TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Implementación de una aplicación cliente para Chromecast.

Autor
IRENE HERRERA LÓPEZ

Director
JORGE NAVARRO ORTIZ

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Septiembre de 2015

Implementación de una aplicación cliente para Chromecast.

Autor
IRENE HERRERA LÓPEZ

Director
JORGE NAVARRO ORTIZ

Departamento de Teoría de la Señal, Telemática y Comunicaciones

—

Granada, Septiembre de 2015

Implementación de una aplicación cliente para Chromecast.

Irene Herrera López

Palabras clave: Chromecast, Android, docencia, Google Cast

Resumen

El proceso evolutivo del ser humano y sus etapas han estado marcadas por avances significativos que han tenido lugar en diferentes ámbitos de la sociedad. Durante los últimos años, la vida del ser humano ha estado ligada a la evolución tecnológica, aprovechando sus ventajas y permaneciendo en una continua búsqueda de nuevas funcionalidades innovadoras que incorporar a su vida diaria.

Así, se han ido insertando dispositivos tecnológicos, cada vez más inteligentes, en el ámbito profesional. En este Proyecto Fin de Grado se pretende aprovechar esta tendencia para hacer del proceso de proyección de material docente una tarea que proporcione al ponente de una mayor libertad frente a las limitaciones que suponen los sistemas cableados actuales.

Para ello, se aunarán las ventajas que presentan los dispositivos de transmisión y reproducción de contenido multimedia, más concretamente, las ventajas de Chromecast, junto a la utilidad de los dispositivos móviles inteligentes o *smartphones*. La conjunción de ambos elementos supondrá una pieza clave para la obtención de una solución *software* que permita al docente o estudiante de la Universidad de Granada proyectar material, con fines educativos, desde su *smartphone* a una pantalla secundaria o panel de proyección de manera inalámbrica, simple y sencilla.

Chromecast application development

Irene Herrera López

Keywords: Chromecast, Android, teaching, Google Cast

Abstract

Human evolution and each one of the periods we have been through, have been marked by significant advances. These great advances have been made in several fields and areas. Over the past few years, people's lives have been bound to technological evolution. They have been taking advantage of the benefits that technology can bring into their lives while they keep looking for new innovative usefulness.

Technological devices have managed to make their way in the integration into the professional environment. These devices have turned out to be smarter as the time passes by. The purpose of this Bachelor's Thesis is to make the most of this trend or movement so we can convert a daily task for teaching staff into something more comfortable. With this project, we want to provide them with freedom to move around the class without wires while they project their work onto a large screen to the audience. This will suppose a better practise to show how to integrate technology into teaching in higher education.

We are going to join and combine the benefits of devices which allow people to play multimedia content on a display by streaming it through a network and smartphones. In particular, we are experimenting with a tiny HDMI stick called Chromecast, which popularity has spread far and wide already. Both elements working in conjunction can get the job done easier. They will be the key to achieve a software solution which permits professors and students screen their presentations from a smartphone to a larger display.

Finally, we will like to mention that the software solution we are expecting to reach will use wireless devices so we can keep it simple to use. Also, it is all about granting access, teaching, learning and collaboration anywhere. This product will link professors to their students and to teaching content, resources and systems to help them improve their own instruction and customized teaching methods.

D. **Jorge Navarro Ortiz**, Profesor del Área de Telemática del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Implementación de una aplicación cliente para Chromecast***, ha sido realizado bajo su supervisión por **Irene Herrera López**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a X de Septiembre de 2015.

El director:

Jorge Navarro Ortiz

Agradecimientos

Han sido muchas las personas que han compartido conmigo esta etapa y a las que me gustaría expresar mi gratitud por las palabras de cariño, apoyo y confianza que siempre me han dedicado. Y no sólo durante el periodo que ha abarcado el desarrollo de este Trabajo Fin de Grado, sino más importante aún, durante todo el proceso formativo que ha supuesto este Grado. No ha sido un camino fácil.

Un sincero agradecimiento a mi tutor, Jorge. Gracias por presentarme la oportunidad de emprender este viaje durante el cual me he dado cuenta de que, con tesón y pasión por el trabajo, no hay ningún lugar al que no se pueda ir, ninguna cima que no se pueda alcanzar.

El sentimiento y la satisfacción de superar obstáculos y alcanzar mis metas también son motivos por los que estar agradecida. Eso es algo que también le debo al resto de profesores con los he tenido la suerte de compartir el camino.

No puedo olvidarme de mi familia y mis amigos, aquellos que me quieren de verdad y lo hacen de manera desinteresada. Gracias Franci por el aliento necesario en los momentos precisos. Gracias por los buenos recuerdos.

Gracias a todos.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción y referencias bibliográficas

"Las enseñanzas orales deben acomodarse a los hábitos de los oyentes."
Aristóteles, filósofo griego (384 a.C. - 322 a.C.)

El presente proyecto puede considerarse como un proyecto de investigación en el ámbito de la tecnología educativa, por lo que en un primer lugar se procederá con una introducción sobre el uso innovador de las TICs (Tecnologías de la Información y Comunicación) en la docencia universitaria.

1.1. Introducción y contexto

La tecnología se ha convertido en un elemento muy importante de cambio del sistema educativo de hoy en día. Y no solamente se refiere este cambio al uso de Internet en las aulas, sino lo que Internet permite hacer tanto a estudiantes como profesores. Años atrás se extendió la concepción de que los dispositivos móviles y redes sociales, entre otros, no deberían estar implicados en una clase, ya que se entendía la filosofía de la enseñanza como una labor puramente humana. Sin duda, la enseñanza ha sido y sigue siendo un proceso interactivo en el que se han usado a lo largo de los años diferentes fuentes que sirven para acentuar el potencial de esta actividad. Por todo ello, se puede considerar la tecnología como una herramienta personal más que como un medio para educar. La tecnología no pretende reemplazar el valor humano, más bien servir como una extensión de éste y configurar nuevos entornos para la formación [1]. En palabras de V. Cerf, *"nuestra tarea es abrazar la riqueza de la evolución de las comunicaciones y dirigirla en direcciones positivas y productivas para el beneficio de todos los que habiten el planeta"*.

Y es que la humanidad ha venido midiendo su progreso históricamente, en términos de tecnología, con el resultado de que cada era ha sobrepasado más rápidamente a las anteriores [2]. A lo largo de la historia, las diferentes tecnologías siempre han ido cambiando las diferentes sociedades donde se han ido implantando. La tecnología puede proporcionar la habilidad a los profesores y estudiantes para adaptarse a ese ambiente cambiante en este sistema educativo a través de la tecnología. El uso adecuado de estas técnicas deben buscar mejorar el proceso de transmisión del conocimiento y, por ende, la calidad de la educación [3].

Algunas de las ventajas más significativas que puede tener la incursión de las TICs en la educación son las siguientes [4]:

- Aumentar la atención, la motivación y la participación del alumnado.
- Facilitar la compresión de los temas, la enseñanza, el aprendizaje y la consecución de objetivos.
- Favorecer la renovación metodológica.
- Aumentar la satisfación, la motivación y la autoestima del docente.

Además se han creado nuevos escenarios virtuales para la comunicación, la enseñanza y el aprendizaje en un entorno digital global interconectado a través de Internet. En este contexto de nodos virtuales, la expansión y evolución hacia la *era de las pantallas* ha modificado los modos de acceso y uso de las redes, adoptando un carácter multidispositivo y con mayor movilidad, pudiendo jugar los dispositivos móviles un papel fundamental en este proceso. Y es en este punto cuando introduciremos el concepto de aprendizaje móvil o *m-learning*. Según algunos autores, el aprendizaje móvil puede entenderse como una evolución del *e-learning* en un contexto en el que se posibilita al alumnado el aprovechamiento de las ventajas de las tecnologías móviles como soporte al proceso de aprendizaje. También se entiende por este concepto aquel aprendizaje que se desarrolla total o parcialmente utilizando dispositivos móviles *(Dyson, Litchfield, Lawrence y Beale, 2005)*. Junto al crecimiento y diversificación del uso de la tecnología móvil es necesario prestar atención a la aparición y rápido desarrollo de las aplicaciones móviles, que proporcionan una serie de características tecnológicas que pueden mejorar los procesos de enseñanza y aprendizaje (portabilidad, movilidad e interactividad) [5]. A pesar de ello, existen estadísticas que destacan el bajo nivel de uso que se hace de la tecnología móvil en los procesos formativos en la actualidad [6].

1.2. Motivaciones

La sociedad se ha acostumbrado a tener acceso permanente y ubicuo a la comunicación e información. Ello ha supuesto un giro en el concepto de

percepción de la realidad y en los hábitos de la comunidad y, por supuesto, también en las aulas y la docencia. Y no es sólo la sofisticación tecnológica lo que nos interesa, sino también la revolución creativa que todo ello supone.

Desde las reuniones en las ágoras de la Antigua Grecia hasta los proyectores multimedia inalámbricos, pasando por pizarras convencionales y transparencias de acetato, han sido múltiples vicisitudes las que se han superado. Pero, ¿cuál es el estado actual de los sistemas de proyección de material de apoyo a la docencia? La mayoría se rigen por dos configuraciones, una primera en la que el aula dispone de un proyector y un ordenador conectado a éste, a disposición de estudiantes y profesorado, para que puedan introducir en él sus dispositivos de almacenamiento y memoria externa. El segundo escenario más común depende de la disponibilidad de un ordenador portátil, y los correspondientes adaptadores, por parte del ponente que va a realizar su clase magistral.

A diferencia de otras TICs, los teléfonos móviles ya están en las manos de alumnos y profesores, ya que son casi universalmente accesibles. Este hecho representa un menor coste que el equipamiento de las aulas con ordenadores y proyectores inalámbricos. Aunque no está ampliamente extendido dentro de los sistemas de proyección de material docente, podría suponer un aspecto a tener en cuenta gracias, entre otras ventajas, a la movilidad y flexibilidad que estos aportan.

Y es por esto por lo que se ha pretendido encontrar una solución que aunara la comodidad de las tecnologías inalámbricas y móviles junto al equilibrio calidad-precio. Todo ello sin olvidar que es importante aprovechar las dotaciones tecnológicas de las que ya disponen la mayoría de las aulas en el ámbito universitario.

1.3. Estudio del estado del arte

Como ya alude el título de este estudio, hemos elegido como componente principal para la solución al sistema de proyección de material docente, el dipositivo Chromecast. Esta herramienta proporcionará la ventaja de eliminar las barreras que presentan los cables. Si bien realizaremos un análisis en profundidad de este pequeño dispositivo en capítulos posteriores, en este apartado se llevará a cabo un estudio de mercado en el cual destacaremos aquellas alternativas que pudieran competir con Chromecast. Para cerrar, y en forma de resumen, se analizarán las ventajas y se defenderá así la elección de Chromecast.

Apple TV

Apple TV [7] es un receptor digital multimedia diseñado por Apple. Este dispositivo en forma de caja permite reproducir todo el contenido de los

dispositivos iOS y Mac vía AirPlay. Además proporciona acceso a diferentes contenidos, disponibles en iTunes.

Su configuración es sencilla, lo único necesario es conectar con un cable HDMI (que se vende por separado) el Apple TV al televisor o monitor. Si además se dispone de un dispositivo iOS, al ponerlo cerca del Apple TV se sincronizarán de forma automática los ajustes de la cuenta de iTunes.

Apple TV combina a la perfección con un iPad junto a las aplicaciones disponibles para ésta, pudiendo sustituir en las aulas a las pizarras electrónicas. Aunque supondría una mejora en el dinamismo y flexibilidad de las lecciones, requiere que tanto el docente como el alumnado dispongan de dispositivos iOS para poder interactuar en el aula, lo que puede suponer un gasto extra. Uno de los objetivos que se buscan alcanzar con la nueva solución propuesta es que sea una alternativa económica y que no conlleve a comprar dispositivos específicos que fuercen a sustituir aquellos que ya poseen los ponentes y las aulas.

Fire TV y Fire TV Stick

El dispositivo Fire TV [8] es equivalente al Apple TV, pero distribuido por Amazon. Además de ser un reproductor de contenido multimedia, incorpora reconocimiento de voz (desde el mando de control remoto) para poder controlar ciertas funciones del dispositivo, así como la posibilidad de conectarle una memoria externa USB 3.0 para ampliar su capacidad de almacenaje.

La configuración y conformación del escenario en un aula es similar al Apple TV, con la ventaja de que, aunque los dispositivos Android también funcionan con Apple TV, tienen menos limitaciones en cuanto a sistema operativo y compatibilidad se refiere.

Por otro lado, también distribuido por Amazon, tenemos la alternativa Fire TV Stick [9] que tiene un formato de *stick* o "pincho". Incorpora también un mando de control remoto aunque sin la funcionalidad de reconocimiento de voz.

Otros productos similares son los de Roku, *Roku 3* [10] (similar a Apple TV y Fire TV) y la versión en "pincho", *Roku Streaming Stick* [11]. Las diferencias que presentan estos dispositivos se recogen en una tabla en la figura 1.1, al final de este apartado.

MatchStick

Como última alternativa posible cabe mencionar una de las más recientes incorporaciones al mercado de los reproductores multimedia en formato *stick* o "pincho", MatchStick [12], basado en Firefox OS. Promete ser una alternativa más abierta, tanto en software como en hardware, y adaptable. Los responsables del producto aseguran la compatibilidad con las aplicaciones existentes para Chromecast (sin cambio alguno en el código), mientras que

las nuevas aplicaciones se podrán adaptar fácilmente con pequeñas modificaciones. Además, como curiosidad, el dispositivo puede ser personalizado con un logo y colores a elegir.

1.3.1. Resumen

Una vez analizadas las distintas propuestas disponibles en el mercado, pasaremos a defender, brevemente, la elección del dispositivo Chromecast.

Aunque no es la solución más económica ya que los precios de MatchStick rondan entre los 18 y 25 dólares, sí es más fácil de conseguir (actualmente no están disponibles los dispositivos MatchStick en el sitio web oficial). Otra razón es la amplia fuente de recursos y documentación disponible gracias a sus desarrolladores y a la propia Google. En cuanto a características técnicas, y teniendo en cuenta los objetivos de este trabajo, cubren todas las necesidades que pudiéramos tener. Presenta la desventaja de no soportar la banda de 5 GHz, lo que pudiera suponer alguna limitación para desarrollos futuros.

	Apple **Apple TV**	Amazon **Fire TV**	Roku **Roku 3**	Roku **Streaming Stick**	Amazon **Fire TV Stick**	Firefox **MatchStick**	Google **Chromecast**
Precio	79 €	99 $	$99.99	$49.99	39 $	$18 - $25	35 €
Conexión	HDMI	HDMI	HDMI	HDMI	HDMI	HDMI	HDMI
Procesador	Chip A5 (un núcleo)	Cuatro núcleos, 1,7 GHz	Doble núcleo	Un núcleo	Doble núcleo	Doble núcleo, 1,2 GHz	Un núcleo
Memoria	2 GB	2 GB	-	512 MB	1 GB	1 GB	512 MB
Almacenaje	8 GB	8 GB	Expansible (microSD)	256 MB	8 GB	4 GB	2 GB
Puertos e interfaces	HDMI, sonido óptico, Ethernet 10/100BASE-T y micro USB	HDMI, sonido óptico, Ethernet 10/100, USB 2.0, DC Jack	HDMI, Ethernet 10/100BASE-T y USB	HDMI, micro USB	HDMI, micro USB	HDMI, micro USB	HDMI, micro USB
Alimentación	Universal, 6W	Universal	12V - 1A	5V - 1A	Micro USB	Micro USB	Micro USB
Red inalámbrica	802.11a/b/g/n	802.11a/b/g/n	802.11a/b/g/n	802.11a/b/g/n	802.11a/b/g/n	802.11b/g/n (2,4 GHz)	802.11b/g/n (2,4 GHz)
Video	H.264, MPEG-4 y JPEG	H.264, H.263 MPEG4-SP Y VC1	MP4 (H.264), MKV (H.264)	H.264, MP4, MKV	H.264	H264, H263, VPX, MPEG4	H.264
Resolución máx	1080p	1080p	1080p	1080p	1080p	1080p	1080p
Audio	HE-AAC, AAC, MP3, AIFF, WAV	AAC, AC-3, E-AC-3, HE-A, PCM, MP3	AAC, MP3	AAC, MP3	AAC-LC, AC3, eAC3, FLAC, MP3, PCM	MP3, AAC, Vorbis, FLAC, APE	HE-ACC, LC-AAC, CELT/Opus, MP3, Vorbis
Fotografía	JPEG, GIF, TIFF	JPG, PNG	JPG, PNG	Desconocido	JPEG, PNG, GIF, BMP	Desconocido	JPEG, PNG, GIF, BMP, WEBP
Control remoto	Mando Apple Remote	Mando Fire TV Voice	Mando incluído	Mando incluído	Mando Fire TV	Desconocido	Smartphone

Figura 1.1: Comparación de las especificaciones de las distintas alternativas propuestas.

1.4. Estructura de la memoria

En esta sección detallaremos la estructura de esta memoria, compuesta por siete capítulos y una serie de anexos y manuales que tienen como fin facilitar al lector conocimiento adicional y necesario sobre ciertos aspectos, así como manuales de instalación, explotación y uso.

- Capítulo 1. Introducción y referencias bibliográficas. En este primer capítulo se sitúa al lector en el contexto adecuado para este proyecto. Tras una breve introducción, se definen una serie de motivaciones que marcarán el eje de objetivos que se pretenden alcanzar. Además, se detallarán algunas de las posibles alternativas a Chromecast que se encuentran en el mercado. Para finalizar el capítulo, se resumirá la estructura del presente estudio y se listarán las principales fuentes bibliográficas.

- Capítulo 2. Planificación y estimación de costes. Aquí se detallarán todos los aspectos relacionados con la planificación temporal de cada una de las fases del proyecto, así como el presupuesto asociado a cada una de ellas y al resto de recursos que han sido necesarios.

- Capítulo 3. Análisis de requisitos. En el tercer capítulo se realizará una introducción a la solución que queremos alcanzar y, para ello, describiremos los requisitos (tanto funcionales como no funcionales) que pretendemos abarcar.

- Capítulo 4. Estudio técnico de las herramientas utilizadas. Este capítulo pretende establecer las bases del fundamento teórico del dispositivo Chromecast, para así conocer mejor las capacidades del dispositivo objeto de estudio en este proyecto. También se realizará un breve estudio sobre el lenguaje de programación para aplicaciones del sistema operativo Android y su historia. Nos centraremos en ciertos aspectos característicos para la programación Android haciendo uso de la SDK de *Google Cast*.

- Capítulo 5. Desarrollo e implementación. En este capítulo se definirá y explicará más detalladamente el grueso de este estudio, el desarrollo de la aplicación cliente para Chromecast. Veremos los aspectos lógicos del diseño y se comentará el código fuente de los distintos bloques que dan forma a la aplicación final.

- Capítulo 6. Pruebas y validación. Una vez finalizada la implementación de la solución *software*, será sometida a una batería de pruebas para comprobar y validar su correcto funcionamiento.

– Capítulo 7. Conclusiones y líneas futuras. Este capítulo será el cierre de este estudio. Se incluirá un resumen en el que destacaremos los principales aspectos y conclusiones obtenidos a lo largo de todo el proceso de este trabajo. Además, se recogerán una serie de posibles mejoras para futuras versiones de la aplicación que se ha implementado.

– Anexos. Por último, esta momeria se complementará con unos anexos finales que se han considerado interesante y que ayudarán al lector de este estudio a comprender y conocer con más detalle algunos aspectos importantes.

1.5. Principales referencias utilizadas

A continuación se exponen las fuentes bibliográficas principales que se han consultado como apoyo a la resolución de este proyecto. Al final de esta memoria se puede consultar una lista más extensa con más recursos bibliográficos consultados.

– *Android Developers*. Sitio oficial que proporciona las SDKs, guías para desarrolladores y referencias para Android. Disponible en: `http://developer.android.com/intl/ja/index.html`

– *Google Cast SDK*. Forma parte de la anterior referencia, *Android Developers*, pero la referenciamos por separado ya que ha sido una fuente de recursos muy importante para la consecución de nuestros objetivo. Disponible en: `https://developers.google.com/cast/`

– *Java Language Specifications*. Manual de consulta del lenguaje de programación *Java*. Disponible en: `https://docs.oracle.com/javase/specs/`

– *Manual de PHP*. Especificaciones del lenguaje de programación PHP. Disponible en: `https://secure.php.net/manual/es/index.php`

Capítulo 2

Planificación y estimación de costes

2.1. Fases de desarrollo

A continuación se van a pormenorizar las distintas etapas o fases que se han recorrido hasta alcanzar la solución final de este trabajo.

- **Especificación de requisitos**
  En esta primera fase, se definieron los requisitos, tanto funcionales como no funcionales, que se pretenden alcanzar al finalizar este trabajo. Se marcan las líneas a seguir y se establece el dispositivo Chromecast como requisito en el anteproyecto realizado por el tutor.

- **Toma de contacto con el dispositivo Chromecast**
  Una vez adquirido el dispositivo Chromecast, se ponen a prueba sus distintas posibilidades y capacidades. Se analizan las aplicaciones disponibles en el mercado que, de alguna manera, interactúan con el mismo y se estudian las respuestas en diferentes entornos de red. El objetivo de esta etapa consiste en la familiarización con el dispositivo.

- **Estudio del estado del arte**
  Una vez conocidos los usos, limitaciones y ventajas de Chromecast, se procede a la comparación con el resto de dispositivos similares, es decir, posibles alternativas, presentes en el mercado. Además, se realiza una investigación acerca del uso y la repercusión de la tecnología en el ámbito docente.

- **Análisis técnico del Chromecast**
  Para justificar las ventajas, inconvenientes y limitaciones que se han ido detectando hasta el momento, es necesario analizar en profundidad las características físicas y técnicas, así como los detalles del fundamento teórico que subyacen al dispositivo objeto de este estudio.

- **Aprendizaje de programación de aplicaciones móviles para Android**
  Una vez tomada la decisión acerca del sistema operativo más idóneo para la resolución del problema propuesto, son necesarias unas nociones básicas de programación de aplicaciones móviles para el sistema operativo Android. Pese a la familiaridad con el lenguaje de programación *Java*, la adaptación a Android requiere determinados conocimientos que se adquirirán durante esta etapa.

- **Estudio de las APIs y SDK necesarias. Evaluación de sus posibilidades**
  Conocidos los principios generales de programación para Android, es necesario analizar en detalle las herramientas específicas para el desarrollo de aplicaciones con el dispositivo Chromecast. Esta tarea es esencial antes de proceder con el diseño de la propia solución, ya que se determinarán aquellos aspectos que se pueden lograr, así como las limitaciones de la tecnología.

- **Análisis de los requerimientos y diseño de la solución**
  En este apartado se presentarán los objetivos que se esperan alcanzar con el presente proyecto. Las etapas anteriores, como ya se ha mencionado, resultarán clave para definir las metas y los límites que debemos sortear.

- **Implementación**
  La implementación de la aplicación móvil o aplicación emisora, con sus respectivos bloques pertinentes (cliente y servidor), así como de la aplicación receptora, conforman el grueso de este estudio.

- **Fase de pruebas parciales y pruebas finales**
  Esta etapa será paralela a los últimos estados de la fase de implementación. El objetivo de las pruebas parciales (o de regresión) es asegurar y comprobar el correcto funcionamiento del *software* y que los cambios que se realicen en el código (e.g. introducción de nuevas funcionalidades) no afectan al comportamiento del resto de la aplicación. En esta etapa se detectan y aislan posibles errores para su posterior corrección. Como última etapa de esta fase, una vez se haya alcanzado la solución final, se procederá a una batería de distintas pruebas sobre la misma.

- **Redacción de la documentación**
  La redacción de la documentación, junto a la revisión bibliográfica, son tareas que se realizan durante todo el periodo de este Trabajo Fin de Grado. Además, son los pilares para la consecución y complementación del mismo.

Una vez identificadas las distintas etapas, las representaremos gráficamente (figura 2.1) mediante un diagrama de *Gantt* en el que se indica, aproximadamente, el tiempo estimado en cada una de las mencionadas fases.

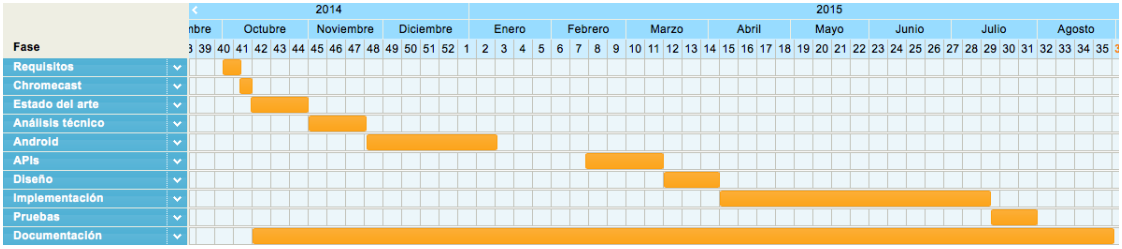


Figura 2.1: Diagrama de Gantt

Para complementar el diagrama de *Gantt*, se ha confeccionado una tabla (ver tabla 2.1), en la que se ha asignado una estimación de horas a cada tarea.

Tarea	**Tiempo estimado (horas)**
Requisitos	12
Chromecast	8
Estado del arte	40
Análisis técnico	40
Android	70
APIs	30
Diseño	30
Implementación	160
Pruebas	30
Documentación	100
	Total: 520 horas

Cuadro 2.1: Estimación temporal para cada una de las fases del proyecto.

2.2. Recursos

En este apartado se van a detallar los distintos recursos que se han usado a lo largo del desarrollo de este estudio.

2.2.1. Humanos

- D. Jorge Navarro Ortiz, Profesor Contratado Doctor del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad

de Granada, en calidad de tutor del proyecto.

- Irene Herrera López, alumna de Grado en Ingeniería de Tecnologías de Telecomunicación, especialidad de Telemática, como autora de este proyecto.

2.2.2. Hardware

- Ordenador personal, MacBook Pro con procesador Intel Core i7 a 2,9 GHz. Este ordenador se ha usado para el desarrollo de las aplicaciones y para alojar los siguientes servicios:
  - Base de datos, MySQL.
  - Servidor FTP, ProFTPD.
  - Servidor Web, Apache.
- Ordenador portátil Sony VAIO, con procesador Intel Pentium Dual-Core a 1,73 GHz, como ordenador auxiliar de soporte.
- Chromecast.
- Monitor LCD con entrada HDMI.
- Smartphone Sony Ericsson Xperia, modelo MT11i y versión de Android 4.0.4.
- Smartphone Huawei, modelo P8 con versión de Android 5.0.
- Smartphone LG Nexus 4, versión de Android 5.1.1.
- Línea de acceso a Internet.

2.2.3. Software

- Sistema operativo *Mac OS X Yosemite* (versión 10.10.4).
- Sistema operativo *Linux Ubuntu 12.04 LTS*, instalado en el ordenador auxiliar.
- Sistema operativo Android, versión 4.0.4, para realizar las pruebas pertinentes en la mínima versión soportada por la aplicación.
- Sistema operativo Android, versiones 4.0.4, 5.0 y 5.1.1.
- Entorno de desarrollo *Android Studio*.
- Android SDK, *Software Development Kit*, conjunto de herramientas necesarias para el desarrollo de aplicaciones Android.

- *Google Cast SDK*, que permite desarrollar aplicaciones para controlar televisores o sistemas de sonido.
- Aplicación *Chromecast*, versiones para Mac OS X, Android e iOS.
- Servidor *XAMPP*, conjunto constituido por un sistema de gestión de bases de datos MySQL, servidor web Apache, servidor FTP y los intérpretes para lenguajes de script PHP y Perl.
- Procesador y editor universal de LaTeX, *Texmaker*

2.3. Estimación de costes

Una vez detallados los recursos necesarios para la realización del presente proyecto y calculadas las horas invertidas de cada una de las fases que lo conforman, se va a realizar una estimación del coste del mismo.

2.3.1. Recursos humanos

En la tabla 2.2 se detalla la relación de horas asociadas a los recursos humanos que se han listado en apartados anteriores, así como el coste que representa el total del tiempo invertido.

	Coste (€/hora)	**Cantidad (horas)**	**Total (€)**
Tutorías	80	15	1200
Trabajo personal	25	520	13000
			Total: 14200 €

Cuadro 2.2: Coste asociado a los recursos humanos.

Desglosando el total de horas del trabajo personal que se definió en la tabla 2.1, se ha realizado el gráfico de la figura 2.2, en el que se pormenoriza el coste asociado a cada una de las fases que conforman este trabajo en función de las horas que se han dedicado a cada una de ellas.

2.3.2. Recursos hardware y software

En la tabla 2.3 se recogen los costes asociados a los recursos hardware y software que se han empleado en el desarrollo de este trabajo.

A parte del precio de los sistemas operativos correspondientes a ordenadores y *smartphones*, implícito en el precio final del dispositivo, el *software* adicional que se ha utilizado es gratuito. Cabe destacar la cuota de cinco euros que hay que abonar para poder registrar tanto el dispositivo Chromecast como la aplicación desarrollada.

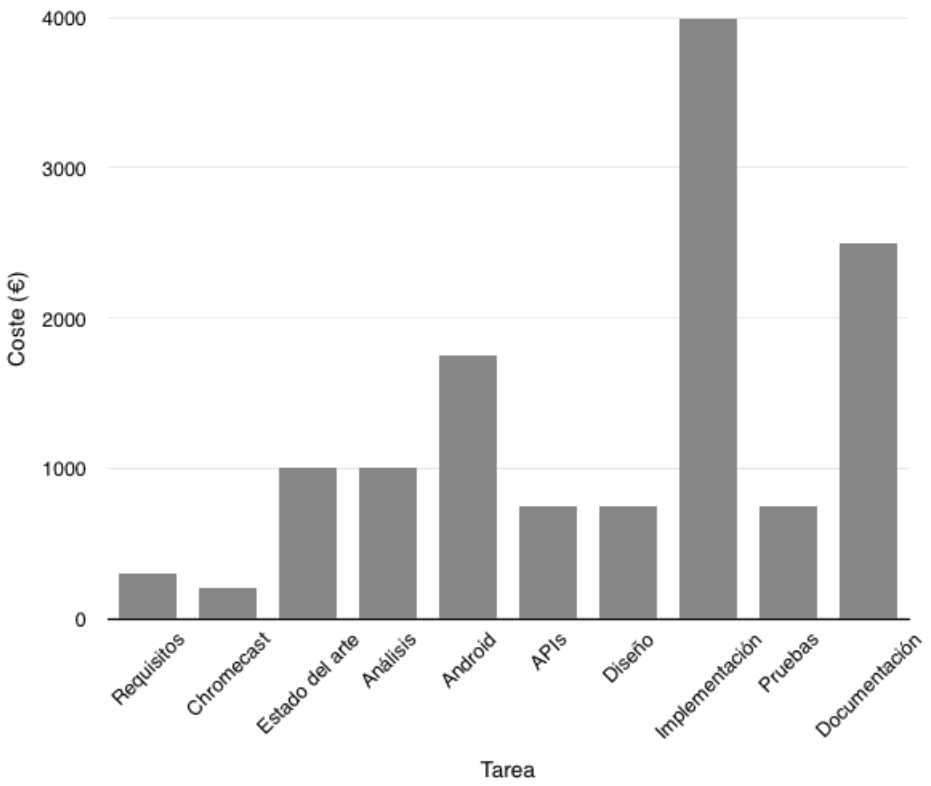


Figura 2.2: Coste asociado a cada una de las fases del estudio.

Recurso	**Coste**	**Cantidad / vida media**	**Total**
Ordenador personal	1200 €	60 meses	335 €
Ordenador de pruebas	600 €	40 meses	150 €
Chromecast	35 €	36 meses	10 €
Monitor con entrada HDMI	300 €	60 meses	50 €
Proyector	350 €	60 meses	60 €
Conexión ADSL	45 €/ mes	-	450 €
Smartphone LG Nexus 4	350 €	36 meses	97 €
Smartphone Huawei P8	499 €	36 meses	139 €
Smartphone Sony Xperia	150 €	36 meses	41 €
Google Cast SDK Developer Console	5 €	-	5 €
			Total: 1337 €

Cuadro 2.3: Coste asociado a los recursos hardware y software.

2.4. Presupuesto total

Haciendo la suma total de las diferentes partes del presupuesto que hemos ido detallando a lo largo de este capítulo, el coste estimado para este Trabajo Fin de Grado asciende a un total de 15537 euros (tabla 2.4).

Recursos humanos	14200 €
Recursos materiales	1337 €
Total:	**15537 €**

Cuadro 2.4: Presupuesto total del proyecto

Capítulo 3

Análisis de requisitos

En anteriores capítulos se realizó una introducción acerca de los objetivos que se pretenden alcanzar con este estudio. En este apartado, procederemos a detallar los requisitos funcionales y no funcionales que motivarán la fase posterior de diseño y desarrollo.

Antes de comenzar con la descripción de los requerimientos, vamos a recordar los objetivos básicos de este proyecto. La idea principal es desarrollar una aplicación cliente para el dispositivo Chromecast, que permita a los usuarios enviar contenido multimedia (con fines docentes) a una pantalla secundaria, como pudiera ser un monitor o panel de proyección. Además, se pretende que ese proceso sea sencillo y transparente para el docente o ponente. Buscando la comodidad de éste, se procuran reducir las conexiones cableadas para así dotar al usuario de una mayor movilidad. Como último objetivo, y no menos importante, se intenta alcanzar una solución que suponga un ahorro económico en cuanto a inversión en la infraestructura del aula se refiere, por lo que se utilizará un *smartphone* para alojar la aplicación cliente.

En resumen, este trabajo se sitúa en el área de desarrollo de aplicaciones móviles para su uso en el ámbito de la docencia.

3.1. Requisitos funcionales

A continuación se va a realizar una descripción más completa del problema a resolver, es decir, qué debe hacer la aplicación, bajo qué condiciones debe funcionar o cuáles serán los criterios para comprobar que dicha aplicación funciona.

- **Control de sesiones.** La aplicación tendrá que ser capaz de conectarse a una base de datos en la que se alojará la información referente a los usuarios que tienen acceso a la aplicación.

- **Gestor de archivos.** La aplicación debe permitir y facilitar al usuario la búsqueda del material que desea proyectar. Dicho material deberá estar almacenado en la memoria externa (entiéndase memoria externa como tarjeta de almacenamiento del dispositivo móvil).
- **Conexión inalámbrica.** La aplicación ha de gestionar de manera transparente al usuario la conexión inalámbrica, en nuestro caso, con el dispositvo Chromecast. Además, el usuario ha de saber mediante un indicador visual el estado de dicha conexión en cualquier momento.
- **Envío de material docente.** Como último requisito y propósito final de la aplicación, esta deberá permitir al usuario enviar contenido a un servidor a partir del cual, el Chromecast será capaz de proyectarlo en una pantalla o panel de proyección al que se haya conectado con el objetivo de ser visualizado.

3.2. Requisitos no funcionales

Una vez definidos los requisitos funcionales, pasaremos a los requisitos no funcionales.

- **Chromecast.** Este es uno de los requisitos no funcionales más restrictivos que se presentan en este proyecto. Este pequeño, pero a la vez potente, dispositivo fue presentado en el anteproyecto por el tutor como una condición o cláusula indispensable para la obtención de la solución final. Como motivación adyacente, el coste económimo y las diferentes posibilidades que presentan las APIs de Google suponen una gran ventaja frente a otros dispositivos similares.
- **Android.** Enlazando con el punto anterior, la documentación, APIs y recursos disponibles sobre el sistema operativo Android y su compatibilidad y funcionamiento con el Chromecast han sido un punto clave para elegir esta plataforma como la adecuada para la implementación de esta primera versión de la aplicación. Además, se intenta alcanzar al máximo número de *smartphones* posibles, por lo que se establecerá como mínima versión de Android la versión 4.0.3 (*IceCreamSandwich*), que corresponde con la API de nivel 15. Al abarcar APIs de menor nivel, se alcanzarían más dispositivos pero nos encontraríamos con la desventaja de que algunas funcionalidades no estarían disponibles. Por ello, al establecer como requisito mínimo la API de nivel 15, aseguramos que nuestra aplicación funcione en, aproximadamente, el 94 % de los dispositivos Android activos en *Google Play Store*. En la figura 3.1 podemos comparar el nivel de alcanzabilidad que tendría nuestra aplicación en función del mínimo nivel de API (o mínima versión Android) que elijamos.

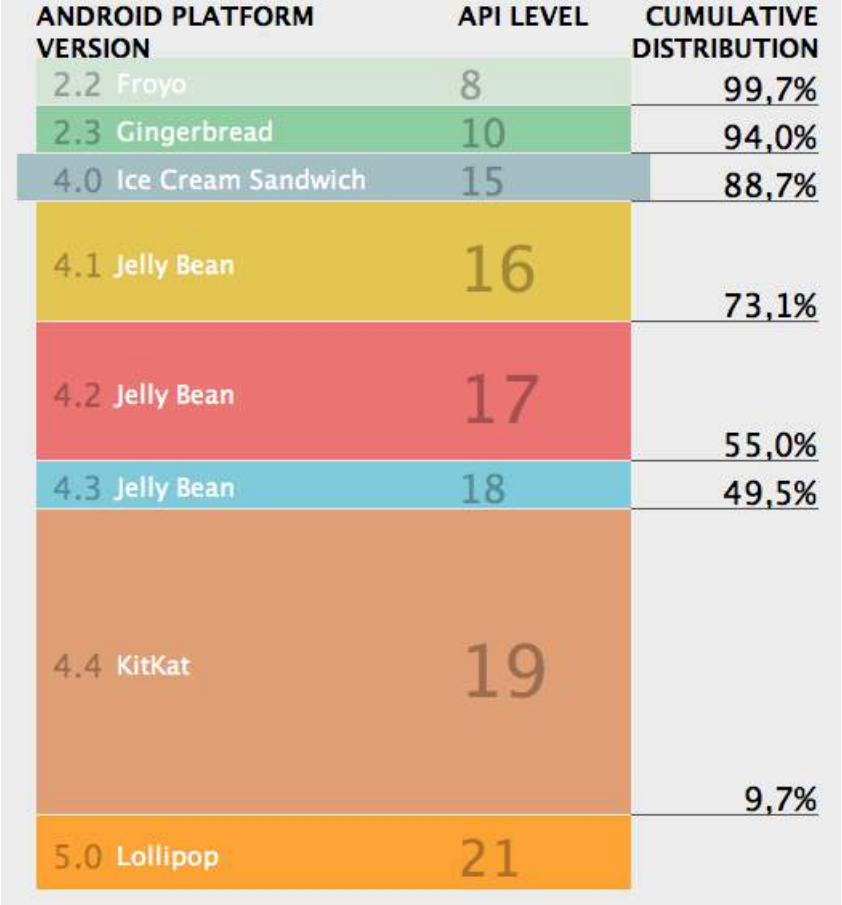


Figura 3.1: Nivel de alcanzabilidad de nuestra aplicación en función del nivel de API elegido.

- **Usabilidad.** Se debe facilitar la interacción entre el usuario y el sistema, para ello la interfaz de usuario ha de ser intuitiva y de manejo sencillo. En aquellos casos en los que se considere necesario, se añadirán mensajes para guiar al usuario en el proceso.

Capítulo 4

Estudio técnico de las herramientas utilizadas

4.1. Chromecast

En este apartado vamos a centrarnos en el dispositivo Chromecast, elemento principal del presente estudio, analizando sus características técnicas y especificaciones.

4.1.1. Presentación

Este dispositivo viene presentado en una caja (figura 4.1a), junto al resto de elementos necesarios para su funcionamiento, como son el cable de alimentación de corriente, el cabezal de adaptador de corriente y un adaptador HDMI (figura 4.1b). Tiene unas dimensiones de 72 mm x 35 mm x 12 mm y un peso de 34 gramos.

En primer lugar, en el extremo opuesto al puerto de salida HDMI (figura 4.2a) del dispositivo Chromecast, podemos observar el puerto USB de alimentación (figura 4.2b). El dispositivo requiere de alimentación externa, vía USB o haciendo uso del adaptador que incluye el set. Se recomienda, por comodidad, el uso del cable de USB, siempre que la televisión, proyector o cualquier otro receptor disponga de un puerto USB para ello. Justo al lado de la entrada de USB, podemos ver un pequeño botón que sirve para reiniciar el dispositivo, manteniéndolo pulsado unos 25 segundos.

En la cara posterior (figura 4.3), se detalla una serie de información relevante acerca del dispositivo. Los datos más importantes, y que más destacan, son el modelo del dispositivo (H2G2-42, en nuestro caso) y el número de serie del producto. Este último, el número de serie, será necesario a la hora de desarrollar aplicaciones, como es el caso de este trabajo.

(a) Contenido de la caja.

(b) Detalle del contenido de la caja.

Figura 4.1: Dispositivo Chromecast.

4.1.2. Estudio técnico.

Una vez que hemos inspeccionado el exterior del pequeño dispositivo, podemos analizar el interior de éste. Bastaría con una herramienta que disponga de un extremo lo suficientemente fino para introducirlo, a modo de palanca, en la franja de separación de las dos piezas de recubrimiento que conforman el dispositivo Chromecast. Así, una vez abierto, podemos ver en la imagen 4.4 los distintos componentes.

(a) Puerto HDMI. (b) Puerto USB.

Figura 4.2: Puertos de entrada y salida de Chromecast.

Figura 4.3: Parte posterior del dispositivo Chromecast

El componente resaltado en color rojo es un módulo de la familia AzureWave, modelo AW-NH387, que proporciona conectividad WLAN 802.11 b/g/n, Bluetooth y FM. Este módulo se caracteriza por su pequeño tamaño, bajo consumo y soporte de múltiples interfaces y sistemas operativos. Además, cumpliendo con el estándar IEEE 802.11b/g/n (2,4 GHz), utiliza las tecnologías de modulación de banda base DSSS (Direct Sequence Spread Spectrum), OFDM (Orthogonal Frequency Division Multiplexing), DBPSK (Di), CCK y QAM. Entre el resto de sus características, podemos destacar el avanzado nivel de seguridad que proporciona con WEP/WPA/WPA2/IEEE 802.11i.

Por otro lado, el elemento resaltado en color naranja corresponde con un chipset Marvell DE3005-A1, del que no disponemos de mucha más información al respecto. Según el análisis del autor Brian Klug [13], el SoC (*System*

(a) Diferentes partes del dispositivo Chromecast.

(b) Parte delantera de la placa madre.

(c) Parte trasera de la placa madre.

Figura 4.4: Detalle de la composición de Chromecast.

on a Chip) de Marvell que incluye el dispositivo, es una versión reducida del modelo 88DE3100 que, por ejemplo, incluye Google TV. En el caso del Chromecast sólo incluiría un core y, además, trabajaría sobre Android en vez de sobre Chrome.

Finalmente, haremos referencia a los módulos de memoria que están dispuestos en la otra cara de la placa madre.

Con color amarillo se ha resaltado la memoria flash de tipo NAND de 16 Gb (2 GB) de la familia Micron, cuyo modelo es MT29F16G08MAA. También de la familia Micron, en color verde, podemos ver la memoria SDRAM DDR3L de 512 MB, modelo D9PXV. La diferencia de las memorias SDRAM DDR3 con la memoria que incorpora este dispositivo es que, ésta última, tienen un voltaje ligeramente inferior.

Como último elemento destacable, y para terminar con el estudio técnico

del dispositivo Chromecast, éste cuenta con un disipador de calor de aluminio. Aún con este sistema incorporado, si exponemos a nuestro dispositivo a un uso prolongado, puede alcanzar una alta temperatura.

4.1.3. Consumo de potencia

Como hemos comentado en el apartado anterior, es necesario conectar el Chromecast a un puerto USB o a la red eléctrica ya que el puerto HDMI no proporciona suficiente corriente (+5 V y un máximo de 50 mA [14]).

Si medimos la potencia entre el puerto microUSB y el propio Chromecast obtenemos los datos recogidos en la tabla 4.1 [13].

	Reposo	Activo
BVolt	4,96 V	5,06 V
SVolt	26,78 mV	43,38 mV
Volt	4,99 V	5,10 V
Curr	267,30 mA	415,90 mA

Cuadro 4.1: Consumo del Chromecast en reposo y en estado activo

4.1.4. Configuración

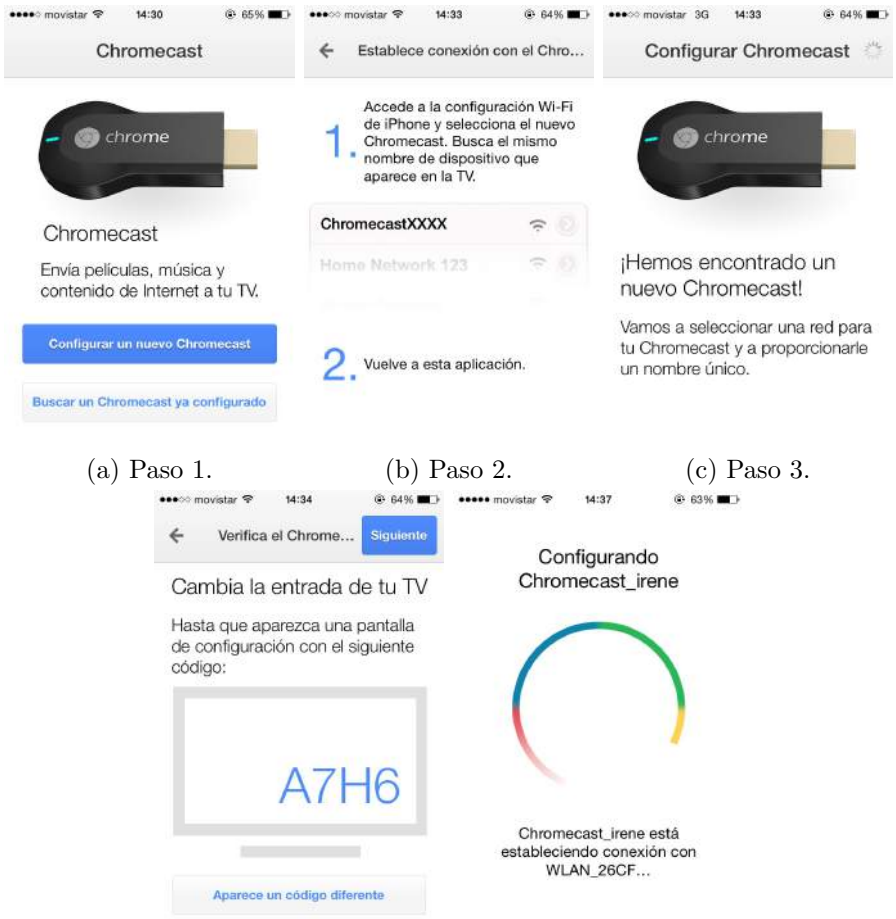
La configuración del software del Chromecast es simple y se puede realizar desde la correspondiente aplicación en la plataforma deseada, tanto en dispositivos móviles como de escritorio. Según el sitio web oficial [15], actualmente es compatible con teléfonos o tablets Android con Android 2.3 o versiones superiores, iPhone, iPad y iPod Touch con iOS 7.0 o versiones superiores aunque no es compatible enviar contenido desde Chrome para móviles. En cuanto a ordenadores, los requisitos del sistema son los siguientes: Windows 7 o versiones superiores, MAC OS 10.7 o superior o Chrome OS.

En este caso, realizamos la configuración usando la aplicación Chromecast para iOS desde un iPhone que se puede descargar desde la App Store [16].

Una vez hayamos conectado el Chromecast, éste creará una red inalámbrica temporal propia y la aplicación se conecta a ella haciendo uso de la utilidad de descubrimiento de puntos de acceso WiFi. Una vez realizada la configuración, se vuelve a la red a la que se estaba conectado anteriormente.

4.1.5. Funcionamiento

El Chromecast estará disponible siempre y cuando el televisor o monitor al que esté conectado esté encendido. Una curiosidad de este dispositivo es que soporta HDMI-CEC (*Consumer Electronic Control*), lo que permite el control remoto a través de la conexión HDMI. En las primeras versiones,



(a) Paso 1. (b) Paso 2. (c) Paso 3.

(d) Paso 4. (e) Paso 5.

Figura 4.5: Configuración del Chromecast haciendo uso de la aplicación para iOS.

esta funcionalidad simplemente permitía activar y cambiar al modo HDMI del televisor cuando se detectaba actividad, estando encendido el televisor. Pero con la actualización 27946 del firmware, se ha desbloqueado el control remoto [17], con lo que se permite controlar la reproducción desde el mando del televisor. Esta funcionalidad no está disponible para todos los televisores ni para todas las aplicaciones compatibles con Chromecast.

Cuando el Chromecast está en estado inactivo, se muestra una pantalla (figura 4.6) que indica que está preparado para reproducir contenido y además presenta información relevante de la red a la que está conectado, como el nombre y el nivel de la señal. El fondo de pantalla se puede configurar desde la aplicación [18], pero por defecto muestra fotos aleatorias que se descarga de un servidor de Google cada 60 segundos.

Figura 4.6: Proyección en estado inactivo.

Dos de las funcionalidades más comunes son reproducir vídeo en una segunda pantalla desde un dispositivo más pequeño y duplicar la pantalla, ya sea móvil, tableta u ordenador personal.

Smartphones y tabletas

Existe una amplia variedad de aplicaciones que son compatibles con Chromecast que podemos consultar en el sitio web oficial [19].

Este dispositivo reproduce el contenido multimedia directamente desde Internet y la fuente [20]. Esto significa que el teléfono móvil sólo actúa como control remoto. Por ejemplo, si queremos reproducir un video desde la aplicación móvil de YouTube, en cuanto pulsamos el icono de Chromecast, el dispositivo Chromecast establece una conexión directa con los servidores de YouTube.

En cuanto a la calidad de la imagen depende en mayor parte de la conexión a Internet de la que dispongamos. Chromecast automáticamente ajusta la resolución y el bitrate basándose en el ancho de banda disponible.

Tmbién es posible duplicar la pantalla de nuestro dispositivo desde la opción 'Enviar pantalla' de la aplicación del smartphone o tableta.

Navegador Chrome desde un ordenador

Al igual que se describió en el apartado anterior, podemos reproducir contenido multimedia desde aplicaciones compatibles con Chromecast.

En este caso también se puede enviar el contenido de una pestaña del navegador web Chrome, así como su audio, directamente al receptor. Basta con añadir la extensión Cast al navegador Chrome, disponible en el store de Google Chrome [21].

Hay disponibles tres opciones para la configuración de la calidad de la imagen, como podemos ver en la figura 4.7.

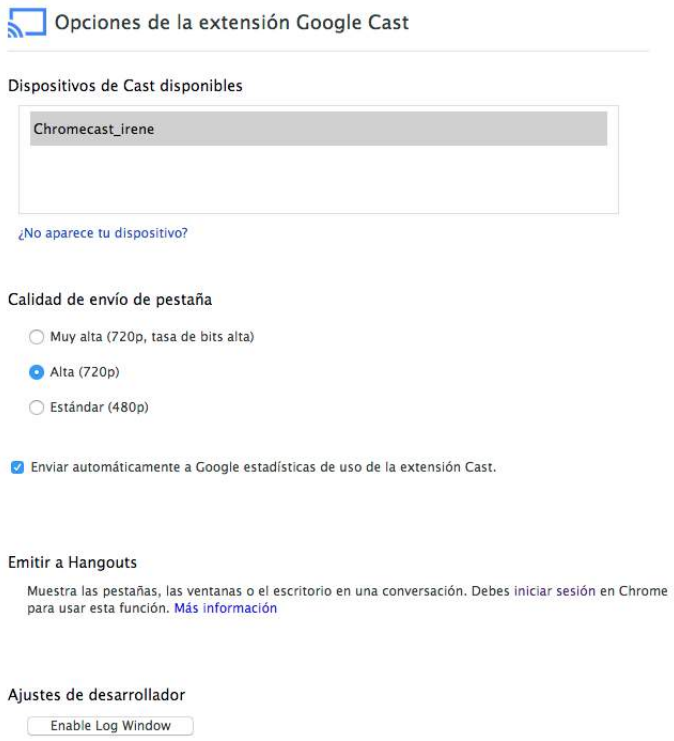


Figura 4.7: Opciones de configuración de la extensión Cast para Chrome.

Además existe una latencia de alrededor de un segundo en la conexión, dependiendo también de la calidad de la conexión.

También podemos reproducir videos almacenados localmente en nuestro ordenador. Cabe destacar que el video es transcodificado y comprimido antes de alcanzar el monitor. Como hemos visto, la máxima resolución que se soporta es 720p, en dos opciones: bitrate alto o bajo. Esto significa que, aunque la visualización sea aceptable, no tiene tantos detalles como el archivo original. Sin embargo, en la versión beta de la aplicación, se alcanzan los 1080p de resolución y, además, se pueden configurar otros parámetros más avanzados (figura 4.8).

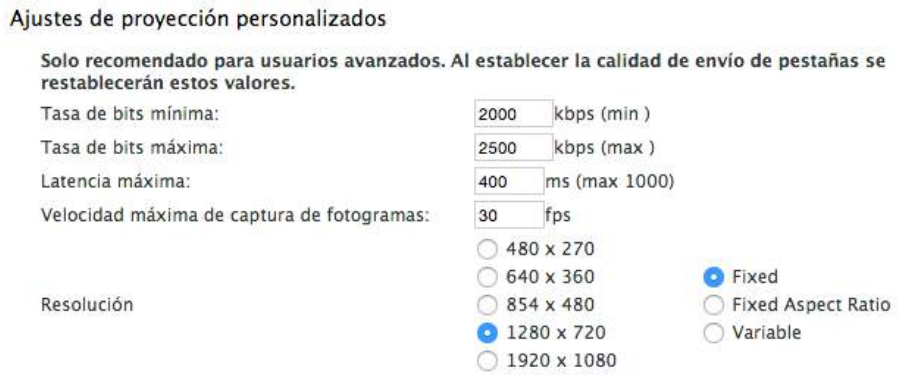


Figura 4.8: Opciones de configuración de la versión beta.

Modo invitados

En la aplicación para Chromecast de Android está disponible el modo invitados [22] (desactivado de manera predeterminada), que permite enviar contenido a usuarios que no estén conectados a nuestra red Wi-Fi y que se encuentren dentro de la zona de cobertura Wi-Fi (aproximadamente en un radio de 7,5 metros).

Google utiliza huellas digitales Wi-Fi para saber si existe algún dispositivo cerca de nuestro Chromecast con el modo invitados activado. Una vez se haya detectado el dispositivo móvil, el Chromecast le enviará automáticamente un código PIN de cuatro dígitos que se genera aleatoriamente, emitiendo breves tonos ultrasónicos que no son perceptibles para nuestro oído. Si este proceso falla, será necesario que el usuario introduzca manualmente este código, que podrá consultar en el monitor al que esté conectado el Chromecast.

Los dispositivos Android que utilizan Android 4.3 (JB MR2) o versiones posteriores admiten el envío de contenido Chromecast en modo invitados sin tener que establecer conexión Wi-Fi. Este modo también está disponible para la versión iOS de la aplicación Chromecast.

Otra desventaja que presenta este modo es que no todas las aplicaciones compatibles con Google Cast funcionan con este modo. Las aplicaciones que envían contenido multimedia local de un teléfono o tableta al Chromecast no funcionan con el modo invitados.

4.1.6. Experimentos, resultados y descubrimientos

Para comprender mejor el funcionamiento de este dispositivo, podemos examinar los paquetes intercambiados a través de la red entre el Chromecast y un dispositivo de control remoto, como puede ser un smarphone o una

tableta. Mientras que el Chromecast encripta la mayor parte del contenido, el dispositivo de control remoto envía paquetes de control a los servidores remotos en texto plano, lo que le hace vulnerable a ataque de replay (o ataques de playblack) y a ataques de secuestro de sesión.

En el presente apartado procederemos a evaluar los protocolos de red que usa Chromecast, así como el flujo de paquetes entre el cliente, Chromecast y servidores externos. Para establecer el entorno de pruebas disponemos del propio Chromecast conectado al puerto HDMI de un monitor LCD, un móvil Android como dispositivo de control remoto y un ordenador con las funciones de punto de acceso para monitorizar el tráfico, todos ellos conectados a la misma red. El escenario descrito se muestra en la figura 4.9.

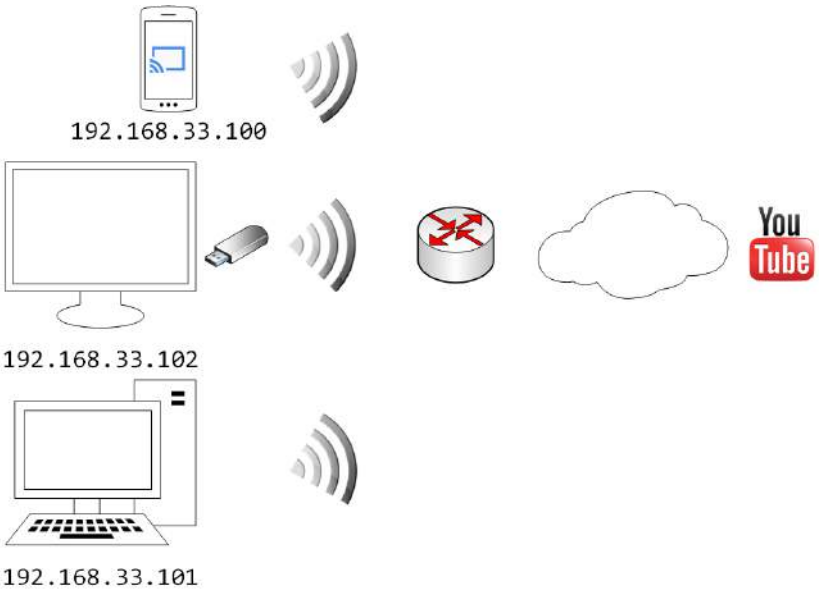


Figura 4.9: Escenario de pruebas.

En la figura 4.10 se listan los protocolos más significativos que toman parte en las comunicaciones durante una sesión con Chromecast.

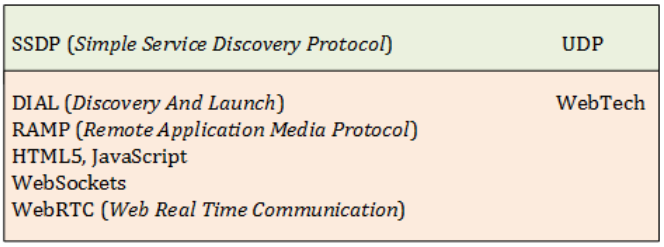


Figura 4.10: Pila de protocolo - Chromecast.

Cuando enviamos contenido a nuestro Chromecast, éste carga una página web ligera usando HTML5, JavaScript y CSS. Esta página web cargará el contenido usando la etiqueta `video` de HTML5 y esperará comandos de parada, pausa o reanudación. Como ya hemos mencionado anteriormente, para proyectar contenido desde el navegador web Chrome, necesitamos la extensión Google Cast. Al hacer uso de esta extensión, el Chromecast carga la página web actual con WebRTC [23].

A continuación vamos a detallar algunos de los paquetes más significativos.

Servicio de descubrimiento DIAL

El dispositivo Chromecast opera con el protocolo DIAL [24]. El servicio de descubrimiento de DIAL habilita al cliente a descubrir al servidor DIAL (Chromecast) en su red local y obtener acceso a los servicios DIAL.

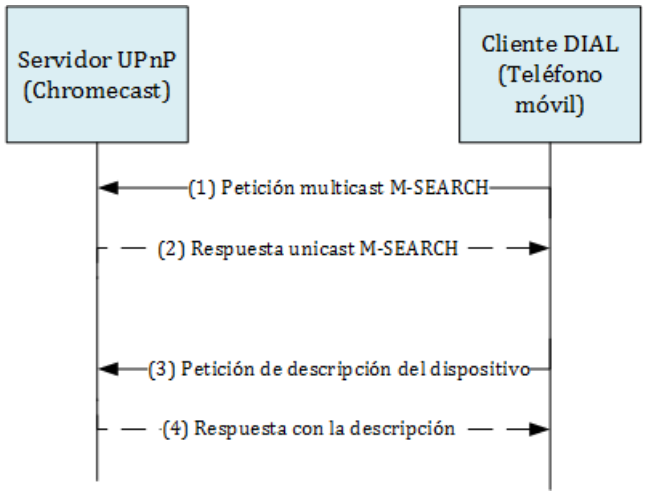


Figura 4.11: Servicio de descubrimiento del protocolo DIAL.

- Petición M-SEARCH.
  Los clientes que desean descubrir los servidores Chromecast envían una petición sobre UDP a la dirección IP 239.255.255.250 y el puerto UDP 1900. En este paquete multicast se incluye el parámetro *Search Target* definido en la especificación DIAL `um:dial-multiscreen-org:service:dial:1`, como podemos ver en la figura 4.12. El parámetro *Time To Live* (TTL) está fijado a 1 y eso significa que el paquete no se va a enrutar hacia otras subnets.

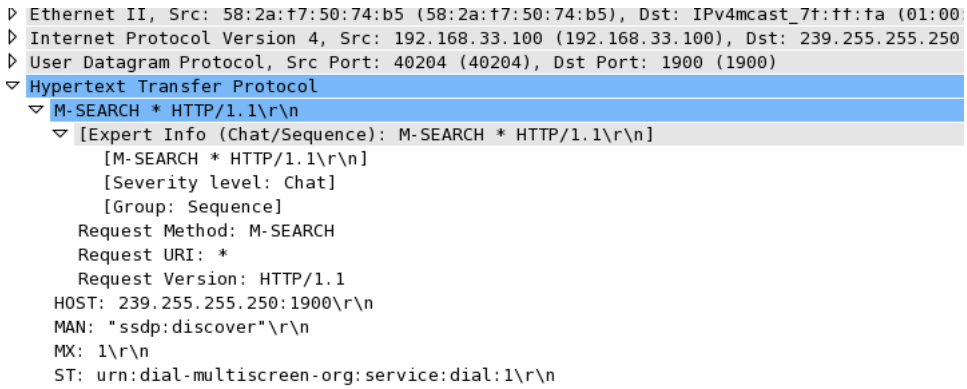


Figura 4.12: Información de la petición de descubrimiento.

- Respuesta M-SEARCH.
  El servidor SSDP/UPnP [25] que reciba la petición anterior, responderá con un mensaje unicast en el que se incluye la URL HTTP absoluta para la descripción del dispositivo raíz.

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=1800
DATE: {currentDate}
EXT:
LOCATION: http://{localIP}:8008/ssdp/device-desc.xml
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: baed804a-1dd1-11b2-8973-d7a6784427e5
SERVER: Linux/3.8.13, UPnP/1.0, Portable SDK for UPnP devices/1.6.18
X-User-Agent: redsonic
ST: {ST}
USN: uuid:{uuid}::{ST}
BOOTID.UPNP.ORG: 7339
CONFIGID.UPNP.ORG: 7339
```

Figura 4.13: Información de la respuesta de descubrimiento.

- Petición de descripción del dispositivo.
  Una vez recibida la respuesta M-SEARCH, el cliente DIAL enviará una petición HTTP de tipo GET hacia la URL recibida en el anterior paquete.

Figura 4.14: Información de la petición de descripción del dispositivo.

Como podemos observar en la figura 4.14, viaja en texto plano el sistema operativo y la versión instalada en el dispositivo de control remoto, así como la marca y modelo de este. En el apéndice C podemos ver otro ejemplo de petición HTTP de tipo GET en el que la información devuelta viaja en texto plano.

- Respuesta de descripción del dispositivo.
  Si la petición tiene éxito, la respuesta HTTP contendrá la cabecera `Application-URL`. También se pueden observar en la figura 4.15 campos como `friendlyName` que coincide con el nombre que le hayamos asignado a nuestro Chromecast, `URLBase` con la dirección IP local del Chromecast dentro de nuestra red.

Servicio DIAL REST

Este servicio (ver figura 4.18) permite al cliente realizar peticiones, lanzar y gestionar aplicaciones. Además, representa dichas aplicaciones como fuentes o recursos identificados por URLs. En primer lugar se intercambian dos mensajes para comprobar si existe la aplicación, en este caso de ejemplo YouTube, para lanzarla posteriormente.

- Petición del cliente
  El cliente DIAL que desee información sobre una aplicación, deberá enviar una petición HTTP de tipo GET a la URL de la aplicación.

```
190 25.899929      192.168.33.102          192.168.33.100
HTTP/1.1 200 OK\r\n
  [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
    Request Version: HTTP/1.1
    Status Code: 200
    Response Phrase: OK
  Access-Control-Allow-Methods:GET, POST, DELETE, OPTIONS\r\n
  Access-Control-Expose-Headers:Application-URL\r\n
  Application-URL:http://192.168.33.102:8008/apps/\r\n
  Access-Control-Allow-Origin:https://www.google.com\r\n
  Content-Length:1095\r\n
  Content-Type:application/xml\r\n
  \r\n
  [HTTP response 1/8]
  [Time since request: 0.004742000 seconds]
  [Request in frame: 185]
  [Next request in frame: 203]
  [Next response in frame: 216]
eXtensible Markup Language
  <?xml
  <root
    xmlns="urn:schemas-upnp-org:device-1-0">
    <specVersion>
    <URLBase>
      http://192.168.33.102:8008
      </URLBase>
    <device>
      <deviceType>
        urn:dial-multiscreen-org:device:dial:1
        </deviceType>
      <friendlyName>
        Chromecast2762
        </friendlyName>
      <manufacturer>
        Google Inc.
        </manufacturer>
      <modelName>
        Eureka Dongle
        </modelName>
```

Figura 4.15: Información de la respuesta de descripción del dispositivo.

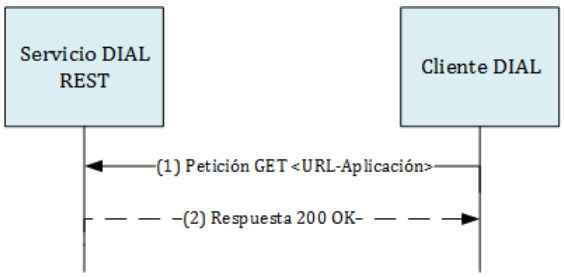


Figura 4.16: Servicio DIAL REST.

```
Frame 276: 311 bytes on wire (2488 bits), 311 bytes captured (2488 bits)
Ethernet II, Src: 58:2a:f7:50:74:b5 (58:2a:f7:50:74:b5), Dst: Azurewav_9f:a7:22 (6c:ad:f8:9f:a7:22)
Internet Protocol Version 4, Src: 192.168.33.100 (192.168.33.100), Dst: 192.168.33.102 (192.168.33.102)
Transmission Control Protocol, Src Port: 35234 (35234), Dst Port: 8008 (8008), Seq: 254, Ack: 1384, Len: 245
Hypertext Transfer Protocol
  GET /apps/YouTube HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /apps/YouTube HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /apps/YouTube
      Request Version: HTTP/1.1
  Origin: package:com.google.android.youtube\r\n
  Host: 192.168.33.102:8008\r\n
  Connection: Keep-Alive\r\n
  User-Agent: com.google.android.youtube/10.25.55(Linux; U; Android 5.0; es_ES; HUAWEI GRA-L09 Build/HUAWEIGRA-L09) gzip\r\n
  \r\n
```

Figura 4.17: Petición del servicio DIAL REST.

- Respuesta del cliente
  La respuesta tendrá formato de texto/xml y contendrá información relacionada con la aplicación sobre la cual se ha realizado la pregunta.

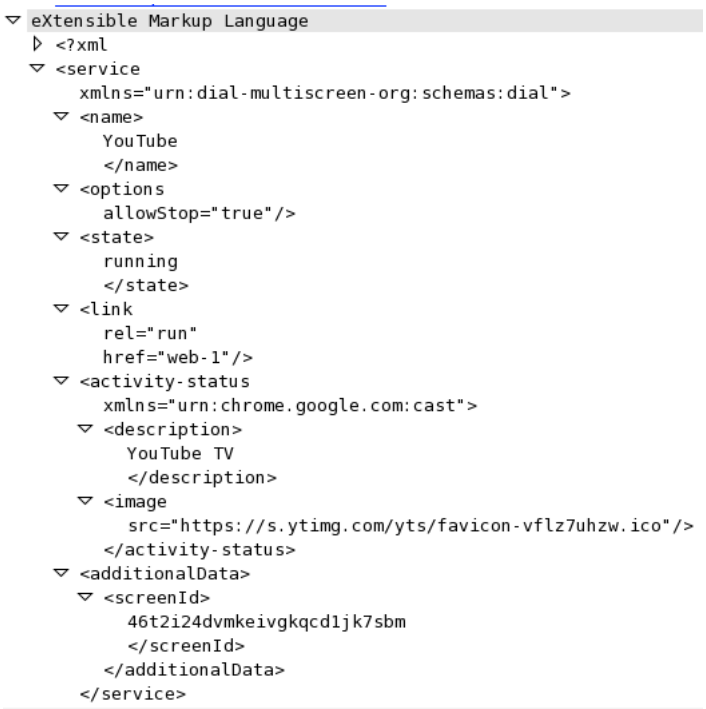


Figura 4.18: Respuesta del servicio DIAL REST.

Para conocer los detalles y significado de los parámetros incluidos en este paquete, puede consultar el anexo B al final del presente documento.

Protocolo mDNS

Chromecast hace uso de los servidores mDNS [26] para permitir el descubrimiento de los servicios de la API de Google necesarios para su funcionamiento.

- Petición mDNS Un ejemplo de dicha petición se puede ver en la figura 4.19.
- Respuesta mDNS La primera respuesta debe especificar el nombre de dominio con el formato `friendlyName._googlecast._tcp.local` donde `friendlyName` es el nombre que hemos elegido para nuestro Chromecast. Las respuestas adicionales se muestran en las figuras4.20 y4.21

```
▷ Frame 167: 392 bytes on wire (3136 bits), 392 bytes captured (3136 bits)
▷ Ethernet II, Src: Azurewav_9f:a7:22 (6c:ad:f8:9f:a7:22), Dst: IPv4mcast_fb (01:00:5e:00:00:fb)
▷ Internet Protocol Version 4, Src: 192.168.33.102 (192.168.33.102), Dst: 224.0.0.251 (224.0.0.251)
▷ User Datagram Protocol, Src Port: 5353 (5353), Dst Port: 5353 (5353)
▽ Domain Name System (query)
    [Response In: 1194]
    Transaction ID: 0x0000
  ▷ Flags: 0x0000 Standard query
    Questions: 2
    Answer RRs: 0
    Authority RRs: 3
    Additional RRs: 0
  ▽ Queries
    ▷ Chromecast2762._googlecast._tcp.local: type ANY, class IN, "QM" question
    ▷ Chromecast2762.local: type ANY, class IN, "QM" question
```

Figura 4.19: Petición mDNS.

```
▽ Chromecast2762._googlecast._tcp.local: type TXT, class IN, cache flush
      Name: Chromecast2762._googlecast._tcp.local
      Type: TXT (Text strings) (16)
      .000 0000 0000 0001 = Class: IN (0x0001)
      1... .... .... .... = Cache flush: True
      Time to live: 4500
      Data length: 107
      TXT Length: 35
      TXT: id=85f04e90af639d94259575d5ae561e42
      TXT Length: 5
      TXT: ve=04
      TXT Length: 13
      TXT: md=Chromecast
      TXT Length: 18
      TXT: ic=/setup/icon.png
      TXT Length: 17
      TXT: fn=Chromecast2762
      TXT Length: 4
      TXT: ca=5
      TXT Length: 4
      TXT: st=0
      TXT Length: 3
      TXT: rs=
```

Figura 4.20: Respuesta de tipo TXT.

```
▽ Chromecast2762._googlecast._tcp.local: type SRV, class IN, cache flush, priority 0, weight 0,
    Service: Chromecast2762
    Protocol: _googlecast
    Name: _tcp.local
    Type: SRV (Server Selection) (33)
    .000 0000 0000 0001 = Class: IN (0x0001)
    1... .... .... .... = Cache flush: True
    Time to live: 120
    Data length: 28
    Priority: 0
    Weight: 0
    Port: 8009
    Target: Chromecast2762.local
▽ Chromecast2762.local: type A, class IN, cache flush, addr 192.168.33.102
    Name: Chromecast2762.local
    Type: A (Host Address) (1)
    .000 0000 0000 0001 = Class: IN (0x0001)
    1... .... .... .... = Cache flush: True
    Time to live: 120
    Data length: 4
    Address: 192.168.33.102 (192.168.33.102)
```

Figura 4.21: Respuestas de tipo SRV y de tipo A.

Curiosidades

Con el fin de descubrir y analizar las posibles vulnerabilidades de Chromecast, hemos realizado un ratreo de puertos con la herramienta Nmap [27], un programa de código abierto. Existen dos puertos abiertos mientras el Chromecast está en estado inactivo, el 8008 (escuchando conexiones HTTP) y el 8009 (para el servicio ajp13).

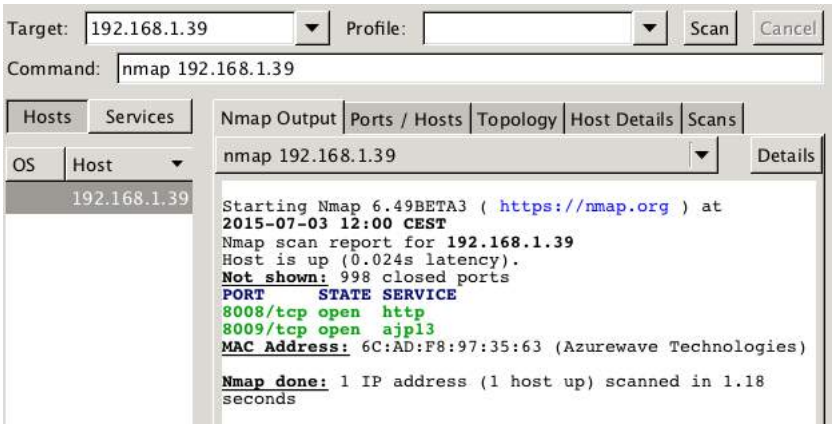


Figura 4.22: Análisis con Nmap.

Además, Chromecast mantiene un par de conexiones a los servidores de Google cuando se conecta a la red inalámbrica local, aunque no se haya comenzado la proyección de contenido multimedia. Una conexión es para recuperar las imágenes que se muestran en el monitor como fondo de pantalla. Para este fin, Chromecast se conecta a un servidor de Google para descargar la imagen. Aunque los paquetes están encriptados con TLS v1.2, se puede diferenciar que paquete corresponde a la imagen a partir del tamaño total de los paquetes entrantes cada, aproximadamente, 60 segundos.

4.2. Android

4.2.1. Historia

Android podría considerarse como una de las plataforma móvil más extendida en el mundo, un todo formado por el sistema operativo en sí y una serie de servicios y aplicaciones que lo complementan. El sistema operativo está basado en un kernel de Linux. Google es el principal responsable de esta plataforma aunque existe un proyecto conocido como AOSP (*Android Open Source Project*) que se encarga de colaborar con el desarrollo de esta plataforma.

Fue fundada por Andy Rubin, Rich Miner, Nick Sears and Chris White en 2003 en Palo Alto (California) y, posteriormente, adquirida por Google en el año 2005.

A pesar de que, cuando nació Android, uno de los objetivos principales era desarrollar un sistema operativo para cámaras digitales, pronto se dieron cuenta de que debían abordar un mercado más amplio y extendido como el de los dispositivos móviles. Desde la primera publicación de la versión Android 0.5 (Milestone 3) hasta la más reciente versión Lollipop (v. 5.0), la compañía ha ido ampliando su mercado a otros dispositivos como televisiones inteligentes, tabletas o vestibles, así como manteniendo el espíritu y la filosofía del continuo desarrollo.

Como podemos ver en la figura 4.23, el sistema operativo para móvil más usado es Android con un 47,45 % del total de usuarios móviles [28].

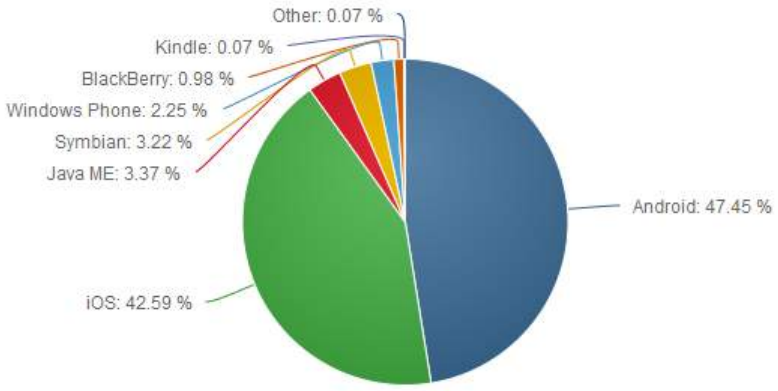


Figura 4.23: Sistemas operativos para dispositivos móviles

Si nos centramos en las estadísticas de España, Android siguió siendo el sistema operativvo móvil líder en el año 2014 [29], tal y como se muestra en la figura 4.24.

Si por el contrario, consultamos datos de Estados Unidos, la llegada del

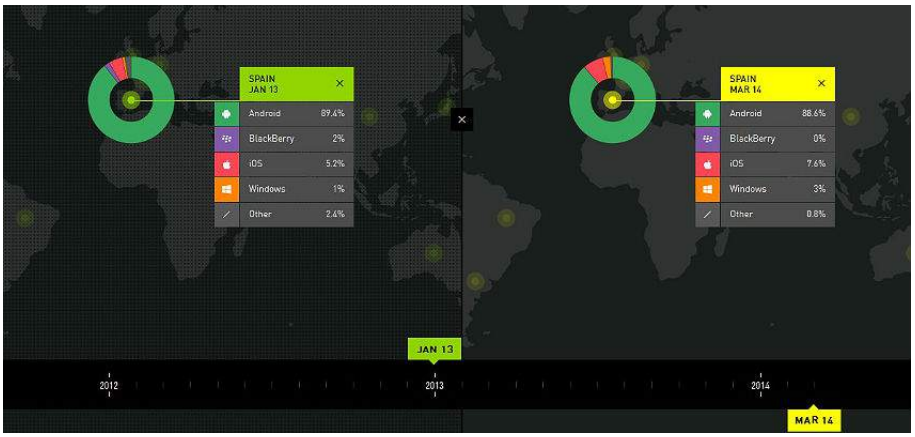


Figura 4.24: Ránking de sistemas operativos móviles a nivel estatal

nuevo terminal de iOS, el iPhone 6 e iPhone 6 Plus, ha marcado un nuevo registro en la compañía Apple, posicionándola a la cabeza en las listas de ventas [30].

4.2.2. Conceptos básicos

En este apartado resumiremos los conceptos básicos necesarios para comprender la estructura y funcionamiento general de una aplicación desarrollada para Android.

- **View.** Constituye el componente básico en el que se apoyan todos los elementos que componen una interfaz. Todos los elementos que generan interfaces heredan de la clase *View*.

- **Activity.** Elemento encargado de mostrar la interfaz de usuario e interactuar con él. Responde a los eventos generados por el usuario como, por ejemplo, pulsar botones. Heredan de la clase *Activity*. Pueden existir tantas actividades como el desarrollador considere necesarias pero, generalmente, una aplicación debe contar con una actividad principal que se instanciará en primer lugar al arrancar la aplicación, y una actividad por cada tarea implementada. En la figura 4.25 se muestra el ciclo de vida de una actividad en Android.

- **Services.** No tienen interfaz visual y se ejecutan en segundo plano. Es el componente encargado de realizar tareas que deben seguir ejecutándose cuando nuestra aplicación no está en primer plano. Extienden de la clase *Service*.

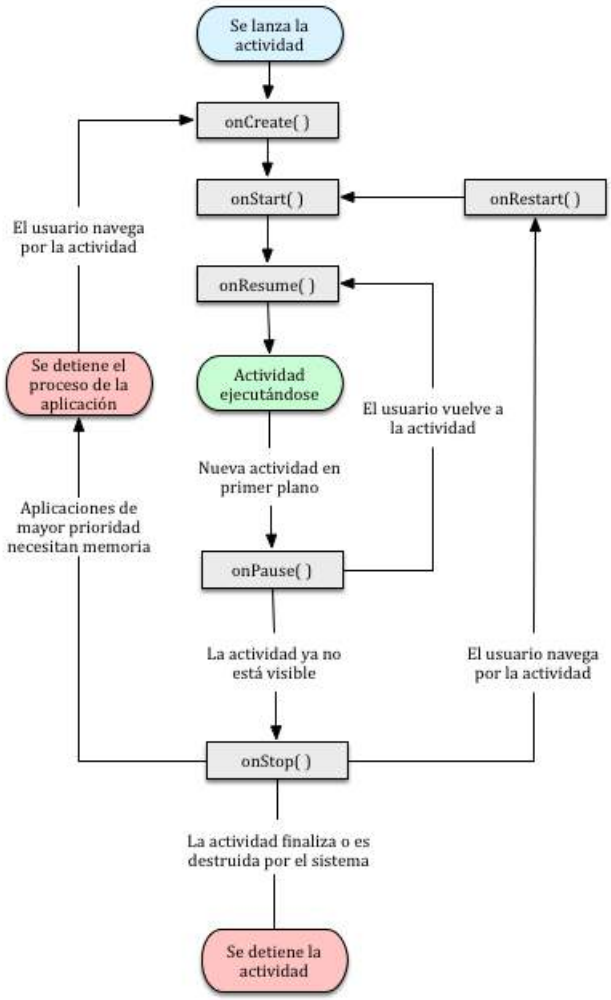


Figura 4.25: Ciclo de vida de una actividad en Android

- **Content Provider.** Este componente facilita a distintas aplicaciones el acceso a grupos de datos. Extienden de la clase *ContentProvider* para implementar los métodos de la interfaz, mientras que hacen uso de la clase *ContentResolver* para acceder a dicha interfaz. Android incluye una base de datos SQLite que es usada para almacenar y consultar datos.

- **Broadcast Receiver.** Reacciona tras recibir un mensaje. Extiende de la clase *BroadcastReceiver* y, aunque no tiene interfaz de usuario, puede lanzar actividades como respuesta un evento. También es posible que se haga uso de *NotificationManager* para informar al usuario. Por ejemplo,

se puede registrar un evento que defina el cambio en el teléfono, e.g. si se recibe una llamada.

- **Intent.** Este elemento permite establecer la comunicación y transferencia de datos entre objetos de la clase *Activity* o *Service*. También permite iniciar otras actividades o lanzar otras aplicaciones.

En la figura 4.25 se muestra el ciclo de vida de una actividad en Android. Durante el desarrollo de aplicaciones para Android existen una serie de métodos que gestionan diferentes tareas y que están presentes en la mayoría de las actividades que vamos a implementar. Vamos a detallar algunos de los más importantes:

- *`public void onCreate (Bundle savedInstanceState)`* En este método se inicializa nuestra actividad. En este punto realizaremos una llamada al método `setContentView(int)` para definir el recurso de nuestro *layout* o diseño. Además se hará uso del método `findViewById(int)` para recuperar los complementos necesarios para el usuario para poder interactuar con la interfaz. Además se detalla la rutina de la actividad al iniciarse.

- *`public void onStart ()`* Este método es llamado cuando la actividad se hace visible al usuario.

- *`public void onRestart ()`* Se llama a este método una vez la actividad haya sido detenida, justo antes de que dicha actividad vuelva a ser lanzada de nuevo.

- *`public void onResume ()`* Este método se usa cuando el usuario empiece a interactuar con la actividad.

- *`public void onPause ()`* Llamaremos a este método cuando el sistema esté a punto de reanudar una actividad previa. Las implementaciones de este método deben ser rápidas ya que la actividad siguiente no se reanudará hasta que este método finalice y lance un resultado.

- *`public void onStop ()`* Este método se invoca cuando la actividad ya no está visible para el usuario porque otra actividad ha sido reanudada y se superpone a ésta. Este caso se da cuando se lanza una nueva actividad, una actividad existente se recupera o cuando la propia actividad es destruida o detenida.

- *`public void onDestroy ()`* Esta es la última llamada que se realiza antes de que se destruya la actividad. Esto puede suceder porque la actividad ha finalizado (es decir, se ha realizado una llamada al método `finish()`) o, porque el sistema destruya temporalmente la actividad para ahorrar espacio.

- *public void onPostCreate (Bundle savedInstanceState)* Se invoca cuando la configuración o inicio de una actividad se ha completado.

- *public boolean onCreateOptionsMenu (Menu menu)* Este método sirve para inicializar el contenido de las opciones del menú de la actividad.

- *public boolean onOptionsItemSelected (MenuItem item)* Se invoca cuando se selecciona una de las opciones del menú de la actividad.

En el caso de que queramos trabajar de manera fácil y flexible con hebras en nuestra aplicación, haremos uso de la clase *AsyncTask*. Cuando ejecutamos una tarea o hebra asíncrona, ésta pasa por los siguientes cuatro estados:

- *public void onPreExecute ()* Este método se invoca antes de que se ejecute la tarea. Este paso normalmente se usa para configurar parámetros de la hebra.

- *public String doInBackground (String... params)* Este método se invoca justo después de que finalice método `onPreExecute`. Se usa para ejecutar en un segundo plano aquellas tareas que pueden tardar más tiempo y en este punto se pasan los parámetros necesarios para la tarea. Los resultados se devuelven al último método.

- *public String onProgressUpdate (Integer... progress)* Se usa para mostrar al usuario mensajes de progreso mientras se ejecutan las tareas en segundo plano.

- *public void onPostExecute (String result)* Se invoca una vez hayan finalizado las tareas que estaban ejecutándose en segundo plano.

4.2.3. Primera aplicación en Android

Entorno de desarrollo

Antes de comenzar con el desarrollo de nuestra primera aplicación en Android, debemos establecer y configurar nuestro entorno de trabajo. Para ello utilizaremos la herramienta Android Studio, el entorno de desarrollo integrado (IDE, *Integrated Development Enviroment*) oficial de Android. Así mismo necesitaremos hacer uso de un kit de desarrollo de software o SDK (*Software Development Kit*) propio de Android, que proporciona todas las herramientas necesarias para crear y compilar las aplicaciones. Ambas dependencias pueden ser descargadas desde la página oficial de desarrolladores de Android como un solo componente. Si, por el contrario, se prefiere usar otro IDE se puede descargar el SDK de manera independiente.

Dependiendo de la plataforma en la que vayamos a trabajar, debemos asegurarnos de cumplir unos requerimientos mínimos de nuestro sistema, que podemos consultar en la web anteriormente referenciada. Cabe destacar entre dichos requisitos (común a los sistemas operativos de Windows, Mac OS X y Linux) la necesidad de tener instalado en nuestra máquina el entorno de desarrollo Java o JDK (*Java Development Kit*). Consiste en un conjunto de herramientas de desarrollo para la implementación de aplicaciones y demás componentes que hacen uso del lenguaje de programación Java.

Una vez hayamos descargado todas las dependencias necesarias e instalado Android Studio, debemos tener en cuenta que, por defecto, no todos los componentes vienen incluidos. Por lo tanto, antes de empezar necesitaremos añadir una serie de paquetes adicionales, haciendo uso de la herramienta *SDK Manager* de Android, accesible desde la barra de herramientas del propio entorno Android Studio.

Podremos comprobar que existe un amplio abanico de paquetes, algunos de ellos ya estarán marcados como instalados. Para poder dar nuestros primeros pasos con Android, como mínimo, deberemos instalar los paquetes correspondientes del apartado **Tools**, tanto *Android SDK Tools* como *Android SDK Platform-tools* y *Android SDK Build-tools* (en su última versión). Además, dentro de la carpeta correspondiente a la última versión de Android, seleccionaremos los paquetes de *SDK Platform* y la imagen de sistema para el emulador como, por ejemplo, *ARM EABI v7a System Image*. Por ahora no será necesario añadir ninguna dependencia más como APIs o librerías adicionales ya que esta primera aplicación se trata de un ejemplo bastante sencillo.

Estructura y contenido del proyecto

Crear nuestro primer proyecto con Android Studio es tan secillo como seleccionar un nuevo proyecto y seguir los pasos del asistente, rellenando en cada paso la información necesaria que se solicite. Cuando hayamos completado todos los pasos, el propio entorno generará una estructura de carpetas que debemos respetar, como la que se muestra en la figura 4.26.

En nuestro caso, para el proyecto llamado HolaMundo, se generará la estructura que se muestra en la siguiente imagen. Durante el proceso de creación del nuevo proyecto hemos seleccionado una actividad vacía (*Blank Activity*) y hemos dejado el nombre por defecto, *MainActivity*.

Una de las carpetas a las que más vamos a recurrir es la carpeta de recursos (*res*). Ésta, a su vez, está dividida en distintas subcarpetas que hemos recogido en la tabla 4.2.

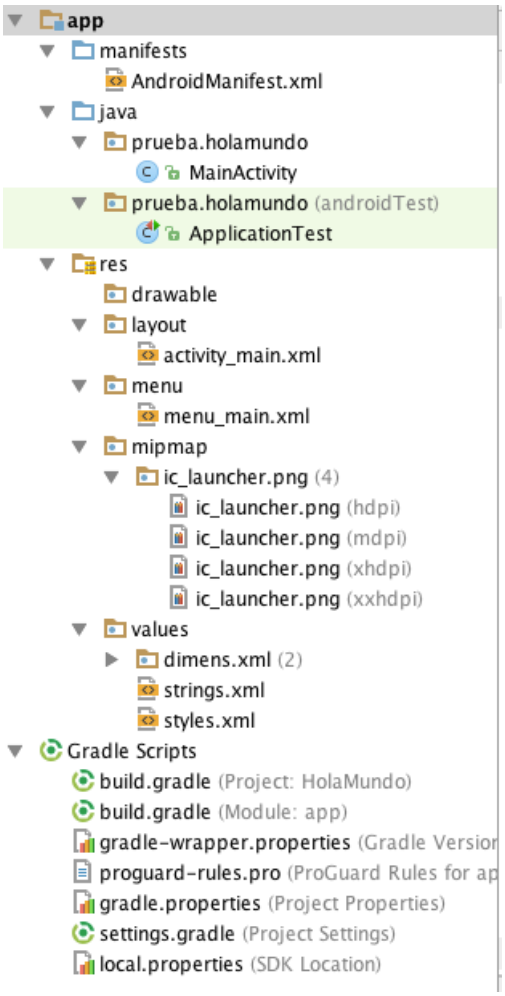


Figura 4.26: Estructura general de un proyecto creado en Android Studio

Recurso	Carpeta	Descripción
Dibujable (*Drawable*)	/res/drawable	Imágenes o archivos XML que describen un objeto dibujable
Diseño (*Layout*)	/res/layout	Incluye archivos XML en las que definimos interfaces gráficas para las distintas actividades o elementos del tipo *fragment*.
Menú (*Menu*)	/res/menu	Se definen las propiedades de los menús.
Mipmap	/res/mipmap	Iconos e imágenes que van a ser usados en la aplicación.
Valores simples (*Simple values*)	/res/values	Esta carpeta incluye a su vez subapartados para definir, por ejemplo, dimensiones (*/res/values/dimens.xml*), cadenas de caracteres (*/res/values/strings.xml*) o colores (*/res/values/colors.xml*) entre otros.

Cuadro 4.2: Estructura de la carpeta res

Además de generarse una estructura, se crean unos archivos por defecto, dentro de los cuales los más importantes son los siguientes:

- activity_ main.xml

  Este archivo XML se encuentra en el directorio **layout** y define la capa de usuario correspondiente a la actividad principal (*MainActivity*). Este archivo está disponible en versión texto o como una previsualización gráfica de la pantalla de la interfaz de usuario (figura 4.27). Por defecto, incluye un elemento *TextView* que muestra el mensaje "Hello world!".

activity_ main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">

    <TextView android:text="@string/hello_world"
                    android:layout_width="wrap_content"
                android:layout_height="wrap_content" />

</RelativeLayout>
```

- MainActivity.java

  Aquí encontramos la definición de la clase por defecto de la actividad que hemos creado. Al construir y, posteriormente, ejecutar la aplicación, la clase *Activity* lanza la actividad y carga el *layout* correspondiente.

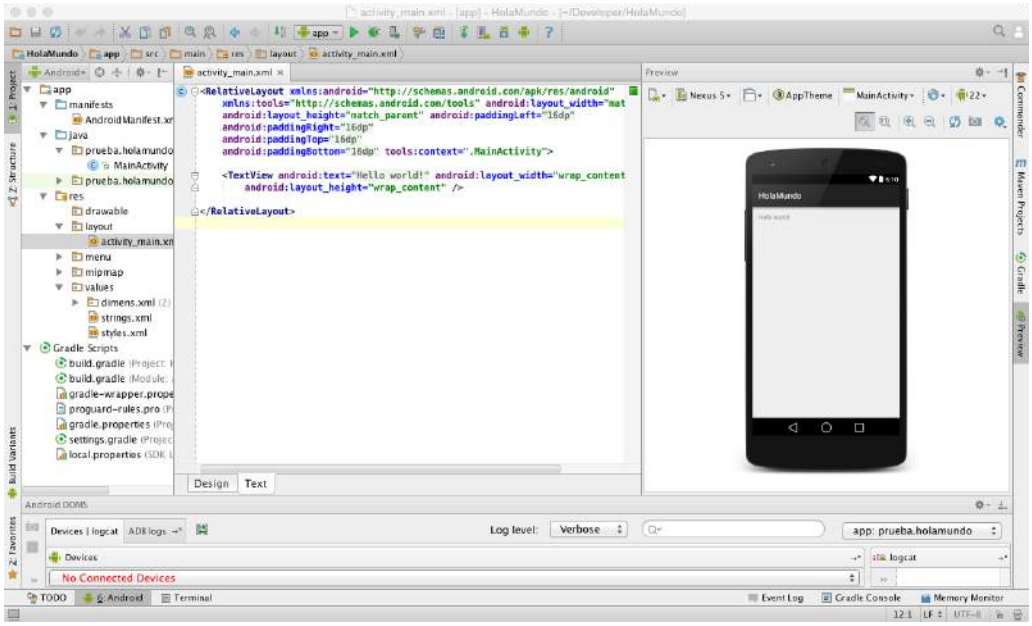


Figura 4.27: Vista de activity_ main.xml

MainActivity.java

```
1  package prueba.holamundo;
2
3  import android.support.v7.app.ActionBarActivity;
4  import android.os.Bundle;
5  import android.view.Menu;
6  import android.view.MenuItem;
7
8  /**
9   * Clase principal que hereda de la clase Activity, por lo que
10  * contara con interfaz de usuario.
11  * /
12 public class MainActivity extends ActionBarActivity {
13
14     @Override
15     protected void onCreate(Bundle savedInstanceState) {
16         super.onCreate(savedInstanceState);
17         /**
18          * Se inicializa cada componente de la actividad
19          * con su correspondiente view
20          */
21         setContentView(R.layout.activity_main);
```

```
22      }
23
24
25      @Override
26      public boolean onCreateOptionsMenu(Menu menu) {
27          // Con esta clase se "infla" el menu;
28          // Anade elementos a la barra de acciones
29          // (action bar) si fuese necesario.
30          getMenuInflater().inflate(R.menu.menu_main, menu);
31          return true;
32      }
33
34      @Override
35      public boolean onOptionsItemSelected(MenuItem item) {
36          // Gestiona los clicks a los elementos de la barra de acciones
37          int id = item.getItemId();
38
39          if (id == R.id.action_settings) {
40              return true;
41          }
42
43          return super.onOptionsItemSelected(item);
44      }
45  }
```

- AndroidManifest.xml

  Este archivo proporciona metadata adicional para la aplicación como, por ejemplo, iconos y la versión de la aplicación. El sistema consulta este archivo durante el proceso de instalación de la aplicación para determinar sus capacidades. En él se deben definir estáticamente todas las actividades, servicios y elementos del tipo *content provider*. Otro aspecto importante es que contiene aquellos permisos que consideremos necesarios para el correcto funcionamiento de nuestra aplicación como, por ejemplo, acceso a Internet. En los siguientes apartados veremos como otro punto indispensable en este archivo será definir las versiones máxima y mínima de la SDK que soportará nuestra aplicación. Por último, una de las partes importantes de este archivo es la etiquetada con el nombre *intent-filter*, definida en el nodo de la actividad que pretendemos que el sistema lance al ejecutar la aplicación. La documentación sobre este archivo puede ser consultada la página oficial.

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="prueba.holamundo" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

- build.gradle

  Este archivo es indispensable para compilar y construir nuestra aplicación. Nuestro proyecto contará con tantos archivos *build.gradle* como módulos haya. El más importante de todos es el correspondiente al módulo de nuestra aplicación, es decir, en nuestro caso, el archivo '*build.gradle (Module app)*'. En este archivo, además de la configuración por defecto, debemos definir aquellas dependencias necesarias.

build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 22
    buildToolsVersion "22.0.1"

    defaultConfig {
        applicationId "prueba.holamundo"
        minSdkVersion 14
        targetSdkVersion 22
        versionCode 1
```

```
12          versionName "1.0"
13      }
14      buildTypes {
15          release {
16              minifyEnabled false
17              proguardFiles getDefaultProguardFile('proguard-android.txt'),
18              'proguard-rules.pro'
19          }
20      }
21  }
22
23  dependencies {
24      compile fileTree(dir: 'libs', include: ['*.jar'])
25      compile 'com.android.support:appcompat-v7:22.1.1'
26  }
```

Ejecución del proyecto

Una vez que ya tenemos todo listo para ejecutar nuestra aplicación, tendremos que elegir entre diferentes opciones, dependiendo de si disponemos o no de un dispositivo Android real. Lo más sencillo es utilizar un dispositivo real con Android, ya que, como veremos en siguientes apartados, hay casos en los que es necesario hacer uso de APIs que no son compatibles con los emuladores. Basta con conectar el dispositivo móvil con nuestro ordenador, asegurándonos de que dicho dispositivo tiene habilitada la opción de depuración USB. Una vez hecho esto, podemos pinchar el botón *Run* de la barra de herramientas y elegir la opción *Choose a running device*. Tras unos segundos, Android Studio instalará y ejecutará y lanzará la aplicación en tu dispositivo.

Si por el contrario decidimos ejecutar la aplicación haciendo uso de un emulador, tenemos la opción de crear un dispositivo Android virtual (AVD, *Android Virtual Device*) o elegir el emulador por defecto de Android Studio. En la sección de manuales de esta memoria, se adjunta un tutorial para crear un AVD. Cuando ya hayamos elegido aquella opción que deseemos, pulsaremos el botón *Run* como en el caso anterior.

4.3. Desarrollo de aplicaciones para Chromecast

Antes de comenzar a desarrollar aplicaciones cliente para Chromecast, deberemos seguir una serie de pasos previos, descritos en la guía de Google Developers [31] y que resumiremos en este apartado.

Una vez hayamos instalado y configurado adecuadamente nuestro dispositivo Chromecast, debemos registrarlo en Google Cast SDK Developer

Figura 4.28: Vista de la aplicación HolaMundo en el emulador

Console [32]. Este paso es imprescindible para realizar las pruebas y empezar a desarrollar aplicaciones ya que, por defecto, los dispositivos Google Cast (como Chromecast o Android TV) no están habilitados para desarrollo. Para ello basta con introducir el número de serie del dispositivo. Una vez que el proceso de registro se haya completado, podremos ver el mensaje Ready for Testing".

Figura 4.29: Proceso de registro del Chromecast completado.

Es imprescindible registrar nuestra aplicación en el mismo sitio web donde habíamos registrado el Chromecast. Con esto, recibiremos una identificador para la aplicación que será utilizado para realizar diferentes llamadas a las APIs. En este proceso debemos tomar la decisión sobre qué tipo de aplicación receptora vamos a necesitar. Podemos consultar los detalle de este proceso en la página web de Google Cast [33]. Ese número identificativo lo usaremos en la aplicación emisora, que lo enviará al Chromecast y éste a su vez lo volverá

a enviar a ls consola de desarrollo de Cast, que reconocerá la aplicación. Por último, tomará la URL del número identificativo correspondiente y lanzará la aplicación receptora en el Chromecast.

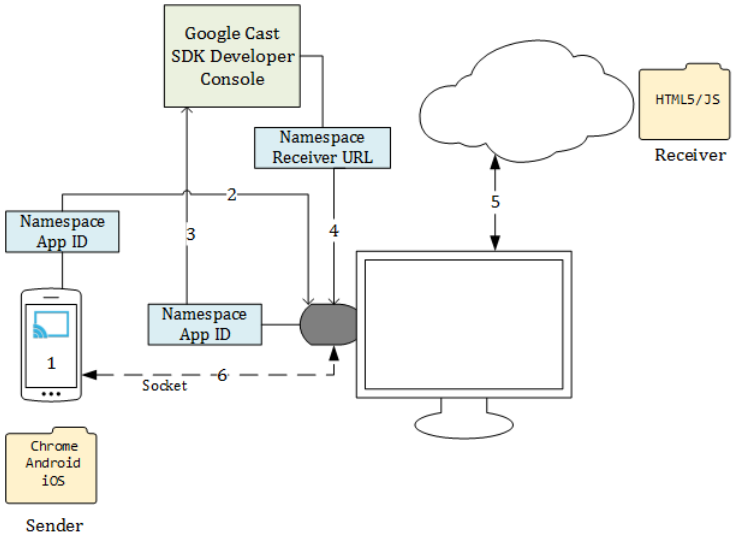


Figura 4.30: Flujo de comunicación con el dispositivo Chromecast.

También es necesario descargar las APIs (*Application Programming Interface*) necesarias para poder enviar contenido, así como la SDK de Google Cast. Estas librerías permiten desarrollar aplicaciones emisoras para Android, iOS y Chrome, y para aplicaciones receptoras en JavaScript, escritas para el navegador Chrome.

- Aplicación emisora (sender) Android.
  Será necesario añadir las siguientes librerías como dependencias de nuestro proyecto en Android:
  - Librería *Google Play Services* [34]. En concreto será necesario instalar una versión superior a la 4.2.
  - Librería *v7 mediarouter* [35]. Ésta proporciona los componentes necesarios para controlar el enrutamiento de los canales multimedia y el flujo desde un dispositivo hacia monitores externos.
- Aplicación emisora (sender) Chrome.
  Para usar la API del sender de Chrome hay que añadir `https://www.gstatic.com/cv/js/sender /v1/cast_sender.js` en nuestra página de Chrome.
- Aplicación receptora (receiver).
  La API del receiver estará disponible una vez que registremos nuestro

dispositivo. Para ello, tendremos que incluir en nuestra aplicación receptora
`https://www.gstatic.com/cast/sdk/libs/receiver/2.0.0/cast_receiver.js`.

A continuación procederemos con la descripción de los diferentes bloques que conforman una aplicación para Chromecast.

4.3.1. Aplicación sender

Esta aplicación será la encargada de lanzar la aplicación receptora (receiver) en el Chromecast y podrá comunicarse con el receiver intercambiando mensajes. Es una especie de aplicación de control remoto que controla a la aplicación receptora. Para determinar el tipo de aplicación sender que mejor se adapta a nuestro proyecto podemos consultar la figura 4.31.

Android sender

En este apartado es muy importante mencionar que para realizar pruebas es indispensable disponer de un dispositivo real de Android, ya que los emuladores no soportan la funcionalidad Cast. Una vez mencionado esto, definiremos, a alto nivel, las pautas para el desarrollo de una aplicación emisora para Android.

- La aplicación emisora inicia el descubrimiento de dispositivos con MediaRouter:
  `MediaRouter.addCallback`.
- MediaRouter informa a la aplicación emisora de la ruta que el usuario ha seleccionado:
  `MediaRouter.Callback.onRouteSelected`.
- La aplicación emisora recupera la instancia CastDevice:
  `CastDevice.getFromBundle`.
- La aplicación emisora crea el objeto GoogleApiClient:
  `GoogleApiClient.Builder`.
- La aplicación emisora se conecta con GoogleApiClient:
  `GoogleApiClient.connect`
- La SDK confirma si se ha conectado con GoogleApiClient:
  `GoogleApiClient.ConnectionCallbacks.onConnected`.
- La aplicación emisora lanza la aplicación receptora:
  `Cast.CastApi.launchApplication`.

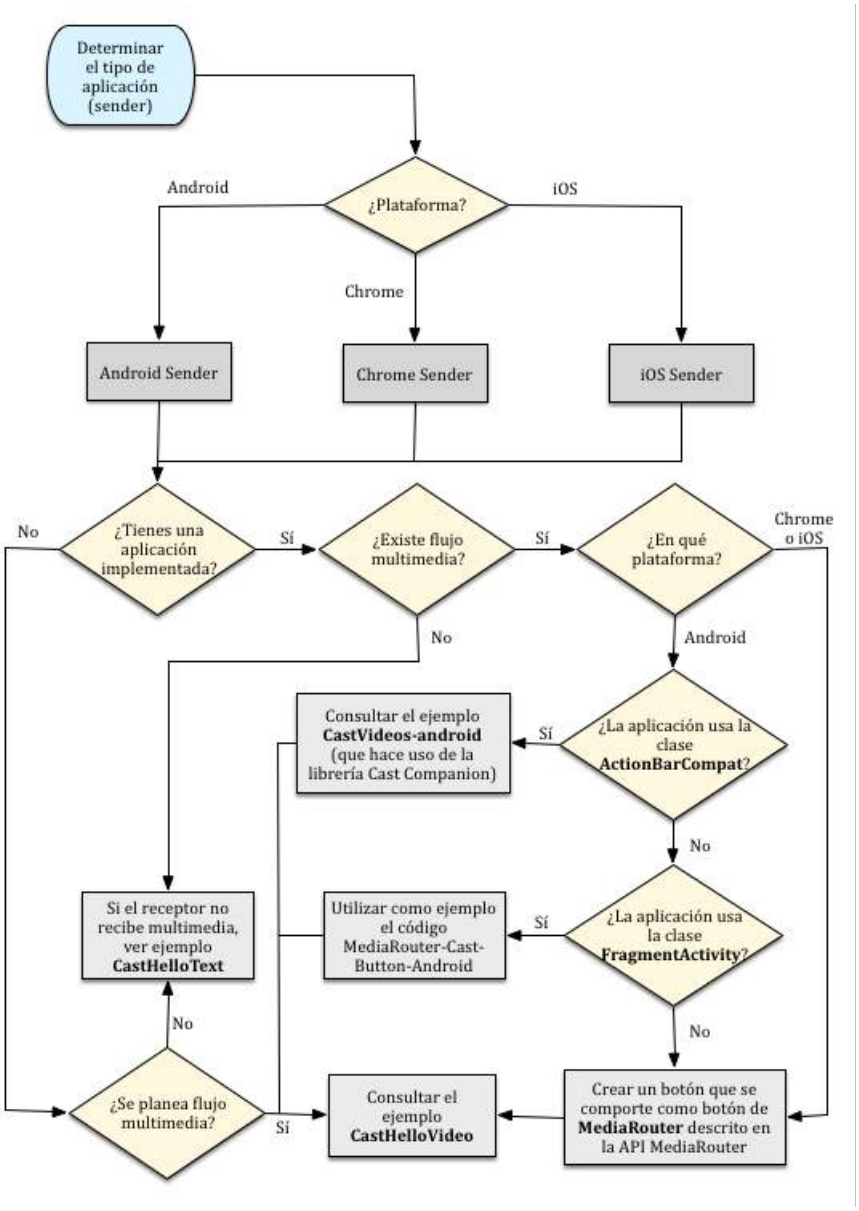


Figura 4.31: Proceso de selección del tipo de sender.

- La SDK confirma si se ha conectado con la aplicación receptora: `ResultCallback<Cast.ApplicationConnectionResult>`.
- La aplicación emisora crea un canal de comunicación: `Cast.CastApi.setMessageReceivedCallbacks`.
- El emisor envía un mensaje al receptor por el canal de comunicación:

`Cast.CastApi.sendMessage`.

Podemos consultar el sitio web de referencias de la API para Android de Google Cast [36] para una mayor compresión de cada una de las clases mencionadas.

Chrome sender

En concreto, una aplicación emisora o sender en Chrome se refiere a una aplicación web, escrita en HTML/JavaScript que se ejecuta en el navegador web Chrome en los sistemas operativos Mac, Windows, Linux y ChromeOS. No está disponible para las versiones Chrome de Android ni iOS.

Una vez que tengamos instalada la extensión Cast y hayamos completado el proceso de registro, estaremos preparados para desarrollar la aplicación emisora en Chrome.

- Inicialización.
  En primer lugar debemos inicializar la API. Para ello creamos el objeto `sessionRequest` con el número identificativo de nuestra aplicación. Este objeto lo utilizaremos para definir un nuevo objeto `ApiConfig`, junto a las llamadas `sessionListener` y `receiverListener`. Finalmente, realizamos una llamada al método `chrome.cast.initialize` que devolverá un resultado de éxito o de error.

initializeCastApi

```
initializeCastApi = function() {
        var sessionRequest = new chrome.cast.SessionRequest(applicationID);
          var apiConfig = new chrome.cast.ApiConfig(sessionRequest,
    sessionListener,
    receiverListener);
          chrome.cast.initialize(apiConfig, onInitSuccess, onError);
};
```

- Selección de dispositivo.
  Una vez que la llamada al método `initialize` devuelva un resultado de éxito, la función `receiverListener` informará si los dispositivos están disponibles. La selección de los dispositivos es llevada a cabo por la extensión Cast.

receiverListener

```
function receiverListener(e) {
          if( e === chrome.cast.ReceiverAvailability.AVAILABLE) {
          }
}
```

- Lanzamiento.
  Al pulsar el icono de la extensión Cast se lanza la llamada a la API `requestSession`. En esta llamada pasaremos como argumento la función `onRequestSessionSuccess` que devuelve un objeto del tipo `chrome.cast.Session` para el dispositivo en concreto que hayamos seleccionado. Este objeto también puede usarse para mostrar el nombre que hemos definido para nuestro Chromecast.

requestSession

```
chrome.cast.requestSession(onRequestSessionSuccess, onLaunchError);
```

  El método `requestSession` tiene dos argumentos: `onRequestSessionSuccess`, al que se llama cuando la sesión se ha creado con éxito (e.g. el usuario ha seleccionado un Chromecast) y `onLaunchError`, esta situación se da cuando la sesión falla por algún motivo (e.g. cuando el usuario no selecciona ningún Chromecast del desplegable).

OnRequestSessionSuccess

```
function onRequestSessionSuccess(e) {
      session = e;
 }
```

- Control de contenido multimedia.
  En este punto deberíamos poder cargar contenido multimedia en el Chromecast seleccionado, para ello creamos los objetos `MediaInfo` y `LoadRequest` y realizando una llamada al método `loadMedia`.

Control multimedia

```
var mediaInfo = new chrome.cast.media.MediaInfo(currentMediaURL);
var request = new chrome.cast.media.LoadRequest(mediaInfo);
session.loadMedia(request,
   onMediaDiscovered.bind(this, 'loadMedia'),
   onMediaError);

function onMediaDiscovered(how, media) {
   currentMedia = media;
}
```

  La función `onMediaDiscovered` devuelve, en caso de éxito, un objeto `chrome.cast.media.Media`, que proporciona un control absoluto del contenido multimedia con las funciones PLAY, PAUSE, RESUME, STOP y SEEK.

La función `loadMedia` es la función a la que llamaremos cuando se inicialice una sesión con éxito. Esta función habrá que desarrollarla para satisfacer la necesidades de nuestra aplicación emisora o sender.

- Cierre de sesión. A grandes rasgos, este sería el último de los grandes bloques que conforman una aplicación emisora para el navegador Chrome. El usuario puede iniciar este proceso al pulsar el botón '*Stop Casting*' en la extensión Cast para Chrome. Alternativamente, podemos incluir un botón específico para finalizar el envío de contenido multimedia desde nuestra aplicación.

stopApp

```
1 function stopApp() {
2   session.stop(onSuccess, onError);
3 }
```

4.3.2. Aplicación receiver

Una aplicación receptora, o receiver, es una aplicación HTML5/JavaScript que se ejecuta en el dispositivo receptor, como es el caso de Chromecast. Desempeña las siguientes funciones:

- Proporciona una interfaz para visualizar el contenido de la aplicación en la televisión o monitor al que conectemos Chromecast.
- Gestiona los mensajes de la aplicación emisora para controlar el contenido en el dispositivo receptor.
- Gestiona mensajes personalizados que envía el sender y que son específicos de la aplicación.

Como describiremos a continuación, existen distintos tipos de aplicaciones receiver, por lo que deberemos elegir que tipo de aplicación vamos a desarrollar en base a una serie de criterios, tal y como se puede observar en la figura 4.32.

Styled Media Receiver

Chromecast proporciona una aplicación receptora por defecto si lo que deseamos es enviar contenido multimedia como fotos, videos o audio. Este receiver incluye la aplicación html que proporciona el reproductor de video, de audio y de fotografías. Presenta la ventaja de que no tienes que realizar ningún cambio en la parte del receptor, tan solo tenemos que gestionar la parte del sender. Pero por otro lado, la desventaja es que no ofrece flexibilidad, ya que no podemos añadir ni eliminar nada en el Styled Media Receiver.

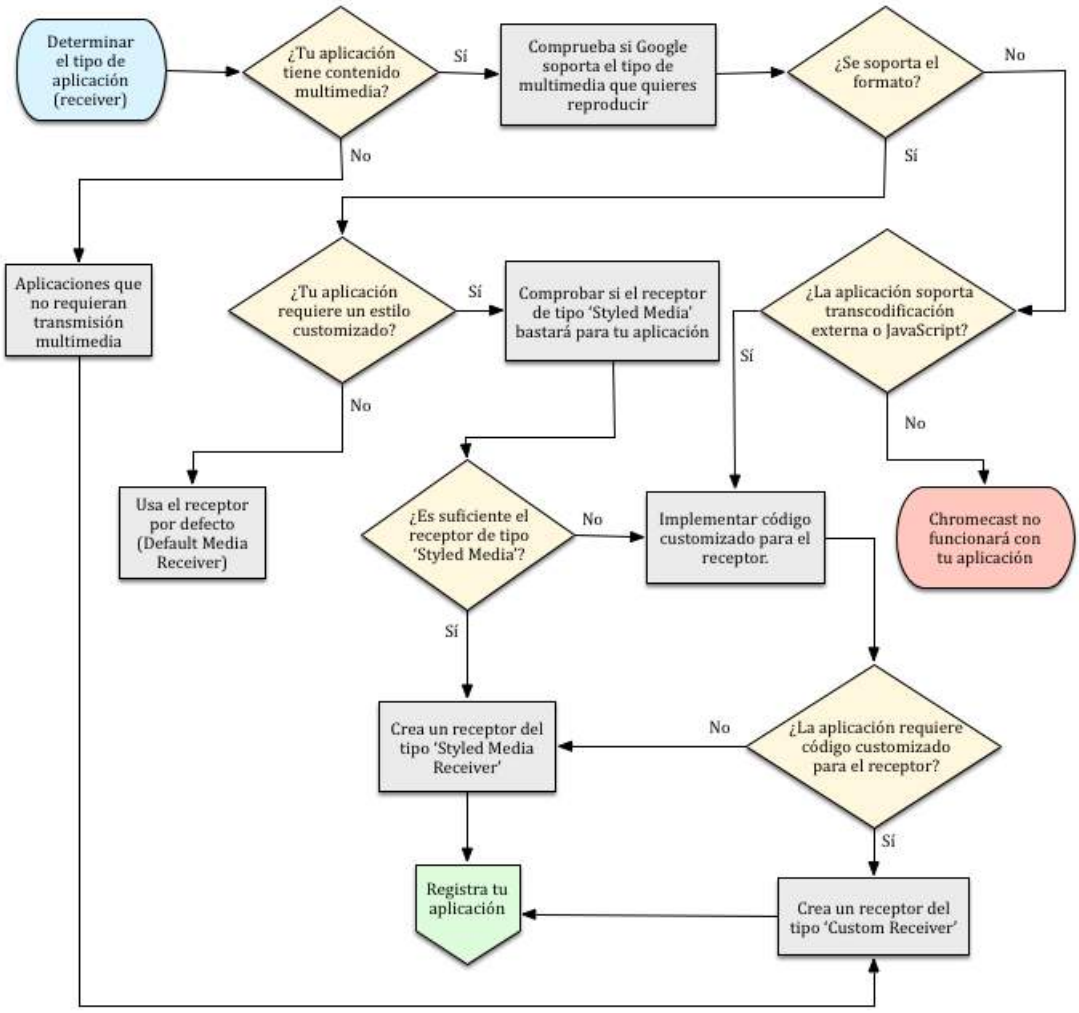


Figura 4.32: Proceso de selección del tipo de receiver.

Custom Receiver

Elegiremos este tipo de receiver si queremos crear una aplicación receptora con una determinada interfaz de usuario y diseño. El principal objetivo es ajustarse a nuestras necesidades, podremos decidir que contenido mostrar y de qué manera hacerlo. Si nuestra aplicación requiere el envío de contenido con un formato no recogido en la lista de formatos multimedia soportados [37], esta será nuestra opción. La única desventaja es que habrá que desarrollar la aplicación desde cero.

Default Media Receiver

Con esta tercera opción, el receiver por defecto, no es necesario que registremos nuestra aplicación, pero tampoco podremos customizar la aplicación. Bastaría con utilizar el número identificativo por defecto.

- Aplicaciones Android:
  `CastMediaControlIntent.DEFAULT_MEDIA_RECEIVER_APPLICATION_ID`
- Aplicaciones Chrome:
  `chrome.cast.media.DEFAULT_MEDIA_RECEIVER_APP_ID`

Desde la aplicación emisora se lanzará este receptor por defecto en el Chromecast y se usará para cargar las URLs.

Capítulo 5

Desarrollo e implementación

5.1. Diseño y arquitectura

5.1.1. Introducción a la arquitectura del sistema

En este capítulo se centrará la atención en el diseño e implementación de la aplicación para Android que se ha desarrollado como solución para este proyecto.

El diseño de esta aplicación se puede dividir en diferentes bloques, los cuales a su vez se subdividen, pudiendo identificar en ellos una parte del lado del cliente y otra del lado del servidor. Por ello se ha considerado adecuado abordar cada bloque funcional por separado, para poder definir y describir sus detalles.

5.1.2. Bloque de autenticación y autorización

El objetivo de un sistema de autenticación es certificar que el usuario que intenta acceder al contenido de nuestra aplicación es realmente quien dice ser. Para ello, utilizaremos una combinación única de identificadores, como son el nombre de usuario y su contraseña asociada. La autenticación no determina qué tareas puede llevar a cabo el usuario o qué datos puede visualizar, ya que sólo identifica y verifica al usuario. Por otro lado, la autorización consiste en dar acceso a una serie de recursos al usuario, previamente autenticado [38]. En nuestro caso, podemos tomar ambos conceptos como un bloque, debido a que el usuario que se autentique correctamente tendrá acceso a la aplicación en su totalidad.

En la figura 5.1 se muestra el proceso que debe seguir la aplicación cuando un usuario quiere acceder a la misma. En primer lugar, se mostrará una pantalla con un formulario para introducir los datos de acceso. Estos datos serán comprobados por el sistema de autenticación.

En esta primera versión de la aplicación se ha diseñado un sistema de acceso sencillo, usando los lenguajes HTML y PHP del lado del servidor y

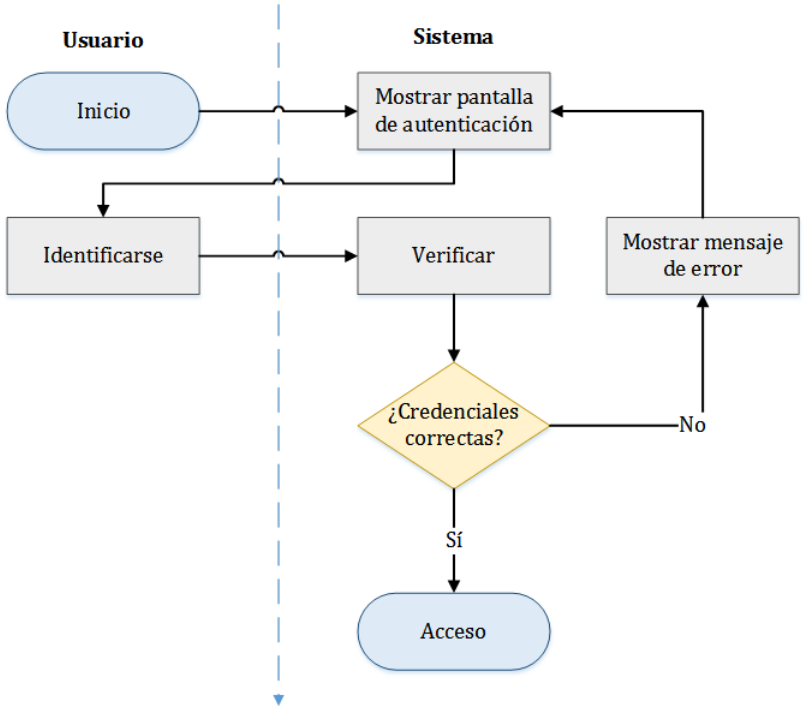


Figura 5.1: Diagrama de estados del sistema de autenticación

MySQL como gestor de la base de datos. Además, haremos uso de un servidor remoto para emular el comportamiento de un sistema de autenticación alojado en la nube. En apartados posteriores describiremos con más detalle la implementación de este sistema pero, a grandes rasgos, en la figura 5.2 se muestra el escenario para el sistema de autenticación.

Primeramente, el usuario (desde su dispositivo móvil) realizará una petición al servidor. Junto a dicha petición, enviará los datos (nombre de usuario y contraseña). El sistema comprobará esos datos, cotejándolos con los alojados en la base de datos, y devolverá una respuesta con el estado de la petición, es decir, si la autenticación es válida o no. En el caso de que el usuario no disponga de credenciales para acceder a la aplicación, se ha habilitado una opción de registro de usuarios.

5.1.3. Sistema de archivos

En este apartado recordaremos que uno de los principales objetivos de la solución software que se plantea es enviar contenido y material docente al dispositivo Chromecast, para su posterior visualización en un proyector o monitor. Ese material estará almacenado en la memoria externa del dispositivo (en la tarjeta SD (*Secure Digital*)) y se mostrará al usuario un listado

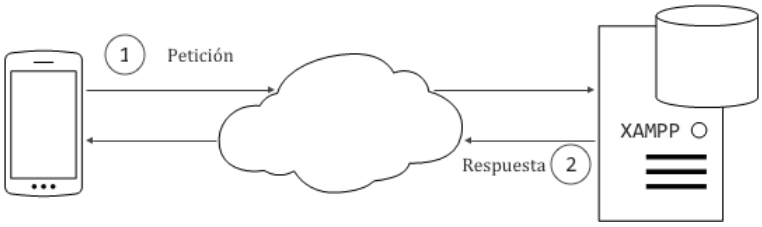


Figura 5.2: Arquitectura del sistema de autenticación

del contenido de la misma para que elija el archivo deseado. Con este fin, se diseñará un sistema de directorio de archivos que permita la navegación por las distintas carpetas disponibles y los archivos que contienen.

Una vez que se haya seleccionado el archivo final, el sistema devolverá su posición, es decir, su ruta completa y el nombre de dicho archivo. Este último valor será utilizado para obtener su extensión y poder discernir, en función a la misma, cuál será el siguiente paso.

En la figura 5.3 podemos observar gráficamente el procedimiento descrito anteriormente. Sólo se aceptarán los archivos PDF (*Portable Document Format*) y aquellas imágenes con extensiones PNG (*Portable Network Graphics*), GIF (*Graphics Interchange Format*), JPG/JPEG (*Joint Photographic Experts Group*).

5.1.4. Envío de material al dispositivo Chromecast

Como ya vimos en capítulos anteriores, la SDK *Cast* es solamente compatible con un número reducido de extensiones de archivos [37]. Por lo tanto, limitaremos el tipo de archivos que se pueden enviar a documentos PDF e imágenes (PNG, GIF, JPG/JPEG). Además, debemos adecuar la información que vamos a enviar a dichos requisitos.

Archivos PDF (*Portable Document Format*)

Para solventar las limitaciones de la SDK *Cast* se ha optado por convertir el archivo de tipo PDF que se desee enviar al Chromecast en imágenes, es decir, obtener una imagen por cada una de las páginas que conformen el archivo PDF. Una vez obtenidas esas imágenes, serán guardadas en una carpeta (que el sistema creará automáticamente) en la tarjeta de almacenamiento, para ser enviadas secuencialmente al dispositivo Chromecast, cuando el usuario así lo solicite. Una vez finalizada la tarea, se eliminará dicha carpeta. De manera paralela, para visualizar también en el dispositivo móvil el documento, será necesario codificar dichas imágenes en base 64 (codificación base64 [39]) para poder ser encapsuladas en un objeto de tipo HTML. Este

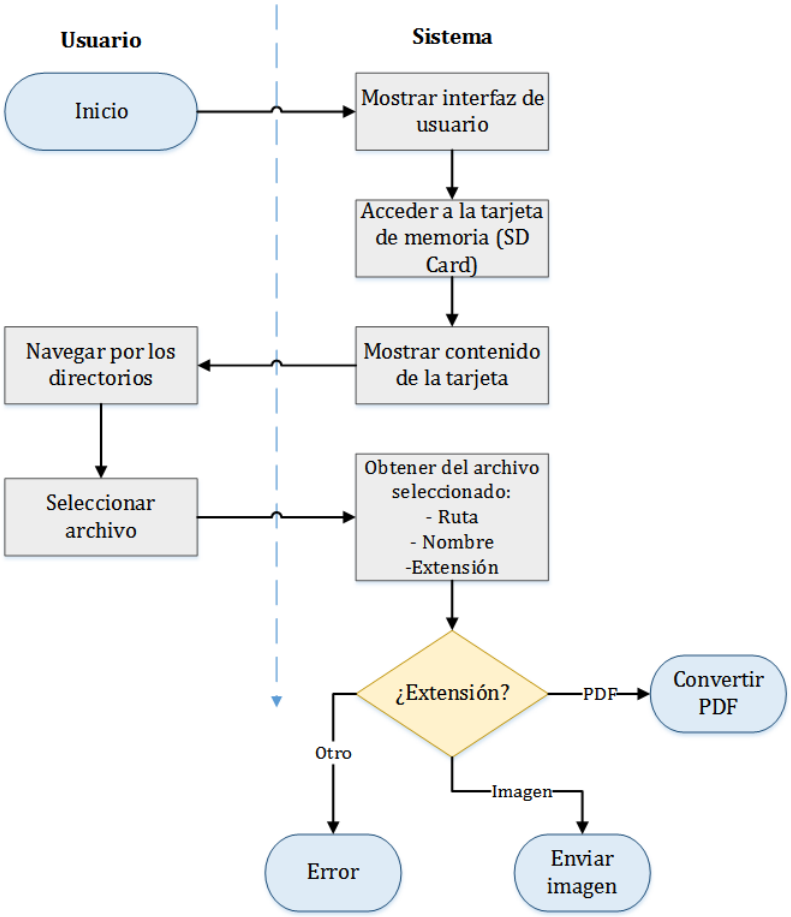


Figura 5.3: Diagrama de estados del sistema de archivos.

proceso, descrito gráficamente en la figura 5.4, será detallado en siguientes apartados para su mejor comprensión.

También podríamos reutilizar el resultado de la codificación en base 64 para enviar cada una de las imágenes, codificadas como una cadena de caracteres, con el método `sendMessage()`, de la SDK *Google Cast*, para posteriormente decodificarlas usando un *Custom Receiver*.

Sin embargo, no es recomendable usar el método `sendMessage` para enviar grandes cantidades de datos, ya que esos canales están indicados para ser usados como canales de control y no para enviar datos. Por esta razón, se ha optado por una alternativa más robusta que consiste en empotrar o implementar un servidor web en nuestra aplicación local, es decir, en el teléfono móvil que realizará las funciones de emisor. Así "serviremos"las imágenes obtenidas de cada una de las páginas del archivo PDF al Chromecast haciendo uso del (*Styled Media Receiver*). Podríamos usar también el receptor

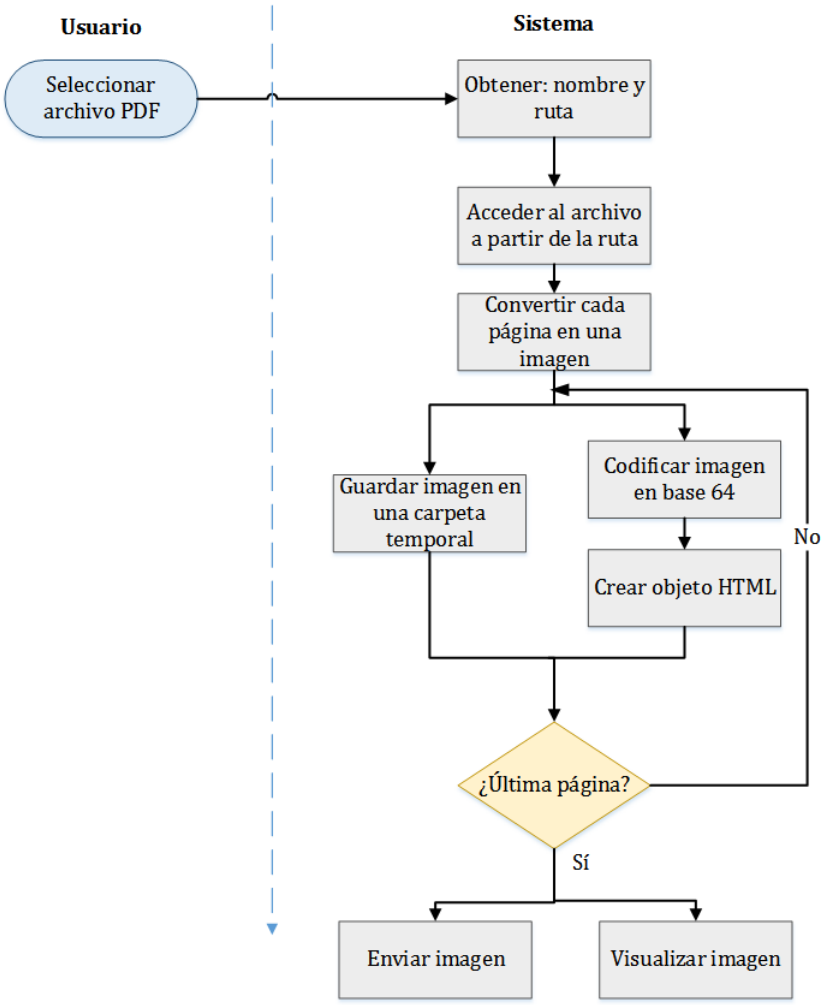


Figura 5.4: Diagrama de estados de la conversión de archivos PDF.

por defecto (*Default Receiver*), pero se ha decidido utilizar el anteriormente nombrado para poder así customizar la interfaz de espera en el proyector o monitor secundario, una vez nos hayamos conectado al dispositivo Chromecast. Configuraremos el servidor para que los archivos multimedia deseados, en este caso imágenes, estén disponibles en el servidor mencionado. Obtendremos una URL (*Uniform Resource Locator*) que apuntará a la dirección IP del servidor que contiene la imagen (dirección IP local del teléfono móvil). Además, esa URL coincidirá con la dirección IP local del teléfono móvil. De esta manera, el dispositivo Chromecast podrá acceder a la imagen a través de dicha URL como si se tratase de cualquier otro contenido de Internet.

5.2. Implementación de la aplicación cliente para Android

5.2.1. Descripción del sistema de autenticación

El primer bloque de esta aplicación cliente consiste en un sistema de autenticación y registro de usuarios basado en nombre de usuario y contraseña. Para implementar dicho sistema se ha elegido PHP (*Hypertext Preprocesor*), ya que es un lenguaje de código abierto y está centrado en la programación de funcionalidades del lado del servidor [40]. Además, será necesaria una base de datos para almacenar la información de los usuarios. En este caso, se ha decidido usar MySQL. Para facilitar el proceso de instalación y configuración de las herramientas necesarias que acabamos de mencionar, utilizaremos XAMPP [41], una distribución de Apache que contiene un servidor web Apache, MySQL y PHP, entre otros recursos.

La forma más sencilla de crear una nueva base de datos, así como la tabla que contendrá los datos, es mediante la opción *phpmyadmin*. Esta herramienta está accesible desde la página principal de nuestro servidor web. Nuestra base de datos, llamada *etsiitcast_db*, estará compuesta por dos columnas, una para almacenar el nombre de usuario y la otra para guardar su contraseña asociada.

Una vez hayamos creado nuestra base de datos, podemos pasar a la programación de la parte web del sistema, que contará con una serie de archivos que describiremos a continuación:

- Archivo de configuración de la base de datos. En este fichero se declararán aquellas variables referentes a la configuración de la base de datos, información que será reusable en el resto de ficheros. Por ejemplo, definiremos el nombre de la base de datos (que deberá coincidir con el nombre que hayamos definido al crear la base de datos), la dirección IP de la máquina que la aloja y el nombre de usuario y contraseña que hemos elegido en la configuración de MySQL y nuestra base de datos.

- Archivo para gestionar la conexión y desconexión de la base de datos. Tomaremos los valores de las diferentes variables del archivo mencionado anteriormente.

```
1 public function connect() {
2    $con = mysql_connect(ip_maquina, usuario_db,
3    pass_db);
4    mysql_select_db(nombre_db);
5    return $con;
6 }
7 public function close() {
```

```
8     mysql_close();
9  }
```

- Archivo en el que se definen los diferentes procedimientos que soporta nuestro sistema de autenticación, como son la validación de datos y el alta de nuevos usuarios. En el caso del alta de nuevos usuarios, se comprobará previamente que el nuevo usuario no exista en la base de datos, ya que cada usuario ha de ser único. En caso de no encontrar ninguna coincidencia, se insertará una nueva fila en nuestra tabla de datos.

```
1  public function adduser($username, $password) {
2     $result = mysql_query("INSERT INTO usuarios(username,
3     passw)
4     VALUES('$username', '$password')");
5     // Comprobamos si ha tenido exito la insercion
6     if ($result) {
7        return true;
8     } else {
9        return false;
10    }
11 }
12
13 // Comprobamos la existencia de un usuario dado el nombre
14 public function isuserexist($username) {
15    $result = mysql_query("SELECT username from usuarios
16    WHERE username = '$username'");
17    $num_rows = mysql_num_rows($result);
18    if ($num_rows > 0) {
19             // Existe una coincidencia
20             return true;
21    } else {
22            return false;
23    }
24 }
25
26 // Validamos los datos para autenticar al usuario
27 public function login($user,$passw){
28    $result=mysql_query("SELECT COUNT(*) FROM usuarios
29    WHERE username='$user' AND passw='$passw' ");
30    $count = mysql_fetch_row($result);
31    if ($count[0]==0){
32         return true;
```

```
33      }else{
34           return false;
35      }
36  }
```

También debemos incluir un archivo que se encargue de devolver mensajes de éxito y/o error en caso de que optemos por la opción de registro (en vez de autenticación) de usuarios. A su vez, definiremos un archivo HTML que muestre una página web sencilla, en la cual se llevará a cabo la tarea del registro de nuevos usuarios desde el navegador web de nuestro dispositivo móvil.

- Archivo de acceso. En caso de que la validación de datos tenga éxito en el paso anterior, este nuevo archivo se encarga de devolver un mensaje de éxito, en formato JSON [42] (*JavaScript Object Notation*), un formato para intercambio de datos.

```
1  if($db->login($usuario,$passw)){
2      $resultado[]=array("logstatus"=>"0");
3          }else{
4          $resultado[]=array("logstatus"=>"1");
5          }
6  echo json_encode($resultado);
```

Debemos alojar estos archivos en la carpeta correspondiente de nuestro servidor para que todo funcione correctamente.

A continuación, se procederá con el análisis de la parte referente a la aplicación de Android.

LoginActivity.java

Esta será la primera actividad que se ejecutará al lanzar nuestra aplicación, y así lo hemos de definir en nuestro archivo `AndroidManifest.xml`. Esta será la clase principal del sistema de autenticación, que servirá para dar acceso a la aplicación. Se incluyen las variables de acceso a la base de datos, como son la dirección IP de la máquina que aloja el servidor y el nombre de la base de datos. Dichos parámetros deberán configurarse de manera estática en el código fuente para asegurar el correcto funcionamiento de la parte del servidor de autenticación.

Esta clase se encarga de recoger los datos introducidos por el usuario (nombre de usuario y contraseña). Para ello se han definido dos campos de texto (`EditText` [43]), uno de tipo *textPersonName* para introducir el nombre de usuario y el otro de tipo *textPassword* para la contraseña del mismo. Al pulsar el botón *Entrar*, se realiza una llamada a un método que comprobará

que los campos no estén en blanco, es decir, vacíos. Si ambos campos están vacíos, la aplicación lanzará un mensaje de error de autenticación; en caso contrario, se procederá a enviar los datos al sistema. Mientras dure el proceso de autenticación, se mostrará un mensaje de espera. Esta tarea será ejecutada por una hebra en segundo plano, por lo que se definirá una clase que herede de la clase *AsyncTask* [44].

Además, contiene el método `loginstatus` que realiza la validación del usuario con los datos dados. En primer lugar, creará una lista para agregar los datos pasados, que serán enviados en una petición POST al sistema para realizar la validación, siendo la respuesta a la petición un objeto con formato JSON, tal y como se describió en el apartado anterior. Si la autenticación tiene éxito, se invocará a la siguiente actividad. Para gestionar y controlar las conexiones y peticiones de tipo HTTP se hará uso de una clase auxiliar, `Httppostaux.java`, que también se encarga de convertir los datos a formato JSON o formato *String*, según sean los requisitos.

En la figura 5.5 se muestra la interfaz de usuario correspondiente a la clase `LoginActivity.java`.

Figura 5.5: Interfaz de usuario de la clase *LoginActivity.java*

5.2.2. Panel lateral de navegación

Una vez hayamos accedido a la aplicación, debemos elegir la opción o tarea que queremos llevar a cabo. Para mostrar las diferentes acciones disponibles, se ha optado por diseñar un menú en forma de panel lateral desplegable, o *navigation drawer* [45]. Un *navigation drawer* es un panel que muestra las opciones posibles de navegación en el extremo izquierdo de la pantalla de nuestro dispositivo. Se puede plegar o desplegar en el momento que el usuario desee y, para ello, tiene dos opciones. La primera sería desplazar el dedo desde el extremo izquierdo de la pantalla hacia la derecha para desplegarlo o desde el extremo derecho del menú hacia la izquierda si éste se encontraba previamente desplegado, para ocultar el menú. Por otro lado, el usuario puede pinchar en el logo de la aplicación situado en la barra superior.

MainActivity.java

En esta actividad definiremos todo lo relativo a la inicialización y control del *navigation drawer*. Lo primero que haremos en esta actividad será inicializar la lista de elementos que formarán parte del menú desplegable. Para ello, definimos un objeto de tipo *ListView* y lo rellenaremos con una lista de elementos, asignando un icono identificativo a cada uno. También invocaremos el método `setOnItemClickListener()` para recibir los eventos que se produzcan al seleccionar alguna de las opciones de la lista.

```
1  @Override
2  protected void onCreate(Bundle savedInstanceState) {
3      super.onCreate(savedInstanceState);
4      setContentView(R.layout.activity_main);
5
6      itemTitle = activityTitle = getTitle();
7      tagTitles = getResources().getStringArray(R.array.nav_options);
8      drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
9      drawerList = (ListView) findViewById(R.id.left_drawer);
10                 ...
11
12     ArrayList<item_object> items = new ArrayList<item_object>();
13     items.add(new item_object(tagTitles[1],
14                     R.drawable.ic_action_person));
15     items.add(new item_object(tagTitles[2],
16                     R.drawable.ic_action_search));
17     items.add(new item_object(tagTitles[3],
18                     R.drawable.ic_action_sd_storage));
19     items.add(new item_object(tagTitles[4],
```

```
                    R.drawable.ic_action_about));

    drawerList.setAdapter(new NavigationAdapter(this, items));
    drawerList.setOnItemClickListener(new DrawerItemClickListener());
        ...
}
```

Cuando el usuario seleccione un elemento de la lista, el sistema realiza una llamada al método `onItemClick()` de la clase `DrawerItemClickListener`. Dentro del método mencionado anteriormente, debemos definir el comportamiento esperado tras seleccionar una opción de las disponibles. En nuestro caso, se creará un nuevo fragmento (*Fragment* [46]). Definiremos un fragmento para cada una de las posibles opciones. Un fragmento es una parte de la interfaz de usuario de la aplicación o un comportamiento determinado que puede ser insertado en una actividad y que se ejecuta junto a ésta.

```
private class DrawerItemClickListener
implements ListView.OnItemClickListener {
    @Override
    public void onItemClick(AdapterView<?> parent,
    View view, int position, long id) {
        selectItem(position);
    }
}

private void selectItem(int position) {

    Fragment fragment = null;
    switch (position) {
                case 1:
                        fragment = new PerfilFragment();
                 break;
                case 2:
                  fragment = new SeleccionarFragment();
                  break;
               case 3:
                      fragment = new GestorFragment();
                     break;
                case 4:
                     fragment = new InfoFragment();
                    break;
                default:
                    break;
```

```
28            }
29
30            if (fragment != null) {
31            FragmentManager fragmentManager =
32                          getFragmentManager();
33                 fragmentManager.beginTransaction().
34                         replace(R.id.content_frame, fragment).commit();
35
36                drawerList.setItemChecked(position, true);
37                 drawerList.setSelection(position);
38                 setTitle(tagTitles[position]);
39                drawerLayout.closeDrawer(drawerList);
40              } else {
41                  Log.e("MainActivity",
42                          "Error al crear el fragmento");
43            }
44  }
```

Como nuestra actividad incluye una barra en la parte superior, *action bar* [47], debemos crear un objeto que herede de la clase `ActionBarDrawerToggle`, para facilitar la interacción entre el icono del *action bar* y el menú desplegable. El icono de la aplicación indicará la existencia del menú desplegable con un icono especial que se añade justo al lado del primero. También haremos uso de este objeto para actualizar el título en el *action bar* al cerrar o abrir el menú desplegable para que así coincida con el título de la opción seleccionada en el mismo.

```
1   drawerToggle = new ActionBarDrawerToggle(
2           this,
3              drawerLayout,
4               R.string.drawer_open,
5              R.string.drawer_close
6   ) {
7              public void onDrawerClosed(View view) {
8                 getActionBar().setTitle(itemTitle);
9              }
10
11               public void onDrawerOpened(View drawerView) {
12                       getActionBar().setTitle(activityTitle);
13
14              }
15  };
16  drawerLayout.setDrawerListener(drawerToggle);
```

Para finalizar, debemos tener en cuenta que para añadir un *navigation drawer*, debemos declarar en el diseño de nuestra interfaz un objeto de tipo *DrawerLayout* como elemento raíz del *layout*. Una vez realizados algunos ajustes de diseño, la vista con el menú desplegado será la que se observa en la figura 5.6.

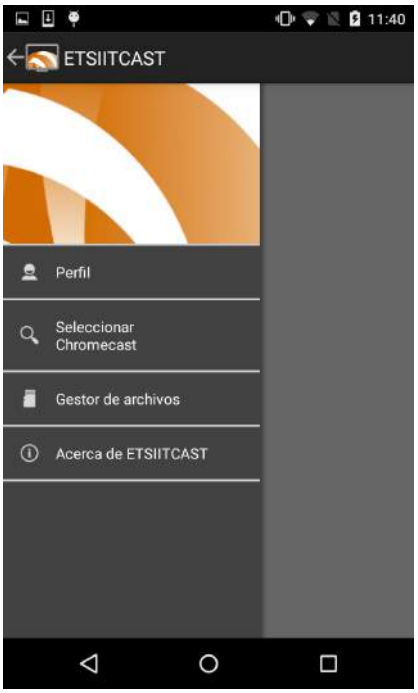


Figura 5.6: Interfaz de usuario de la clase *MainActivity.java*

5.2.3. Perfil del usuario

La primera opción disponible del menú corresponde con el título "Perfil". En este apartado se podrá consultar información referente al usuario y a su conexión a la red inalámbrica. En concreto, se mostrará el nombre con el que el usuario ha accedido al sistema, el SSID (*Service Set Identifier*) con el que identificaremos el nombre de la red a la que estamos conectados y la dirección IP del dispositivo móvil dentro de la misma. Se ha considerado oportuno mostrar esta información para facilitar al usuario localizar cualquier problema de conexión que pueda darse en el sistema. Esta información también será necesaria en caso de que se quiera realizar la configuración del dispositivo Chromecast en la red, como hemos descrito es capítulos anteriores. Por motivos de seguridad, no se mostrará la contraseña proporcionada por el usuario.

También se ha implementado en este apartado un procedimiento para dar de baja a los usuarios, es decir, para eliminar al usuario de nuestra base de datos dado el nombre de usuario con el que se ha accedido a la aplicación.

PerfilFragment.java

Al ser una vista sencilla, crearemos un fragmento en este caso. Aunque un fragmento tiene su propio ciclo de vida, éste depende de la actividad que lo invoca. En este caso, el ciclo de vida del fragmento incluye métodos básicos del ciclo de vida de una actividad, como `onResume`. Vamos a usar este método para deshabilitar el botón *Atrás* del móvil y evitar así que el usuario vuelva a la vista de autenticación (es decir, salir de la aplicación), al pulsarlo. Esta función se añadirá en el resto de bloques de la aplicación, de manera análoga. En caso de que el usuario quiera salir de la aplicación, se ha habilitado un botón de *Cerrar sesión* con ese propósito en esta vista.

```
1  @Override
2  public void onResume() {
3      super.onResume();
4      getView().setFocusableInTouchMode(true);
5      getView().requestFocus();
6      getView().setOnKeyListener(new View.OnKeyListener() {
7              @Override
8              public boolean onKey(View v, int keyCode, KeyEvent event) {
9                      if (event.getAction() == KeyEvent.ACTION_UP
10                             && keyCode == KeyEvent.KEYCODE_BACK) {
11                                 return true;
12               }
13              return false;
14             }
15     });
16 }
```

Para que el sistema nos devuelva la información referente a la conexión inalámbrica haremos uso de la clase *WifiManager* [48]. Esta clase proporciona la API necesaria para gestionar todos los aspectos de la conectividad Wi-Fi. También será necesario crear un objeto de la clase *WifiInfo* [49], que describe el estado de la conexión inalámbrica. En concreto, invocaremos los métodos `getSSID` que devuelve el identificador SSID de la red 802.11 actual, y el método `getIpAddress` para obtener la dirección IP del dispositivo móvil. Debemos convertir el resultado del segundo método en un formato adecuado tipo String o cadena de caracteres.

```
WifiManager wifiMgr = (WifiManager) getActivity().
        getSystemService(Context.WIFI_SERVICE);
WifiInfo wifiInfo = wifiMgr.getConnectionInfo();
String name = wifiInfo.getSSID();
int ip_address = wifiInfo.getIpAddress();
String dir_ip=String.format("%d.%d.%d.%d",(ip_address & 0xff),
        (ip_address >> 8 & 0xff),(ip_address >> 16 & 0xff),
        (ip_address >> 24 & 0xff));
```

Para que este código funcione es imprescindible añadir una serie de permisos para nuestra aplicación en el archivo `AndroidManifest.xml`.

- `ACCESS_WIFI_STATE`. Permite a la aplicación acceder a información sobre las redes Wi-Fi.
- `ACCESS_NETWORK_STATE`. Permite a la aplicación acceder a la información referente a las redes.

En la figura 5.7 se muestra la interfaz de usuario de la clase `PerfilFragment`.

Figura 5.7: Interfaz de usuario de la clase *PerfilFragment.java*

Para finalizar este apartado, se va a detallar la funcionalidad de baja de usuarios, disponible al pulsar el botón "Dar de baja mi usuario". Debido a

que es una acción que no se puede revertir, antes de hacer efectiva la baja del usuario, se mostrará un diálogo para confirmar dicha acción (ver figura 5.8). Esto reducirá las posibilidades de que eliminemos nuestra cuenta de usuario por error.

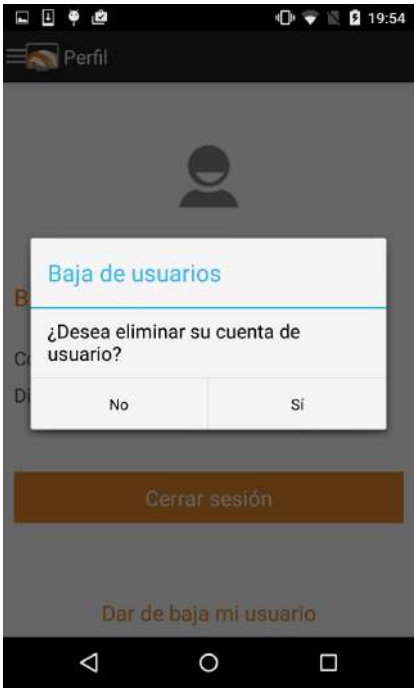


Figura 5.8: Diálogo de alerta antes de proceder con la baja del usuario.

Es importante tener en cuenta que no debemos realizar llamadas o peticiones a la red en la hebra principal de nuestra actividad. Por lo tanto, realizaremos las peticiones HTTP de tipo POST, es decir, nuestra llamada a la red, en un segundo plano. Para ello utilizaremos *AsyncTask*. En la petición POST enviaremos a nuestro servidor los datos que van a servir como criterio para eliminar una entrada la tabla en nuestra base de datos. Concretamente, enviaremos el nombre de usuario con el que hemos accedido a la aplicación, para eliminar la fila que coincida con ese nombre de usuario que hemos pasado.

```
1 // URL a la que vamos a enviar los datos
2 String postReceiverUrl = "http://" + IP_Server + "/etsiitcast_db/
3 deleteuser.php";
4 Log.v(TAG, "postURL: " + postReceiverUrl);
5
```

```
// HttpClient
HttpClient httpClient = new DefaultHttpClient();
// Cbecera POST
HttpPost httpPost = new HttpPost(postReceiverUrl);

// Anadimos los datos que queremos enviar
String username = getActivity().getIntent().getExtras()
        .getString("user");
List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
nameValuePairs.add(new BasicNameValuePair("username", username));

httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

// Ejecutamos la peticion HTTP de tipo POST
HttpResponse response = httpClient.execute(httpPost);
HttpEntity resEntity = response.getEntity();

if (resEntity != null) {
        String responseStr = EntityUtils.toString(resEntity).trim();
}
```

Como vemos al principio del extracto de código detallado anteriormente, se hace referencia al archivo `deleteuser.php`. Este archivo deberá estar almacenado en la misma carpeta en la que recogimos el resto de archivos *php* para asegurar el correcto funcionamiento de la base de datos. En dicho archivo, se gestionará la conexión a la base de datos, de manera similar a la seguida en apartados anteriores.

```
$name = implode(",", $_POST);

$sql = "DELETE FROM usuarios
        WHERE username = '$name'";
```

La variable `$_POST` contiene el array con los datos que hemos enviado desde la aplicación Android. Como no se puede realizar una consulta a la base de datos sobre un *array*, haremos uso de la opción `implode` para unir los elementos de un *array* en un elemento de tipo *string*.

5.2.4. Descubrimiento de dispositivos Chromecast

En este apartado trataremos la segunda opción del menú desplegable de nuestra aplicación, "Seleccionar Chromecast". El fin de este apartado es

detectar los dispositivos Chromecast conectados y disponibles, así como comprender el proceso de descubrimiento de dispositvos. También recurriremos a esta opción en caso de que se produzca algún error en el envío de datos al Chromecast (esta opción será detallada en apartados posteriores) y necesitemos cerrar alguna conexión que se haya quedado establecida tras el error. Al igual que en la opción detallada anteriormente, "Perfil", al elegir "Seleccionar Chromecast"del menú, se creará un fragmento, el cual se encargará de crear o *inflar* el diseño (*layout*) y gestionar la llamada a la actividad `SeleccionarActivity` cuando el usuario pulse el botón que se muestra en la figura 5.9

Figura 5.9: Interfaz de usuario de la clase *SeleccionarFragment.java*

SeleccionarActivity.java

Esta actividad es una clase de prueba para demostrar el uso de `MediaRouteButton` [50], responsable de configurar la visibilidad del botón *Cast* en función del estado de la conexión con el dispositivo Chromecast. También haremos uso de la clase `MediaRouter.Callback`, una interfaz que recibe los eventos producidos por cambios en la conexión y descubrimiento de dispositivos. Estos eventos pueden referirse a alguno de los estados que mencionamos a continuación:

- Estado inicial en el que no hay ningún dispositivo Chromecast disponible. El botón *Cast* o botón que nos indica la presencia de un dispositivo Chromecast disponible y la posibilidad de enviar contenido multimedia a éste, no estará visible por defecto. Así lo definiremos en el diseño de dicho elemento en el *layout* mediante la etiqueta `android.support.v7.app.MediaRouteButton` y la propiedad `android:visibility="gone"`. Por lo tanto, si no se descubre ningún Chromecast disponible, no será visible el botón como podemos observar en la figura 5.16. Para conocer más información, definimos un elemento de la clase `MediaRouter.RouteInfo` para que nos muestre en la consola, o *log* de depuración, más detalles sobre el estado de la operación. En el caso de que no se detecte ningún dispositivo Chromecast, nos devolverá información del teléfono móvil desde el que hemos realizado la llamada de descubrimiento:
```
Found default route: MediaRouter.RouteInfo uniqueId=android/
.support.v7.media.SystemMediaRouteProvider:DEFAULT_ROUTE,
name=Teléfono, description=null, enabled=true,
connecting=false, canDisconnect=false, playbackType=0,
playbackStream=3, volumeHandling=1, volume=3, volumeMax=15,
presentationDisplayId=-1, extras=null, settingsIntent=null,
providerPackageName=android
```

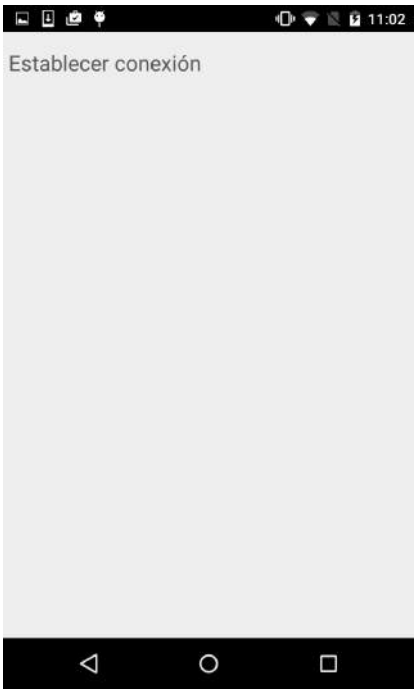


Figura 5.10: No se ha descubierto ningún dispositivo Chromecast

- Se han descubierto o detectado dispositivos disponibles. Cuando se detecta la existencia de un dispositivo Chromecast, se realiza una llamada al método `onRouteAdded` y, como resultado, se hace visible el botón *Cast* como podemos ver en la figura 5.11.

Figura 5.11: Se han detectado dispositivos Chromecast disponibles

```
1 public void onRouteAdded(MediaRouter router,
2 MediaRouter.RouteInfo route) {
3     Log.d(TAG, "onRouteAdded");
4     if (++mRouteCount == 1) {
5           mMediaRouteButton.setVisibility(View.VISIBLE);
6         }
7 }
```

Al seleccionar el botón *Cast*, ahora visible, se mostrará un diálogo con un listado de los dispositivos Chromecast que se hayan descubierto (ver figura 5.12)

- Se selecciona un dispositivo de los disponibles. Una vez que seleccionemos un dispositivo de la lista, se invoca el método `onRouteSelected`.

```
1 public void onRouteSelected(MediaRouter router,
2 MediaRouter.RouteInfo info) {
3     Log.d(TAG, "onRouteSelected");
4     mSelectedDevice = CastDevice.getFromBundle(info.getExtras());
5 }
```

En este caso, la información que muestra el *log* corresponde con el dispositivo Chromecast que se ha seleccionado:

```
onRouteUnselected: info=MediaRouter.RouteInfo uniqueId=
com.google.android.gms/.cast.media.CastMediaRouteProvider
Service:9c97bd62eb0299ab69ed314d524c7432, name=Chromecast_
irene, description=Chromecast, enabled=true, connecting=
true, canDisconnect=false, playbackType=1, playbackStream=-1,
volumeHandling=0, volume=0, volumeMax=20, presentation
DisplayId=-1, extras=Bundle(mParcelledData.dataSize=592),
```

Figura 5.12: Lista de dispositivos Chromecast disponibles descubiertos.

`settingsIntent=null, providerPackageName=com.google.android.gms`

Además, el botón *Cast* cambiará de color para indicar el estado de conectado (figura 5.16).

Figura 5.13: Se ha establecido conexión con el dispositivo Chromecast seleccionado.

- Se desconecta el dispositivo. Por último, si se vuelve a pulsar el botón *Cast* una vez se ha establecido la conexión, se puede finalizar la misma con el Chromecast seleccionado, realizándose una llamada al método `onRouteRemoved`:

```
1 public void onRouteRemoved(MediaRouter router,
2 MediaRouter.RouteInfo route) {
3     Log.d(TAG, "onRouteRemoved");
```

```
4       if (--mRouteCount == 0) {
5             mMediaRouteButton.setVisibility(View.VISIBLE);
6           }
7   }
```

Se mantiene el estado del botón como visible para no forzar la inhabilitación de dicho botón en caso de que, tras la desconexión, sigan existiendo dispositivos disponibles.

Para utilizar esta API de *MediaRouter* es necesario añadir como dependencia al proyecto en *Android Studio* la correspondiente librería (`com.android.support:mediarouter-v7:22.2.1`).

5.2.5. Gestor de archivos

Pasamos al tercer apartado del menú, "Gestor de archivos". A partir de esta opción se accederá al grueso de la aplicación: la selección del archivo deseado y su envío al dispositivo Chromecast. De igual manera que en la sección anterior, será un fragmento inicial el que lanzará, en este caso, un diálogo que mostrará el contenido de la tarjeta de almacenamiento.

GestorActivity.java

Es esta la actividad encargada de definir el sistema de directorio de archivos y de gestionar la función de navegación entre las distintas carpetas y archivos. También se recogerán las variables necesarias que, más tarde, rehusaremos para completar el envío de material al Chromecast.

Haremos uso de la clase *Environment* [51], que proporciona el acceso a distintas variables del entorno, y del método `getExternalStorageDirectory()`. Éste método devuelve el directorio raíz del almacenamiento externo del dispositivo móvil (e.g. `/storage/emulated/0/`).

A *grosso* modo, primero comprobaremos que esa ruta devuelta exista. En caso afirmativo, crearemos una lista en la que iremos almacenando los distintos directorios a los que se van accediendo para conseguir la ruta completa del archivo final que seleccionemos. Si pulsamos el botón *Atrás* disponible, se eliminará el último directorio de dicha lista.

```
1  [...]
2  // Comprobamos si el path existe
3  if (path.exists()) {
4      FilenameFilter filter = new FilenameFilter() {
5          @Override
6              public boolean accept(File dir, String filename) {
```

```
                    File sel = new File(dir, filename);
                      return (sel.isFile() || sel.isDirectory())
                             && !sel.isHidden();

          }
};
[...]

@Override
public void onClick(DialogInterface dialog, int which) {
    chosenFile = fileList[which].file;
           File sel = new File(path + "/" + chosenFile);
            if (sel.isDirectory()) {
             firstLvl = false;
          // Anadimos el directorio seleccionado a la lista
              str.add(chosenFile);
               fileList = null;
              path = new File(sel + "");

               loadFileList();
           }

    // Comprobamos si se ha pulsado el boton "Atras"
          else if (chosenFile.equalsIgnoreCase("Atras")
           && !sel.exists()) {
               // eliminamos de la lista el directorio actual
                String s = str.remove(str.size() - 1);
              // Modificamos la ruta para eliminar el directorio
                path = new File(path.toString().substring(0,
                           path.toString().lastIndexOf(s)));
               fileList = null;
        // Si no hay mas directorios en la lista,
        // nos encontramos en el primer nivel
                 if (str.isEmpty()) {
              firstLvl = true;
                }
                loadFileList();
        }
           // Archivo seleccionado
            else {
                    // Definimos los pasos a seguir en funcion
                    // de la extension del archivo
                    switch (extension) {
```

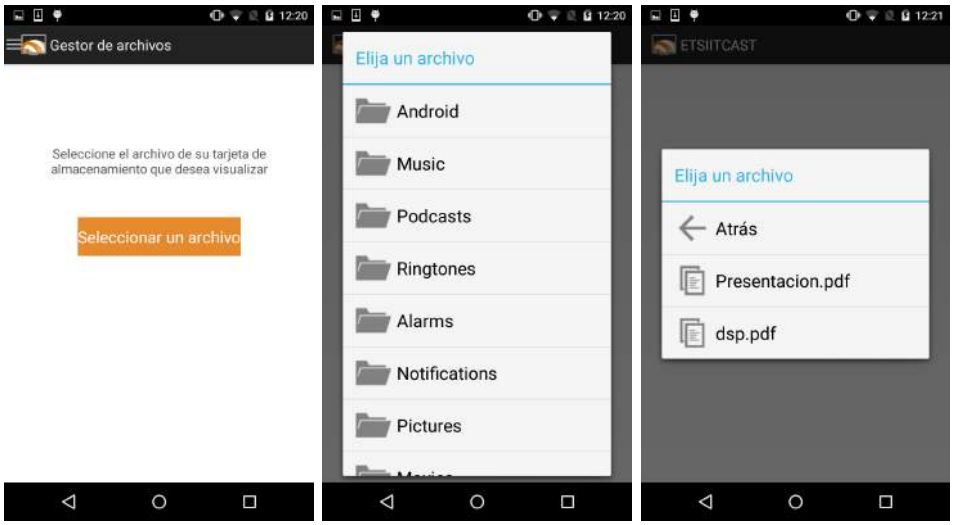
```
50                  case "application/pdf":
51                       Intent intent = new Intent(GestorActivity.this,
52                       ConvertPDF_Activity.class);
53                            intent.putExtra("param1", ruta);
54                          intent.putExtra("param2", chosenFile);
55                          startActivity(intent);
56                   break;
57
58                    case "image/png":
59                         Intent intent_png = new Intent(GestorActivity.this,
60                         CastImg_Activity.class);
61                   intent_png.putExtra("param2", chosenFile);
62                          intent_png.putExtra("param1", ruta);
63                          intent_png.putExtra("param3", extension);
64                          startActivity(intent_png);
65                         break;
66                          // Idem para el resto de extensiones de
67                          // tipo "image"
68                         default:
69                          Toast.makeText(GestorActivity.this,
70                            "Por favor, elija otro archivo",
71                            Toast.LENGTH_LONG).show();
72                          break;
73                     [...]
74             }
75  }
```

Una vez que hayamos seleccionado un archivo, calcularemos la extensión de éste a partir de su nombre, que será del tipo *nombre_archivo.extensión*. Para ello definimos el método `getExtension (String chosenFile)` que devuelve un objeto de tipo *String* con el formato MIME (*Multipurpose Internet Mail Extensions*) correspondiente para cada extensión a partir de la subcadena localizada a partir del punto que separa el nombre de la propia extensión.

```
1  public String getExtension(String chosenFile){
2      String type= " ";
3      if (chosenFile.substring(chosenFile.lastIndexOf("."),
4      chosenFile.length()).equals(".png")) {
5                       type = "image/png";
6              }
7            else if (chosenFile.substring(chosenFile.lastIndexOf("."),
8            chosenFile.length()).equals(".gif")){
9                type = "image/gif";
```

```
10              }
11              else if (chosenFile.substring(chosenFile.lastIndexOf("."),
12              chosenFile.length()).equals(".jpg")){
13                type = "image/jpeg";
14              }
15              else if (chosenFile.substring(chosenFile.lastIndexOf("."),
16              chosenFile.length()).equals(".pdf")){
17                type = "application/pdf";
18             }
19      return type;
20  }
```

Para finalizar con esta clase, mostramos unas capturas de pantalla de la interfaz de usuario de esta actividad en la figura 5.14.



(a) Interfaz de usuario de *GestorFragment.java* (b) Directorio raíz de archivos. (c) Contenido del directorio seleccionado

Figura 5.14: Diferentes vistas del sistema de archivos.

ConvertPDF_Activity.java

En esta actividad, transparente para el usuario, se llevan a cabo las tres tareas principales de la aplicación: conversión del archivo PDF, configuración del servidor web y envío de las imágenes resultantes al dispositivo Chromecast. Por lo tanto, para facilitar la descripción de este apartado, se subdividirá a su vez en diferentes secciones.

- Conversión y renderizado de los documentos PDF.
  Para llevar a cabo esta tarea vamos a recurrir al proyecto *Pdf-renderer*

[52] (con licencia LGPL-2.1), para facilitar el proceso de conversión de los documentos de tipo PDF.

De la clase anterior (`GestorActivity.java`) obtendremos la ruta y el nombre del archivo para localizar el elemento que deseamos convertir. Una vez localizado, pretendemos visualizarlo tanto en la pantalla de nuestro dispositivo como en la pantalla secundaria (mediante el dispositivo Chromecast). Para la primera tarea, inicializaremos un objeto de tipo `WebView` [53] que, básicamente, consiste en un complemento para visualizar páginas web. Dicha página web la crearemos a partir de un objeto de tipo HTML que contendrá las diferentes páginas del documento, codificadas en base 64. Para que la actividad tenga acceso a Internet para cargar la página web, debemos añadir el siguiente permiso:
`<uses-permission android:name=.ªndroid.permission.INTERNET/>`.

Los pasos que vamos a seguir para convertir un documento de tipo PDF son los que enumaramos a continuación:

1. Crear objeto de tipo `PDFFile` a partir de un *buffer* de bytes que conforman el documento.
   ```
   ByteBuffer bb = ByteBuffer.NEW(data);
   PDFFile pdf = new PDFFile(bb);
   ```
2. Extraer la primera página del documento.
   ```
   PDFPage PDFpage = pdf.getPage(1, true);
   ```
3. Obtener un factor de escala a partir del ancho del elemento `WebView`.
   ```
   final float scale = ViewSize/PDFpage.getWidth()*0.95f;
   ```
4. Convertir la página en una imagen de tipo *bitmap* en función de la escala obtenida anteriormente.
   ```
   Bitmap page = PDFpage.getImage((int)(PDFpage.getWidth()
   * scale), (int)(PDFpage.getHeight() * scale), null, true,
   true);
   ```
5. Guardar la imagen obtenida en un *array* o cadena de bytes.
   ```
   ByteArrayOutputStream stream = new ByteArrayOutputStream();
   page.compress(Bitmap.CompressFormat.PNG, 100, stream);
   byte[] byteArray = stream.toByteArray();
   ```
6. Codificar la cadena bytes en base 64 y lo guardamos como cadena de caracteres.
   ```
   String base64 = Base64.encodeToString(byteArray,
   Base64.NO_WRAP);
   ```
7. Crear un objeto HTML y añadir la imagen codificada en base 64.
   ```
   String html = ‘‘<!DOCTYPE html><html><body bgcolor= \‘‘
   #b4b4b4 \‘‘><img src= \‘‘data:image/png;base64 ‘‘+base64+‘‘
   \‘‘ hspace=10 vspace=10><br>‘‘;
   ```

8. Repetir desde el paso número cuatro para cada una de las páginas que forman el documento e ir añadiendo los resultados al objeto HTML.

La vista final del elemento *WebView* una vez realizada la conversión completa del documento PDF será la que se recoge en la figura 5.15.

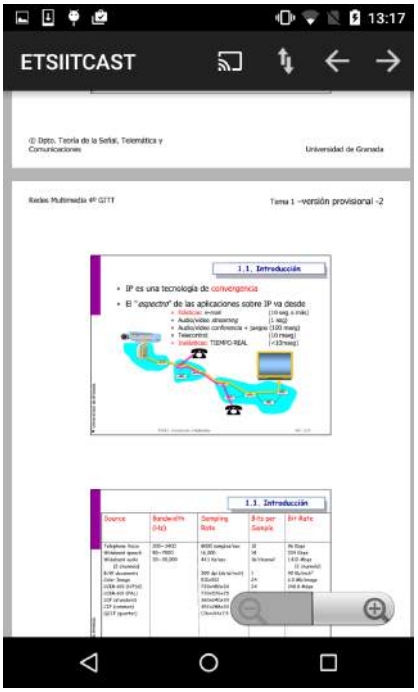


Figura 5.15: Interfaz de usuario una vez convertido el archivo PDF

Además de este proceso de conversión, de forma paralela, debemos ir guardando las imágenes obtenidas a partir de las páginas del documento en una carpeta temporal. El principal motivo de este proceso será recoger las imágenes para que estén disponibles en el servidor web que vamos a implementar. Para ello, tomaremos cada una de las imágenes en formato *bitmap* y las convertiremos a formato JPEG antes de guardarlas. Le asignaremos a cada una un nombre aleatorio y las guardaremos en una carpeta que debemos eliminar al finalizar la actividad, con el fin de no almacenar *basura* no deseada.

```
1 private void SaveImage(Bitmap finalBitmap) {
2         String root = Environment.getExternalStorageDirectory()
3         .toString();
```

```
4            File myDir = new File(root + "/saved_images");
5             myDir.mkdirs();
6            Random generator = new Random();
7            int n = 10000;
8            n = generator.nextInt(n);
9            String fname = "Imagen-"+ n +".jpg";
10           File file = new File (myDir, fname);
11           if (file.exists ()) file.delete ();
12           try {
13                    FileOutputStream out =
14                    new FileOutputStream(file);
15               finalBitmap.compress(Bitmap
16               .CompressFormat.JPEG, 90, out);
17                out.flush();
18               out.close();
19        } catch (Exception e) {
20            e.printStackTrace();
21        }
22    }
23
24 public static boolean deleteDirectory(File path) {
25         if( path.exists() ) {
26             File[] files = path.listFiles();
27                 if (files == null) {
28                return true;
29                    }
30                    for(int i=0; i<files.length; i++) {
31                if(files[i].isDirectory()) {
32                    deleteDirectory(files[i]);
33                }
34                else {
35                    files[i].delete();
36                }
37             }
38             }
39            return( path.delete() );
40 }
```

- Servidor web.
  En apartados anteriores se justificó el uso de un servidor web del lado del cliente o emisor (*sender*) para servir las imágenes. Entre las diferentes opciones disponibles, se ha optado por la herramienta *Nanohttpd* [54], un servidor web simple y fácil de empotrar en una aplicación. Es un proyecto de código abierto y escrito en Java, disponible en su repositorio

de *GitHub* [55] y que añadiremos como una librería a nuestro proyecto, incluyendo el archivo `NanoHTTPD.java` que conforma el *core*. Una vez hecho esto, definiremos una clase que herede de `NanoHTTPD.java` para inicializar (en el puerto 8080) y configurar el servidor web.

Como queremos servir las imágenes que anteriormente hemos guardado en la carpeta *saved_images*, accederemos directamente a ese directorio. Ya que previamente desconocemos el número de páginas que componen el archivo PDF y, por tanto, el número de imágenes que se han guardado, obtendremos la lista de archivos que contiene dicha carpeta. Recorreremos la lista para ir guardando en una variable de tipo `FileInputStream` cada una de las imágenes. Esa variable, junto con el formato de la imagen, conformarán la respuesta del método, con la que conseguimos servir la imagen al servidor. El objetivo es emular un sistema para pasar diapositivas, por lo que el archivo que estamos sirviendo en un instante dado dependerá de la variable `incremento`, que podremos variar con los botones para pasar o retroceder página. Así indicaremos la página actual que queremos servir. Si accedemos desde un navegador web a la dirección IP del teléfono móvil, podremos visualizar la imagen que estemos serviendo en ese momento.

Es importante tener en cuenta que debemos lanzar el servidor web al comienzo de la actividad (método `onCreate()`) y cerrarlo al salir de la misma (método `onDestroy()`) para su correcto funcionamiento.

```
1  public class webserver extends NanoHTTPD {
2            public webserver(){
3                   super(8080);
4           }
5
6             @Override
7           public Response serve(String uri, Method method,
8           Map<String, String> header,Map<String, String>
9           parameters, Map<String, String> files) {
10                      Response res;
11                String mediasend = "image/jpeg";
12                 FileInputStream fis = null;
13                 String root1 = Environment
14                 .getExternalStorageDirectory().toString() +
15                  "/saved_images";
16                 File f = new File(root1);
17                File file[] = f.listFiles();
18
19                          for (int i = 0; i < file.length; i++) {
```

```
20                                  Log.d("Files", "FileName:" +
21                                  file[i].getName());
22                              try {
23                                fis = new FileInputStream(Environment
24                                .getExternalStorageDirectory()
25                                  + "/saved_images" + "/" +
26                                  file[incremento].getName());
27
28                              }catch(FileNotFoundException e){
29                          e.printStackTrace();
30                                      }
31
32              }
33              res = new NanoHTTPD.Response(etsiit.etsiitcast_def
34              .NanoHTTPD.Response.Status.OK, mediasend, fis);
35              return res;
36          }
37      }
```

- Conexión y envío de las imágenes al dispositivo Chromecast.
  Para establecer la conexión con el Chromecast, se hará de manera análoga a la que se explicó en el apartado correspondiente a la clase `SeleccionarActivity.java`, es decir, mediante la API *Media Router*. Esta API habilita las conexiones entre los dispositivos Android y el Chromecast para poder reproducir contenido multimedia en este último. Debemos añadir la librería de los servicios de *Google Play* para el correcto funcionamiento de la SDK de *Google Cast*: `com.google.android.gms:play-services:7.5.0`.

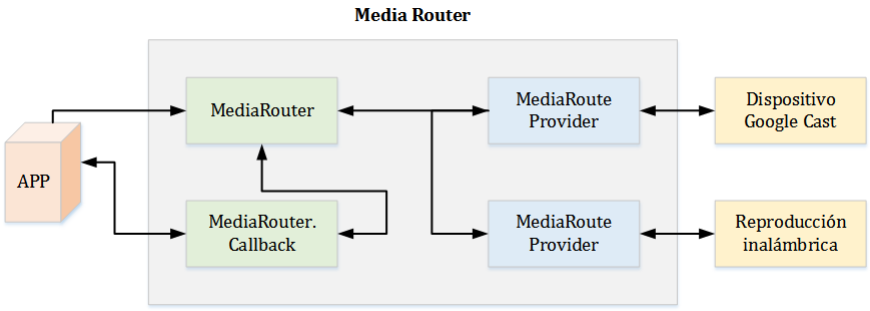


Figura 5.16: Diagrama del uso de clases de la API *Media Router*.

En este caso, para comodidad del usuario, se habilitará el botón *Cast* en la barra superior de acciones (*action bar*). En esa misma barra añadiremos tres botones más, dos para avanzar y retroceder las páginas del

documento y un tercero para comenzar la proyección en la pantalla secundaria (monitor, proyector...). Estos botones se definirán en el archivo `menu_pdf.xml`.

```
@Override
public boolean onCreateOptionsMenu( Menu menu ) {
        super.onCreateOptionsMenu(menu);
          getMenuInflater().inflate(R.menu.menu_pdf, menu);

          MenuItem mediaRouteMenuItem = menu.findItem
           (R.id.media_route_menu_item);
    MediaRouteActionProvider mediaRouteActionProvider =
             (MediaRouteActionProvider) MenuItemCompat
             .getActionProvider(mediaRouteMenuItem);
    mediaRouteActionProvider.setRouteSelector
             (mMediaRouteSelector);
           return true;
}
```

Una vez hayamos instanciado la barra de acciones, debemos asignar una tarea para cada botón en caso de que sean seleccionados.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

        switch (item.getItemId()) {
              case R.id.start_cast:
                 if (!mPDFIsLoaded)
                      startPDF();
                  else
                      Log.d(TAG, "error");
                    return true;
              case R.id.pag_sig:
                      // incremento + 1
                   mediaserver.getIncremento();
                  ipdevice = ipdevice + "/"
                  + System.currentTimeMillis();
                   startPDF();
                   return true;
              case R.id.pag_atr:
                      // incremento - 1
```

```
20                      mediaserver.decremento();
21                      ipdevice = ipdevice + "/"
22                      + System.currentTimeMillis();
23                      startPDF();
24                      return true;
25                  default:
26                       return false;
27              }
28  }
```

La dirección IP (`ipdevice`) se obtiene de la misma manera que se explicó en la clase `PerfilFragment.java` con la diferencia de que añadiremos el puerto en el que se ha lanzado el servidor web (8080 en nuestro caso). Además, añadimos la hora actual en milisegundos al final de la dirección IP con el método `System.currentTimeMillis()`. Si no incluyéramos esta marca temporal, el receptor siempre estaría almacenando en la caché la misma imagen ya que la URL que estamos pasando es siempre la misma para todas las imágenes. Por ello, hay que forzar al receptor a recargar la imagen añadiendo una marca temporal (o *timestamp*) o cualquier otro marcador a la URL, para que el navegador reconozca la URL como un nuevo recurso.

Con el método `startPDF()` iniciaremos el envío de contenido al Chromecast. Para ello haremos uso de la clase `MediaInfo` [56] de la SDK de *Google Cast*, que agrega información sobre el elemento multimedia. Crearemos una instancia de esta clase mediante el constructor, que será usado por la clase `RemoteMediaPlayer` para cargar el contenido multimedia en la aplicación receptora.

```
1   private void startPDF() {
2           MediaMetadata mediaMetadata = new MediaMetadata
3           (MediaMetadata.MEDIA_TYPE_PHOTO);
4             MediaInfo mediaInfo = new MediaInfo.Builder(ipdevice)
5                       .setContentType( "image/jpeg" )
6                 .setStreamType(MediaInfo.STREAM_TYPE_BUFFERED)
7                  .setMetadata( mediaMetadata )
8                   .build();
9              try {
10               mRemoteMediaPlayer.load(apiClient, mediaInfo,
11               true)
12                      .setResultCallback( new ResultCallback
13                      <RemoteMediaPlayer.MediaChannelResult>() {
14                         @Override
```

```
15                          public void onResult( RemoteMediaPlayer
16                          .MediaChannelResult mediaChannelResult ) {
17                               if( mediaChannelResult.getStatus()
18                               .isSuccess() ) {
19                                  mPDFIsLoaded = true;
20                               }
21                           }
22                      } );
23             } catch( Exception e ) {
24              }
25 }
```

Para lanzar la aplicación receptora, primero crearemos una instancia de la clase `Cast.CastOptions` para configurar los parámetros necesarios. A continuación, crearemos un objeto de tipo `GoogleApiClient`, la interfaz necesaria para la integración de los servicios de *Google Play* y la conexión del cliente a los mismos.

```
1  private void launchReceiver() {
2          Cast.CastOptions.Builder apiOptionsBuilder =
3          Cast.CastOptions
4                .builder(selectedDevice, mCastClientListener);
5           ConnectionCallbacks mConnectionCallbacks =
6           new ConnectionCallbacks();
7            ConnectionFailedListener mConnectionFailedListener =
8            new ConnectionFailedListener();
9             apiClient = new GoogleApiClient.Builder( this )
10                .addApi( Cast.API, apiOptionsBuilder.build() )
11               .addConnectionCallbacks( mConnectionCallbacks )
12                .addOnConnectionFailedListener
13                (mConnectionFailedListener)
14               .build();
15            apiClient.connect();
16 }
```

CastImg_Activity.java

Esta actividad será la encargada de enviar archivos de tipo imagen al dispositivo Chromecast. Se reutilizan los métodos y bloques correspondientes al servidor web y la conexión y envío de imágenes del apartado anterior. Se procederá de igual forma pero de manera simplificada, ya que solamente se trata de un archivo de imagen. No es necesario almacenar la imagen en ninguna carpeta, porque se sirve dicho archivo desde la ruta donde esté almacenado.

Además, deshabilitaremos los botones para pasar o retroceder páginas, ya que es una sola imagen. Tampoco será necesario un objeto de tipo `WebView`, con un elemento `ImageView` bastará para visualizar la imagen seleccionada en el dispositivo móvil tal y como podemos observar en la figura 5.17.

Figura 5.17: Interfaz de usuario de la clase *CastImg_Activity.java*.

Las dimensiones máximas de la imagen deberán ser 4096x4096, en caso contrario, se producirá un error de conversión, ya que sea del tipo que sea la imagen (dentro de las extensiones permitidas) se realizará una conversión a formato *bitmap* para una correcta visualización.

Capítulo 6

Pruebas y validación

La metodología que se ha seguido para completar la primera versión totalmente funcional de esta aplicación ha sido un modelo de desarrollo en espiral, en el que se ha alcanzando la solución a partir de distintas iteraciones (ver figura 6.1). Cada una de ellas ha ido aportando funcionalidades al sistema diseñado y se han podido aislar los problemas mediante la realización de test unitarios. Todo ello ha facilitado la validación de la aplicación completa.

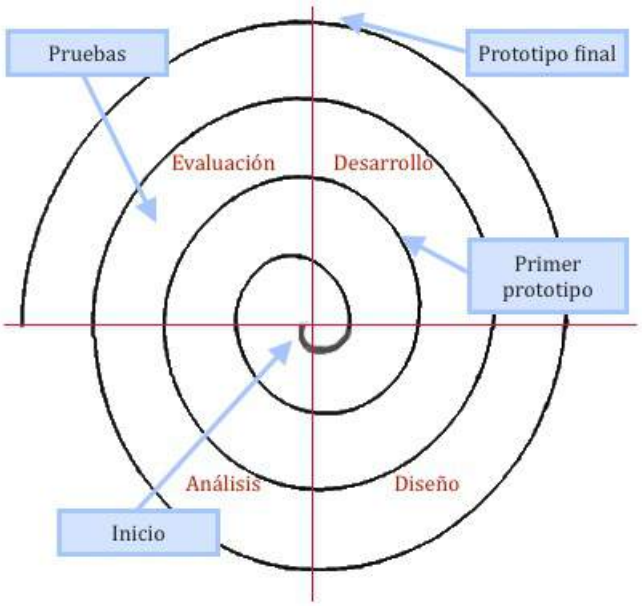


Figura 6.1: Modelo de desarrollo de software en espiral

Esta fase de análisis y control de calidad podría definirse como una de las etapas críticas en el ciclo de vida del desarrollo de un producto *software*. Actualmente, en el área del desarrollo de aplicaciones, se considera impor-

tante la existencia de un equipo o departamento de control de calidad y, que no se trate de los propios desarrolladores los que realicen las pruebas de su código. Esto se debe a que a veces los desarrolladores de *software* no son, no somos, conscientes de los propios errores. Por lo tanto, se considera un trabajo, aunque independiente, también cruzado y colaborativo.

6.1. Pruebas de rendimiento

El entorno de *Android Studio* proporciona una serie de herramientas para depurar los procesos de Android y realizar un análisis de rendimiento (*profiling*). Son diversos los análisis que se pueden realizar a un *software*, dependiendo de las características de la aplicación y los requisitos de ésta. En nuestro caso, vamos a comprobar los resultados de los análisis básicos.

- **Estadísticas de red**. Para obtener los datos vamos a utilizar la herramienta de depuración DDMS (*Dalvik Debug Monitor Server*) [57]. Con esta opción, podremos ver el uso de red para conocer cuando se están realizando peticiones a ésta. También podemos controlar cuándo se efectúan transferencias de datos.

  Por ejemplo, realizamos una primera prueba en la que accedemos a la aplicación, seleccionamos el menú *Perfil* y cerramos la sesión. Repetimos este proceso dos veces y observamos la gráfica de la figura 6.2. En dicha figura se puede identificar el proceso de inicio de sesión con el pico de mayor amplitud (recordemos que se ha repetido el proceso dos veces, por lo que se observan dos picos mayores, correspondientes cada uno de ellos a un inicio de sesión). La tasa de tráfico emitido (con la etiqueta TX en la gráfica), corresponde al envío de credenciales al sistema de autenticación. Por otro lado, y casi de manera simultánea, observamos el pico del tráfico recibido (con la etiqueta RX), referente a la respuesta del mencionado sistema de autenticación. Podemos deducir a partir de esta gráfica que el inicio de sesión no implica

  En una segunda prueba, en la cual se ha procedido a la proyección de un documento PDF, obtenemos la gráfica de la figura 6.3.

  Podemos identificar cada pico en la gráfica con el envío de una imagen a la aplicación receptora, es decir, cada vez que pulsamos el botón para pasar página en la interfaz de usuario de la aplicación emisora. Si sumamos los datos recogidos en ambas pruebas, conseguimos el total de paquetes recibidos y emitidos por la aplicación, como se ve en la figura 6.4.

  Aprovechando los resultados que podemos obtener gracias a esta herramienta, podemos comprobar la diferencia entre el tamaño de los datos almacenados y los datos que nuestra aplicación transmite. En concreto,

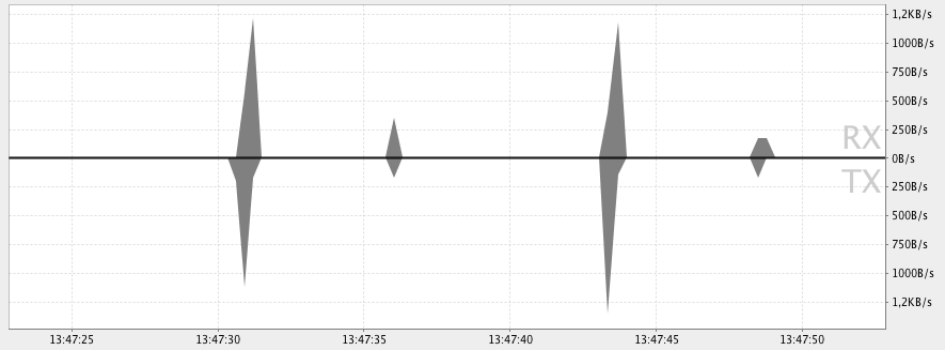


Figura 6.2: Estadísticas de red al iniciar y cerrar sesión.

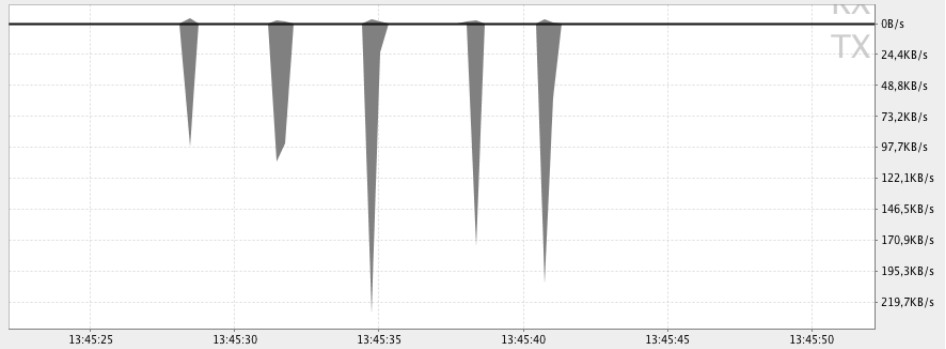


Figura 6.3: Estadísticas de red al proyectar un documento PDF.

Tag	RX bytes	RX packets	TX bytes	TX packets
Total	6.586	84	321.285	234

Figura 6.4: Estadística total de red.

se ha realizado una prueba en la que se ha proyectado un documento PDF compuesto por seis páginas. El tamaño total de las imágenes obtenidas a partir de dichas páginas es de 646 KB. Si calculamos el tamaño total de los paquetes emitidos por la aplicación vemos que asciende a 683,05 KB. Podemos deducir que el excedente que marca la diferencia entre ambas cifras puede deberse a mensajes de control.

Para dar por finalizado el análisis de red, podemos consultar los datos que se han consumido durante una sesión de proyección con la aplicación. Para ello existen diferentes aplicaciones, disponibles en el *Play Store* para Android, de monitorización y gestión del consumo de da-

tos. En concreto, se ha usado la aplicación *Traffic Monitor* [58], de la cual hemos obtenido los datos que se muestran en la figura 6.5 para un consumo de datos haciendo uso de la red Wi-Fi.

Figura 6.5: Consumo de datos de la aplicación.

- **Información del sistema**. Este apartado, también disponible en la herramienta DDMS, muestra información relacionada con el estado de los recursos utilizados en el dispositivo mientras se está ejecutando la aplicación. En la figura 6.6 podemos observar la carga de la CPU. Esta imagen en concreto se obtuvo tras realizar una sesión de proyección de un documento PDF con el *smartphone* LG Nexus 4. Se puede analizar que la aplicación (pocesos con las etiquetas *etsiit.etsiitcast_def(user)* y *etsiit.etsiitcast_def(kernel)*) no generan demasiada carga en el dispositivo.

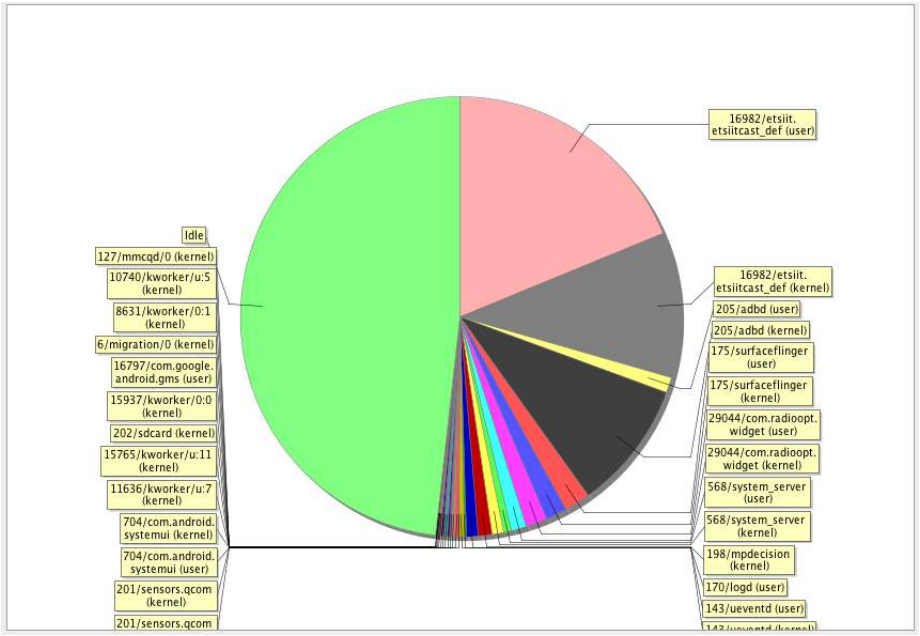


Figura 6.6: Carga de CPU.

Sin embargo, el propio entorno *Android Studio* presenta la capacidad de visualizar la carga de CPU desde su interfaz principal. Vamos a llevar a cabo una sesión de proyección de un documento de tipo PDF recogeremos los datos obtenidos por esta herramienta.

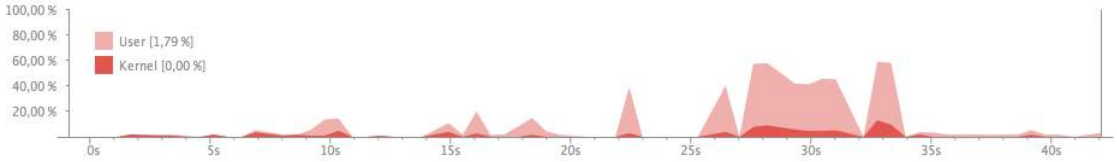


Figura 6.7: Visualización de la carga de la CPU con Android Studio.

En la figura 6.7 se puede identificar el periodo en el que se está realizando el visionado del documento PDF, correspondiente al intervalo que va desde los 25 segundos hasta los 35 segundos, aproximadamente. De manera orientativa, y para realizar comparaciones, el pico que observamos en torno a los 10 segundos coincide con el momento de autenticación de usuario, en el que tienen lugar las peticiones al servidor. Podemos concluir que el nivel de carga de CPU aumenta significativamente cuando el usuario inicia un proceso de proyección de documentos. Mientras el usuario no realice un pase de diapositivas, la carga de CPU podría considerarse mínima.

- ***Heap* y memoria dinámica**. El término *heap* hace referencia a un espacio de memoria en tiempo de ejecución que se usa para almacenar las instancias de clases, objetos y arrays [59]. Esto resulta relevante para controlar el uso de la pila de un proceso.

  En la figura 6.8 se muestran, entre otros datos, el tamaño del *heap*, la asignación de memoria o el porcentaje de memoria usada. También está disponible la opción de representar de manera gráfica el tamaño en función del tipo de objeto que seleccionemos de la lista.

  De manera análoga al apartado anterior, también está disponible esta opción en el entorno de *Android Studio*. Si comparamos las figuras 6.7 y 6.9, podemos identificar nuevamente la proyección del material PDF en el intervalo de los 25 a los 35 segundos, mientras que el inicio de sesión (autenticación), coincide con los 10 segundos. En esta gráfica se puede observar como el tamaño de la pila permanece constante hasta que, de manera similar al caso de carga de CPU, en el momento en el que el usuario comienza la proyección, el tamaño de la pila crece desordenadamente. Si el usuario realiza una pausa entre pases de diapositivas, podemos percibir como el tamaño de la pila se estabiliza, tal y como se visualiza en el último periodo de la figura 6.9.

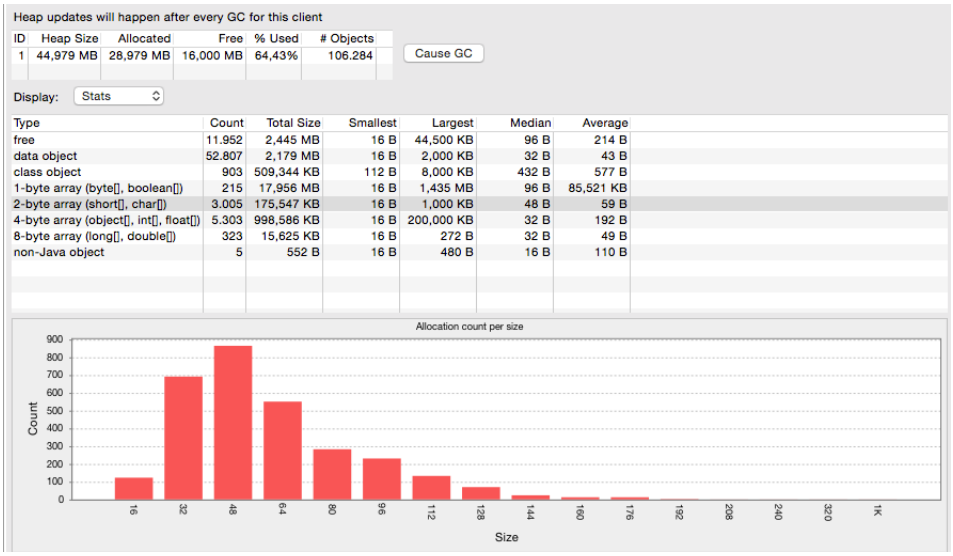


Figura 6.8: Estadísticas del *heap*.

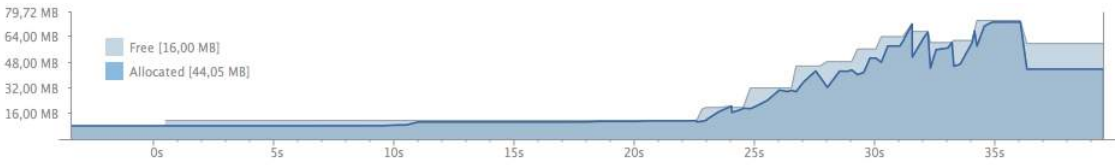


Figura 6.9: Gráfica de la memoria asociada y la memoria disponible.

- **Hebras**. El último apartado que vamos a analizar corresponde a la vista de las hebras, disponible en la herramienta DDMS. Aquí se mostrarán los procesos que se encuentran en ejecución en el instante de la captura. En la imagen 6.10 se muestra una tabla en la que se recogen los datos obtenidos, donde el parámetro *utime* corresponde al tiempo que tarda en ejecutarse el código del usuario, mientras que *stime* hace referencia al tiempo empleado en ejecutar el código del sistema.

6.2. Pruebas de compatibilidad

La finalidad de estas pruebas es comprobar el funcionamiento de la aplicación bajo diferentes configuraciones de *hardware* y diferentes versiones del sistema operativo Android que debe soportar. Antes de pasar a las propias pruebas, vamos a recordar el historial de versiones de Android, listadas en la tabla que se muestra en la figura 6.11.

En la tabla de la figura 6.12 podemos ver las características técnicas de los distintos dispositivos móviles con los que se han realizado las pruebas. Las pruebas se han pasado satisfactoriamente para los tres dispositivos, con la salvedad de que para la versión de Android *Ice Cream Sandwich* (4.0.4) no se visualiza el icono correspondiente a la aplicación en la barra de acciones, junto al icono del menú desplegable. En cuanto a los tiempos de respuesta, son similiares en los distintos casos de uso.

6.3. Pruebas de mantenimiento y usabilidad

Esta solución *software* se ha diseñado siguiendo un esquema modular, de tal manera que pueda ajustarse a cambios en los requerimientos de la misma en un posible futuro. Así, podrán añadirse nuevas funcionalidades fácilmente. Por otro lado, esta herramienta puede ser usado fácilmente para los usuarios para los que se ha diseñado, con el apoyo de la documentación aportada.

6.4. Limitaciones

Durante la fase de pruebas se detectó que algunas imágenes no se visualizaban correctamente en el dispositivo móvil y, por lo tanto, tampoco en el proyector una vez enviadas al Chromecast. En la figura 6.13 se muestra el resultado, siendo la imagen original de la foto malformada la que se muestra en la figura 6.14.

Estos errores son provocados por la funcionalidad de renderizado de la librería que se ha elegido para convertir las páginas del documento PDF a imágenes.

ID	Tid	Status	utime	stime	Nombre
1	15275	Runnable	279	76	main
2	15283	Runnable	0	0	-
3	15281	Runnable	0	0	-
4	15282	Runnable	0	0	-
5	15284	Wait	0	0	Signal Catcher
6	15285	Runnable	9	6	JDWP
7	15286	Wait	11	12	ReferenceQueueDaemon
8	15287	Wait	34	17	FinalizerDaemon
9	15288	Wait	0	0	FinalizerWatchdogDaemon
10	15289	Wait	0	0	HeapTrimmerDaemon
11	15296	Runnable	244	233	GCDaemon
12	15297	Runnable	2	4	Binder_1
13	15298	Runnable	3	3	Binder_2
14	15302	Runnable	318	110	RenderThread
15	15307	Wait	5	0	Binder_3
16	15372	Runnable	2	1	AsyncTask #1
17	15384	Wait	34	12	hwuiTask1
18	15421	Runnable	0	0	AsyncTask #2
19	15432	Runnable	0	0	Chrome_DBThread
20	15433	Runnable	0	0	Chrome_FileThread
21	15437	Runnable	1	6	Chrome_IOThread
22	15439	Runnable	0	0	Thread-4776
23	15436	Runnable	0	0	Chrome_CacheThread
24	15434	Runnable	0	0	Chrome_FileUserBlockingThread
25	15435	Runnable	0	0	Chrome_ProcessLauncherThread
26	15441	Wait	0	0	Thread-4780
27	15442	Wait	0	0	AsyncTask #3
28	15445	Wait	0	0	CleanupReference
29	15446	Runnable	828	78	AsyncTask #4
30	15447	Runnable	0	0	NanoHttpd Main Listener
31	15451	Runnable	7	0	hwuiTask2
32	15444	Runnable	0	0	Thread-4786

Figura 6.10: Hebras o procesos en ejecución.

Versión	Fecha de lanzamiento	Nombre	Nivel de API
1.5	Abril de 2009	Cupcake	3
1.6	Septiembre de 2009	Donut	4
2.0	Octubre de 2009	Éclair	5
2.0.1	Diciembre de 2009	Éclair	6
2.1	Enero de 2010	Éclair	7
2.2-2.2.3	Mayo de 2010	Froyo	8
2.3-2.3.2	Diciembre de 2010	Gingerbread	9
2.3.3-2.3.7	Febrero de 2011	Gingerbread	10
3.0	Febrero de 2011	Honeycomb	11
3.1	Mayo de 2011	Honeycomb	12
3.2-3.2.6	Julio de 2011	Honeycomb	13
4.0-4.0.2	Octubre de 2011	Ice Cream Sandwich	14
4.0.3-4.0.4	Diciembre de 2011	Ice Cream Sandwich	15
4.1-4.1.2	Julio de 2012	Jelly Bean	16
4.2-4.2.2	Noviembre de 2013	Jelly Bean	17
4.3-4.3.1	Agosto 2013	Jelly Bean	18
4.4-4.4.4	Octubre de 2013	KitKat	19
4.4W-4.4W.2	Junio de 2014	KitKat	20
5.0-5.0.2	Noviembre de 2014	Lollipop	21
5.1-5.1.1	Marzo de 2015	Lollipop	22
6.0	Próximamente	Marshmallow	23

Figura 6.11: Historial de versiones del sistema operativo Android.

	Sony Ericsson Xperia	Huawei P8	LG Nexus 4
Versión Android	4.0.4	5.0	5.1.1
Chipset	Qualcomm MSM8255 Snapdragon S2	HiSilicon Kirin 930	Qualcomm APQ8064 Snapdragon S4 Pro
CPU	1 GHz Scorpion	Quad-core 2 GHz Cortex-A53 & quad-core 1.5 GHz Cortex-A53	Quad-core 1.5 GHz Krait
GPU	Adreno 205	Mali-T628 MP4	Adreno 3204
RAM	512 MB	3.0 GB	2.0 GB
Resolución pantalla	480x854 píxeles	1080x1920 píxeles	768x1280 píxeles

Figura 6.12: Características técnicas de los dispositivos con los que se han realizado las pruebas.

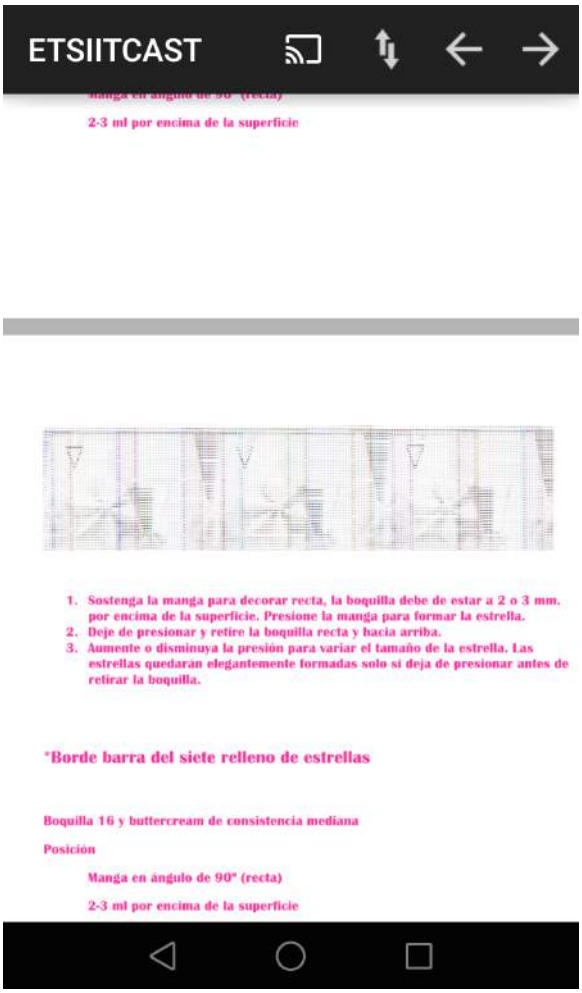


Figura 6.13: Error en el renderizado de imágenes en un documento PDF.

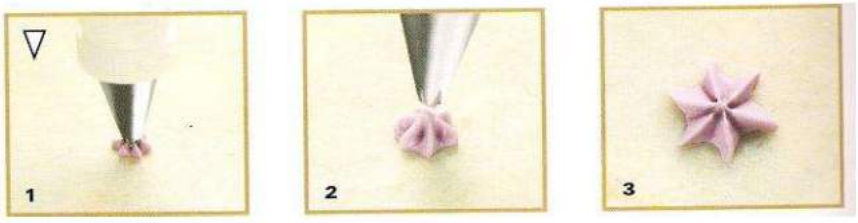


Figura 6.14: Imagen original.

Capítulo 7

Conclusiones y líneas futuras

7.1. Conclusiones

Una vez alcanzado el final del recorrido propuesto en este estudio, podemos concluir que la solución obtenida ha resultado satisfactoria, ya que se han cumplido los requisitos iniciales. En resumen, se ha incorporado la versatilidad del dispositivo Chromecast en el ámbito docente, rompiendo las limitaciones que las conexiones cableadas imponían en el ponente o usuario final de esta aplicación.

Por otro lado, debemos recordar que la incorporación del Chromecast a la infraestructura del aula supondría una reducción en el presupuesto que se destinaría a los ordenadores, cuya función sería equivalente.

En este capítulo, no sólo se quiere hacer referencia a la solución *software* alcanzada y su efectividad, sino que se pretenden recuperar los aspectos referentes a la tecnología y su implicación tanto en la sociedad, en general, como en la educación que se introdujeron al principio de este documento. Durante el desarrollo de este estudio ha surgido la reflexión sobre los límites de los avances de la tecnología, ¿cuáles serán las tendencias educativas del futuro? En palabras de Alberto J. Cañas, "*primero habrá que pensar cuál debe ser la educación del futuro y luego ver qué tecnología necesitamos y no al revés*".

Es imposible predecir las tecnologías que existirán dentro de 20 o 30 años, pero sí se pueden observar tendencias, sin olvidar que la clave es pensar cómo queremos que sea la educación de ese futuro y, llegado ese momento, emplear la tecnología existente [60].

7.2. Vías futuras

Aunque se han alcanzado los objetivos que se propusieron en un principio, en este apartado se presentan una serie de mejoras o funcionalidades adicionales que pueden ser objeto de líneas de trabajo futuras.

- Soporte para otros sistemas operativos como iOS o *Windows Phone*. La principal línea futura de trabajo sería expandir el sorporte a otros sistemas operativos móviles, para así alcanzar a un mayor número de usuarios. Así, eliminaríamos una de las limitaciones que presenta esta solución, tener acceso a un *smartphone* Android.

- Integración con el sistema de autenticación de la Universidad de Granada. De esta manera, el usuario podría acceder a la aplicación con la misma cuenta de usuario que posee en la Universidad, sin necesidad de crear una cuenta adicional. También se plantea la posibilidad de que el ponente no perteneciese a la Universidad de Granada, por lo que se le facilitaría el acceso mediante una opción alternativa para invitados o visitantes. Así se conseguiría además un sistema de autenticación más robusto.

- Sistema de recordatorio de contraseña. Otra mejora relacionada con el sistema de autenticación y la base de datos que alberga la información acerca de los usuarios registrados sería habilitar la opción de que el sistema, mediante el envío de mensajes, facilitara la contraseña correspondiente al usuario en caso de que este no la recordase.

- Compatibilidad con otros formatos. En el caso de que futuras actualizaciones de la SDK de *Google Cast* no fuesen compatibles con nuevos formatos, se podría emular el proceso seguido para los documentos de tipo PDF con archivos cuyo formato sea *Powerpoint*, *Word* o sus equivalentes para otros sistemas ofimáticos. Bastaría con implementar un método de conversión a imágenes para cada una de las páginas o incorporar una librería existente con el mismo propósito.

- Conexión con la nube. Se ha planteado la posibilidad de, en versiones futuras, habilitar la conectividad de la aplicación con herramientas de almacenamiento en la nube, como *Google Drive* o *Dropbox*. Así, el usuario podría acceder al material docente que pretende proyectar sin necesidad de que éste se encuentre almacenado en la tarjeta de memoria de su *smartphone*.

- Limitación en la capacidad de almacenamiento. Tanto en la solución actual (tarjeta de almacenamiento externa) como en la posible alternativa de almacenamiento en la nube, la capacidad disponible para el usuario puede ser limitada. Debemos tener en cuenta que, para realizar la conversión de los documentos PDF a imágenes, es necesario un espacio extra (temporal) para almacenar dichas imágenes. En el caso de que el usuario no dispusiera de suficiente espacio libre en el momento de guardar las imágenes, no se completaría el proceso de proyección. De esta manera, una alternativa eficiente sería diseñar e implementar

un modelo para almacenar dichas imágenes en la memoria caché del dispositivo móvil. Actualmente no se ha podido alcanzar este objetivo por limitaciones de la tecnología y la SDK disponible.

7.3. Valoración personal

Como apartado final a este estudio se realizará una evaluación o valoración personal acerca del mismo. Este apartado no sólo supone el cierre de este proyecto, sino que también significa la conclusión de una etapa así como el comienzo de un nuevo camino.

Han sido diversas las vicisitudes que se han afrontado a lo largo de este Proyecto Fin de Grado, en cuanto a aprendizaje y trabajo autónomo se refiere. Esto ha resultado en un mayor nivel de satisfacción al alcanzar y superar los objetivos propuestos en el anteproyecto. Para ello, han sido muy beneficiosos todos los conocimientos adquiridos durante los años de preparación de este Grado. Ha sido necesario realizar una tarea de abstracción, para así llegar a separar o aislar aquellos conceptos precisos que se han adquirido de manera transversal durante todas las asignaturas cursadas, siendo más recurrentes aquellas materias estudiadas durante el último periodo de especialización, Telemática. Sin olvidar que ha sido necesario aprender nuevos conceptos y lenguajes de programación que no se habían visto durante la carrera, para así completar este proyecto, que puede contemplarse como un reto.

Además, este proyecto ha servido para ver la posibilidad de inmersión en el mercado real, ya que esta solución podría ser incluida fácilmente en un ámbito de explotación concreto. Esto, junto a la idea de facilitar ciertas tareas a los usuarios (ya sea a nivel de aplicación o a nivel docente) y el enriquecimiento personal y formativo que ha implicado, ha dado forma a un sentimiento muy gratificante.

Bibliografía

[1] Andrew T. Kemp, John Preston, C. Steven Page, Rebecca Harper, Benita Dillard, "Technology and Teaching: A Conversation among Faculty Regarding the Pros and Cons of Technology", *The Qualitative Report, 2014, Volume 19, Article 6, 1-23.*

[2] Terceiro, J., "Sociedad Digital", *1996, Madrid, Alianza.*

[3] Alonso Andrade, Andrés Mauricio Rodríguez, Geovanny Chabur Sánchez y Ricardo Andrés Almeida Delgado, "Las TICs como estrategia didáctica en la docencia universitaria", *Universidad San Buenaventura, Santiago de Cali, Colombia, 2011.*

[4] María Domingo y Pere Marquès, .ªulas 2.0 y uso de las TICs en la práctica docente", *Revista Científica de Educomunicación, Comunicar, nº 37, v. XIX, 2011.*

[5] Cristina Villalonga Gómez, Dra. Carmen Marta-Lazo, "Modelo de integración educomunicativa de 'Apps' móviles para la enseñanza y aprendizaje", *Revista de Medios y Educación, nº 46, Enero 2015.*

[6] Dra. Carmen Rocío Yot Domínguez, Dr. Carlos Marcelo García, "¿Despega el m-learning? Análisis de la disposición y hábitos de los usuarios", *Revista de Medios y Educación, nº 46, Enero 2015.*

[7] Apple, Apple TV. `https://www.apple.com/es/appletv/`

[8] Amazon, Amazon Fire TV. `http://www.amazon.com/Fire-TV-streaming-media-player/dp/B00CX5P8FC`

[9] Amazon, Amazon Fire TV Stick. `http://www.amazon.com/dp/B00GDQ0RMG/ref=fs_ftvs`

[10] Roku, Roku 3. *https://www.roku.com/products/roku-3*

[11] Roku, Roku Streaming Stick. *https://www.roku.com/products/streaming-stick*

[12] MatchStick. The Streaming Stick Built on Firefox OS http://www.matchstick.tv

[13] Google Chromecast Review - An awesome $35 HDMI dongle http://www.anandtech.com/show/7186/google-chromecast-review-an- awesome-35-hdmi-dongle/2

[14] Wikipedia. La enciclopedia libre. High-Definition Multimedia Interface https://es.wikipedia.org/wiki/High-Definition_Multimedia_Interface

[15] Chromecast. Cómo enviar contenidos a tu TV. http://www.google.es/chrome/devices/chromecast/learn.html

[16] Apple Store https://itunes.apple.com/es/app/chromecast/id680819774?mt=8

[17] Chromecast now works with your TV's remote control. https://medium.com/@jankoroettgers/chromecast-now-works-with-your-tv- s-remote-control-b8572fe2d0b1

[18] Chromecast. Fondo de pantalla para tu Chromecast. https://www.google.com/chromecast/backdrop/

[19] Chromecast. Tus aplicaciones favoritas ahora en Chromecast. http://www.google.es/chrome/devices/chromecast/apps.html

[20] Faltpanelshd. Review: Google Chromecast. http://www.flatpanelshd.com/review.php?subaction=showfull&id=1376286082#3

[21] Chrome Web Store. Google Cast. https://chrome.google.com/webstore/detail/google-cast/boadgeojelhgndaghljhdicfkmllpafd?hl=es

[22] Ayuda de Chromecast - Modo invitados de Chromecast. https://support.google.com/chromecast/answer/6109292?hl=es&ref_topic=6109288

[23] WebRTC Challenge - Chromecast. http://www.webrtcchallenge.com/built-with-webrtc/google-chromecast/

[24] DIAL - DIscovery And Launch protocol specification. Version 1.7.2 https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGlhbC1tdWx0a XNjcmVlbi5vcmd8ZGlhbHxneDo4MDQ4YjgzYjQ3MTAwYjU

[25] UPnP Forum. UPnP Device Architecture 2.0. http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf

[26] S. Cheshire, M. Krochmal, "Multicast DNS", *RFC 6762, Internet Engineering Task Force*, Febrero 2013.

[27] Nmap Security Scanner. https://nmap.org

[28] CEU, Universidad Cardenal Herrera. Ranking de sistemas operativos más usados para 2015. https://blog.uchceu.es/informatica/ranking-de-sistemas-operativos-mas-usados-para-2015/.

[29] Orange. Ohmyphone! El blog de tecnologías de Orange. http://ohmyphone.orange.es/iphone/sistema-operativo/asi-esta-el-ranking-de-sistemas-operativos-ios-remonta.html

[30] iOSMac. iPhone supera a Android en ventas en EEUU. http://iosmac.es/iphone-supera-a-android-en-ventas-en-eeuu.html

[31] Google Developers - Cast. https://developers.google.com/cast/docs/developers

[32] Google Cast SDK Developer Console. https://cast.google.com/publish/#/overview

[33] Google Developers. Cast: Registration. https://developers.google.com/cast/docs/registration

[34] Google Developers. Google APIs for Android: Setting up Google Play Services. https://developers.google.com/android/guides/setup

[35] Android Developers. Support Library Features: v7 mediarouter library. http://developer.android.com/tools/support-library/features.html#v7-mediarouter

[36] Google Developers. Google APIs for Android: com.google.android.gms.cast. https://developers.google.com/android/reference/com/google/android/gms/cast/package-summary

[37] Google Developers. Cast: Supported Media for Google Cast. https://developers.google.com/cast/docs/media

[38] Boston University. Information Services & Technology. Understanding Authentication, Authorization and Encryption. http://www.bu.edu/tech/about/security-resources/bestpractice/auth/

[39] S. Josefsson, "The Base16, Base32, and Base64 Data Encondings", *RFC 4648, Internet Engineering Task Force*, Octubre 2006.

[40] Manual de PHP. https://secure.php.net/manual/es/

[41] XAMPP. https://www.apachefriends.org/es/index.html

[42] ECMA-404 The JSON Data Interchange Standard. http://json.org

[43] Android Developers - EditText. http://developer.android.com/intl/ja/reference/android/widget/EditText.html

[44] Android Developers - AsyncTask. http://developer.android.com/intl/ja/reference/android/os/AsyncTask.html

[45] Android Developers - Creating a Navigation Drawer. https://developer.android.com/intl/ja/training/implementing-navigation/nav-drawer.html

[46] Android Developers - Fragment. https://developer.android.com/intl/ja/reference/android/app/Fragment.html

[47] Android Developers - Action Bar. https://developer.android.com/intl/ja/guide/topics/ui/actionbar.html

[48] Android Developers - WifiManager. http://developer.android.com/intl/ja/reference/android/net/wifi/WifiManager.html

[49] Android Developers - WifiInfo. http://developer.android.com/intl/ja/reference/android/net/wifi/WifiInfo.html

[50] Android Developers - MediaRouteButton. http://developer.android.com/intl/ja/reference/android/app/MediaRouteButton.html

[51] Android Developers - Environment. http://developer.android.com/intl/ja/reference/android/os/Environment.html

[52] Java.net. The source for Java Technology Collaboration - Pdf-renderer. https://java.net/projects/pdf-renderer/

[53] Android Developers - WebView. http://developer.android.com/intl/ja/reference/android/webkit/WebView.html

[54] Nanohttpd. Tiny, easily embeddable HTTP server in Java. http://nanohttpd.org

[55] GitHub. NanoHttpd Repository. https://github.com/Nanohttpd/nanohttpd

[56] Google APIS for Android. Cast - MediaInfo. https://developers.google.com/android/reference/com/google/android/gms/cast/MediaInfo

[57] Android Developers - Using DDMS. http://developer.android.com/intl/ja/tools/debugging/ddms.html

[58] Google Play. Traffic Monitor. https://play.google.com/store/apps/details?id=com.radioopt.widget

[59] JoseVazquez.net - Definicón de Java Heap y Java Heap Dump. http://www.josevazquez.net/definicion-de-java-heap-y-java-heap-dump/

[60] Fundación Telefónica, "20 Claves Educativas para el 2020. ¿Cómo debería ser la educación del siglo XXI?", *Encuentro Internacional de Educación*.

Apéndice A

ARMADA 1500 (88DE3100)

A.1. Diagrama de bloques del procesador multimedia ARMADA 1500 (88DE3100)

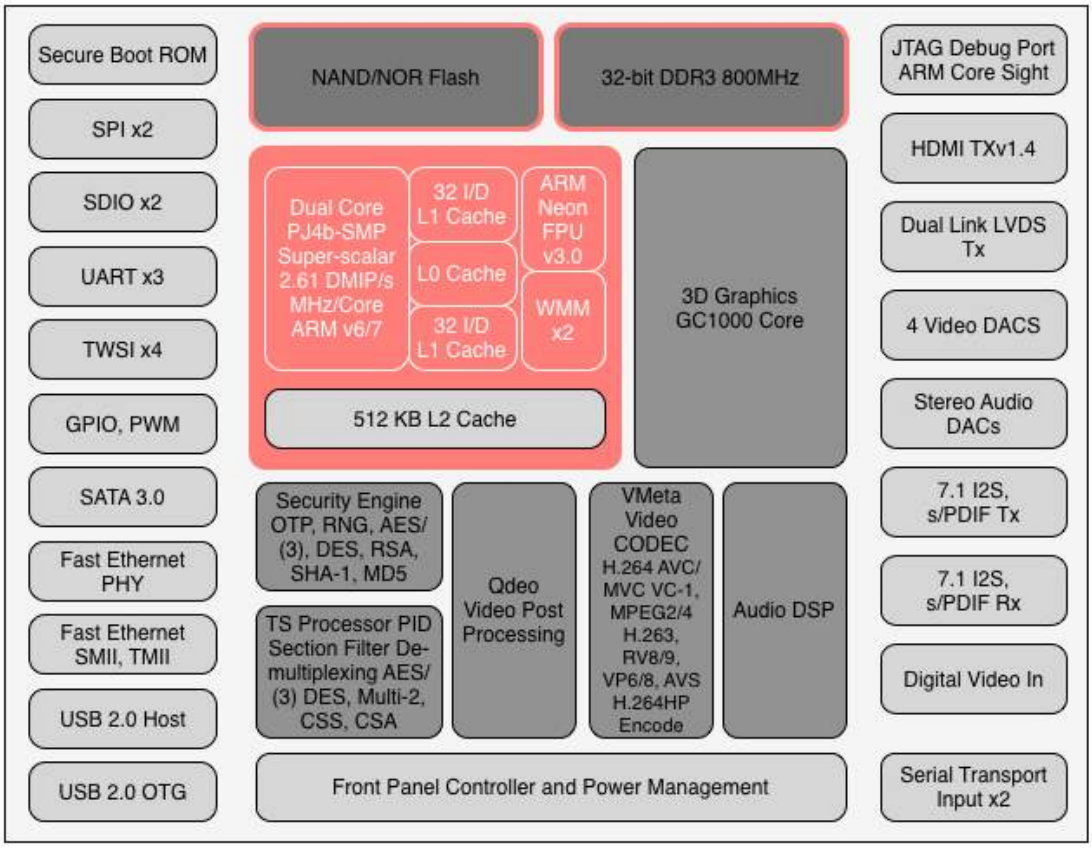


Figura A.1: Diagrama de bloques del SoC ARMADA 1500 (88DE3100)

Apéndice B

Servicio DIAL REST

B.1. Servicio DIAL REST - Solicitud de información sobre una aplicación. Respuesta del servidor.

En la figura B.1 se recogen los posibles parámetros que pueden encontrarse en la respuesta del servidor ante una petición HTTP de tipo GET por parte del cliente DIAL sobre una aplicación en concreto.

Como ejemplo, en la figura B.2 se muestra un paquete en el que el parámetro `state` se encuentra fijado al valor `stopped`.

Elemento o @atributo	Definición
`name`	Contiene el nombre de la aplicación.
`options`	
`@allowStop`	Valores válidos: • true: indica que se soporta la operación `DELETE` (para detener el envío). Es el valor por defecto. • false: Indica que no soporta la operación `DELETE`.
`state`	Valores válidos: • `running`: indica que la aplicación está instalada y está iniciándose o en ejecución. • `stopped`: indica que la aplicación está instalada pero no se está ejecutando. • `instalable=<URL>`: indica que la aplicación no está instalada pero está disponible para su instalación. El valor `<URL>` debe contener la URL HTTP absoluta. Cualquier otro valor es inválido y debe ser ignorado.
`link`	Este elemento es opcional y debería ser incluido cuando la aplicación se esté ejecutando y el servidor DIAL soporte peticiones `DELETE`.
`@rel`	El valor de este atributo debe ser run.
`@href`	Este atributo contiene el nombre del recurso de la aplicación que se está ejecutando, e.g. "run" o "pid-25352".
`additionalData`	Contiene cero o más elementos que la aplicación envía, al ser lanzada, al servidor DIAL.

Figura B.1: Tabla de posibles parámetros y atributos.

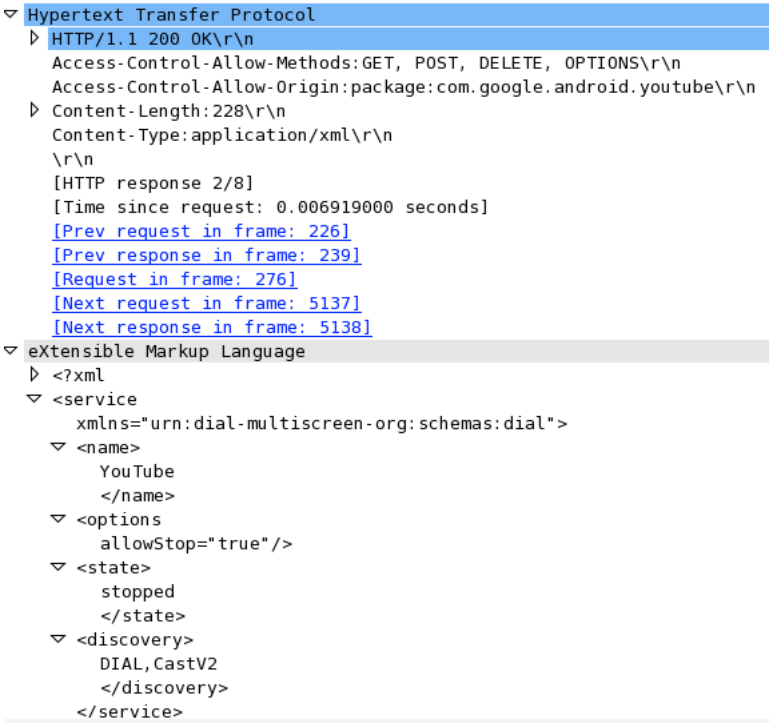


Figura B.2: Respuesta del servidor. Servicio DIAL REST.

Apéndice C

Información detallada del dispositivo

C.1. Petición de información detallada sobre el dispositivo Chromecast.

Figura C.1: Petición HTTP tipo GET de información.

C.2. Respuesta: información detalla del dispositivo Chromecast.

HTTP/1.1 200 OK


```
{"bssid":"f8:d1:11:c1:99:f2",
"build_version":"32904",
"cast_build_revision":"1.14.32904",
"closed_caption":{},
```

```
"connected":true,
"detail":
{"icon_list":[{"depth":32,
"height":55,
"mimetype":"image/png",
"url":"http://192.168.33.102:8008/setup/icon.png",
"width":98}],
"locale":{"display_string":"espaol"},
"manufacturer":"Google Inc.",
"model_name":"Eureka Dongle",
"timezone":{"display_string":"Hora de verano de Europa central (Madrid)",
"offset":120}},
"ethernet_connected":false,
"has_update":false,
"hdmi_control":true,
"hotspot_bssid":"FA:8F:CA:81:8B:07",
"ip_address":"192.168.33.102",
"locale":"es",
"location":{"country_code":"ES","latitude":255.0,"longitude":255.0},
"mac_address":"6C:AD:F8:9F:A7:22",
"name":"Chromecast2762",
"noise_level":-89,
"opencast_pin_code":"3105",
"opt_in":{"crash":true,"location":false,"opencast":true,"stats":true},
"public_key":"MIIBCgKCAQEAxtJ5sy8xw9Iyo+0OT5qc7ofmVF8z
ekblKX1VRNOkBwwutTySaZcWDOEh5DLrkQHp+4xq9POrc1o+k
TuiUtRo8u3ouH/O5EfEQWo+GtG3kAqXfEbmUgW5xF9o2kZnqZo
WuncBwy1lj8I5F1HGGTAORaKqF9XTvcpl6oq1B7VZiufI5yFyxizZB
MhXIgq3nsmL42tCaAjGIQUbYNLv9J/c1egPPAuQxt+4N6YB1NrX
eDonyd6o7nAqBsLr2joWE0kzoW+Z8Gh98ig+mbYnyZAGAA+o8m
qLBP5IHj3ssO9Tihw2nYtC09PrgBjvm68wFo0NPVgeQ8GCDcZZv+
e8cttSRwIDAQAB",
"release_track":"stable-channel",
"setup_state":60,
"setup_stats":{"historically_succeeded":true,
"num_check_connectivity":0,
"num_connect_wifi":0,"num_connected_wifi_not_saved":0,
"num_initial_eureka_info":0,"num_obtain_ip":0},
"signal_level":-41,
"ssdp_udn":"85f04e90-af63-9d94-2595-75d5ae561e42",
"ssid":"test",
"time_format":2,
"timezone":"Europe/Madrid",
```

```
"tos_accepted":true,
"uma_client_id":"B1EC2E52-06CB-748B-35C7-99F3DEBA5054",
"uptime":224.59,
"version":6,
"wpa_configured":true,
"wpa_id":0,
"wpa_state":10}
```

Apéndice D

Desarrollo de una aplicación sender para Android

En el presente anexo se va a describir más detalladamente los bloques más importantes que componen una aplicación emisora o sender para Android haciendo uso de la SDK de Google Cast.

D.1. Añadir el botón Cast

Se recomienda añadir un botón Cast y, para ello, hay diferentes maneras de conseguirlo. A continuación se muestra una de esas tres opciones que consiste en añadir dicho botón a la barra superior de la aplicación o *ActionBar* haciendo uso de la clase `MediaRouteActionProvider`.

Menu

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto" >
    <item
        android:id="@+id/media_route_menu_item"
        android:title="@string/media_route_menu_title"
        app:actionProviderClass="android.support.v7.app.
                        MediaRouteActionProvider"
        app:showAsAction="always"/>
    ...
</menu>
```

La actividad principal debe extender de `ActionBarActivity`

MainActivity

```
public class MainActivity extends ActionBarActivity {
...
}
```

onCreate

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  ...
  mMediaRouter = MediaRouter.getInstance(getApplicationContext());
}
```

Constructor MediaRouteSelector

```
mMediaRouteSelector = new MediaRouteSelector.Builder()
  .addControlCategory(CastMediaControlIntent.categoryForCast
  ("ID_APLICACION"))
  .build();
```

Por último, el objeto `MediaRouteSelector` se asigna a `MediaRouteAction Provider` en el menú *ActionBar*. Así, `MediaRouter` hará uso del selector para filtrar el descubrimiento de dispositivos que se muestran al usuario cuando éste pulse el botón Cast.

onCreateOptionsMenu

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  super.onCreateOptionsMenu(menu);
  getMenuInflater().inflate(R.menu.main, menu);
  MenuItem mediaRouteMenuItem = menu.findItem
          (R.id.media_route_menu_item);
  MediaRouteActionProvider mediaRouteActionProvider =
    (MediaRouteActionProvider) MenuItemCompat.
    getActionProvider(mediaRouteMenuItem);
  mediaRouteActionProvider.setRouteSelector(mMediaRouteSelector);
  return true;
}
```

D.2. Gestión de la selección de dispositivo

Cuando el usuario selecciona un dispositvo de la lista, se informa a la aplicación realizando una llamada a la clase `MediaRouter.Callback`. Dicha llamada debería realizarse en el método `onStart()` y eliminarse en el método `onStop()`.

MediaRouter.Callback

```
private class MyMediaRouterCallback extends MediaRouter.Callback {

  @Override
  public void onRouteSelected(MediaRouter router, RouteInfo info) {
    mSelectedDevice = CastDevice.getFromBundle(info.getExtras());
    String routeId = info.getId();
    ...
  }

  @Override
  public void onRouteUnselected(MediaRouter router, RouteInfo info) {
    teardown();
    mSelectedDevice = null;
  }
  ...
  @Override
  protected void onStart() {
          super.onStart();
         mMediaRouter.addCallback(mMediaRouteSelector,
         mMediaRouterCallback,
     MediaRouter.CALLBACK_FLAG_REQUEST_DISCOVERY);
}
  @Override
  protected void onStop() {
          mMediaRouter.removeCallback(mMediaRouterCallback);
          super.onStop();

}
```

D.3. Lanzamiento del receptor

Una vez que la aplicación conoce el dispositivo que el usuario ha seleccionado, la aplicación sender puede lanzar la aplicación receiver en ese dispositivo. Para ello se invoca a la SDK de Google Cast usando `GoogleApiClient`. A continuación, la aplicación puede extablecer una conexión.

GoogleApiClient

```
Cast.CastOptions.Builder apiOptionsBuilder = Cast.CastOptions
                .builder(mSelectedDevice, mCastClientListener);

mApiClient = new GoogleApiClient.Builder(this)
                        .addApi(Cast.API, apiOptionsBuilder.build())
```

```
                        .addConnectionCallbacks(mConnectionCallbacks)
                        .addOnConnectionFailedListener
                        (mConnectionFailedListener)
                        .build();
...
mApiClient.connect();
```

Para conocer el estado de la conexión, es decir, si dicha conexión ha tenido éxito o por el contrario ha fallado, pueden incluirse las siguientes llamadas:

Estado de la conexion

```
private class ConnectionCallbacks implements
      GoogleApiClient.ConnectionCallbacks {
  @Override
  public void onConnected(Bundle connectionHint) {
    if (mWaitingForReconnect) {
      mWaitingForReconnect = false;
      reconnectChannels();
    } else {
      try {
        Cast.CastApi.launchApplication(mApiClient, "ID_APLICACION", false)
          .setResultCallback(
             new ResultCallback<Cast.ApplicationConnectionResult>() {
            @Override
            public void onResult(Cast.ApplicationConnectionResult result) {
                Status status = result.getStatus();
                if (status.isSuccess()) {
                  ApplicationMetadata applicationMetadata =
                        result.getApplicationMetadata();
                  String sessionId = result.getSessionId();
                  String applicationStatus = result.getApplicationStatus();
                  boolean wasLaunched = result.getWasLaunched();
                  ...
                } else {
                  teardown();
                }
            }
        });

      } catch (Exception e) {
        Log.e(TAG, "Error al lanzar la aplicacion", e);
      }
    }
  }
```

```
34
35    @Override
36    public void onConnectionSuspended(int cause) {
37      mWaitingForReconnect = true;
38    }
39  }
40
41  private class ConnectionFailedListener implements
42        GoogleApiClient.OnConnectionFailedListener {
43    @Override
44    public void onConnectionFailed(ConnectionResult result) {
45      teardown();
46    }
47  }
```

D.4. Canal multimedia

Para usar el canal multimedia es necesario crear una instancia de `RemoteMediaPlayer`.

RemoteMediaPlayer

```
1  mRemoteMediaPlayer = new RemoteMediaPlayer();
2  mRemoteMediaPlayer.setOnStatusUpdatedListener(
3                              new RemoteMediaPlayer
4                              .OnStatusUpdatedListener() {
5    @Override
6    public void onStatusUpdated() {
7      MediaStatus mediaStatus = mRemoteMediaPlayer.getMediaStatus();
8      boolean isPlaying = mediaStatus.getPlayerState() ==
9              MediaStatus.PLAYER_STATE_PLAYING;
10     ...
11   }
12 });
13
14 mRemoteMediaPlayer.setOnMetadataUpdatedListener(
15                             new RemoteMediaPlayer
16                             .OnMetadataUpdatedListener() {
17   @Override
18   public void onMetadataUpdated() {
19     MediaInfo mediaInfo = mRemoteMediaPlayer.getMediaInfo();
20     MediaMetadata metadata = mediaInfo.getMetadata();
21     ...
22   }
```

```
23 });
```

Si las aplicaciones emisora y receptora se han conectado con éxito, se puede proceder a la creación del canal multimedia.

Creacion del canal multimedia

```
1  try {
2   Cast.CastApi.setMessageReceivedCallbacks(mApiClient,
3           mRemoteMediaPlayer.getNamespace(), mRemoteMediaPlayer);
4  } catch (IOException e) {
5    Log.e(TAG, "Excepcion al crear el canal multimedia", e);
6  }
7  mRemoteMediaPlayer
8    .requestStatus(mApiClient)
9    .setResultCallback(
10     new ResultCallback<RemoteMediaPlayer.MediaChannelResult>() {
11       @Override
12       public void onResult(MediaChannelResult result) {
13         if (!result.getStatus().isSuccess()) {
14           Log.e(TAG, "Fallo al solicitar el estado.");
15         }
16       }
17     });
```

Para cargar el contenido multimedia se hará uso de la instancia `MediaInfo`.

Cargar contenido multimedia

```
1  MediaMetadata mediaMetadata = new
2  MediaMetadata(MediaMetadata.MEDIA_TYPE_MOVIE);
3  mediaMetadata.putString(MediaMetadata.KEY_TITLE, "Mi video");
4  MediaInfo mediaInfo = new MediaInfo.Builder(
5      "http://tu.servidor.com/video.mp4")
6      .setContentType("video/mp4")
7      .setStreamType(MediaInfo.STREAM_TYPE_BUFFERED)
8      .setMetadata(mediaMetadata)
9                .build();
10 try {
11   mRemoteMediaPlayer.load(mApiClient, mediaInfo, true)
12       .setResultCallback(new ResultCallback<RemoteMediaPlayer.
13       MediaChannelResult>() {
14     @Override
15     public void onResult(MediaChannelResult result) {
16       if (result.getStatus().isSuccess()) {
17         Log.d(TAG, "Contenido cargado con exito");
```

```
18        }
19      }
20       });
21  } catch (IllegalStateException e) {
22    Log.e(TAG, "Se ha producido un problema durante
23    la carga del contenido multimedia", e);
24  } catch (Exception e) {
25    Log.e(TAG, "Se ha producido un problema
26    al abrir el contenido multimedia", e);
27  }
```

Una vez que la reproducción del contenido multimedia esté en curso, la aplicación emisora puede controlar dicha reproducción.

Control de la reproduccion

```
1  mRemoteMediaPlayer.pause(mApiClient).setResultCallback(
2  new ResultCallback<MediaChannelResult>() {
3    @Override
4    public void onResult(MediaChannelResult result) {
5      Status status = result.getStatus();
6      if (!status.isSuccess()) {
7         Log.w(TAG, "No es posible pausar la reproduccion: "
8                 + status.getStatusCode());
9      }
10   }
11 });
```