir transmisión en directo de canales a través de enlaces en formato M3U8.

2. El sistema permitirá al usuario poder cambiar de canal en tiempo real por pantalla o dispositivo de control remoto.

3. El sistema permitirá desplegar un menú para poder visualizar la lista de canales completa que contiene el sistema

4. El sistema permitirá consultar la información del programa actual que se está retransmitiendo en el canal que se está reproduciendo.

5. El sistema permitirá consultar la información existente de la información del EPG para todos los canales.

6. El sistema permitirá controlar el volumen del dispositivo.

7. El sistema podrá mostrar y seleccionar las diferentes pistas de audio y subtítulos que tiene un canal.

8. El sistema tendrá la funcionalidad de ver programas anteriores grabados en servidores externos en los canales que estén autorizados.

9. El sistema permitirá volver a un menú inicial para salir del reproductor.

2.3.2 Requisitos no funcionales

A continuación se muestra una lista de los requisitos de tipo no funcional que debería de cumplir esta aplicación.

1. La aplicación debe cargar rápidamente.

2. La aplicación debe responder de forma fluida a la interacción del usuario.

3. La retransmisión debe ser de forma fluida, sin cortes y con buena calidad.

4. La interfaz debe ser intuitiva y fácil de usar para todo tipo de usuarios, independientemente de sus habilidades tecnológicas o edad.

5. La aplicación se ajustará a distintas versiones de Android, así como distintos tamaños de pantalla de dispositivos.

6. La aplicación se ajustará a cambios de orientación en el dispositivo.

7. La aplicación será funcional para usar distintos tipos de codecs de audio y vídeo.

8. El sistema podrá recuperarse automáticamente de fallos o caídas de Internet.

9. El sistema tendrá una tasa de fallos y bloqueos minúscula.

10. El sistema ajustará la transmisión dependiendo de la velocidad de red.

2.4 Historias de usuario

Una historia de usuario [31] es una descripción sencilla y genérica de una funcionalidad del software desde la experiencia del usuario. Define lo que un usuario necesita, lo que ayuda a priorizar el trabajo y a aumentar el valor para el cliente.

Las historias de usuario son una herramienta común en desarrollos ágiles, que permiten a los equipos de desarrollo trabajar en iteraciones, centrándose en ofrecer a los usuarios finales pequeñas funcionalidades lo antes posible. Sigue el siguiente formato:

- **ID.** Identificador único de la HU
- **Título.** Título descriptivo de la HU
- **Descripción.** Cómo [rol de usuario] quiero [objetivo] para poder [beneficio]
- **Estimación.** Estimación de coste de la implementación en días.
- **Valor de negocio.** Valor que aporta la historia de usuario al cliente.
- **Dependencia.** Relaciones con otras historias de usuario.
- **Criterios de aceptación.** Condiciones específicas que deben cumplirse para considerar que la historia está completada. Estas condiciones suelen ser verificables y proporcionan un conjunto claro de expectativas para el equipo de desarrollo. Esta parte se dejará para más adelante en el plan de pruebas.

Para la realización de historias de usuario en un marco de desarrollo ágil, se seguirá el modelo INVEST [32] para asegurar la calidad de las mismas y que cumplan las siguientes características:

- **Independiente:** Las historias de usuario deben ser lo más independientes posible entre ellas. Esto facilita la planificación y la priorización, permitiendo implementar las historias de usuario en cualquier orden y sin necesidad de depender de otras historias de usuario.

● **Negociable:** Las historias de usuario pueden ser modificadas o refinadas durante el proceso de desarrollo para poder ajustar los requisitos según sea necesario.

● **Valiosa:** Cada historia de usuario debe tener un valor para el usuario final.

● **Estimable:** Se debe poder estimar el esfuerzo necesario para poder realizar cada historia de usuario. Esto incluye tanto tiempo como recursos.

● **Pequeña:** Las historias de usuario deben ser lo más pequeñas posible, pero lo suficiente para aportar valor al cliente. Se debe poder implementar en un periodo corto de tiempo.

● **Testable:** Debe ser posible saber si una historia de usuario ha sido completada satisfactoriamente. Para conseguir esto, se deben definir claramente los criterios de aceptación y a partir de ellos, comprobar si la historia de usuario cumple con los requisitos propuestos.

Para poder priorizar y categorizar [33] las historias de usuario según los valores que aportan. Se divide en cuatro categorías según el nivel de prioridad:

● **Must have (Debe tener):** Son funcionalidades críticas en la aplicación que son imprescindibles para el éxito del proyecto. Sin estas funcionalidades el proyecto no estaría completo. Son elementos de alta prioridad. Para indicar que una historia de usuario pertenece a esta categoría se le asignará la letra *M en el apartado de valor.*

● **Should have (Debería tener):** Son funcionalidades importantes pero no críticas. Proporcional valor, pero si se eliminase el producto seguiría siendo funcional. Para indicar que una historia de usuario pertenece a esta categoría se le asignará la letra *S en el apartado de valor.*

● **Could have (Podría tener):** Son funcionalidades opcionales que incrementarían el valor del producto y estaría bien tenerlas, pero no son críticas ni esenciales. Para indicar que una historia de usuario pertenece a esta categoría se le asignará la letra *C en el apartado de valor.*

● **Won´t have (No debería tener):** Son funcionalidades que no se van a implementar, ya sea porque no aportan el suficiente valor o porque son muy costosas en relación al beneficio que va a ocasionar. Para indicar que una historia

de usuario pertenece a esta categoría se le asignará la letra *W en el apartado de valor.*

En la tabla 2.1, se muestran las distintas historias de usuario ordenadas por un identificador y con datos como son el título, descripción, tiempo que se estima que se va a tardar en completar en días y valor de la misma.

ID	Título	Descripción	Estimación (días)	Valor
1	Reproducción de canales en directo	Como usuario, quiero poder reproducir transmisiones en directo a través de enlaces M3U8 para poder ver transmisión en tiempo real.	9	M(5)
2	Interfaz reproducción	Como usuario, quiero tener una interfaz de reproducción intuitiva para poder controlar el reproductor de forma sencilla	8	M(5)
3	Cambio de canal en tiempo real	Como usuario, quiero poder cambiar de canal para poder ver diferentes canales ya sea interactuando con el dispositivo o con un control remoto	5	M(5)
4	Visualización de lista de	Como usuario, quiero poder ver la lista	4	S(4)

	canales	completa de canales para poder seleccionar uno de ellos fácilmente.		
5	Consulta de información del programa actual	Como usuario, quiero poder consultar la información del programa actual mientras reproduzco ese canal para poder consultar la información del programa.	3	S(4)
6	Consulta de información de EPG	Como usuario, quiero consultar la guía electrónica de programación(EPG) para poder ver la programación futura de los canales.	4	S(4)
7	Control de volumen	Como usuario quiero poder controlar el volumen del dispositivo mientras veo la transmisión para poder ajustarlo según mis preferencias.	1	S(4)
8	Selección de pistas de audio y subtítulos.	Como usuario, quiero poder seleccionar diferentes pistas de audio y subtítulos para adaptar la experiencia de usuario.	5	S(4)

9	Visualización de programas grabados anteriores	Como usuario, quiero poder ver contenido grabado en servidores para ver programas anteriores.	5	C(3)
10	Volver al menú principal	Como usuario, quiero poder volver al menú inicial para salir del reproductor	2	M(5)

Tabla 2.1: Historias de usuario

2.5 Planificación

Ya descritas las historias de usuario, en las que tenemos una estimación del esfuerzo necesario que tendrá que invertirse para completar la tarea, además de tener la prioridad que tiene cada una de ellas, se pasará a planificar los Sprints que se van a realizar y las tareas que compondrán cada uno de estos. La duración de las tareas se medirán en días.

En la tabla 2.2, se muestran las distintas tareas que se deberán de llevar a cabo para el cumplimiento del proyecto. Además de las historias de usuario que se han mencionado anteriormente las cuales se muestran ahora las tarea relacionadas con ellas, se han incluido otras 3 que serían el análisis del proyecto, diseño y documentación del mismo. Las tres tareas irían precediendo a las tareas de implementación de las historias de usuario y su realización es necesaria para el proyecto. De hecho, para comenzar con la primera tarea de implementación (Configuración inicial del proyecto), antes se debe haber terminado el proceso de diseño del proyecto. A su vez, para empezar esta tarea, se debe de concluir el análisis del mismo, la cual será la inicial. Por último, a la vez que empiezan las tareas que involucran el análisis del proyecto, paralelamente empiezan las tareas de documentación del proyecto, las cuales, como se indica posteriormente, durarán todo el proyecto.

Id	Historia de usuario	Tareas	Precedentes	Duración (Días)
A	Análisis del proyecto	1. Investigar estado del arte 2. Investigar tecnologías 3. Obtener requisitos para la aplicación 4. Planificar 5. Estimar costes	-	10
B	Diseño del proyecto	1. Realizar diagramas 2. Concluir arquitectura de la aplicación 3. Hacer diseño de la interfaz	A	10
C	Documentación del proyecto	1. Redactar memoria	-	Totalidad Proyecto
D	Configuración inicial del proyecto	1. Crear un proyecto Java en Android Studio 2. Añadir al proyecto un repositorio en Github	B	1
E	Reproducción de canales en directo	1. Investigar y seleccionar una biblioteca adecuada para la reproducción de M3U8 2. Configurar la biblioteca en el proyecto	D	9

		3. Implementar carga de enlaces M3U8 4. Desarrollar el módulo de reproducción 5. Realizar pruebas de reproducción 6. Documentar la funcionalidad		
F	Interfaz de reproducción	1. Diseñar y desarrollar una interfaz de reproducción de vídeo 2. Asegurar que la interfaz sea intuitiva y sencilla 3. Realizar ajustes	D	8
G	Cambio de canal en tiempo real	1. Implementar funcionalidad de cambio de canal en tiempo real sin interrumpir la transmisión 2. Optimizar la velocidad de cambio de canal 3. Mostrar información actualizada del cambio de canal 4. Realización de pruebas 5. Implementar funcionalidad cambiar canal a partir de un	E	5

		dispositivo de control remoto(teclado o mando)		
H	Visualización de lista de canales	1. Crear un menú que muestre la lista completa de canales 2. Implementar la funcionalidad de desplazamiento a través de la lista 3. Hacer que la lista sea fácil de entender 4. Realizar pruebas de usabilidad 5. Comprobar que se muestran todos los canales	E,F	4
I	Consulta de información del programa actual	1. Implementar funcionalidad para mostrar el título, hora de inicio y fin y descripción del programa actual 2. Actualizar la información del programa actual al cambiar de canal 3. Realizar pruebas y validación	G	3
J	Consulta de	1. Implementar una	G,H,I	4

	información del EPG	interfaz para mostrar el EPG del canal 2. Mostrar la información de las próximas 24 horas 3. Permitir la navegación a través del EPG 4. Comprobar que la lista de EPG se muestra correctamente		
K	Control de volumen	1. Añadir control de volumen en la interfaz de usuario 2. Hacer que el control sea intuitivo 3. Validar el control	F	1
L	Selección de pistas de audio y subtítulos	1. Capturar las diferentes pistas de audio y subtítulos que tengan incluidas cada enlace M3U8 2. Hacer el diseño e implementación de un menú para mostrar las diferentes pistas 3. Permitir la selección de las pistas 4. Cambiar la información de las pistas al cambiar de canal	I,J	5

		5. Realizar pruebas de funcionamiento		
M	Visualización de programas grabados anteriores	1. Filtrar los programas que tienen grabación en el servidor 2. Mostrar información que permita identificar esos programas 3. Permitir reproducir un programa grabado 4. Realizar pruebas para comprobar si se muestra correctamente	J,K	5
N	Volver al menú principal	1. Añadir un botón que permita volver al menú principal 2. Asegurar que la navegación sea intuitiva 3. Validar la usabilidad	F	2

Tabla 2.2: Definición de las tareas a realizar

Una vez se han definido las tareas a realizar para cada historia de usuario, tenemos que el total de días en los que se estima que concluirá el proyecto son 93, incluyendo las tareas de análisis y diseño. Además, las tareas relacionadas con la documentación irán en paralelo a estas. Por lo tanto tenemos que cada Sprint tendrá una serie de tareas que aportarán valor al producto final. Las tareas relacionadas con análisis, diseño e implementación añadirán un valor de 5 al proyecto respectivamente, ya que pertenecen a la categoría de Must Have mencionada anteriormente debido a que es obligatorio la realización de estas fases.

En la tabla 2.3, se muestran los distintos Sprints, así como la fecha de inicio y fin de cada uno de ellos, junto con las historias de usuario que se van a implementar en cada uno y el valor añadido que aportará el conjunto de historias de usuario que conforman el Sprint.

Sprint	Fecha Inicio	Fecha Fin	Historias de usuario	Valor
1	19/02/2024	01/03/2024	A,C	10
2	04/03/2024	15/03/2024	B	5
3	18/03/2024	29/03/2024	D,E	10
4	1/04/2024	12/04/2024	F,N	10
5	15/04/2024	26/04/2024	G,H,K	13
6	29/04/2024	08/05/2024	I,J	8
7	09/05/2024	22/05/2024	L,M	7

Tabla 2.3:Planificación de Sprints

El valor final de cada Sprint se calculará como la suma de los valores de las tareas que lo componen. La duración de cada Sprint, como se estableció en un principio, será de 2 semanas, lo que supondrá un total de 10 días laborales por cada Sprint.

Para la planificación del proyecto, se han priorizado las actividades que dan más valor al proyecto, siendo estas fundamentales para la realización del mismo, siempre cuadrando su duración a la determinada por el Sprint. Con esto se pretende, que si no se llega a completar todas las tareas, que entonces, al menos las que aportan mayor valor queden terminadas.

En la ilustración 2.17 se muestra un diagrama de Gantt [34] con todas las tareas de implementación a realizar ordenadas por una línea temporal.

Fecha inicial de planing 19-2-2024	Duracion total: 93 dias naturales	PLANING DE TIEMPOS

Id	c	tarea	f. ini	f.fin	dur
1	1	Análisis	19-2-2024	1-3-2024	12
2	1	Diseño	4-3-2024	15-3-2024	12
3	1	Documentación	19-2-2024	22-5-2024	94
4	1	Configuración inicial	18-3-2024	19-3-2024	2
5	1	Reproducción de canales	20-3-2024	29-3-2024	10
6	1	Interfaz de reproducción	1-4-2024	10-4-2024	10
7	1	Volver menú principal	11-4-2024	12-4-2024	2
8	1	Cambio canal	15-4-2024	19-4-2024	5
9	1	Visualización lista canales	22-4-2024	25-4-2024	4
10	1	Control de volumen	25-4-2024	26-4-2024	2
11	1	Consulta programa actual	29-4-2024	2-5-2024	4
12	1	Consulta información EPG	3-5-2024	8-5-2024	6
13	1	Selección pistas	9-5-2024	15-5-2024	7
14	1	Visualización de programas gr	16-5-2024	22-5-2024	7

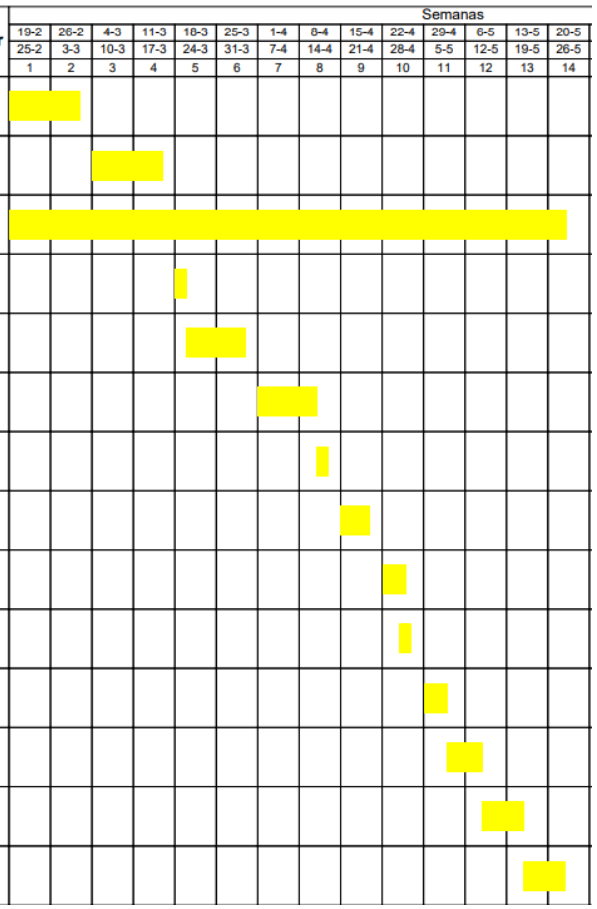

Ilustración 2.17: Diagrama de Gantt

Como se puede comprobar en el diagrama de Gantt todas las tareas son planificadas de forma secuencial. Esto es debido a que el proyecto se realiza de forma individual, por lo que hasta que no se acabe una tarea no se comenzará con la siguiente.

A continuación se muestra un enlace que contiene la imagen en google Drive para que se pueda observar de forma más clara.

https://drive.google.com/file/d/11HcLzo9R3ufmD5eAzCHH63hnQlyUvgEv/view?usp=sharing

2.6 Estimación de costes

En este apartado se analizarán todos los costes resultantes de la ejecución del proyecto, tanto a nivel de componentes como de personal.

A nivel de software, todo el proyecto se realizará con programas de licencia libre, por lo que en esta parte el coste será nulo.

En la tabla 2.4, podemos observar la estimación de los costes de dispositivos hardware, así como una descripción de los dispositivos que se van a usar junto con el tiempo de vida útil que se estima que tiene el dispositivo.

La duración de la vida útil de los dispositivos hardware se han basado en informes de la industria, que se detallarán en el apartado de referencias bibliográficas al final del documento. [35]

Hardware	Descripción	Coste	Vida Útil
Lenovo Ideopad Gaming 3	Ordenador propio. AMD Ryzen 7, Nvidia GeForceGTX 16 GB RAM	800	4 años
Lenovo Tab M10 Plus	Tablet para pruebas Android, 4GB RAM 10.61”	154	3 años

Tabla 2.4: Coste hardware

Una vez obtenido el conjunto de dispositivos que usaremos a nivel hardware para la realización del proyecto junto con su precio, se pasará a calcular la amortización de cada uno de ellos contando que el tiempo de utilización de los mismos va a ser de 3 meses. Entonces se obtiene que el total del ordenador será 50 €, mientras que la tablet supondrá 12.83 €.

Por otra parte, se ha determinado el coste según el rol de cada tipo de trabajador, como se muestra en la tabla 2.5 a continuación.

En esta tabla se muestra el tipo de personal que se ha ido desarrollando durante el proyecto junto con una estimación del tiempo dedicado en comparación con el total.

Recurso Humano	Sueldo	Porcentaje dedicado	Total
Programador	2100€/mes	60%	3780€
Analista	2420€/mes	25%	1815€
Jefe de proyecto	2950€/mes	15%	1327,5€
			6922,5€

Tabla 2.5: Coste recursos humanos

Partiendo de estos datos, se tiene que el total de coste relacionado con recursos humanos es de 6922,5 €.

El sueldo de cada tipo de trabajador puede variar según distintos aspectos como la experiencia o la zona donde se trabaje, aunque se ha hecho una estimación según los datos que tenemos. Además se ha acreditado de fuentes externas que permiten conocer los salarios medios. Estas fuentes se mostrarán en las referencias externas. [36]

Por último, a esto se le debe sumar un pequeño margen de costes indirectos relacionados con gastos como el mantenimiento de la zona de trabajo, luz, agua y otros gastos ocasionados. Este margen será de alrededor del 10 %.

Por último, se incluirá un pequeño beneficio que se espera obtener a partir de la realización del proyecto, que será de un 10 % del coste total del producto.

En la tabla 2.6, se indica el coste total de los recursos que se utilizan para la realización del proyecto en conjunto, junto con el beneficio esperado.

Recurso	Coste
Software	0 €
Hardware	62.83 €
Recursos Humanos	6922.5€
Costes indirectos	698,53€
Beneficio esperado	768,38
Coste total	8452,24€

Tabla 2.6:Costos totales

En total, podemos observar que el coste total del proyecto será de 8452.24 €, incluyendo un 10% de ganancia que se espera obtener tras la realización del mismo.

3. Diseño

Tras haber realizado la etapa de análisis y haber obtenido las historias de usuario, los requisitos funcionales y de calidad que debería tener la aplicación junto con la planificación para la realización de las tareas, seguiré con la etapa de diseño en la cual se pasará a explicar el modelo y especificaciones del producto a través de una serie de diagramas y mockups.

3.1 Clases del proyecto

Para la realización del proyecto, se extrae la información de APIs públicas que guardan información sobre listas de canales de televisión y la programación de los mismos. En concreto, para este proyecto se utilizarán los datos que ofrece un repositorio de GitHub abierto y que contiene una lista de canales IPTV gratuitos y de acceso libre junto con los datos de EPG. A continuación se muestra un enlace del repositorio de GitHub.

https://github.com/LaQuay/TDTChannels?tab=readme-ov-file

La estructura de los datos se organiza a través de dos clases principales que son Canal y Programa. Estas clases permitirán almacenar la información de forma estructurada y permiten el acceso y gestión de los datos de forma rápida y sin necesidad del uso de una base de datos, debido a que la información es cambiante y se actualiza con frecuencia.

La extracción de los datos se realizará mediante dos tareas asíncronas (ObtenerListaCanales y ObtenerProgramación), que se encargará de realizar solicitudes HTTP, procesar las respuestas en formato JSON o XML y convertirlas a los objetos convenientes.

Para comprender mejor este proceso, se presenta una descripción de ambas clases junto con un diagrama para cada una de ellas que muestra el proceso de extracción y procesamiento de la información.

3.1.1 Clase Canal

La clase Canal guardará información acerca de los canales de televisión. En la tabla 3.1 tenemos los atributos que guarda la clase Canal junto con una descripción de cada uno de ellos.

Atributo	Descripción
NombreCanal	Guarda el nombre del canal
Web	Guarda el enlace de la web del canal
Logo	Guarda el enlace a la imagen del logo
EpgId	Contiene el nombre que relaciona el canal con la programación
Formato	Almacena el formato de transmisión del canal
Url	Almacena la URL para la transmisión del canal

Tabla 3.1: Clase Canal

3.1.2 Clase Programa

La clase Programa servirá para guardar la información relacionada con cada uno de los programas que se emiten. En la tabla 3.2 tenemos los atributos que guarda la clase Programa junto con una descripción de cada uno de ellos.

Atributo	Descripción
Título	Almacena el título del programa
Descripción	Guarda la descripción del programa
Icono	Guarda el enlace del icono del programa

Categoría	Guarda la categoría a la que pertenece el programa
Inicio	Almacena la fecha de inicio del programa
Fin	Almacena la fecha de fin del programa
Canal	Guarda el identificador del canal en el que se retransmite ese programa

Tabla 3.2: Clase Programa

3.2 Diagrama de casos de uso

En este apartado se representa el diagrama de casos de uso, que muestra las interacciones entre el usuario y las funcionalidades del sistema. Los casos de uso representan las finalidades que el usuario puede conseguir al usar el sistema. Este diagrama permite identificar los requisitos funcionales de manera visual y comprensible. El actor que se muestra en el diagrama es el usuario, que será el que interactúa con el sistema para llevar a cabo las acciones.

La ilustración 3.1 muestra el diagrama de casos de uso del proyecto. En ella se presentan las funcionalidades importantes que debería de poder realizar el usuario una vez esté finalizado el proyecto.

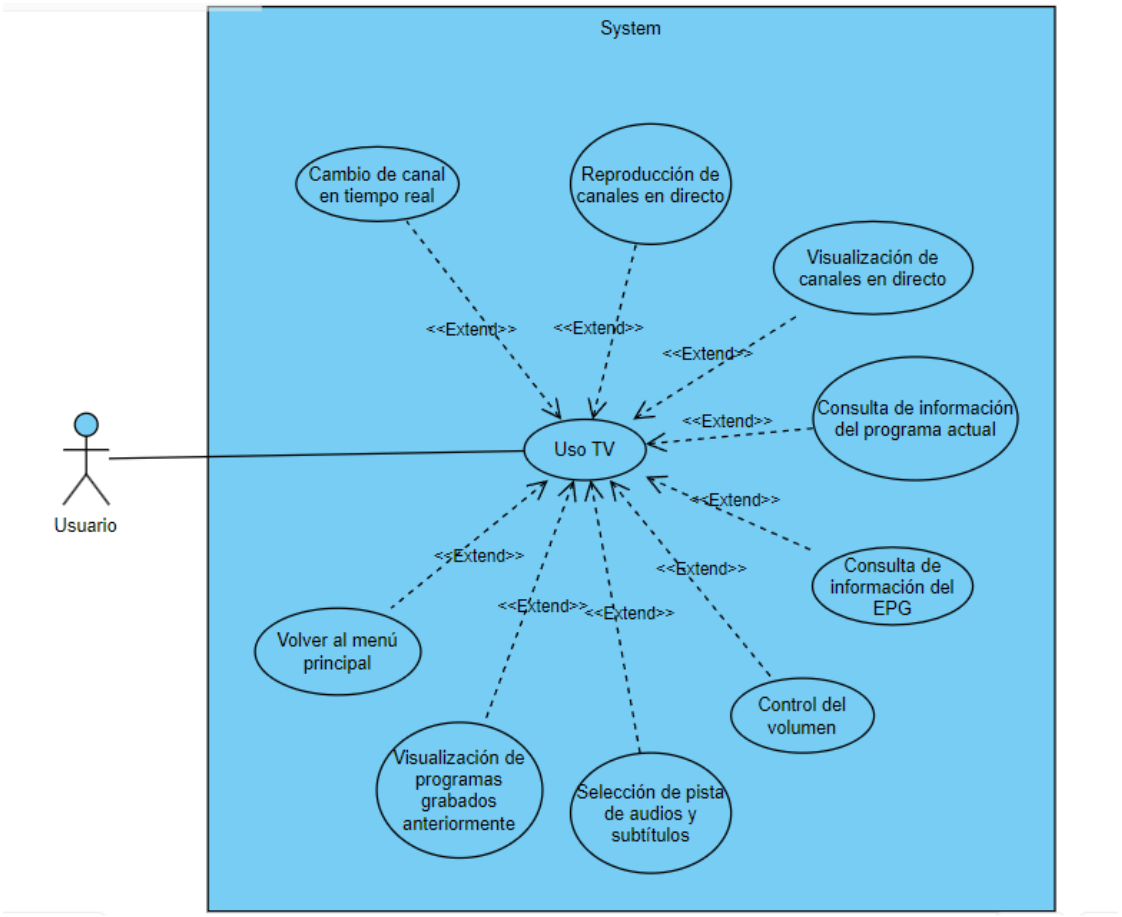

Ilustración 3.1. Diagrama de casos de uso

3.3 Diagramas de secuencia

En este apartado mostraremos los diagramas de secuencia, los cuales nos permiten representar el intercambio de mensajes entre las instancias del modelo y el comportamiento dinámico de la aplicación móvil. En estos diagramas de secuencia se hará un énfasis especial en el tiempo, ordenando las acciones según el mismo.

3.3.1 Extracción de información de la API

La ilustración 3.2 representa un diagrama de secuencia que muestra la serie de pasos que hay que seguir para pasar de tener la información en las APIs públicas hasta llegar a tenerlas guardadas de forma estructurada en cada una de las clases.

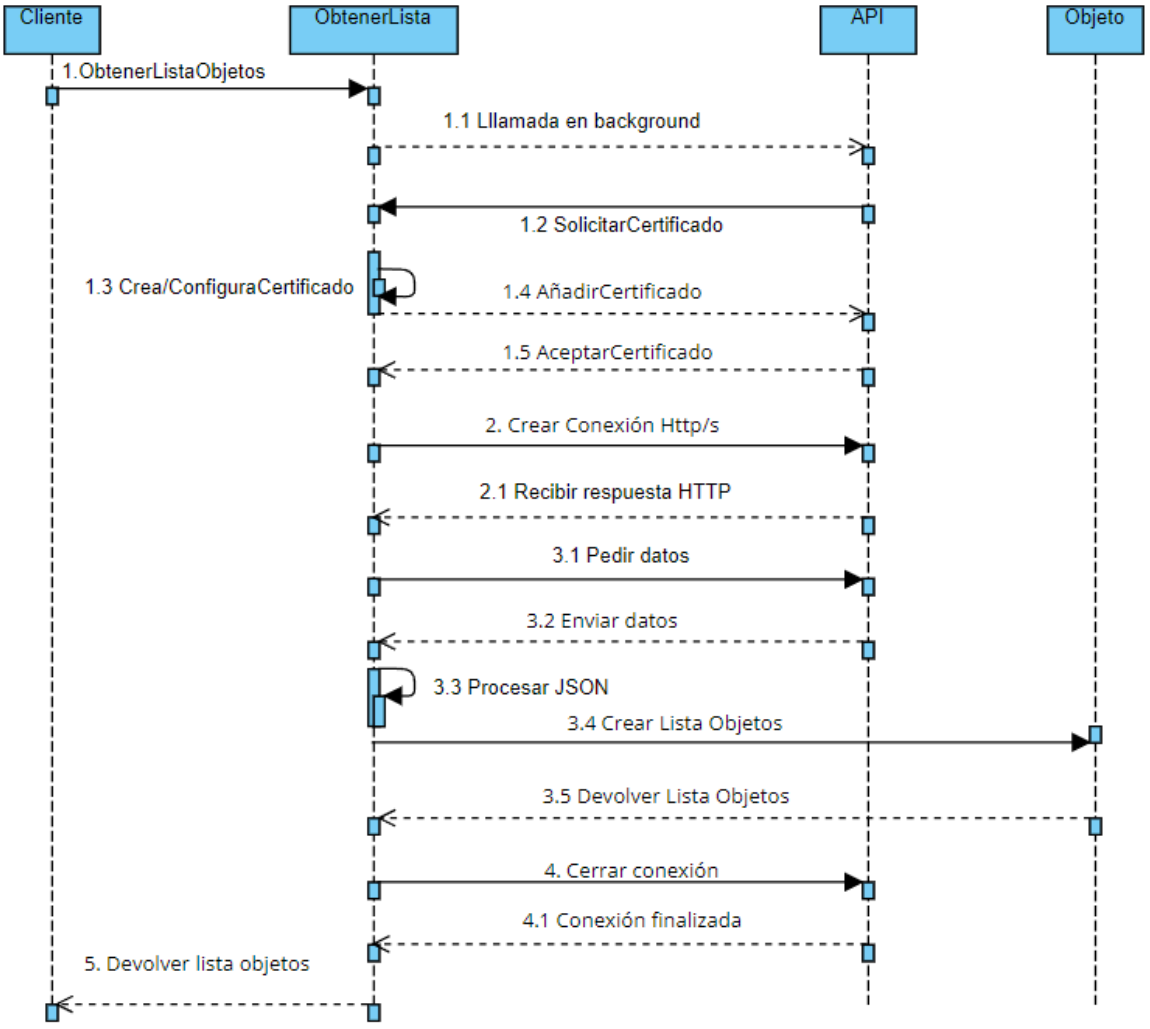

Ilustración 3.2: Diagrama de secuencia Extracción Información API

En este diagrama de secuencia podemos observar una generalización que puede servir para la explicación tanto de obtener canales como de obtener programación. En ambas ocasiones comenzamos llamando al método en background para que el hilo principal no quede colapsado y rompa la ejecución. Siguiendo con la ejecución, se podrá pedir certificado si la conexión es segura. Para esto se creará un certificado, junto con un Keystore [37] que permitirá establecer conexión a través del protocolo https. En este caso, podremos usar una conexión http, por lo que no será necesario el uso de certificado. Después de esto, podremos pedir los datos a la API, que los devolverá en formato JSON o XML. Dependiendo del formato, se procesa de una forma distinta. Una vez procesado, pasaremos a serializar el objeto a partir de la información con una serie de librerías que nos ofrece

gran ayuda para ello como es Gson. Por último devolveremos la lista de objetos serializados y cerraremos sesión con el servidor.

3.3.2 Inicio de la aplicación

Seguiremos con la ilustración 3.3 que muestra el diagrama de secuencia para el inicio de la aplicación, en el que engloba el proceso desde que se arranca la aplicación hasta que empieza a reproducirse el primer canal.

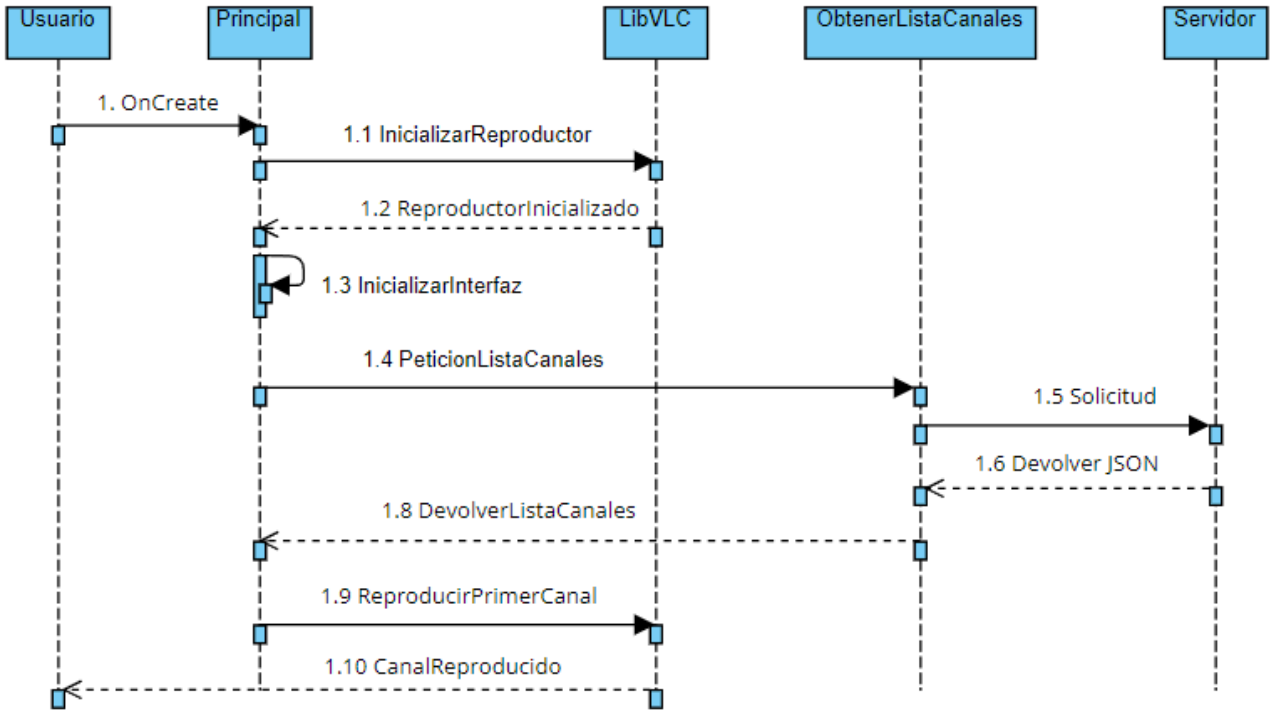

Ilustración 3.3: Diagrama de secuencia Inicio de la Aplicación

En este diagrama se representan una serie de clases las cuales son:

●**Usuario.** Representa al usuario que interactúa con la aplicación.

●**Principal.** Es la clase que gestiona el funcionamiento principal de la aplicación.

●**LibVLC.** Representa a la biblioteca que permite la reproducción de directos.

●**ObtenerListaCanales.** Representa a la clase que gestiona la petición de la información de los canales al servidor y la posterior inicialización de las clases a partir de dicha información.

●**Servidor.** Representa al backend que contiene la información de los canales.

Cuando el usuario abre la aplicación, se ejecuta el método onCreate() de la clase principal, que es el punto de partida de la aplicación. Esta clase se encarga de inicializar el reproductor, además de inicializar también todos los componentes necesarios para la interfaz, como por ejemplo algunos botones necesarios para la interacción o diferentes temporizadores.

Una vez iniciado todo esto, se realizará una llamada a la clase ObtenerListaCanales para que sea ella misma la que haga una petición al servidor que contiene toda la información de la lista de canales y que devolverá en formato JSON todo el contenido. La propia clase de ObtenerListaCanales procederá a parsear toda la información a las diferentes clases para devolverlas a la clase Principal para su posterior tratado.

Una vez la clase principal recibe toda la información de los canales, pasará a reproducir el primer canal. Por último, el usuario percibirá la reproducción del mismo, acabando el flujo principal de inicio de la aplicación.

3.3.3 Seleccionar canal a partir del menú

En la ilustración 3.4 se muestra el diagrama de secuencia que recoge el flujo de interacciones que tienen los diferentes objetos desde que se pulsa el botón del menú hasta que se sintoniza la retransmisión del nuevo canal.

A continuación se muestran los diferentes actores que participarán en el diagrama.

- **Usuario.** Persona que interactúa con el dispositivo.
- **Actividad.** Pantalla que muestra el diálogo.
- **Diálogo.** Componente que muestra el menú de canales.
- **Adaptador.** Adaptador que se usa dentro del diálogo para mostrar los canales.

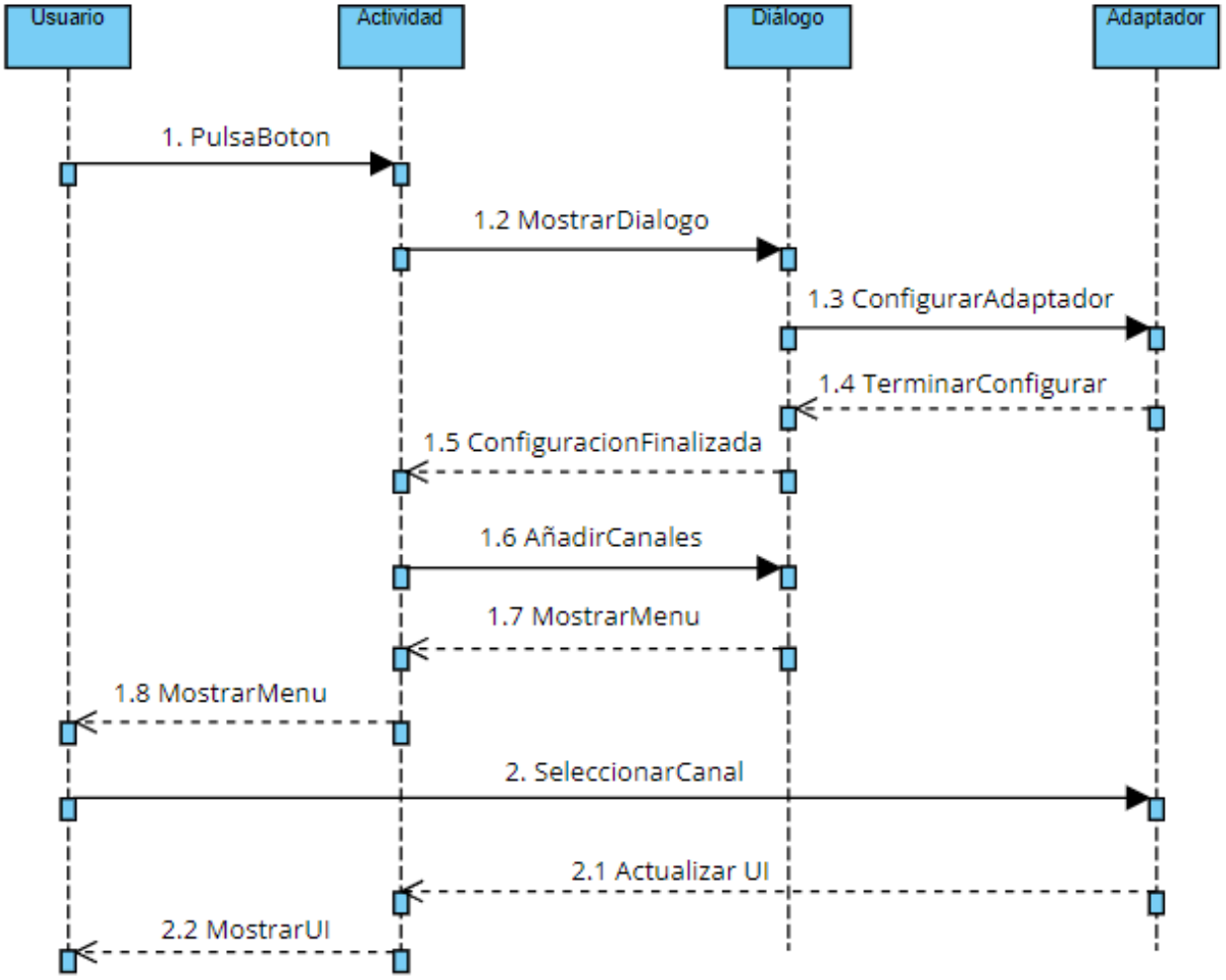

Ilustración 3.4: Diagrama de secuencia Seleccionar canal a partir del menú

En el diagrama de secuencia se puede observar cómo el usuario que interactúa con el dispositivo al pulsar el botón para mostrar el menú, la actividad encargada mostrará el menú, llamado diálogo el cual será configurado a partir de un adaptador. Una vez terminada la configuración del menú, la actividad se encargará de añadir los canales al menú para su posterior visualización. Una vez se ha mostrado el menú al usuario, él mismo se encargará de seleccionar el canal y el adaptador notificará a la actividad que se ha actualizado la interfaz para que esta sea la encargada de mostrar la nueva interfaz al usuario. Con esto último acabaría el flujo.

3.3.4 Reproducción de canal

La ilustración 3.5 representa el diagrama de secuencia que muestra el flujo que se sigue para realizar el cambio de canal a través de los botones que se situarán en cada lateral de la interfaz.

En este diagrama participarán los siguientes actores:

●**Usuario.**Será el encargado de presionar el botón.

●**Actividad.** Será la encargada de controlar la interfaz.

●**MediaPlayer.** Se encarga de reproducir el vídeo.

●**Canal.** Contiene los datos para la reproducción del streaming.

●**Handler.** Temporizador que introduce un pequeño retraso para mejorar la experiencia visual al cambiar de canal.

●**Botón.** Botón que pulsará el usuario.

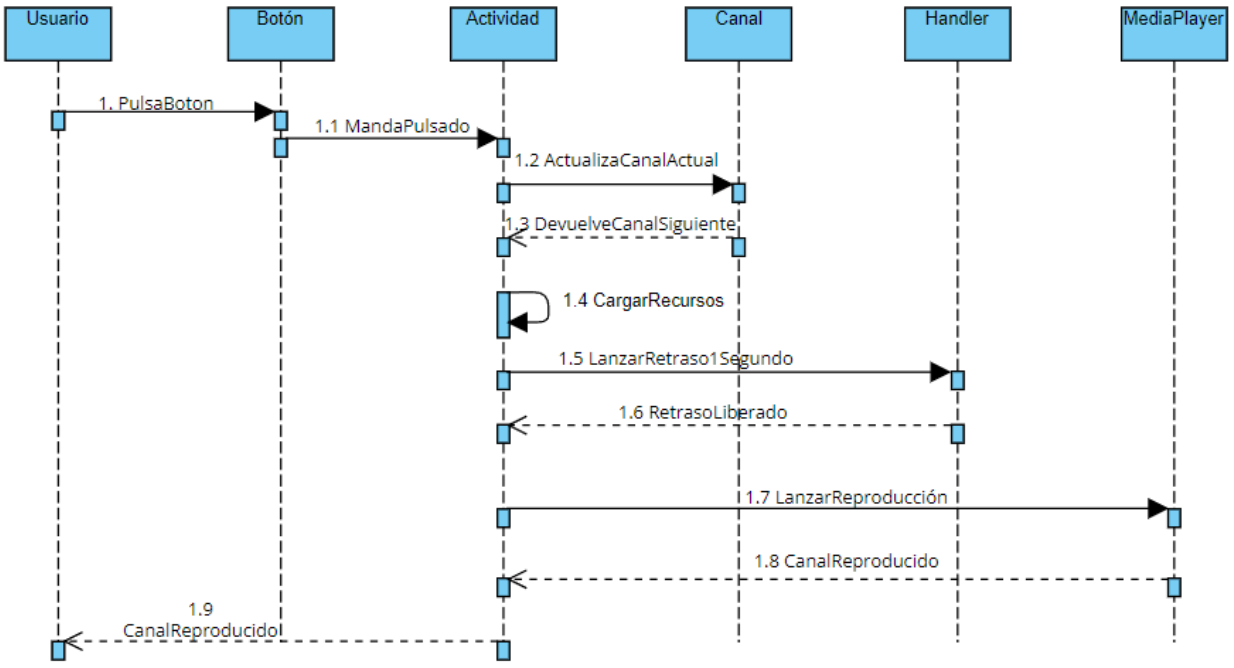

Ilustración 3.5:Diagrama de secuencia Reproducción de canal

En este diagrama se puede ver cómo el usuario es el encargado de pulsar el botón, ya sea el del siguiente canal o el del anterior canal. La clase de botón recibirá el evento y lo comunicará a la actividad encargada de realizar toda la gestión. Esta se encargará de actualizar el canal que se está reproduciendo en ese momento y se

buscará el nuevo para su posterior reproducción. La propia actividad una vez se ha obtenido el siguiente canal procederá a cargar los recursos tales como el nombre del canal, el logo o la información del programa que se está reproduciendo. En esta parte entra en acción el Handler [38], el cual programa una breve pausa mientras se completa la carga de los recursos. Esto se hace con el objetivo de que la transición no sea demasiado brusca y sea más estética. Por último, al terminar la pausa, se notifica a la actividad para proceder a la reproducción del siguiente canal. De esta forma, se visualiza una transición más fluida mientras se cargan los recursos antes de cambiar de canal.

3.3.5 Manejo de eventos del Control Remoto(Teclado o Mando)

La ilustración 3.6 representa el diagrama que muestra cómo la aplicación maneja los distintos eventos que se le pueden mandar a través de teclas de control o de teclado para interactuar entre los canales o hacer otras acciones.

Los actores que participan en este diagrama son:

- **Usuario.** Será el encargado de interactuar con el control remoto.
- **Actividad.** Se encargará de gestionar todo el proceso.
- **MediaPlayer.** Se encarga de reproducir el vídeo.
- **Handler.** Lanzará un pequeño retraso.

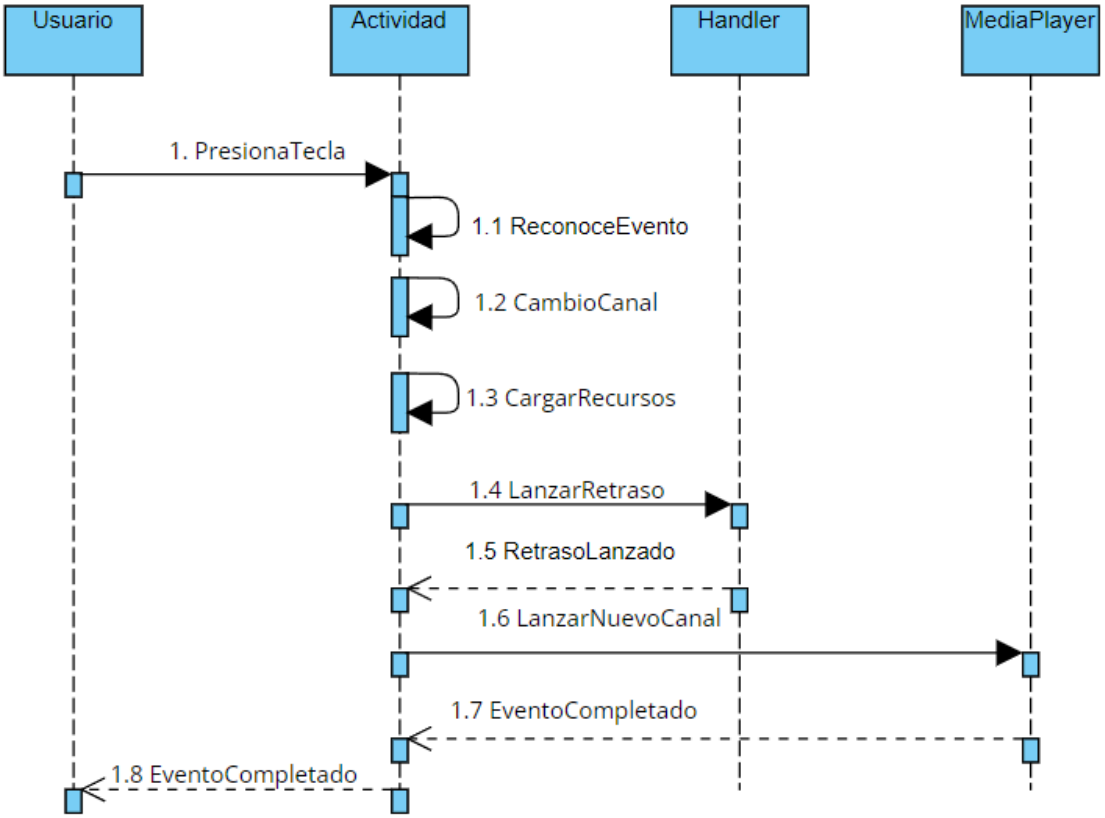

Ilustración 3.6: Diagrama de secuencia Manejo de eventos del Control Remoto(Mando o Teclado)

En este diagrama de secuencia se muestra cómo el usuario manda un evento a través de un control remoto y la actividad principal recoge dicho evento. Una vez lo tiene, reconoce que tipo es para procesar la acción que debe realizar. Estos eventos estarán destinados principalmente al cambio de canal, por lo que después de reconocer a qué canal tiene que moverse, procede a cargar los recursos, a lanzar un pequeño retraso para que se vea más estético y a notificar al MediaPlayer [39] para que reproduzca el nuevo canal, el cual podrá ver el usuario una vez haya acabado este proceso.

3.3.6 Gestión de la visibilidad de la interfaz de reproducción

En este diagrama se muestra cómo se controla la visibilidad de los distintos elementos de la interfaz para el manejo de la reproducción en función de la interacción con el usuario y el temporizador. La idea que se pretende realizar en la aplicación respecto a este ámbito es que la interfaz de usuario aparezca visible al

arranque de la aplicación alrededor de unos cinco segundos. En ese tiempo, si no ha habido ninguna interacción por parte del usuario desaparecerá. En otro caso, seguirá activa y los cinco segundos se reiniciará. En resumen, cada interacción del usuario hace que se muestre la interfaz y se reinicie el temporizador. Después de cinco segundos sin interacción la interfaz se volverá invisible automáticamente.

A continuación vemos los actores que forman parte de este diagrama de secuencia.

- **Usuario.** El usuario interactúa con la pantalla para visibilizar la interfaz.
- **Actividad.** Será la encargada de la gestión de la visibilidad.
- **Handler.** Será el que invoque un retraso de 5 segundos.
- **Planificador.** Gestionará los distintos handlers.
- **Vista.** Será donde se oculten los distintos elementos de la interfaz.

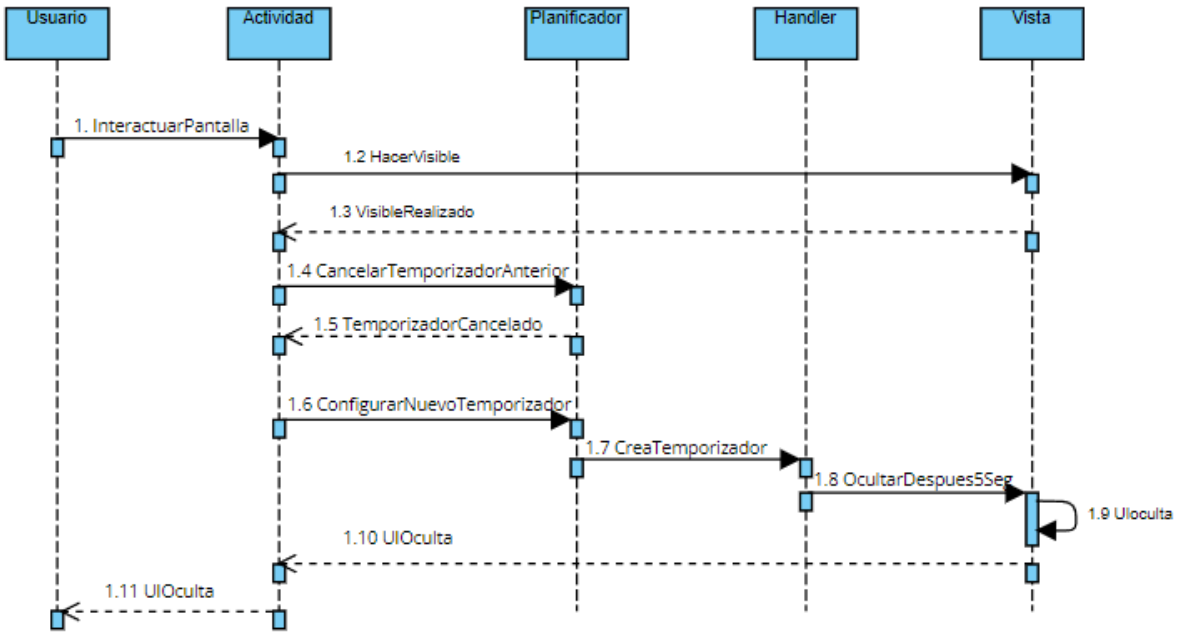

Ilustración 3.7. Diagrama de secuencia Gestión de la Visibilidad de la Interfaz de Reproducción

En la ilustración 3.7, se muestra el diagrama de secuencia comentado anteriormente. Podemos ver cómo el usuario es el encargado de interactuar con el dispositivo a través de la pantalla con un pulsado para hacer visible los controles del reproductor por medio de la interfaz de usuario. Una vez el usuario pulsa, los

controles aparecen en la pantalla y la actividad, que será la encargada de la gestión de todo el proceso, notificará al planificador para que cancele el temporizador que había anteriormente y configure uno nuevo que terminará a los cinco segundos. El planificador creará la instancia del temporizador y a los 5 segundos mandará un aviso a la vista principal para que se oculte de nuevo los controles de la interfaz de usuario, haciendo visible al mismo este proceso a través de la propia vista y de la actividad.

3.3.7 Actualización de la información del programa actual en la UI

Por último, en la ilustración 3.8 podemos observar el diagrama que gestiona cómo se filtra y se actualiza el contenido que recoge la información del programa que se está retransmitiendo a tiempo real para que el usuario pueda ver la información del contenido del canal que está viendo en ese momento, como puede ser el título, descripción y horario del programa.

Este proceso ocurrirá cada vez que el usuario interaccione con la pantalla y muestre los controles de la interfaz.

Los actores que formarán parte de este diagrama de secuencia serán los siguientes.

- **Actividad.** Será el encargado de realizar el proceso.
- **Programa.** Tendrá toda la información acerca de los programas.
- **Canal.** Tendrá información acerca de los canales.
- **TextView.** Servirá para que el usuario pueda ver la información del programa actual.

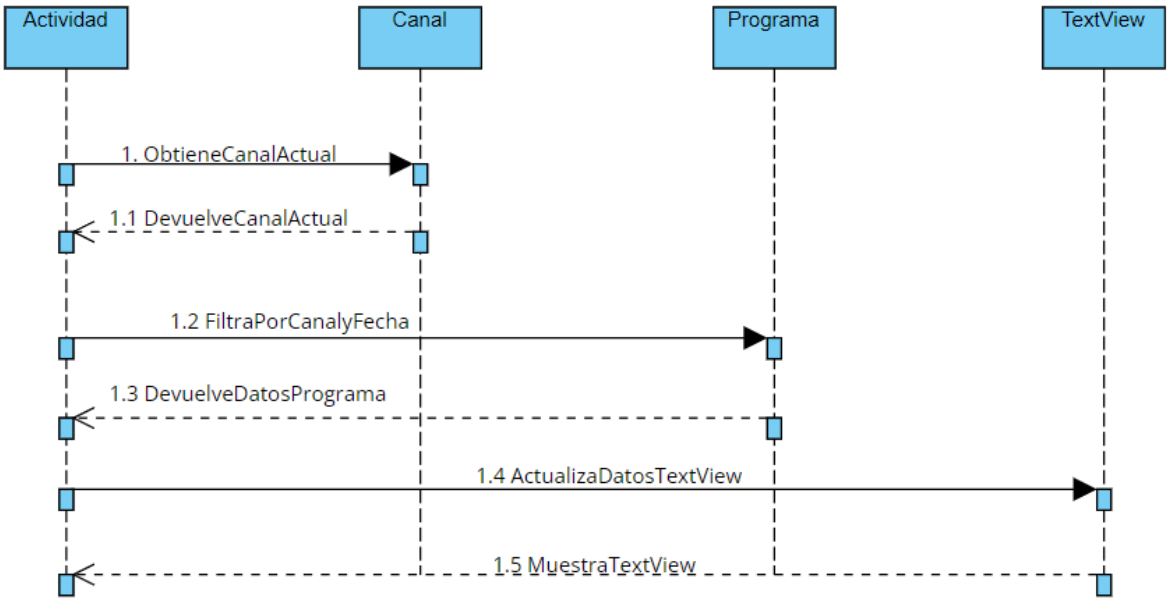

Ilustración 3.8. Diagrama de secuencia Actualización de Información del programa actual en la UI

En este diagrama de secuencia podemos ver como cuando se llama a este método al mismo tiempo que aparecen los controles del reproductor, la actividad consultará el canal que se está transmitiendo en ese momento.

Una vez lo tiene, consultará en la lista de programas e irá filtrando y quedándose con el programa que coincida con el canal actual y además que la hora de retransmisión englobe al momento actual. Una vez ha filtrado eso, devolverá los datos del programa y la actividad será la encargada de enviar al TextView [40] los datos nuevos para que se actualicen y se muestran en la interfaz.

3.4 Arquitectura de la aplicación

La arquitectura de la aplicación se ha planteado usando un enfoque similar al patrón Modelo-Vista-Presentador [41], que es comúnmente utilizado en los proyectos de tecnologías Android. Este tipo de patrón permite una separación de responsabilidades entre los distintos componentes y ayuda a realizar pruebas y a mantener el código en futuras versiones. Ahora se pasa a explicar algunas características sobre este patrón y cómo se llevará a cabo en el código.

El patrón Modelo-Vista-Presentador divide la aplicación en tres componentes distintos, que son:

● **Modelo.** Es el encargado de la lógica de negocio, el control de los datos y las operaciones relacionadas con los mismos, como puede ser el almacenamiento de la información en las clases o la recuperación de la información de ellas. En la aplicación, este componente contiene varias clases que manejan el almacenamiento y obtención de la información a partir de la información que nos proporcionan las APIs externas. Estas clases serían:

○ **Programa.** Guardará la información relacionada con los programas de televisión, incluyendo el título, descripción, hora de inicio y fin.

○ **Canal:** Guardará la información de los canales de televisión como el nombre, identificador y la URL que se utiliza para la transmisión del canal.

○ **ObtenerProgramación.** Será la clase que se encargará de obtener la información relacionada con los programas de la API externa. Esta clase encapsula toda la lógica de obtención y procesamiento de los datos, ya que la clase de Programa será creada por esta entidad, para que luego posteriormente sea usada en la parte del presentador.

○ **ObtenerListaCanales.** El comportamiento de esta clase será similar a la anterior pero esta vez con la información relacionada con los canales. Será la encargada de obtener toda la información de la API y crear la lista de objetos de tipo Canal para su posterior uso. Todo este proceso irá encapsulado en esta clase.

● **Vista.** La vista es la parte de la aplicación que interactúa directamente con el usuario. Este componente estará representado esencialmente por la actividad principal del proyecto. Este componente es el que se encarga de crear y mostrar todos los elementos visuales que hay en la interfaz, además de ser el responsable de recibir todos los eventos producidos externamente por el usuario, como puede ser clics de botones o toques en la pantalla. Por último, será el encargado de mostrar la información actualizada que le proporciona el componente del Presentador.

En este componente se realizarán funciones específicas como mostrar el vídeo según el enlace que se le proporciona a partir de la interfaz que proporciona para ello el motor de LibVLC, mostrar la información actualizada del canal y programa que se está reproduciendo y mostrar los distintos menús para que el usuario pueda interactuar con la lista de canales o pistas de subtítulos y audios.

● **Presentador.** El componente de presentador es el intermediario que hay entre los componentes de la vista y el modelo. Es el que se encarga de recibir interacciones por parte del usuario desde la vista, realizar el procesamiento y actualizar la vista con los datos necesarios que incluye el componente del modelo. En la arquitectura que se ha planteado para esta aplicación parte de la responsabilidad de este componente recaerá en la actividad principal y en algunas otras clases, aunque hay una separación clara en distintas funciones de los componentes de vista y presentador.

El componente del presentador gestionará tareas como la interacción con el modelo. En esta parte, el presentador se encargará de solicitar los datos de los canales y programas al componente del modelo que guardará los datos. Posteriormente realizará cualquier manejo necesario en los datos, como puede ser el filtrado para obtener el programa actual y posteriormente pasará a notificar a la vista para que muestre los datos actualizados. Otra de sus funciones sería manejar las interacciones por parte del usuario, como puede ser el cambio de canal o el pulsado para mostrar los distintos menús. Cuando recibe alumnos de estos eventos, se encargará de obtener la información necesaria de la parte del modelo y pasar a notificar a la vista para que presente la información nueva.

La principal diferencia con respecto al patrón Modelo-Vista-Controlador se presenta en el componente del Presentador y Controlador. Por una parte, en el componente del presentador, se gestiona toda la lógica de presentación además de contener toda la lógica de interfaz de usuario, haciendo que el componente de la vista únicamente se centre en mostrar datos y capturar los eventos del usuario.

Por otra parte, en el MVC, el controlador responde directamente a los eventos del usuario. Además no necesariamente contiene toda la lógica de la interfaz, ya que la vista puede manejar algunos aspectos de la actualización de la interfaz.

En la ilustración 3.9, podemos ver un esquema gráfico de la arquitectura Modelo-Vista-Presentador vista anteriormente. Podemos ver que el componente de presentador se comunica con ambos componentes y sirve de controlador entre ambos.

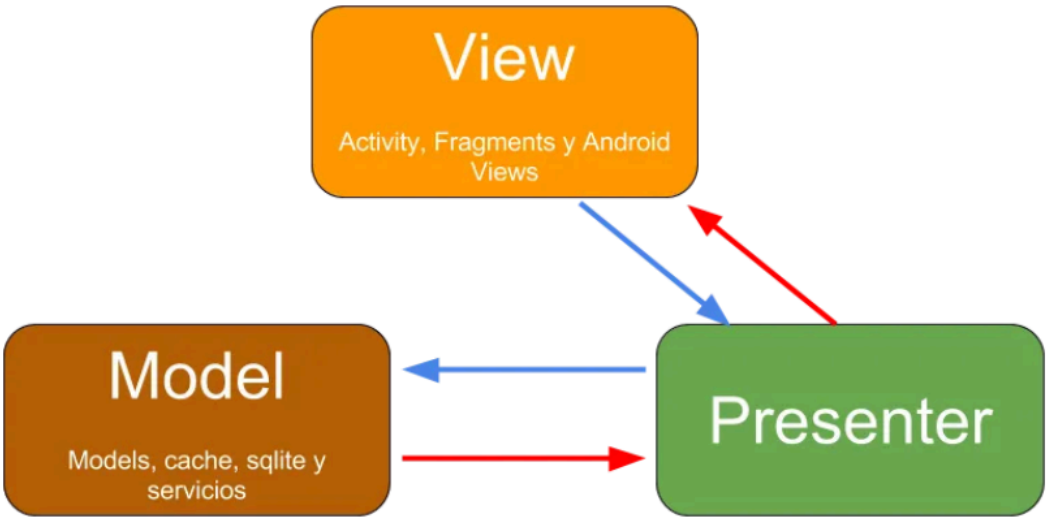

Ilustración 3.9: Representación gráfica del Patrón Modelo-Vista-Presentador

Este tipo de arquitectura proporciona una serie de ventajas como puede ser:

● **Facilidad de pruebas:** Al tener separación entre los distintos componentes, la parte del presentador puede ser probado de forma separada al resto de componentes, lo cual ayuda a encontrar fallos en el proyecto.

● **Separación de responsabilidades:** El hecho de que se separe el código en distintos componentes hace que cada uno de ellos esté especializado en distintas funciones separadas entre ellos, lo que facilita la gestión y mantenimiento del código.

● **Mantenibilidad:** Los cambios que se produzcan en los distintos componentes pueden ser realizados sin afectar a las demás partes de la aplicación. Esto hace que la aplicación sea escalable y permite añadir nuevas funcionalidades a la aplicación.

● **Reutilización de código:** Al tener la separación entre distintos componentes, es más fácil reutilizar código entre ellos.

Para concluir, podemos ver que el patrón Modelo-Vista-Presentador es una buena forma para la implementación de la arquitectura que suele usarse en proyectos de tecnologías Android y que permite tener una clara separación de responsabilidades. Esto permitirá en el futuro el seguimiento y mejora de la aplicación.

3.5 Diseño de la Interfaz

A continuación se va a presentar un mockup del diseño que va a presentar la interfaz del prototipo del proyecto. La interfaz de usuario de la aplicación se va a diseñar de forma que sea lo más simple, accesible y fácil de navegar para el usuario, ya que el objetivo fundamental de esta aplicación es la reproducción correcta de los canales y el control de los mismos una vez esté sintonizado el canal es menos usado.

Se comienza a detallar los componentes principales de la interfaz de usuario y la disposición de los mismos.

3.5.1 Barra de información inferior

En la parte inferior de la interfaz se va a situar una barra horizontal ocupando todo el ancho de la pantalla que proporcionará información relevante al usuario sobre el canal y la información del programa actual que se está transmitiendo. Esta barra estará dividida en diferentes secciones, englobando los siguientes elementos:

- **Logo del canal:** En la parte izquierda de la barra de información se mostrará el logo del canal que se está visualizando. Ocupará todo el alto de la barra y esto permitirá al usuario identificar de forma rápida y sencilla el canal en el que se encuentra.

- **Nombre del canal:** Situado a la derecha del logo del canal, se encontrará el nombre del mismo para identificar de forma completa en el canal en el que estamos.

- **Título del programa:** Debajo del nombre del canal se mostrará el título del programa que se está transmitiendo en ese momento.

- **Descripción del programa:** Bajo el título del programa aparecerá un apartado en el que se añadirá una breve descripción del programa del que se está transmitiendo.

- **Horario del programa:** Por último, al lado del título del programa aparecerá un fragmento que contendrá la hora de inicio y la hora de fin del programa.

Esta barra inferior proporcionará una vista rápida que hará que el usuario pueda obtener toda la información relevante de lo que está visualizando sin interrumpir la experiencia de visualización de la televisión.

3.5.2 Botones de navegación superior

En la parte superior derecha se encontrarán tres botones situados de forma consecutiva horizontalmente que darán acceso a los siguientes componentes de la aplicación.

- **Menú de canales:** Este botón permitirá desplegar un menú de los canales disponibles que puede ver el usuario y además permite el cambio de canal por parte del mismo.

- **Menú de pistas de audio:** Este botón despliega las diferentes pistas de audio que tiene un canal y permite cambiar la pista actual por cualquiera que se elegida de entre todas las que tiene.

- **Menú de pistas de subtítulos:** Este botón despliega las diferentes pistas de subtítulos que tiene un canal y permite cambiar la pista actual por cualquiera que se elegida de entre todas las que tiene.

3.5.3 Botones de cambio de canal

En cada lateral de la interfaz se encontrará un botón que permitirá el navegar entre canales de forma rápida.

- **Botón de canal anterior:** Se situará en el lateral izquierdo y permitirá retroceder al canal anterior del actual.

- **Botón de canal siguiente:** Se situará en el lateral derecho y permitirá avanzar al siguiente canal del actual.

Estos botones permitirán una navegación fluida sin necesidad de abrir el menú de canales, lo que mejora significativamente la experiencia de usuario.

El diseño de esta interfaz se basa en principios de usabilidad y se ha planteado de forma que todas las funciones estén accesibles fácilmente y sean intuitivas para la experiencia de usuario. Además, se ha estructurado la organización de los botones y textos de forma cuidadosa para evitar la sobrecarga de información y permitir que el usuario disfrute de la reproducción sin interrumpir su experiencia.

En la ilustración 3.10, podemos observar una representación gráfica del mockup que servirá para mostrar cómo se va a representar los distintos elementos que componen la interfaz de control del reproductor.

3.5.4 Mockup Interfaz

Ilustración 3.10: Mockup diseño de la interfaz.

3.6 Plan de pruebas

Debido a que estamos usando una metodología SCRUM y como se ha comentado anteriormente en el apartado de historias de usuario, cada una de ellas llevará implícita una serie de criterios de aceptación. Estos criterios de aceptación serían breve descripciones sobre expectativas que deberían de cumplir cada una de las historias de usuario cuando la aplicación sea usada por el mismo. Las pruebas deberán cumplir estos criterios para que se consideren pasadas con éxito.

La tabla 3.3 muestra el conjunto de pruebas que debería de pasar cada historia de usuario para que se considere exitosa cada una de ellas. Para ello, posteriormente se realizará una batería de pruebas para que la aplicación quede lo más robusta y fiable posible.

ID	Título	Criterios de aceptación
1	Reproducción de canales en directo	**1.** El sistema carga y reproduce el canal seleccionado a través de enlace M3U8. **2.** El usuario puede pausar, detener y reanudar la transmisión. **3.** El cambio de canal es rápido y fluido
2	Interfaz reproducción	**1.** El sistema tendrá una interfaz sencilla e intuitiva para su manejo.
3	Cambio de canal en tiempo real	**1.** El usuario cambia de canal sin interrumpir la transmisión actual. **2.** El cambio de canal fluido. **3.** Se muestra la información actualizada del nuevo canal.
4	Visualización de lista de canales	**1.** Se muestra un menú con la lista completa de canales. **2.** Se puede desplazar a través de la lista. **3.** La lista es fácil de entender.
5	Consulta de información del programa actual	**1.** El usuario puede acceder a la información del programa actual. **2.** Se muestra el título del programa, la hora de inicio y fin y la descripción del programa.

		3. Al cambiar de canal, se actualiza la información del programa.
6	Consulta de información de EPG	**1.** El usuario puede acceder al EPG de todos los canales. **2.** Se muestra la programación de las próximas 24 horas. **3.** Se puede navegar fácilmente a través de la lista de EPG.
7	Control de volumen	**1.** El usuario puede cambiar el volumen mientras reproduce un canal. **2.** El control es intuitivo.
8	Selección de pistas de audio y subtítulos.	**1.** El sistema muestra las diferentes pistas de audio y subtítulos disponibles para cada canal. **2.** El usuario puede seleccionar para usar las diferentes pistas disponibles.
9	Visualización de programas grabados anteriores	**1.** El sistema muestra los diferentes programas grabados para los canales disponibles. **2.** El usuario puede seleccionar un programa disponible para reproducirlo. **3.** La reproducción del programa es fluida.

10	Volver al menú principal	**1.** El usuario puede volver al menú principal fácilmente.

Tabla 3.3: Criterios de aceptación del Plan de pruebas

4. Implementación

En este apartado, se detallan los aspectos técnicos de la aplicación a desarrollar. Se explicará cómo se han integrado las distintas funcionalidades detallando las clases principales que componen la aplicación y el proceso por el cual se conectan con los distintos datos que se recuperan de servidores externos. Se describe el proceso añadiendo fragmentos del código, capturas de pantalla y distintas explicaciones para mostrar cómo se han resuelto los diferentes desafíos de la implementación.

4.1 Estructura de la información del servidor

Antes de explicar las clases y métodos que componen la aplicación, es necesario entender cómo se maneja la información que viene de servidores externos. La aplicación recupera la información en formato JSON de dos servidores web distintos, que proporcionan la información necesaria para crear la lista de canales y la lista de programas. Esta información es esencial en nuestra aplicación y su uso correcto, ya que contiene todo lo necesario para la reproducción del vídeo y para mostrar toda la información que debe poseer el usuario mientras está usando la aplicación.

Debido a que el contenido del JSON es muy extenso, se mostrará un breve árbol que ilustrará para cada JSON la estructura y el tipo de campos que tiene, lo cual es fundamental para entender cómo funcionan estos datos y cómo se utilizan los mismos dentro de la aplicación.

4.1.1 JSON Lista de Canales

Se comenzará revisando la estructura que tiene el JSON que guarda la información acerca de los canales de televisión.

En la ilustración 4.1, se muestra un fragmento del JSON que muestra cómo se van estructurando estos datos.

Ilustración 4.1: Estructura JSON Canales

Como presenta la ilustración, podemos observar que existen varios grupos de canales dependiendo del contenido que traten. En este caso, el canal que se presenta estaría dentro del grupo de canales de tipo generalista. Para este canal, podemos ver la información que guarda. En ella incluye el nombre del canal, un enlace web, una url para la obtención del logo o el identificador que se asignará en el JSON de la lista de programas. Por otra parte, tenemos una lista de opciones, la cual guardará una serie de enlaces de retransmisión y el formato al que pertenece. Para la realización del proyecto, nos quedaremos con los canales los cuales el primer elemento de esta lista tenga formato de reproducción de M3U8. Se hace este filtro debido a la excesiva cantidad de canales que presenta este JSON.

4.1.2 JSON Lista de Programas

La ilustración 4.2 muestra la estructura que sigue el JSON que recoge la información acerca de los programas.

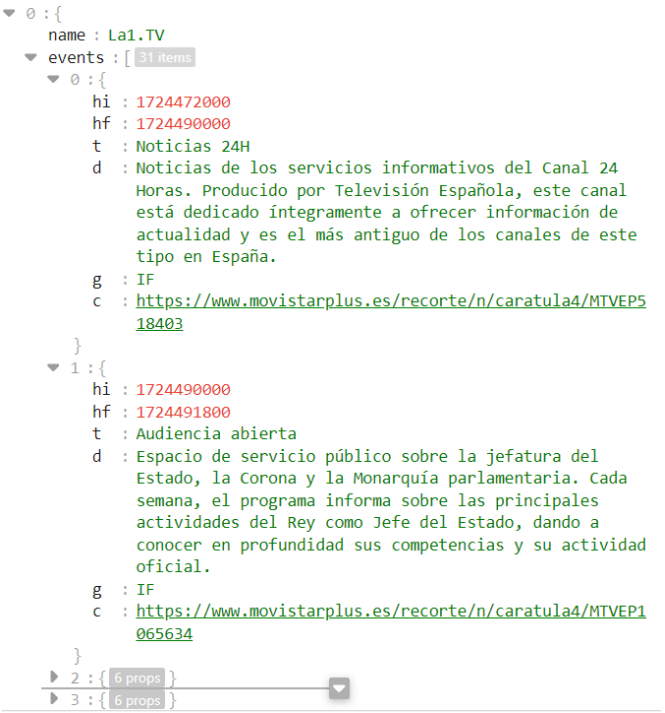

Ilustración 4.2: Estructura Inicial JSON Programas

En esta ilustración podemos ver como hay una lista de objetos que guarda el identificador del canal y una lista de eventos, que será los que recogen la información de los programas para ese canal. Cada evento contiene información relevante como la hora de inicio y fin, el título del programa, la descripción, la categoría a la que pertenece o el logo del programa. Alguno de estos datos nos servirán de gran ayuda en nuestro proyecto.

4.2 Mecanismos para la Obtención y Tratamiento de Datos de Canales y Programación

Las clases “ObtenerListaCanales” y “ObtenerProgramacion” son las encargadas de establecer la conexión con los servidores remotos, además obtienen la información en formato JSON, y por último, procesan los datos para utilizarlos estructurados dentro de la aplicación.

Ambas clases realizan el proceso siguiendo los mismos pasos a la hora de abrir conexión, leer y procesar los datos, aunque se presentan algunas diferencias entre ellas. Veremos a continuación las distintas fases por las que se debe pasar para convertir la información de la red en clases estructuradas.

4.2.1 Apertura de la conexión

Las dos clases implementan la lógica de establecer conexión con los servidores que guardan los datos JSON. Esta operación se realiza en el método “doInBackground”, que se ejecuta en un hilo separado al hilo principal de la interfaz de usuario. Esto se realiza necesariamente para evitar que la interfaz de usuario quede bloqueada mientras se está realizando esta operación tan costosa en tamaño y recursos y mantener la fluidez de la aplicación.

En ambas clases, la conexión inicia creando un objeto que proporciona la API de Java que permite realizar conexiones seguras a través del protocolo HTTPS. Este protocolo es fundamental para asegurar la seguridad e integridad de la información que se transporta, ya que se realiza un cifrado que permite proteger la información contra ataques por parte de otros usuarios no autorizados.

En la ilustración 4.3, podemos observar cómo gestiona la clase de obtención de la lista de canales el proceso. En esta clase, se ha incorporado una comprobación para saber si la conexión necesita el uso de un certificado. Cuando esto ocurre, la clase intenta cargar un certificado que tiene que estar en la carpeta de recursos de la aplicación. Este certificado se utiliza para autenticar al servidor y establecer un canal de comunicación seguro para la transmisión de la información. Si se encuentra el certificado correctamente, se generará un almacén para las claves de seguridad que importa el certificado.

```
@Override
protected List<Channel> doInBackground(String... urls) {
    String urlString = urls[0];
    HttpsURLConnection urlConnection = null;
    try {
        // Intentar cargar el certificado desde res/raw
        InputStream certInputStream = null;
        try {
            certInputStream = context.getResources().openRawResource( id: 0);
        } catch (Exception e) {
            Log.e( tag: "Error",  msg: "Certificado no encontrado", e);
        }

        if (certInputStream != null) {
            // Si el certificado está disponible, configurarlo
            CertificateFactory certificateFactory = CertificateFactory.getInstance( type: "X.509");
            Certificate certificate = certificateFactory.generateCertificate(certInputStream);

            // Crear un KeyStore y agregar el certificado
            KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
            keyStore.load( stream: null,  password: null);
            keyStore.setCertificateEntry( alias: "cert", certificate);
```

Ilustración 4.3: Apertura conexión Obtención Datos I

En la ilustración 4.4, podemos comprobar cómo continúa el proceso de apertura de la conexión. En este punto, una vez tenemos la clave del certificado almacenada en un objeto, creamos un gestor que validará el certificado y permitirá la conexión segura.

```
// Crear un TrustManager que confíe en el certificado
TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(
        TrustManagerFactory.getDefaultAlgorithm());
trustManagerFactory.init(keyStore);

// Crear un contexto SSL que use el TrustManager personalizado
SSLContext sslContext = SSLContext.getInstance( protocol: "TLS");
sslContext.init( km: null, trustManagerFactory.getTrustManagers(), random: null);

// Configurar la conexión para usar el contexto SSL personalizado
urlConnection = (HttpsURLConnection) new URL(urlString).openConnection();
urlConnection.setSSLSocketFactory(sslContext.getSocketFactory());
```

Ilustración 4.4: Apertura conexión Obtención Datos II

Aunque este caso en nuestra aplicación no se va a dar porque esta información es de OpenSource y no hace falta certificado, se ha implementado para añadirle valor a la aplicación, ya que si se cambia de fuente de datos, podría ser necesario incluir un certificado y acudir a esta implementación.

Cuando no es necesario un certificado, la conexión se inicia de manera más sencilla, sin aplicar ningún tipo de configuración de seguridad. Si no hay ningún error en el proceso, la conexión se abrirá para la transmisión de información.

4.2.2 Lectura de los datos

Una vez se ha obtenido la conexión con el servidor, se comienza a leer los datos. Para esta tarea, se utiliza un objeto que permite crear un flujo de entrada de datos desde la conexión. Posteriormente, se añaden los datos a un buffer que facilita la lectura de los datos e incrementa el rendimiento del procesamiento. Una vez tenemos toda la información añadida al buffer, se convierte a formato JSON, lo que permite su uso posteriormente en distintas tareas de la aplicación.

4.2.3 Procesamiento de los datos JSON

Una vez transformada la información obtenida del servidor a formato JSON, las clases se encargan de construir las clases con esos datos para poder usarlas posteriormente en la aplicación. Este proceso se facilita mucho gracias a una librería que permite manipular y trabajar con datos en formato JSON, como lo es GSON.

Para este procesamiento, a partir de las cadenas de texto que contienen los datos extraídos del servidor que se han formado anteriormente, se construye un objeto de tipo JSON. Una vez tenemos este objeto, lo recorremos para obtener la información que necesitamos para nuestro proyecto. Por último se crearán objetos que representarán los canales o los programas respectivamente, y que guardarán los datos extraídos de forma estructurada.

En la ilustración 4.5, podemos apreciar las distintas fases que sigue el procesamiento de los datos en el caso de la obtención de los canales. Podemos ver como se recorre un array que contiene la información y por último, creamos un objeto que representa el canal y que se añade a una lista que contendrá todos ellos.

```
for (int i = 0; i < countriesArray.size(); i++) {
    JsonObject country = countriesArray.get(i).getAsJsonObject();
    JsonArray ambitsArray = country.getAsJsonArray( memberName: "ambits");

    // Recorrer los ambits
    for (int j = 0; j < ambitsArray.size(); j++) {
        JsonObject ambit = ambitsArray.get(j).getAsJsonObject();
        JsonArray channelsArray = ambit.getAsJsonArray( memberName: "channels");

        // Recorrer los canales
        for (int k = 0; k < channelsArray.size(); k++) {
            JsonObject channelObj = channelsArray.get(k).getAsJsonObject();

            String nombreCanal = channelObj.has( memberName: "name") && !channelObj.get("name").isJsonNull() ?
                    channelObj.get("name").getAsString() : "";
            String web = channelObj.has( memberName: "web") && !channelObj.get("web").isJsonNull() ?
                    channelObj.get("web").getAsString() : "";
            String logo = channelObj.has( memberName: "logo") && !channelObj.get("logo").isJsonNull() ?
                    channelObj.get("logo").getAsString() : "";
            String epgId = channelObj.has( memberName: "epg_id") && !channelObj.get("epg_id").isJsonNull() ?
                    channelObj.get("epg_id").getAsString() : "";

            JsonArray optionsArray = channelObj.getAsJsonArray( memberName: "options");
            String format = !optionsArray.isEmpty() && !optionsArray.get(0).isJsonNull() ?
                    optionsArray.get(0).getAsJsonObject().get("format").getAsString() : "";
            String url = !optionsArray.isEmpty() && !optionsArray.get(0).isJsonNull() ?
                    optionsArray.get(0).getAsJsonObject().get("url").getAsString() : "";

            if(url!=null&&!url.isEmpty()&& url.endsWith(".m3u8")){
                Channel channel = new Channel(nombreCanal, web, logo, epgId, format, url);
                channels.add(channel);
```

Ilustración 4.5: Procesamiento datos Lista Canales

Este procesamiento permite que los datos obtenidos del servidor se conviertan en objetos estructurados que podremos usar en nuestro proyecto en funciones como visualizar la lista de canales u obtener el programa actual.

4.3 Estructura de datos para Canales y Programas

Las clases Canal y Programa sirven para almacenar los datos de los canales de televisión y programas respectivamente. Los canales guardarán información como el nombre, url o logo mientras que el programa guardará información como título, descripción y horario.

Estas clases permiten acceder a los datos de manera eficiente y usarlos en cualquier parte de la aplicación

4.4 Funcionamiento principal

La clase Principal es la más importante en el proyecto. Será la que genere toda la interfaz del reproductor. Además esta clase será la encargada de recibir todos los datos estructurados en las clases y procesar la información. Por último, también se encargará de crear el reproductor y asignarle todos los recursos.

Esta clase tendrá un elevado número de atributos, ya que será la encargada de crear todos los botones, los layouts que contendrán toda la información y el reproductor. Se mostrarán a continuación los más importantes:

```
private LibVLC mLibVLC = null;
6 usages
private VLCVideoLayout mVideoLayout = null;
13 usages
private MediaPlayer mMediaPlayer = null;
3 usages
private ManagerDialogo dialogManager;
4 usages
private Button btnAvanzar, btnRetroceder, menu,pistasAudio,pistasSubtitulos;
2 usages
private List<Programa> listaProgramas;
4 usages
ImageView logoImageView;
12 usages
private List<Channel> listaCanales;
18 usages
private static Channel selectedCanal;
8 usages
Calendar calendar = Calendar.getInstance();
3 usages
LinearLayout info;
```

Ilustración 4.6: Atributos Clase Principal

En la ilustración 4.6, se detallan los principales elementos que componen la clase. Entre ellos se encuentran el motor del reproductor, se define el área que muestra la reproducción y se configura el mecanismo de control del reproductor. Por otra parte, se crearán las clases necesarias para el almacenamiento de la información.

Se creará además un componente que se encargará de gestionar los diálogos. Seguidamente se instancian atributos globales que guardan información acerca de lo que se está reproduciendo en este momento. También se creará el área de información que recogerá la distinta información del programa actual así como el espacio donde se añadirá el logo del canal.

Por último, se configurarán mecanismos de sincronización para el control y visualización de los elementos de la interfaz y de control.

4.4.1 Método OnCreate()

En el método OnCreate se inicializan todos los elementos de la interfaz. Además, aquí inicializamos todo lo necesario para el reproductor, como podemos ver en la ilustración 4.7.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_principal);
    final ArrayList<String> args = getStrings();
    mLibVLC = new LibVLC( context: this, args);
    mMediaPlayer = new MediaPlayer(mLibVLC);
    ControladorMediaPlayer playerInterface = new ControladorMediaPlayer(mMediaPlayer);
    mVideoLayout = findViewById(R.id.video_layout);
```

Ilustración 4.7: Método OnCreate() Actividad Principal Inicialización Reproductor

Un detalle importante que debemos tener en cuenta es que la librería necesita una serie de argumentos para su correcto funcionamiento, los cuales veremos a continuación en la ilustración 4.8

```
private ArrayList<String> getStrings() {
    final ArrayList<String> args = new ArrayList<>();
    View decorView = getWindow().getDecorView();
    int uiOptions = View.SYSTEM_UI_FLAG_FULLSCREEN;
    decorView.setSystemUiVisibility(uiOptions);

    args.add("-vvv");
    args.add("--rtsp-tcp");
    args.add("--network-caching=150");
    args.add("--file-caching=150");
    args.add("--clock-jitter=0");
    args.add("--live-caching=0");
    args.add("--clock-synchro=0");
    args.add("--drop-late-frames");
    args.add("--skip-frames");
    return args;
}
```

Ilustración 4.8: Argumentos necesarios librería VLC

Estos argumentos tienen como funciones principales:

●- - **rtsp-tcp.** Usa forzosamente TCP para la transmisión ya que asegura la entrega de paquetes.

● -- **network-caching=150.** Configura la cantidad de tiempo en milisegundos que la librería almacenará datos en los buffers antes de reproducirlos.

● -- **file-caching=150.** Es similar al anterior, pero en este caso con archivos locales.

● -- **clock-jitter=0.** Se utiliza para minimizar el retraso en la reproducción.

● -- **live-caching=0.** Se utiliza a 0 para que la librería intente reproducir lo más cerca posible del instante de tiempo real.

● -- **clock-synchro=0.** Desactiva la sincronización automática del reloj de VLC.

● -- **drop-late-frames.** Se activa para omitir la reproducción de frames de reproducción que llegan en un tiempo tardío, para asegurarnos que siempre estamos en el directo.

● **-- skip-frames**. Se utiliza para saltar frames en general cuando el rendimiento es muy limitado o la decodificación del vídeo es muy exigente.

Por otra parte, en el método OnCreate() a parte de inicializar los valores de todos los atributos se crearán una serie de listener para cada botón que permitirá

llamar a los métodos necesarios para cada uno de ellos, como puede ser pasar al siguiente canal, retroceder al anterior, desplegar cualquier menú…

En la ilustración 4.9, se instancian los listener de cada uno de los botones que aparecen en la interfaz. En ellos se puede ver que cada uno tiene una funcionalidad distinta, como puede ser avanzar de canal o retroceder al anterior.

```
menu.setOnClickListener(v -> {
    mostrarDialog();
});
pistasAudio.setOnClickListener(view -> {
    handler.post(() -> mostrarDialogTracks( tipoTrack: 0));
});
pistasSubtitulos.setOnClickListener(view -> {
    handler.post(() -> mostrarDialogTracks( tipoTrack: 2));
});
btnAvanzar.setOnClickListener(v -> {
    avanzarCanal();
    touch();
});
btnRetroceder.setOnClickListener(v -> {
    retrocederCanal();
    touch();
});
mVideoLayout.setOnClickListener((v) -> {
    touch();
});
```

Ilustración 4.9: Inicialización Listener OnCreate() Actividad Principal

En la ilustración 4.10, podemos visualizar que hay un método al que se llama cuando se presiona la interfaz de vídeo o los botones de cambio de canal. Este método es muy importante debido a que con él, cada vez que interactuemos con la interfaz pulsando en ella aparecerá la misma y mientras la usamos no desaparecerá. Sin embargo, cuando dejemos de usarla, directamente a los cinco segundos sin interacción por parte del usuario en la misma, la interfaz se ocultará automáticamente. A continuación vemos la implementación de este método:

```
public void touch(){
    Log.d( tag: "Touch", msg: "touch");
    mostrar();
    beeperHandle.getAndAccumulate( x: null, (x,y)->{
        if (x != null) x.cancel( mayInterruptIfRunning: true);
        return scheduler.scheduleWithFixedDelay(() -> ocultar(), initialDelay: 5 , delay: 5, SECONDS);
    });
}
```

Ilustración 4.10: Método para ocultar/visualizar Interfaz

Por otra parte, los métodos de *mostrar()* y *ocultar()* tienen una fácil implementación y sólo realizan la acción de volver visible o invisible todos los elementos de la interfaz, pero dará un toque muy estético y profesional a la aplicación.

4.4.2 Método ReproducirVídeoAsíncrono

En la ilustración 4.11, podemos observar la implementación del método el cual será el responsable de reproducir un canal a partir de un objeto que representa un canal.

```
private void reproducirVideoAsincrono(Channel c,int delay){
    calendar.clear();
    calendar.setTime(new Date());
    // Cambiar la URL del Media y reproducir el video

    GlideApp.with( activity: this).load(c.getLogo()).into(logoImageView);
    cambiarInfo();

    channelHandler.removeCallbacksAndMessages( token: null);

    // Programa la tarea para lanzar el canal después de un tiempo de espera (por ejemplo, 1 segundo)
    channelHandler.postDelayed(() -> {
        lanzarCanal(c,  timeShift: 0);
    }, delay);
}
```

Ilustración 4.11: Método reproducción video Actividad Principal

En este método, al iniciar se actualiza un objeto que maneja la fecha actual para que la información nueva que se va a mostrar sea la correcta. Por otra parte se actualiza la imagen del logo del canal al que se ha cambiado. Este cambio se refleja en la interfaz. Posteriormente, se actualizarán los datos del cuadro de texto de información, que ahora contendrá la información del programa nuevo que se está visualizando. A continuación, se eliminan todas las tareas y llamadas pendientes del controlador de tareas para que no haya problemas con la actual y se añade una tarea nueva que lanzará el nuevo canal con un pequeño retraso personalizable que gestionará la sincronización.

4.4.3 Método LanzarCanal

Este método será el encargado de gestionar la reproducción del vídeo en el reproductor. En la ilustración 4.12, se observa cómo funciona este método.

Al iniciar, se comprueba si las vistas donde se añade el vídeo están conectadas al reproductor. Si se comprueba que no están conectadas, se crean y se añaden. Por otra parte, se ajusta al tamaño del marco en el cual se reproduce el contenido.

Posteriormente, se crea un manejador de eventos que permite detectar posibles errores de distinto tipo durante la reproducción del canal.

A continuación, se crea un objeto para representar el contenido del canal usando la URL del mismo. Se habilita la decodificación por hardware debido a la mejora de rendimiento que produce en el reproductor.

Por último, comienza la reproducción.

```
private void lanzarCanal(Channel c, int timeShift){
    if (!mMediaPlayer.getVLCVout().areViewsAttached()) {
        mMediaPlayer.attachViews(mVideoLayout, dm: null, ENABLE_SUBTITLES, USE_TEXTURE_VIEW);
        mMediaPlayer.getVLCVout().setWindowSize(mVideoLayout.getWidth(),mVideoLayout.getHeight());
    }
    mMediaPlayer.setEventListener(event -> {
        if (event.type == MediaPlayer.Event.SeekableChanged) {
            // Manejar el error aquí
            int errorCode = event.type;
            String errorMessage = event.toString();
            Log.d( tag: "MediaPlayerError", String.valueOf(event.getSeekable()));
            // Puedes imprimir o manejar el mensaje de error según tus necesidades
            Log.d( tag: "MediaPlayerError",  msg: "Error Code: " + errorCode + ", Message: " + errorMessage);
        }
    });
    final Media media = new Media(mLibVLC, Uri.parse(c.getUrl()));
    media.setHWDecoderEnabled( enabled: true, force: true);
    mMediaPlayer.setMedia(media);
    media.release();
    mMediaPlayer.play();
}
```

Ilustración 4.12: Método lanzamiento canal Actividad Principal

4.4.4 Método Cambiar Información

En la ilustración 4.13 se muestra la implementación del método que actualiza la información que aparece en el marco inferior de la interfaz que recoge datos como el nombre del canal, título del programa actual, descripción…

```
private void cambiarInfo() {
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat( pattern: "HH:mm");
    String horario = "";
    String programa = "";
    String descripcion = "";
    String canalNombre = selectedCanal.getEpgId();
    Log.d( tag: "CanalNombre",canalNombre);
    for (Programa p : listaProgramas) {
        if (p!=null&&p.getInicio() != null && p.getFin() != null && p.getTitulo() != null) {
            if (p.getChannel()!=null&&p.getChannel().equals(canalNombre)) {
                if (calendar.getTime().getTime()/1000 >= Long.parseLong(p.getInicio()) &&
                        calendar.getTime().getTime()/1000 <= Long.parseLong(p.getFin())) {
                    programa=p.getTitulo();
                    Date fechaInicio = new Date(Long.parseLong(p.getInicio()) * 1000);
                    Date fechaFin = new Date(Long.parseLong(p.getFin()) * 1000);
                    horario = simpleDateFormat.format(fechaInicio) + " - " + simpleDateFormat.format(fechaFin);
                    descripcion=p.getDescripcion();
                }
            }
        }
    }
    TextView canalNombreView = findViewById(R.id.canalNombre);
    TextView programaView = findViewById(R.id.programa);
    TextView descripcionView = findViewById(R.id.descripcion);
    canalNombreView.setText(selectedCanal.getNombreCanal());
    programaView.setText(programa+" "+horario);
    descripcionView.setText(descripcion);
}
```

Ilustración 4.13: Método cambiar Información Actividad Principal

En este método, al comenzar inicializamos las variables necesarias para poder guardar los datos que nos interesan. Además se usa un tipo de formato de fecha que guarda el horario del programa y que solo usa hora y minutos. Posteriormente se hace un filtrado por toda la lista de programas y mirando si coincide con el canal actual y con la fecha y hora actual. En ese caso guardaremos los datos y asignaremos a cada cuadro de texto la información pertinente.

4.4.5 Método DispatchKeyEvent

Por último, para terminar de ver los métodos más importantes de esta clase, veremos el método sobrescrito de dispatchKeyEvent [42].

En la ilustración 4.14 podemos ver la implementación de este método. Se utiliza para poder recoger eventos de dispositivos físicos externos. Se ha programado este método para poder dar funcionalidad a posibles dispositivos de control, aunque hay que personalizar este método dependiendo del tipo de control que se use, ya que cada tecla está relacionada con un número que cambiará según el modelo o dispositivo.

```java
@Override
public boolean dispatchKeyEvent(KeyEvent event) {
    int action = event.getAction();
    int keyCode = event.getKeyCode();

    switch (keyCode) {
        case KeyEvent.KEYCODE_DPAD_UP:

        case 133:
            if (action == KeyEvent.ACTION_DOWN) {
                // Acción cuando se presiona la tecla de flecha arriba
                avanzarCanal();
                return true; // Indica que has manejado el evento
            }
            break;

        case KeyEvent.KEYCODE_DPAD_DOWN:
            // Puedes manejar otros códigos de teclas según tus necesidades
        case 134:
            if (action == KeyEvent.ACTION_DOWN) {
                retrocederCanal();
                // Acción cuando se presiona la tecla de flecha abajo
                return true; // Indica que has manejado el evento
            }
            break;
```

Ilustración 4.14: Método capturador eventos Actividad Principal

Como podemos ver en el código, este método captura los eventos enviados por el usuario a través de un mando o un teclado. Entonces, dependiendo de qué tipo de evento sea se puede realizar la acción que se precise.

4.5 Controlador de Diálogos

La clase ManagerDialogo gestiona la creación y presentación de diálogos personalizados en una aplicación Android. Tiene dos métodos que se utilizarán para mostrar la lista de canales que tiene la aplicación y permite que el usuario seleccione entre ellos y para mostrar la lista de pistas de audios y subtítulos disponible para el canal en el que nos encontremos respectivamente.

A continuación, veremos la implementación de cada uno de estos métodos.

4.5.1 Diálogo Lista Canales

En la ilustración 4.15 observamos el método que será el encargado de mostrar la lista de canales disponibles para que el usuario pueda interactuar entre ellos.

```
// Convertir la lista de nombres a un array
CharSequence[] items = nombresCanales.toArray(new CharSequence[0]);
// Crear un AlertDialog.Builder con diseño personalizado
AlertDialog.Builder builder = new AlertDialog.Builder(context);
LayoutInflater inflater = LayoutInflater.from(context);
View dialogView = inflater.inflate(R.layout.dialog_custom_layout,  root: null);
builder.setView(dialogView);
// Configurar el diálogo con la lista de canales y un listener de clics
ListView listViewCanales = dialogView.findViewById(R.id.listViewCanales);
 jfet0001 *
ArrayAdapter adapter = new ArrayAdapter(context, R.layout.list_item_canal, R.id.textViewNombreCanal, items) {
     jfet0001 *
    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
        View itemView = super.getView(position, convertView, parent);

        // Obtener el logo del canal y establecerlo en el ImageView
        ImageView imageViewLogo = itemView.findViewById(R.id.imageViewLogo);
        Channel canal = listaCanales.get(position);
        GlideApp.with(context) GlideRequests
                .load(canal.getLogo()) GlideRequest<Drawable>
                .into(imageViewLogo);
        return itemView;
    }
};
listViewCanales.setAdapter(adapter);
```

Ilustración 4.15: Creación Diálogo Lista Canales

Esta parte del método será la encargada de crear el diálogo que contiene la lista. En ella podemos ver cómo a partir de un array de objetos de canales se crea un diálogo con un diseño predeterminado definido en un archivo XML que veremos a continuación. Posteriormente, para cada uno de los ítems que componen el diálogo se añade información del mismo como es el nombre o la imagen del logo.

```
listViewCanales.setOnItemClickListener((parent, view, position, id) -> {
    if (position >= 0 && position < listaCanales.size()) {
        Channel selectedCanal = listaCanales.get(position);
        listener.onCanalSelected(selectedCanal);
    }
    dialog.dismiss();
});

dialog = builder.show();
```

Ilustración 4.16: Listener Diálogo Lista Canales

Por último, como observamos en la ilustración 4.16, una vez ha sido creado el diálogo completo con toda la información, se configura un listener para cada ítem, el cual al ser pulsado notificará a la interfaz para que cambie el canal actual. Posteriormente, se cerrará el diálogo

4.5.2 Diálogo Lista Tracks

En la ilustración 4.17 vemos el método el cual se usará para crear, configurar y permitir cambiar la pista actual que tiene el canal reproducido. Este método se utilizará tanto para pistas de audios como de subtítulos, dependiendo del botón de la interfaz que se pulse. Pasamos a ver la implementación.

```
public void mostrarDialogTracks(int tipoTrack) {
    // Crear una lista de nombres de tracks
    List<String> nombreTracks = new ArrayList<>();
    IMedia.Track[] tracks = mMediaPlayer.getTracks(tipoTrack);
    if (tracks != null && tracks.length != 0) {
        for (IMedia.Track track : tracks) {
            nombreTracks.add(track.name);
        }
    }
    // Convertir la lista de nombres a un array
    CharSequence[] items = nombreTracks.toArray(new CharSequence[0]);
    // Crear un AlertDialog.Builder con diseño personalizado
    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    LayoutInflater inflater = LayoutInflater.from(context);
    View dialogView = inflater.inflate(R.layout.dialog_custom_layout,  root: null);
    builder.setView(dialogView);
    // Configurar el diálogo con la lista de tracks y un listener de clics
    ListView listViewCanales = dialogView.findViewById(R.id.listViewCanales);
    jfet0001
    ArrayAdapter adapter = new ArrayAdapter(context, R.layout.item_tracks, R.id.textViewNombreCanal, items) {
        jfet0001
        @NonNull
        @Override
        public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {
            return super.getView(position, convertView, parent);
        }
    };
    listViewCanales.setAdapter(adapter);
    listViewCanales.setOnItemClickListener((parent, view, position, id) -> {
        if (position >= 0 && position < tracks.length) {
            mMediaPlayer.selectTrack(tracks[position].id);
        }
```

Ilustración 4.17: Método Diálogo Tracks

En este método el parámetro principal es el tipo de track que entra por cabecera. En él se podrá diferenciar dos tipos: “0” y “2”, siendo pistas de audio y pista de subtítulos respectivamente. Se utilizará el objeto de la clase de la librería del reproductor que contiene la lista de pistas para cada canal. La creación del diálogo será igual que el anterior por lo que se obviará la explicación de este proceso.

Por último, una vez creado el diálogo, se creará un listener para cada track que aparezca y se usará un método nativo del objeto del reproductor para cambiar al

track seleccionado. Al pulsar sobre alguno de ellos, se cerrará el diálogo y se habrá cambiado la pista.

4.5.3 Estilo Diálogos

Ambos métodos utilizarán el mismo estilo de diálogo, siendo el mismo archivo XML, lo que dará simplicidad a los procesos y permitirá reutilizar gran parte del código.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:background="@color/marcoMenu"
    android:padding="16dp"
    tools:ignore="ExtraText">

    <ListView
        android:id="@+id/listViewCanales"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:dividerHeight="1dp"
        android:divider="@color/dividerColor"
        android:background="@android:color/transparent"
        android:padding="8dp"
        />
</LinearLayout>
```

Ilustración 4.18: XML Diálogos

En la ilustración 4.18 se observa este código XML que muestra la vista que contiene toda la lista de tracks o canales. Dentro del mismo, se creará un una lista de vistas que contendrá la lista de ítems, separados por un barra horizontal y con color de fondo para que sea más estético, como veremos a continuación en la ilustración 4.19.

Ilustración 4.19: Estilo Diálogos.

5. Pruebas y resultados

5.1 Pruebas

En esta sección se describen las distintas pruebas que se han realizado para asegurar el correcto funcionamiento de la aplicación y comprobar que cumple con los requisitos que se indicaron al comienzo del proyecto. Las pruebas se han llevado a cabo durante toda la implementación del proyecto, incluyendo pruebas unitarias, pruebas de integración y pruebas funcionales.

1. **Pruebas unitarias.** Se han realizado pruebas unitarias para todos los componentes de la aplicación, en especial el modelo y el presentador. En especial se trata de que todos los componentes por separado realicen las tareas correctamente, para su posterior integración con el resto de código. En estas pruebas se ha comprobado que se capturan excepciones correctamente y que se devuelven los resultados correctamente. Este tipo de pruebas se han realizado mayormente en las clases de obtención de información, en la que se ha asegurado que se extraen correctamente los datos de las APIs externas.

2. **Pruebas de integración.** Estas pruebas de integración permiten asegurar que los distintos módulos interactúan correctamente entre sí. Algunas de las pruebas serían verificar si se actualizan correctamente los datos en la interfaz cuando cambian los datos de la API o si la vista recibe y presenta los cambios correctamente tras recoger los eventos del usuario.

3. **Pruebas funcionales.** Se han realizado pruebas para comprobar el correcto funcionamiento de la aplicación. Estas pruebas permiten comprobar la experiencia de usuario y asegurar que se cumplen correctamente todas las funcionalidades presentadas.

5.2 Resultados

En este apartado se presentan los resultados obtenidos tras la fase de pruebas. Hemos comprobado que la aplicación es funcional, cumpliendo con los objetivos del proyecto y permitiendo la transmisión de los canales de forma efectiva.

A continuación se muestran distintas capturas que ilustran aspectos visuales de la aplicación.

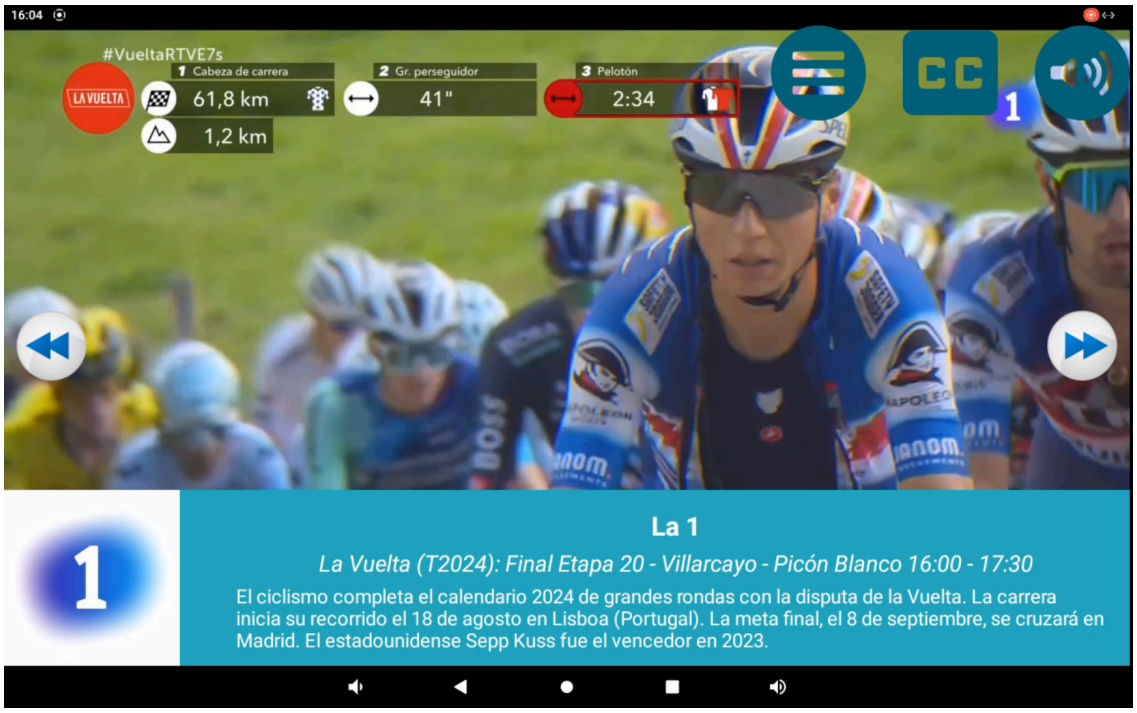

Ilustración 5.1 Reproductor Interfaz Inicio

En la ilustración 5.1 se puede observar la interfaz de usuario que se presenta inicialmente, donde aparecen los distintos botones que incluye la interfaz y el cuadro que incluye la información de la transmisión.

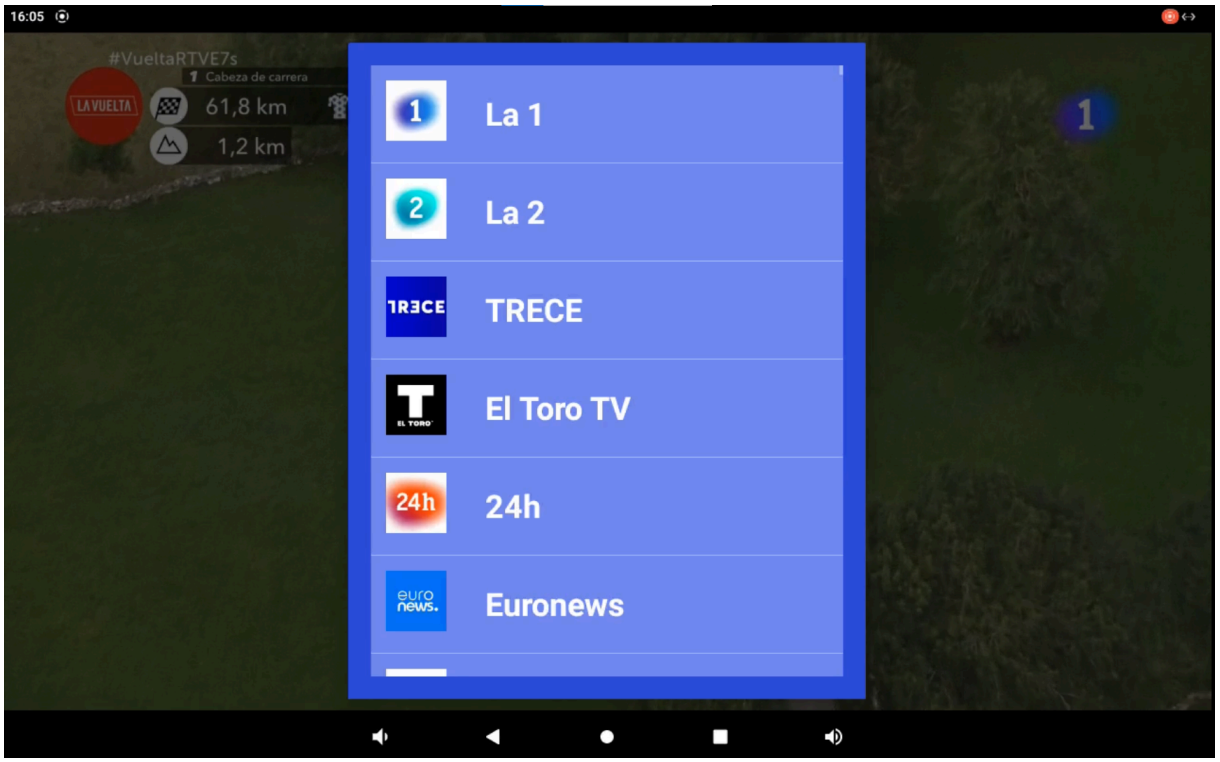

Ilustración 5.2 Menú lista Canales Reproductor

En la ilustración 5.2 podemos ver el menú de la lista de canales que guarda la aplicación.

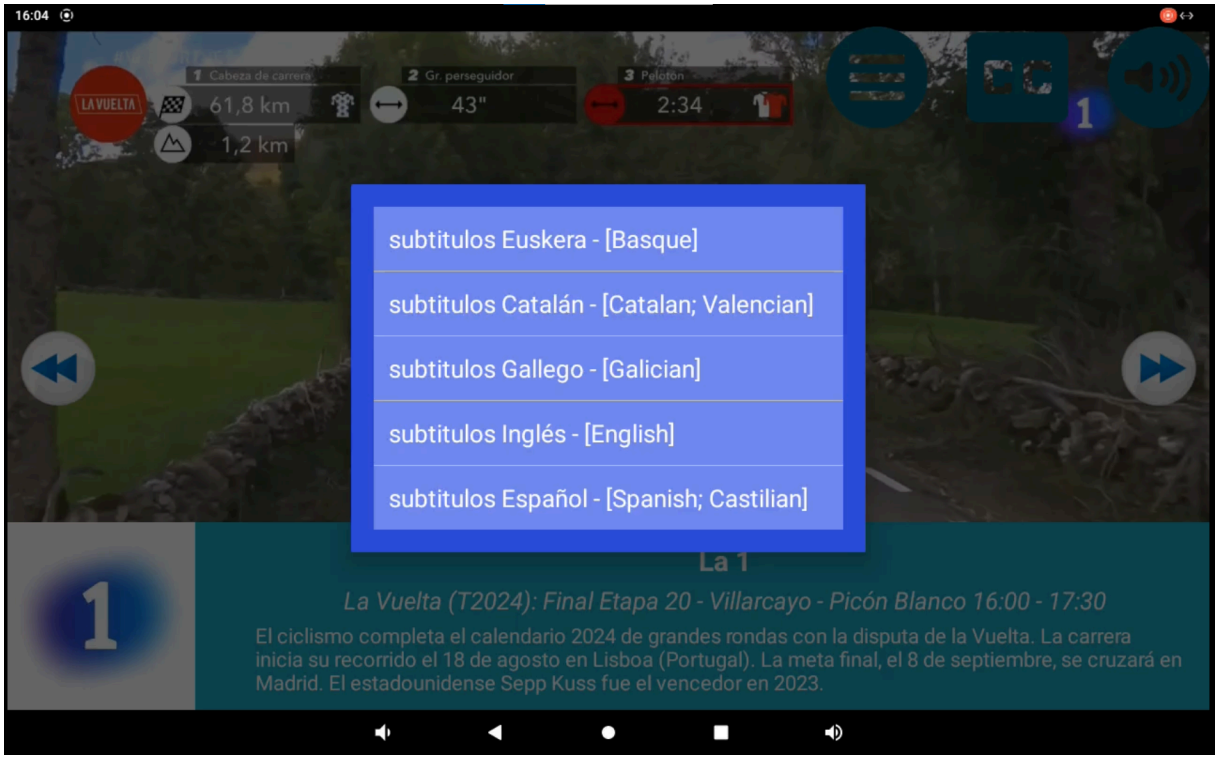

Ilustración 5.3 Menú pistas subtítulos reproductor

Seguimos con el menú de pistas de subtítulos que guarda en canal reproducido, que se visualiza en la ilustración 5.3

Ilustración 5.4 Menú pistas audio reproductor

Aquí observamos en la ilustración 5.4 el menú de pistas de audio para el canal actual.

Por último podemos ver el correcto funcionamiento de los subtítulos en la ilustración 5.5

Ilustración 5.5 Subtítulos reproductor

Además, se adjunta el enlace a un vídeo que muestra el funcionamiento de la aplicación. Aquí podemos ver la interfaz de usuario y la navegación por ella, cambiando de canales o desplegando los distintos menús que presenta la interfaz.

https://drive.google.com/file/d/1N10rnR2v_iUTW1kxkbnPIeV3ySrFC7ik/view?usp=sharing

En conclusión, la aplicación ha cumplido con los criterios de aceptación definidos anteriormente dando una buena experiencia de usuario. Las pruebas realizadas aseguran que la aplicación es funcional y fácil de usar, lo que garantiza que los usuarios puedan usarla sin ningún tipo de problema.

6. Conclusiones y trabajo futuro

En conclusión, pienso que el objetivo que se planteó acerca del proyecto se ha cumplido, logrando obtener una versión funcional que permite a los distintos usuarios ver la televisión desde sus dispositivos Android. La aplicación muestra una interfaz sencilla pero que recoge y permite realizar todas las funcionalidades existentes que presenta la aplicación, lo que hace que la experiencia sea muy intuitiva. Respecto a la planificación que se presentó en el análisis del proyecto no se ha podido seguir completamente debido a compromisos de tema laboral. No obstante, se ha podido finalizar el proyecto en el tiempo dado para ello.

Por otra parte, algunos de los requisitos propuestos al inicio del proyecto fueron resueltos directamente por las características del dispositivo Android, como puede ser el control de volumen, el cual hasta estando en la aplicación permite configurarlo como se desee. Sin embargo, uno de los objetivos que se pidió no ha podido ser resuelto por la naturaleza de los datos, ya que en este caso, los enlaces a los canales IPTV que se han usado no recogen grabación de programas anteriores, por lo que no se ha podido implementar esa parte. Eso podría ser una posible mejora para la siguiente versión. Aún así, he quedado satisfecho con el proyecto, ya que está diseñado para que se pueda usar a largo plazo, mientras los servidores estén disponibles y abiertos a proporcionar datos de canales y programas.

Por último, gracias a este proyecto se ha aprendido de forma significativa. En especial he adaptado un lenguaje que ya conocía por los estudios del grado a un entorno nuevo y que no conocía como es Android.

Por tanto, ha sido una experiencia muy enriquecedora y que seguro contribuirá en mi crecimiento académico y profesional.

Para la mejora de la aplicación en futuras versiones, se podría incorporar la funcionalidad que no se pudo realizar en este proyecto como fue la capacidad de reproducir contenido grabado. Este problema surgió debido a que las fuentes que proporcionaban los datos que se usaban para el proyecto no guardaban ningún tipo de grabación y solamente reproducía el directo. Para poder mejorar el proyecto, se podría buscar otra fuente de datos que sí proporcione datos grabados o explorar distintas opciones de grabado en la nube.

Por otra parte, se podría mejorar la interfaz de usuario, haciéndola más atractiva visualmente, lo que enriquecería la experiencia de usuario en el uso de la aplicación.

Por último, se podría implementar la aplicación para distintas plataformas, como podría ser iOS, la cual tiene un lenguaje de programación específico para su desarrollo. Esto haría que más usuarios pudiesen disfrutar de la aplicación.

Anexo I. Manual de usuario de la aplicación

La aplicación permite a los usuarios ver la televisión a través de una interfaz sencilla y fácil de usar. Permite realizar cambios de canal, visualizar información acerca de lo que se reproduce y además incluye una serie de menús que permiten interactuar con la aplicación.

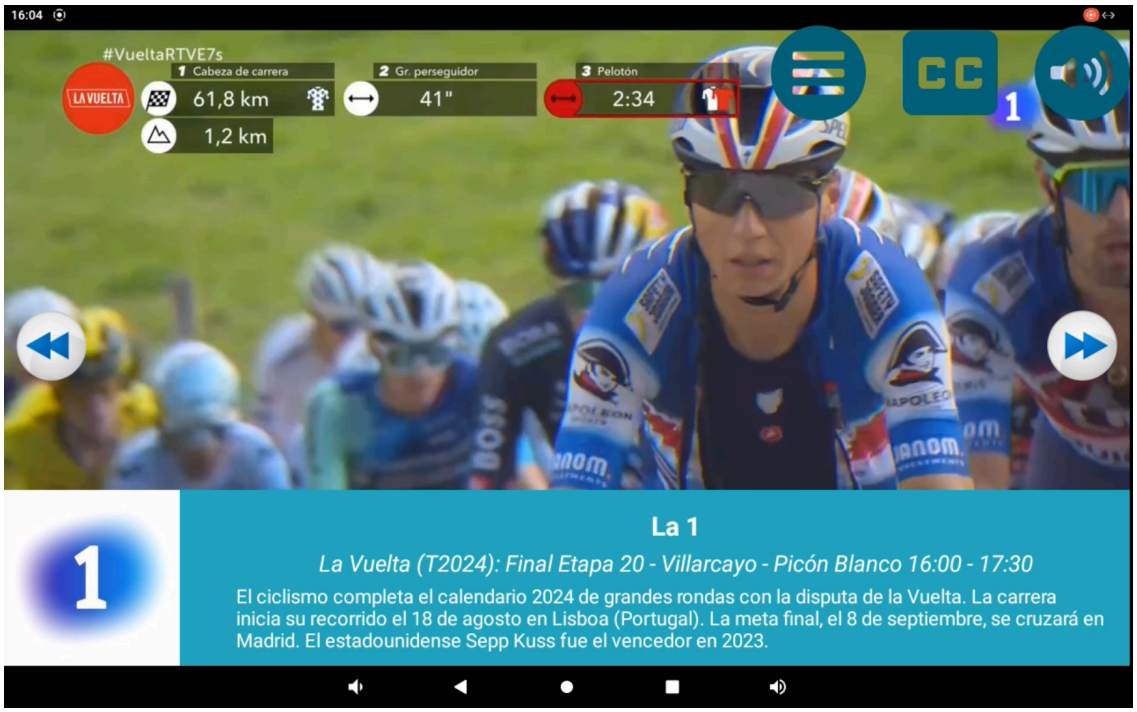

Ilustración 7.1 Interfaz reproducción Manual usuario

●**Pantalla inicial**. Al abrir la aplicación directamente iniciará en el primer canal de la lista.

●**Cambio de canal.** Para poder cambiar de canal podemos utilizar los botones situados en la zona central en los laterales para pasar al canal anterior o al siguiente. Por otra parte, se puede pulsar el primer botón situado en la parte superior de la pantalla, el cual desplegará un menú que contiene la lista de canales y podremos seleccionar un canal para su reproducción.

●**Información reproducción.** El marco de la parte inferior de la interfaz será únicamente informativo, en el cual la información se irá refrescando conforme cambiemos de canal.

●**Control de volumen.** El usuario podrá controlar el volumen del dispositivo a través de los botones que se muestran en la parte inferior de la interfaz, justo debajo del marco informativo.

●**Cambiar la pista de audio.** Para cambiar la pista de audio, el usuario pulsará el tercer botón situado en la parte superior de la interfaz para desplegar un menú con las opciones de audio y así, poder seleccionar uno de ellos.

●**Cambiar la pista de subtítulos.** Para cambiar la pista de subtítulo, el usuario pulsará el segundo botón situado en la parte superior de la interfaz para desplegar un menú con las opciones de subtítulos y así, poder seleccionar uno de ellos.

Anexo II. Manual de instalación del software

Para poder instalar la aplicación en una tablet Android, únicamente es necesario instalar el archivo UJAIPTV.apk en el dispositivo.

Para la compilación de la aplicación a través de un ordenador, se puede utilizar las versiones de aplicaciones y librerías que se han utilizado en el dispositivo que se ha usado para la implementación y la fase de pruebas. Estas versiones serían:

- Android Studio con versión 2023.2.1
- Gradle con versión 8.0
- JDK 1.8

Además, para la realización de pruebas se ha usado una tablet Android con las siguientes especificaciones.

- Android 11 (API 30)
- Pantalla de 15 pulgadas
- Resolución de pantalla Full HD (1920x1800)
- Memoria RAM 4 GB

Anexo III. Listado de entregables

A continuación se enumeran los distintos archivos que se van a entregar.

●TFG_Escudero_Toribio_Juan_Francisco.pdf: Memoría del proyecto

●ProyectoAndroidIPTV.zip: Archivo comprimido que contiene el código fuente

●EnlaceVideo.txt: Enlace a Drive que guarda un vídeo demostrativo del funcionamiento de la aplicación.

●UJAIPTV.apk: Archivo para instalar la aplicación en una tablet Android.

Bibliografía

[1] Android https://wabimovil.es/el-sistema-operativo-android-ventajas-e-inconvenientes/, [Último acceso: 15-03-2024]

[2] M3U8 https://www.adslzone.net/reportajes/software/archivos-m3u/, [Último acceso: 15-03-2024]

[3] Interfaz TV https://www.toptal.com/designers/ui/repensando-la-interfaz-de-usuario-para-la-plataforma-de-la-tv ,[Último acceso: 15-03-2024]

[4] Interfaces daltonismo https://www.linkedin.com/advice/1/what-best-mobile-app-interface-design-practices-uiwlc?lang=es&originalSubdomain=es, [Último acceso: 16-03-2024]

[5] Scrum https://grupoaspasia.com/es/glosario/metodologia-scrum/ , [Último acceso: 17-03-2024]

[6] Kodi https://kodi.tv/ , [Último acceso: 19-03-2024]

[7] VLC Media Player https://www.videolan.org/index.es.html ,[Último acceso: 19-03-2024]

[8] OTT Navigator https://ottnavigator.com/ ,[Último acceso: 19-03-2024]

[9] IPTV https://www.xataka.com/basics/iptv-que-ventajas-desventajas-que-listas-canales ,[Último acceso: 19-03-2024]

[10] Tivimate https://www.xatakamovil.com/aplicaciones/tivimate-trae-cientos-canales-gratis-a-tu-smart-tv-necesidad-antena-asi-funciona ,[Último acceso: 19-03-2024]

[11] Android TV https://www.android.com/intl/es_es/tv/ ,[Último acceso: 30-03-2024]

[12] EPG https://es.wikipedia.org/wiki/Gu%C3%ADa_electr%C3%B3nica_de_programas ,[Último acceso: 30-03-2024]

[13] Media3 https://developer.android.com/media/media3?hl=es-419 [Último acceso: 31-03-2024]

[14] Código abierto https://www.redhat.com/es/topics/open-source/what-is-open-source , [Último acceso: 31-03-2024]

[15] LibVLC https://www.videolan.org/vlc/libvlc.html ,[Último acceso: 31-03-2024]

[16] VideoLAN https://www.videolan.org/index.es.html ,[Último acceso: 07-04-2024]

[17] Glide https://www.glideapps.com/ ,[Último acceso: 07-04-2024]

[18] Bumptech https://github.com/bumptech/glide ,[Último acceso: 07-04-2024]

[19] ImageView https://developer.android.com/reference/android/widget/ImageView , [Último acceso:07-04-2024]

[20] OkHttp https://www.baeldung.com/guide-to-okhttp ,[Último acceso: 07-04-2024]

[21] TCP https://www.hostinger.es/tutoriales/protocolo-tcp ,[Último acceso: 09-04-2024]

[22] Gson https://jarroba.com/gson-json-java-ejemplos/ ,[Último acceso: 09-04-2024]

[23] JSON https://www.json.org/json-es.html ,[Último acceso: 15-04-2024]

[24] Retrofit https://square.github.io/retrofit/ ,[Último acceso: 15-04-2024]

[25] Android SDK https://www.xatakandroid.com/tutoriales/como-instalar-el-android-sdk-y-para-que-nos-sirve ,[Último acceso: 15-04-2024]

[26] Android Studio https://developer.android.com/studio?hl=es-419 ,[Último acceso: 19-04-2024]

[27] IntelliJ Idea https://www.jetbrains.com/idea/ ,[Último acceso: 19-04-2024]

[28] ADB https://developer.android.com/tools/adb?hl=es-419 ,[Último acceso: 19-04-2024]

[29] Apache Cordova https://cordova.apache.org/ ,[Último acceso: 19-04-2024]

[30] Requisito https://es.wikipedia.org/wiki/Especificaci%C3%B3n_de_requisitos_de_software , [Último acceso: 30-04-2024]

[31] Historia de usuario https://www.atlassian.com/es/agile/project-management/user-stories ,[Último acceso: 30-04-2024]

[32] Modelo Invest https://dosideas.com/noticias/metodologias/980-el-modelo-invest-para-crear-historias-de-usuario-efectivas ,[Último acceso: 30-04-2024]

[33] Método MoSCoW https://darioherrera.com/priorizacion-metodo-moscow/ ,[Último acceso: 30-04-2024]

[34] Diagrama de Gantt https://www.atlassian.com/es/agile/project-management/gantt-chart ,[Último acceso: 30-04-2024]

[35] Lenovo https://tecfys.com/blog/post/23-conoces-la-vida-media-de-tu-ordenador#:~:text=Por%20lo%20general%2C%20la%20vida,entre%20los%203%20%2D%205%20a%C3%B1os , [Último acceso: 30-04-2024]

[36] Michael Page https://www.michaelpage.es/ ,[Último acceso: 15-05-2024]

[37] Keystore https://developer.android.com/studio/publish/app-signing?hl=es-419 , [Último acceso: 15-05-2024]

[38] Handler https://developer.android.com/reference/android/os/Handler , [Último acceso: 15-05-2024]

[39] MediaPlayer https://developer.android.com/media/platform/mediaplayer?hl=es-419 , [Último acceso: 05-06-2024]

[40] TextView https://developer.android.com/reference/android/widget/TextView ,[Último acceso: 05-06-2024]

[41] Modelo-Vista-Presentador https://appmaster.io/es/glossary/modelo-vista-presentador-mvp , [Último acceso: 06-06-2024]

[42] Dispatch-Key-Event https://learn.microsoft.com/es-es/dotnet/api/android.views.keyevent.dispatch?view=net-android-34.0 , [Último acceso: 07-08-2024]