

Repositorio multimedia para dispositivos móviles



Máster Universitario en Ingeniería de
Telecomunicación

Trabajo Fin de Máster

Autor:

Juan Carlos Rubio García

Tutor:

Miguel Ángel Lozano Ortega

Julio 2017



Universitat d'Alacant
Universidad de Alicante



Índice de contenidos

Índice de contenidos	I
Índice de figuras.....	III
Índice de tablas	V
Índice de términos.....	VII
Capítulo 1. Introducción	1
1.1. Marco contextual.....	1
1.2. Motivación	2
1.3. Objetivos	2
1.4. Metodología	3
1.5. Organización de la memoria	4
Capítulo 2. Estado del arte	5
2.1. Servicios de transcodificación.....	5
2.2. Comparativa de los servicios de transcodificación	10
2.3. Integración de servicios en la nube	13
2.4. Nicho de mercado.....	14
Capítulo 3. Herramientas	15
Capítulo 4. Desarrollo del proyecto	17
4.1. Arquitectura del sistema.....	17
4.2. Estructuración del código.....	19
4.3. Persistencia de datos.....	21
4.4. Perfiles multimedia	22
4.5. Herramienta de publicación de video.....	25
4.5.1 Control de acceso.....	25
4.5.2 Subida de medios al servidor	31
4.5.3 Conversión de medios.....	34

4.6. Repositorio multimedia.....	37
4.6.1. Back-end: Servicios REST	37
4.6.2. Back-end: Servicio de video streaming	44
4.6.3. Front-end: Cliente Web	45
4.7. Cliente móvil.....	59
4.7.1. Tipos de aplicaciones móviles.....	59
4.7.2. Implementación del cliente móvil	62
Capítulo 5. Experimentación	69
5.1 Despliegue de la aplicación	69
5.2 Cliente web	71
5.3 Cliente móvil.....	74
Capítulo 6. Conclusiones y trabajo futuro	79
6.1. Conclusiones	79
6.2. Trabajo futuro	80
Capítulo 7. Bibliografía y referencias.....	83



Índice de figuras

Figura 1. Teléfonos tradicionales vs teléfonos inteligentes.	1
Figura 2. Coste mensual de transcodificación por servicio.	12
Figura 3. Principales plataformas de servicios en la nube pública en 2016 y 2017. .	13
Figura 4. Arquitectura cliente-servidor del repositorio multimedia.	18
Figura 5. Estructura del proyecto “RepositorioWeb”.	19
Figura 6. Diagrama de flujo de la interfaz web.	20
Figura 7. Archivo de propiedades correspondiente al perfil multimedia 1.	25
Figura 8. Vista web del portal de acceso a la herramienta de publicación.	26
Figura 9. Vista web del formulario de registro de usuarios.	27
Figura 10. Diagrama de flujo del portal de acceso.	28
Figura 11. Posibles respuestas del servlet de registro.	29
Figura 12. Diagrama de flujo del filtro de control de acceso.	30
Figura 13. Vista web del formulario de subida de archivos.	31
Figura 14. Respuesta del servlet al subir un archivo no válido.	32
Figura 15. Diagrama de flujo del proceso de subida de archivos.	33
Figura 16. Proceso de conversión de medios.	34
Figura 17. Respuesta del servlet tras lanzar el proceso de conversión.	35
Figura 18. Ejemplo de flujo de datos generado por la ejecución de ffmpeg.	37
Figura 19. Restlet Client: Herramienta de testeo de los servicios REST.	42
Figura 20. Encabezado de página para la versión sin sesión del repositorio.	46
Figura 21. Encabezado de página para la versión con sesión del repositorio.	46
Figura 22. Repositorio multimedia: Relación cliente-servidor.	47
Figura 23. Vista web del repositorio multimedia: listado de medios.	48
Figura 24. Lista de recursos asociada a un determinado medio.	49
Figura 25. Aspecto del reproductor multimedia.	50
Figura 26. Vista web del listado de perfiles multimedia.	51
Figura 27. Vista web de un perfil multimedia extendido.	51
Figura 28. Principales archivos implicados en la implementación del cliente web. .	52
Figura 29. Diagrama de flujo del algoritmo de monitorización.	54
Figura 30. Monitorización de la conversión (I).	56
Figura 31. Monitorización de la conversión (II).	57

Figura 32. Monitorización de la conversión (III).	58
Figura 33. Monitorización de la conversión (IV).	58
Figura 34. Creación de la aplicación web en un dispositivo Android.	62
Figura 35. Archivos reutilizados para la implementación del cliente móvil.	63
Figura 36. Encabezado de página del cliente móvil.	64
Figura 37. Aplicación híbrida creada en el framework PhoneGap Desktop.	65
Figura 38. Comandos necesarios para instalar el plugin whitelist.....	66
Figura 39. Archivo de configuración del proyecto PhoneGap.	66
Figura 40. Interfaz web del servicio Adobe PhoneGap build.	67
Figura 41. Instalación de la aplicación híbrida en Android.	67
Figura 42. Diagrama de red utilizado para llevar a cabo la experimentación.....	70
Figura 43. Aplicación híbrida en bq Aquaris M5 (Android).	75
Figura 44. Reproducción en el navegador web de bq M5. Portrait y fullscreen.	75
Figura 45. Error en el servidor al reproducir un perfil no soportado.....	76



Índice de tablas

Tabla 1. Planes de contratación de Hybrik.	9
Tabla 2. Precios de los diferentes servicios de transcodificación (I).....	10
Tabla 3. Precios de los diferentes servicios de transcodificación (II).	11
Tabla 4. Comparativa de costes de los diferentes servicios de transcodificación.	12
Tabla 5. Perfiles multimedia (I).	23
Tabla 6. Perfiles predefinidos (II).....	23
Tabla 7. Servicios REST y sus correspondientes métodos y URIs.	40
Tabla 8. Clases java involucradas en la representación de los recursos REST.	43
Tabla 9. URI asociado al servicio de video streaming.	44
Tabla 10. Ventajas y desventajas de las aplicaciones nativas.....	59
Tabla 11. Ventajas y desventajas de las aplicaciones web.	60
Tabla 12. Ventajas y desventajas de las aplicaciones híbridas.	61
Tabla 13. Compatibilidad del repositorio con los principales navegadores web.....	73
Tabla 14. Dispositivos móviles empleados en la experimentación.	74
Tabla 15. Compatibilidad de la aplicación web en cada dispositivo.	78
Tabla 16. Compatibilidad de la aplicación híbrida en cada dispositivo.	78



Índice de términos

A	
AJAX.....	57
API.....	9
AWS.....	8
B	
Back-end.....	21
C	
Ciente móvil.....	3
Cloud computing.....	7
Código QR.....	71
Cordova.....	65
D	
DASH.....	14
DHCP.....	74
DOM.....	56
F	
Front-end.....	21
H	
Herramienta de publicación.....	3
HLS.....	14
HTTP.....	10
J	
JSP.....	19
M	
MAC.....	74
MXF.....	14
O	
On-prem.....	7
OVP.....	10

P	
<i>PDA</i>	1
Perfil multimedia	2
<i>PhoneGap</i>	65
R	
Repositorio multimedia	3
<i>REST</i>	2
<i>RIA</i>	11
S	
SDK	63
Servicio de transcodificación	7
<i>Smartphone</i>	1
T	
<i>Tablet</i>	1
U	
<i>URI</i>	44
V	
<i>Video streaming</i>	10
W	
<i>WebView</i>	65
<i>Wi-Fi</i>	74

Capítulo 1. Introducción

1.1. Marco contextual

Según el informe anual de *Ditrendia* [1], actualmente hay más de 7,9 millones de dispositivos móviles en el mundo, superando así al número de personas en el planeta. La penetración global de teléfonos móviles asciende al 97 %, reduciéndose solo a cuatro las regiones donde ésta es menor al 100 %.

A nivel europeo, 78 de cada 100 habitantes dispone de un teléfono inteligente mientras que en España el porcentaje asciende al 87 %, situando a nuestro país en la primera posición.

Cabe destacar el pronóstico referente al tráfico global de datos. Se estima que éste crecerá, aproximadamente, en 8 veces entre 2015 y 2020. Llama todavía más la atención el auge del vídeo móvil. Se espera que para 2019 el tráfico dedicado a video sobre dispositivos móviles suponga el 72 % de todo el tráfico global.

El término ‘dispositivo móvil’ no engloba solo a los teléfonos móviles tradicionales y a los teléfonos inteligentes (*smartphones*), sino que también incluye a las tabletas (*tablets*), *PDA*s (*Personal Digital Assistant*), reproductores multimedia y videoconsolas portátiles, entre otros.



Figura 1. Teléfonos tradicionales vs teléfonos inteligentes.

La Figura 1 describe las principales diferencias existentes entre los teléfonos móviles tradicionales y los teléfonos inteligentes [2].

Pese a las diferencias, la línea de separación entre los dos tipos de dispositivos no está bien definida, pudiendo llegar a ser los teléfonos tradicionales actuales mejores que los primeros *smartphones*.

Debido a la constante evolución y mejora de los dispositivos, actualmente existe una gran variedad de dispositivos en uso. . No todos los dispositivos tienen, por tanto, la misma capacidad para soportar un determinado formato de vídeo o audio.

1.2. Motivación

Este proyecto pretende desarrollar un sistema de gestión de contenido multimedia, principalmente vídeo, con la finalidad de adaptar cada uno de los medios para dar cobertura a la gran mayoría de dispositivos del mercado.

Con esta herramienta, se pretende que la fuente original de un determinado medio multimedia se despreocupe del formato, codificación y calidad del mismo. Delegando en este servicio la tarea de adaptación a cada dispositivo.

Se denominará perfil multimedia a cada combinación de los parámetros específicos que generen un medio adaptado a cada subconjunto en el que se dividirá el abanico de dispositivos. Así, se podrán cubrir las necesidades de todos ellos, desde los dispositivos más antiguos o limitados hasta los más sofisticados.

1.3. Objetivos

El objetivo principal de este proyecto es crear un sistema de publicación de vídeo, capaz de realizar la adaptación a los diferentes perfiles multimedia predefinidos, para su posterior consumo por dispositivos móviles mediante tecnología *REST (Representational State Transfer)*.



Atendiendo a la funcionalidad que ofrece el sistema, es posible distinguir tres partes bien diferenciadas. A continuación, se enumeran y se describen cada una de ellas.

I. Herramienta para la publicación y conversión de vídeo

Esta primera parte será la encargada de convertir un medio determinado por el usuario a todos los perfiles multimedia predefinidos en la aplicación. La herramienta de publicación será el punto de entrada de los medios al repositorio, gestionando el proceso y registrando la autoría de la publicación mediante un control de acceso.

II. Repositorio multimedia

El repositorio está destinado a almacenar y servir los medios convertidos en la fase anterior. Requiere de los métodos de persistencia y acceso apropiados para que los recursos puedan ser consumidos en cualquier momento. El repositorio debe contar con una interfaz gráfica, basada en web, que permita al usuario listar y reproducir los diferentes medios.

III. Cliente móvil

Por último, el repositorio debe ser accesible por un cliente móvil. Se establece como objetivo desarrollar una aplicación móvil multiplataforma con la capacidad para conectarse al repositorio, navegar en él y reproducir el perfil multimedia que mejor se ajuste a sus necesidades.

1.4. Metodología

Para alcanzar los objetivos propuestos, se va a proceder a desarrollar cada una de las siguientes tareas:

- Estudio de las herramientas disponibles para la conversión de videos en la nube.
- Estudio de los formatos soportados por los distintos tipos de dispositivos para definir así los perfiles multimedia que garanticen la máxima calidad y compatibilidad.
- Elección de un método de persistencia que asegure la disponibilidad de los datos almacenados en el repositorio.

- Diseño e implantación de la herramienta de publicación que permita realizar la subida, conversión y almacenamiento de los medios en el repositorio.
- Creación de los servicios *REST* para dar acceso a los contenidos.
- Diseño de la interfaz web del repositorio, capaz de mostrar y reproducir los diferentes medios, consumiendo los servicios mencionados.
- Integración de la herramienta de publicación y el repositorio multimedia bajo un entorno web que implemente control de acceso mediante un proceso de registro y autenticación de usuarios.
- Estudio de los diferentes tipos de aplicaciones móviles para determinar la opción que ofrezca mayor compatibilidad multiplataforma.
- Creación de la aplicación móvil, implementando así el cliente *REST* que acceda a los contenidos del repositorio.
- Experimentación con el sistema final para su evaluación.

1.5. Organización de la memoria

Este documento se ha estructurado de la siguiente forma:

- Capítulo 1: Introducción, motivación, objetivos y metodología a emplear.
- Capítulo 2: Estado del arte.
- Capítulo 3: Herramientas.
Herramientas y tecnologías empleadas.
- Capítulo 4: Desarrollo del proyecto.
Descripción minuciosa de todo el proceso de diseño y desarrollo.
- Capítulo 5: Experimentación.
Puesta en funcionamiento del servicio en un entorno real para su evaluación con diferentes navegadores y dispositivos móviles.
- Capítulo 6: Conclusiones y trabajo futuro.
Valoración personal y propuesta de futuras líneas de trabajo.
- Capítulo 7: Bibliografía y referencias.

Capítulo 2. Estado del arte

En este capítulo se va a hacer un estudio de las diferentes herramientas presentes en el mercado, que ofrecen unas características similares a las planteadas en este trabajo. En primer lugar, se describen los servicios de transcodificación, así como las principales compañías del sector, para después realizar una comparativa de todas ellas.

2.1. Servicios de transcodificación

Estas herramientas ofrecen a sus clientes la conversión de contenidos multimedia, es decir, cambios en la codificación y formato de los datos, para lograr la compatibilidad entre los diferentes dispositivos, sistemas o aplicaciones.

Se distinguen tres modelos de servicio de transcodificación [3]. El primero es el modelo clásico. Consiste en realizar las tareas de conversión en los equipos de la propia empresa que ofrece el servicio. Este concepto, conocido como “*on-prem*” o local, requiere de la contratación de personal cualificado para el mantenimiento de los equipos, ya que hay que garantizar aspectos clave como son la disponibilidad y la seguridad.

La tendencia actual, en cambio, se orienta al segundo modelo, donde los servicios de transcodificación delegan el procesamiento a terceros [4]. Toda la computación se realiza fuera de las instalaciones de la empresa. Este concepto es conocido como “*cloud computing*” o computación en la nube. Esta alternativa ofrece numerosas ventajas frente a la computación local, principalmente respecto a costes, ya que requiere menos gasto en personal, energía, mantenimiento y seguridad.

Hay un tercer modelo que consiste en la fusión de los dos anteriores, dando lugar a la computación híbrida. De tal modo, una empresa dedicada a la transcodificación, puede efectuar un determinado procesamiento en local o en la nube, dependiendo de sus necesidades.

Respecto a cómo estos servicios cobran a sus clientes, se distinguen tres modelos tradicionales de tarificación:

- Precio por minutos.

El precio está basado en el número total de minutos de los archivos generados. Este dependerá de la duración del archivo original y el número de perfiles a generar. Es común aplicar factores de multiplicación dependiendo de la calidad del video. Generalmente 2x para videos de 720p a 2K (2.048p), y 4x para videos de más de 2K.

- Precio por gigabytes.

El coste del servicio se remite al tamaño de los archivos generados, aunque a veces el total puede incluir también el tamaño del archivo de entrada. En este tipo de tarificación, el coste es independiente del tiempo de procesamiento requerido.

- Precio por el uso exclusivo de una máquina dedicada en la nube.

Generalmente se paga una tarifa mensual fija por una máquina disponible 24 horas al día, 7 días a la semana. No se tarifica ni por minutos ni por gigabytes. Sin embargo, la cantidad de codificaciones que una máquina dedicada pueda realizar no es fija, dependerá del tipo de codificación y el potencial de la máquina alquilada.

Estos tres modelos tienen en común que es el propio servicio de transcodificación el que gestiona el servicio de computación, ya sea porque las máquinas son de su propiedad, como es el caso de *Amazon* y *Azure*, o porque alquilan el servicio a terceros (Como a *Amazon*, *Google* o a *Azure*).

Sin embargo, existe un nuevo modelo, impulsado por *Hybrik*, en el que el servicio de computación deja de estar gestionado por el servicio de transcodificación. Es el cliente el que contrata los dos servicios por separado, pero todo el procesamiento se lleva a cabo en su propia cuenta de *AWS (Amazon Web Services)*. Por consiguiente, el cliente paga a *Hybrik* una cuota mensual basada en el número de instancias simultáneas de procesamiento, y paga a *Amazon* por el consumo real de las máquinas. Este modelo tiene la ventaja de que escala muy bien cuando aumenta la demanda de procesamiento, siendo su precio inferior al resto de competidores. Sin embargo, no es recomendable para baja demanda, ya que el precio mínimo del servicio es de unos 1000 dólares [5].



Estos son algunos de los principales servicios de transcodificación del mercado, dispuestos en orden alfabético:

- *Amazon Elastic Transcoder*
- *Azure Media Services*
- *Bitmovin*
- *Encoding.com*
- *Hybrik*
- *Telestream*
- *Videocloud*
- *Wowza Media System*
- *Zencoder*

A continuación, se describen algunos de ellos, para conocer entre otros, qué ofrecen y como lo hacen.

Zencoder

Propiedad de *Brighcove*, es un servicio de codificación de vídeo y audio en la nube lanzado en 2010. Ofrece una *API (Application Programming Interface)* segura basada en tecnología *REST*, junto a un conjunto de herramientas de codificación de alta calidad para contenidos en directo y bajo demanda. Garantiza una conversión inferior a diez segundos con unas tasas de éxito y disponibilidad superior al 99,9 % [6].

El proceso de conversión de archivos a la carta se realiza en tres fases:

1) Solicitud

La aplicación cliente envía una solicitud para iniciar una tarea de codificación, junto con la dirección del vídeo y los correspondientes ajustes de procesamiento opcionales. A continuación, el servidor responde con un identificador de la tarea de codificación con el que el usuario puede comprobar el estado.

2) Transcodificación

En el servidor, se descarga el vídeo y se convierte a todos los formatos que necesite. Todos los archivos de salida se codifican de forma simultánea.

Posteriormente, los videos generados se suben a una red de distribución de contenidos o a la ubicación previa que haya indicado el usuario.

3) Notificación

Por último, se envía una notificación *HTTP (Hypertext Transfer Protocol)* al usuario para indicar que la tarea de codificación ha terminado y que los archivos multimedia han sido subidos.

Encoding.com

Encoding es otro servicio de transcodificación que ofrece a sus clientes la posibilidad de crear soluciones basadas en nubes públicas, privadas o híbridas. El servicio de almacenamiento lo delega a terceros, entre los que destacan *Amazon Web Services*, *Akamai*, *Microsoft Azure*, *Rackspace*, *Openstack* o *YouTube*.

Entre sus paquetes de servicios, destaca *Vid.ly*. Una *OVP (Online Video Platform)*, de código abierto, que simplifica el proceso de entrega de video. Detecta automáticamente el tipo de dispositivo cliente y ofrece una transcodificación a medida, seleccionando uno de sus 24 perfiles predefinidos.

Además, en su página web hay disponible una amplia gama de recursos, entre ellos numerosa documentación e informes, que ayudan a difundir el estado y uso de la tecnología [7].

Wowza Media Systems

Wowza es un servicio orientado principalmente a video *streaming*. Nacido en 2005 de la mano de *David Stubenvoll* y *Charlie Good*, ofrece soluciones integrales tanto en la nube como en local. Sus principales componentes son [8]:

- *Wowza Gocoder*
Agrupa la aplicación móvil multiplataforma y el *SDK* para codificación de video *streaming* para la mayoría de dispositivos *iOS* y *Android*.
- *Wowza Streaming Engine*
Servidor de medios basado en *Java*, dedicado a transmitir video en vivo o bajo demanda, audio, así como aplicaciones *RIA (Rich Internet Application)*.



El servidor puede transmitir a múltiples tipos de dispositivos cliente de forma simultánea.

- *Wowza Streaming Cloud*
Servicio de *streaming* en la nube totalmente administrado para transmitir en vivo de extremo a extremo o como parte de una plataforma de *streaming* personalizada.
- *Wowza Player*
Reproductor *HTML5* embebido, capaz de transmitir video de alta calidad bajo un optimizado flujo de datos adaptativo dependiente de las condiciones de la red.

Videocloud

Es una plataforma orientada al alojamiento de video destinado a fines comerciales. Creado en 2007, este repositorio ofrece las herramientas para almacenar, transmitir, y compartir videos en un entorno de publicidad, para después administrar, comercializar y monetizar sus contenidos [9]. Entre sus servicios cabe mencionar *VIDEOENCODE*, dedicado a la transcodificación de los archivos multimedia.

Hybrik

Esta empresa fundada en 2015 ofrece un modelo innovador de transcodificación basado en *AWS* para clientes al por mayor. Ofrece tres planes de contratación con un gasto fijo mensual acorde al número de máquinas de procesamiento dedicadas. En la Tabla 1 se muestra el número de máquinas y el coste asociado a cada plan de contratación.

	N.º de máquinas	Precio mensual
Plan 1: <i>Big</i>	<i>10</i>	<i>\$1,000</i>
Plan 2: <i>Bigger</i>	<i>100</i>	<i>\$5,000</i>
Plan 3: <i>Biggest</i>	<i>1000</i>	<i>\$10,000</i>

Tabla 1. Planes de contratación de Hybrik.

2.2. Comparativa de los servicios de transcodificación

Es interesante conocer cuál son los aspectos en que difieren los principales servicios de transcodificación mencionados en el apartado anterior, entre ellos el tipo de tarificación, los diferentes planes que ofrecen y su correspondiente precio.

	<i>Tarificación</i>	<i>Plan</i>	<i>Precio</i>		
Amazon	Por minutos	<i>Video SD</i>	\$0.0015/min		Precio dependiente de la ubicación
		<i>Video HD</i>	\$0.030/min		
Azure	Por minutos	<i>Standard</i>	\$0.015/min		Video bajo demanda
		<i>Premium</i>	\$0.035/min		
Bitmovin	Por GB	<i>Free</i>	\$0/mes	2.5 GB	No extra
		<i>Professional</i>	\$149/mes	100 GB	\$1.75/GB extra
		<i>Small Business</i>	\$645/mes	500 GB	\$1.65/min extra
		<i>Medium Business</i>	\$1290/mes	1000 GB	\$1.55/min extra
		<i>Large Business</i>	\$2580/mes	2000 GB	\$1.45/min extra
		<i>Enterprise</i>	A medida	-	-
Encoding	Por GB (Nubes públicas)	<i>Pro</i>	\$199/mes	50 GB	\$2.0/GB
		<i>Studio</i>	\$399/mes	100 GB	\$1.8/GB
		<i>Enterprise</i>	-	-	\$0.5/GB
	Cuota fija (Nubes privadas)	-	\$2000/mes	-	Potencial no especificado
Hybrik	Cuota fija especial	<i>Big</i>	\$1,000/mes	10 máquinas	+ AWS: Tiempo de cómputo
		<i>Bigger</i>	\$5,000/mes	100 ''	
		<i>Biggest</i>	\$10,000/mes	1000 ''	

Tabla 2. Precios de los diferentes servicios de transcodificación (I).

	<i>Tarificación</i>	<i>Plan</i>	<i>Precio</i>		
Telestream Cloud	Por minutos	<i>Flip</i>	\$99/mes	5,000 minutos	\$0.020/min extra
		<i>BigFlip</i>	\$999/mes	55,000 minutos	\$0.018/min extra
		<i>SuperFlip</i>	\$1,899/mes	120,000 minutos	\$0.016/min extra
		<i>GigaFlip</i>	\$3,999/mes	290,000 minutos	\$0.014/min extra
Videocloud	Cuota fija	<i>Business</i>	400€/mes	-	-
		<i>Enterprise</i>	800€/mes	-	-
Zencoder	Por minutos	<i>Play-As-You-Go</i>	\$0/mes	\$0.05/min	-
		<i>Launch</i>	\$40/mes	1,000 minutos	\$0.04/min extra
		<i>Traction</i>	\$300/mes	10,000 minutos	\$0.03/min extra
		<i>Growth</i>	\$2000/mes	100,000 minutos	\$0.02/min extra

Tabla 3. Precios de los diferentes servicios de transcodificación (II).

En las tablas Tabla 2 y Tabla 3 quedan definidos los diferentes planes que ofrece cada servicio y su precio asociado. Debido al diferente tipo de tarificación, es difícil comparar el coste absoluto del servicio entre sí. Sin embargo, se pueden comparar los servicios entre sí, tal como se expone en el informe realizado por *Jan Ozer* [5]. Éste consiste en usar como referencia la codificación de un video en formato *MXF* de 60 minutos de duración y una tasa binaria de 50 Mbps, a diferentes perfiles de salida en formato *HLS* y *DASH*.

En la Tabla 4 se muestran los resultados obtenidos en el estudio de *Jan Ozer*, en la cual se presentan los costes asociados a cada servicio dependiendo del tiempo de procesamiento mensual. Se puede interpretar, tal como muestra la Figura 2, que, a grandes rasgos, todos presentan un comportamiento lineal. En cambio, la diferencia de precios es considerable. Aquellos como *Hybrik*, con un gasto base fijo, resultan poco rentables para poco tiempo de uso, pero son más adecuados para grandes cantidades de procesamiento.

	Horas de procesamiento mensual							
	1	10	50	100	200	300	400	1000
Hybrik	\$1,001	\$1,011	\$1,056	\$1,112	\$1,225	\$1,337	\$1,450	\$2,957
Amazon (Public Cloud)	\$29	\$286	\$1,428	\$2,856	\$5,712	\$8,568	\$11,424	\$28,560
Azure	\$19	\$189	\$947	\$1,893	\$3,787	\$5,680	\$7,573	\$18,482
Encoding.com	\$199	\$957	\$3,909	\$7,599	\$14,979	\$22,360	\$29,740	\$74,021
Zencoder	\$67	\$504	\$2,520	\$3,360	\$6,720	\$10,080	\$13,440	\$33,600
Bitmovin	\$29	\$307	\$1,390	\$2,689	\$5,198	\$6,975	\$9,620	\$20,990
Telestream Cloud	\$49	\$410	\$1,783	\$3,355	\$6,147	\$9,171	\$11,088	\$27,720

Tabla 4. Comparativa de costes de los diferentes servicios de transcodificación.

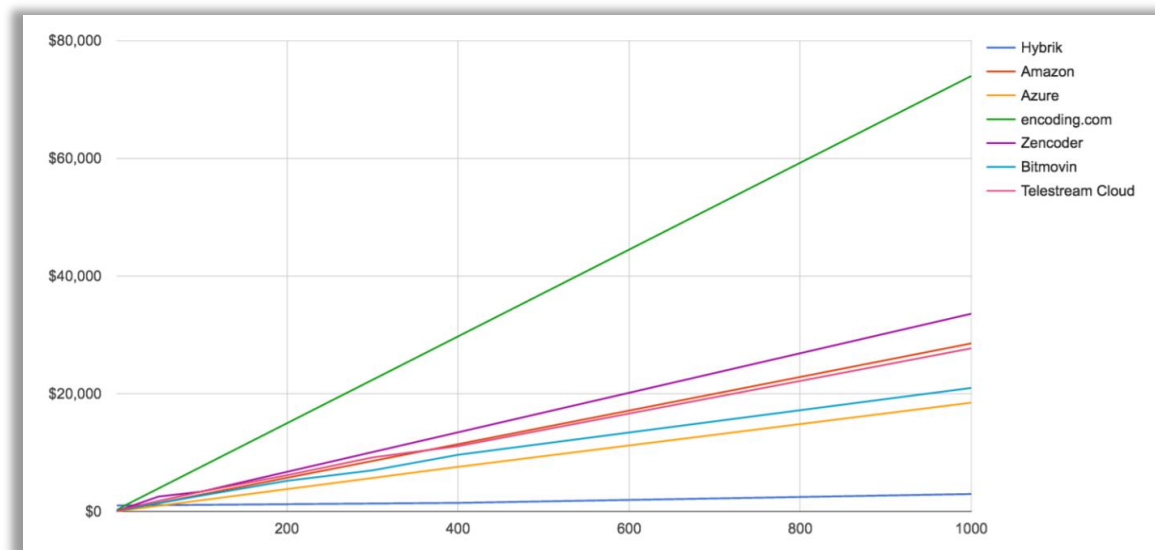


Figura 2. Coste mensual de transcodificación por servicio.

2.3. Integración de servicios en la nube

En este apartado se hace una mención especial a *AWS* por ser la plataforma de servicios integrados de referencia con más del 50 % de cuota de mercado sobre la nube pública, según la encuesta anual realizada por *RightScale* [10]. Ofrece un amplio conjunto de servicios en la nube, entre los que destacan: computación, almacenamiento, redes, bases de datos, o herramientas de desarrollo. Entre sus competidores destacan *Microsoft Azure* y *Google Cloud Platform*, tal como refleja la Figura 3.

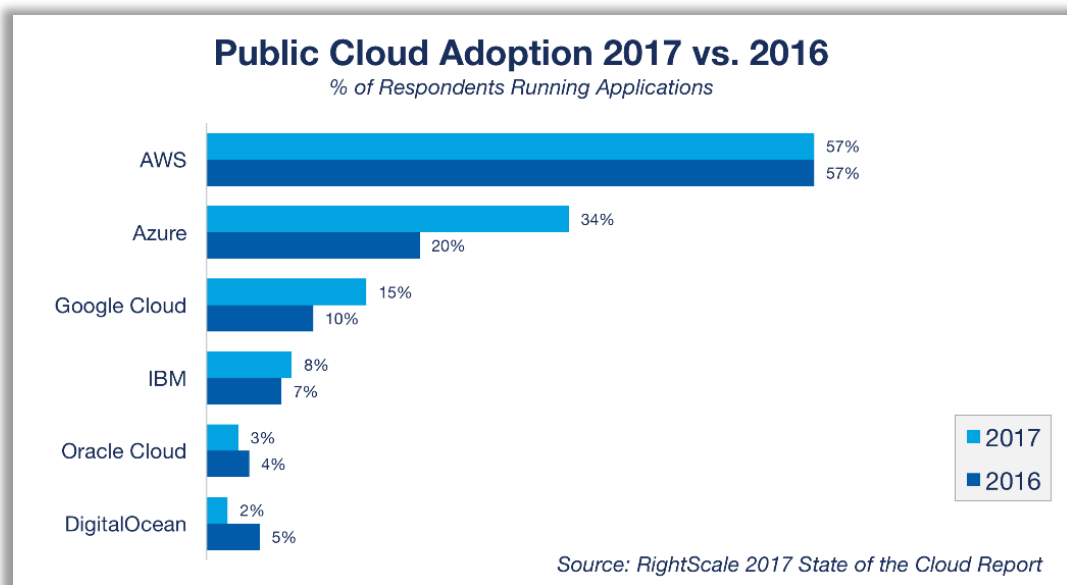


Figura 3. Principales plataformas de servicios en la nube pública en 2016 y 2017.

Estas arquitecturas pueden albergar en la nube todos los servicios necesarios para implementar las principales piezas que componen un repositorio multimedia. Esto incluye el hosting del sitio web, el procesamiento para realizar la conversión, el almacenamiento de los contenidos, y la *API* con la que interacciona la aplicación del cliente final.

2.4. Nicho de mercado

Aunque la tendencia actual es ofrecer servicios en la nube de pago, este trabajo se centra en la elaboración de una herramienta de transcodificación local, gratuita y de código abierto. Un diseño modular, separando la computación, el almacenamiento y los servicios, puede favorecer la migración de la aplicación parcial o completa a la nube en un futuro, sin necesidad de realizar grandes cambios. De este modo, el usuario final puede disfrutar de un sistema de transcodificación personalizable, capaz de proporcionar contenidos compatibles con toda la gama de dispositivos.

En cuanto al uso de la aplicación, puede enfocarse a tres modelos de negocio diferentes, pero compatibles entre sí.

El primero consiste en utilizar la aplicación como repositorio personal o público, con el fin de almacenar medios, para posteriormente poder reproducirlos por cualquier dispositivo, garantizando la máxima compatibilidad.

El segundo modelo, consiste en hacer uso exclusivo del servicio de transcodificación, prescindiendo del almacenamiento. De este modo, un usuario podría utilizar la herramienta para convertir un determinado archivo multimedia de entrada, a uno o varios de salida, utilizando el perfil o perfiles deseados.

Por último, cabría la posibilidad de utilizar la aplicación como un servicio de contenidos a terceros, haciendo transparente al usuario final la ubicación y formato de los contenidos. Por ejemplo, una página web con cierto contenido multimedia podría delegar en la aplicación mediante una *API*, qué perfil de contenidos ofrecer a sus visitantes dependiendo de las características del dispositivo con el que accedan al portal. En este modelo, el propietario de la página web solo tendría que aportar la versión original de cada contenido, despreocupándose de la compatibilidad, ya que está garantizada por la aplicación.



Capítulo 3. Herramientas

En este capítulo se introducen las principales herramientas y tecnologías empleadas a lo largo de este trabajo.

Tecnologías del lado del servidor (*back-end*)

- ***Java***

Es el lenguaje utilizado para el desarrollo del servidor por ser lo suficientemente potente y ofrecer una solución multiplataforma bajo un código orientado a objetos bien estructurado [11].

- ***MySQL***

Gestor de base de datos relacional empleado para almacenar buena parte de la información relevante del repositorio. Se ajusta a las necesidades de persistencia de datos, y se integra correctamente con *Java*. Además, cuenta con la ventaja de ser gratuito [12].

- ***FFmpeg***

Es la herramienta necesaria para la transcodificación de archivos. Admite numerosos formatos y parámetros de ajuste, lo que la convierte en *framework* muy potente. Es multiplataforma, gratuita, y de código abierto [13].

- ***JAX-RS***

Es la *API* del lenguaje de programación Java que proporciona soporte en la creación de servicios web de acuerdo con el estilo arquitectónico *REST* [14].

- ***Apache Tomcat***

Contenedor web con soporte de *JSPs* (*Java Servlet Page*) y *servlets*. Permite desplegar la aplicación *Java* en el servidor para ser consumida por los dispositivos clientes [15].

Tecnologías del lado del cliente (*front-end*)

- ***HTML5, CSS3 y JavaScript***

Son los lenguajes empleados para el desarrollo web: *HTML5* para la estructura, *CSS3* para el estilo y *JavaScript* para aportar funcionalidad dinámica mediante código ejecutado en el cliente.

- ***jQuery / frameworks***

Esta librería de código abierto añade unas capacidades extra a la funcionalidad básica de *JavaScript*. Además, va a permitir el uso de diferentes *frameworks* como: *Bootstrap*, para mejorar el diseño; *jQuery Form Validator*, para la validación de formularios; o *Html5 Lightbox Free*, para la reproducción embebida de medios.

Principales herramientas

- ***Xampp***

Paquete de instalación que contiene *PhpMyadmin* y *Apache Tomcat*. Dispone además de una aplicación de escritorio para gestionar estos servicios [16].

- ***PhpMyadmin***

Permite administrar la base de datos *MySQL* a través de una interfaz web [17].

- ***Restles Client***

Es una extensión de *Google Chrome* que permite testear los servicios *REST*, muy útil durante el proceso de depuración y desarrollo [18].

- ***PhoneGap Desktop / PhoneGap Build***

PhoneGap Desktop es la aplicación de escritorio del *framework PhoneGap* que facilita la creación y simulación de aplicaciones híbridas [19].

PhoneGap Build, en cambio, es la herramienta de gestión de aplicaciones en la nube que permite compilar las aplicaciones móviles creadas con el *framework* [20].

Capítulo 4. Desarrollo del proyecto

Este capítulo recoge el grueso del trabajo, puesto que describe en detalle el diseño e implementación del repositorio multimedia. Para una mejor comprensión del lector, se van a introducir los diferentes apartados en los que se desglosa. Éstos son:

- 1) Arquitectura del sistema
- 2) Estructuración del código
- 3) Persistencia de datos
- 4) Perfiles multimedia
- 5) Herramienta de publicación y conversión de vídeo
- 6) Repositorio multimedia
- 7) Cliente móvil

Los cuatro primeros apartados detallan aspectos clave para entender el contexto en el que se desarrolla la aplicación. Establecen el punto de partida para implementar los tres siguientes, correspondientes a las tres grandes piezas que componen el repositorio; introducidas en los objetivos del proyecto.

En cada una de estas piezas se puede diferenciar la parte dedicada a servidor (*back-end*), de la parte dedicada a cliente (*front-end*). Con la combinación de ambas se consigue proporcionar al usuario una interfaz capaz de interactuar con los servicios ofrecidos por el sistema.

4.1. Arquitectura del sistema

Desde un principio es importante definir de forma clara cómo es la arquitectura del sistema. En la Figura 4 se muestra un esquema básico de los principales módulos que componen el repositorio multimedia. Se puede apreciar la relación que mantienen los elementos entre sí dentro de la arquitectura cliente-servidor. Tanto el cliente web como la APP móvil consumen los servicios web. Sin embargo, solo el cliente web es capaz de acceder a la herramienta de publicación, con la que añadir nuevos contenidos.

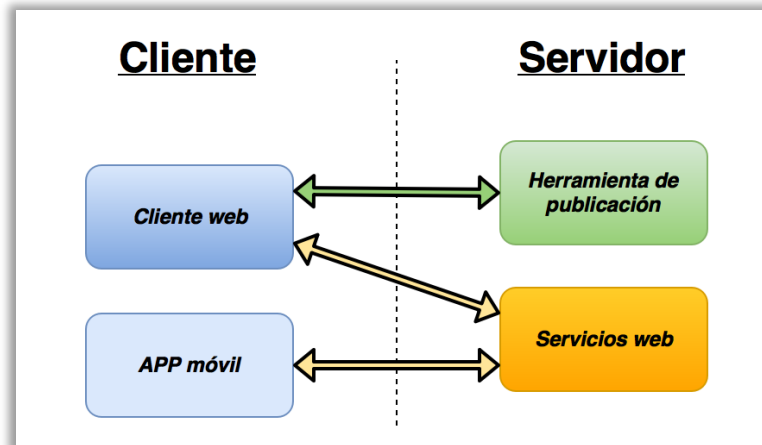


Figura 4. Arquitectura cliente-servidor del repositorio multimedia.

A continuación, se describen cada uno de los módulos de la Figura 4, haciendo hincapié en las funcionalidades que ofrecen y con qué tecnología se han implementado.

- ***Herramienta de publicación***

El servicio está protegido por un portal de acceso. Al superarlo, permite subir al repositorio los archivos multimedia originales para ser convertidos a todos los perfiles definidos. En su implementación interviene toda la lógica de negocio, destacando los *Java Servlets* para interactuar con el cliente web. En la conversión interviene el *framework* multimedia *FFmpeg*.

- ***Servicios web***

Ofrecen, tanto al cliente web como al cliente móvil, la capacidad de acceder a los contenidos del repositorio. Su implementación está basada en tecnología *REST* (para la transferencia de información en general) y en *Java Servlet* (para la transmisión de los archivos mediante *streaming*).

- ***Cliente web***

Crea una interfaz con la que el usuario puede visualizar los contenidos del repositorio, así como publicar nuevos contenidos. Su implementación está basada principalmente en *HTML5*, *CSS3* y *JavaScript*.

- ***APP móvil***

Aplicación derivada del cliente web. Se limita a acceder a los contenidos del repositorio. En su desarrollo destaca el uso del *framework PhoneGap*.

4.2. Estructuración del código

Antes de continuar desglosando el resto de apartados, es interesante introducir cómo está organizado el código de la aplicación principal, para comprender mejor los detalles de implementación.

La aplicación principal, llamada ‘*RepositorioWeb*’, es un proyecto de tipo ‘*Java Web Application*’ creado en el *IDE NetBeans*. Agrupa todos los elementos de la arquitectura a excepción de la *APP* móvil (Ver Figura 4). Estos son: la herramienta de publicación, los servicios web y el cliente web. La *APP* móvil requiere un proyecto secundario, como veremos más adelante.

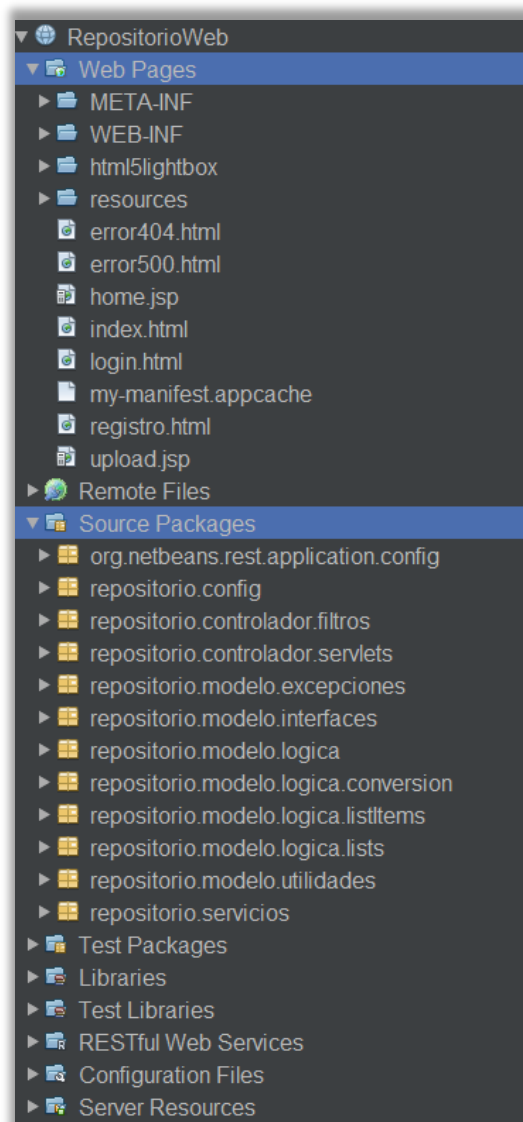


Figura 5. Estructura del proyecto “RepositorioWeb”.

En la Figura 5 se puede observar la estructura del proyecto principal. Se ha organizado el código por paquetes, tal como muestra la entrada ‘*Source Packages*’. Se toma como referencia el Modelo-Vista-Controlador (*MVC*) [21], para separar conceptualmente la funcionalidad de cada paquete.

Dentro del modelo se encuentra la lógica de negocio, incluyendo utilidades, interfaces, excepciones propias y herramientas de conversión. El controlador por su parte agrupa los *servlets* y filtros. Además, se han dedicado otros paquetes exclusivos para la configuración, y para los servicios *REST*. La vista en cambio, está representada por las páginas web contenidas en la entrada ‘*Web Pages*’.

El principio de organización basado en *MVC* ayuda a entender mejor la estructura y funcionamiento de la aplicación. Es posible, sin embargo, que ciertas funcionalidades descritas no se ciñan estrictamente a este modelo, ya que “*las partes del MVC clásico realmente no tienen sentido para los clientes actuales*” [22].

Diagrama de flujo de la aplicación web

En este apartado introductorio, es interesante presentar la estructura de la aplicación web. Conociendo la relación entre los diferentes módulos del proyecto es más sencillo interpretar el funcionamiento global del sistema.

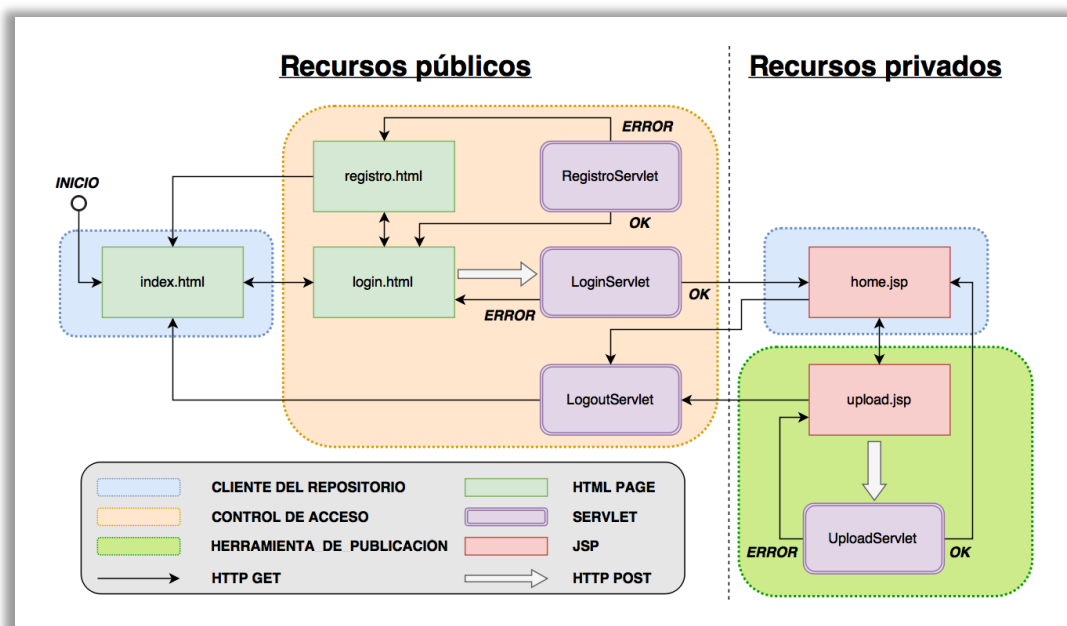


Figura 6. Diagrama de flujo de la interfaz web.

La Figura 6 refleja el diagrama de flujo de la interfaz web que gestiona el repositorio multimedia. En ella se pueden distinguir, en primer lugar, los módulos en los que se divide la aplicación. Destacan: las vistas dedicadas visualizar el repositorio, en azul; la herramienta de publicación de video, en verde; y el portal de acceso que hace de nexo entre los dos anteriores, en naranja. En segundo lugar, se distingue el nombre y tipo de archivo empleado para implementar cada vista o servicio. En tercer lugar, se hace una distinción de aquellos recursos que son públicos o accesibles para cualquier usuario, de los recursos restringidos, protegidos por el portal de acceso. Por último, se especifica cómo se relacionan los elementos de la interfaz entre sí, mediante el tipo de comunicación *HTTP*.

4.3. Persistencia de datos

Los datos utilizados por la aplicación requieren permanencia en el tiempo, así como estar disponibles en todo momento. Por ello, es necesario definir un modelo de persistencia. Dependiendo del tipo de dato que se pretende perpetuar se han seguido diferentes directrices. Estas son:

- **Base de datos local**

Se ha optado por la base de datos relacional, *MySQL*, para el almacenamiento de la información referente a usuarios, medios y sus respectivos recursos multimedia. El uso de una base de datos relacional queda justificado por la necesidad de vincular los datos de cada medio con los de sus correspondientes recursos multimedia. Al mismo tiempo, tanto los medios como los recursos están asociados al usuario que efectúa la publicación.

En esta primera versión, la base de datos estará alojada en un servidor local. Queda propuesta para una versión futura la migración de la base de datos a un servidor accesible a través de Internet, o incluso el uso de un servicio *MySQL* en la nube.

- **Sistema de archivos local**

Mientras que la información asociada a los medios y los recursos multimedia está contenida en la base de datos local, los archivos físicos están almacenados en el sistema de ficheros del propio servidor. De esta forma, se evita guardar en base de

datos los archivos multimedia, reduciendo la información únicamente a la ruta donde están ubicados. Se dedica un directorio exclusivo para el almacenamiento de los medios, diferente al de la aplicación web. Con esto se evita que los archivos desaparezcan cada vez que se limpie y reconstruya el proyecto.

En una versión futura se propone almacenar los archivos multimedia en un servicio de alojamiento en la nube. Con ello se conseguirá descentralizar el servidor de los datos, creando un sistema más persistente y seguro.

- **Archivos de propiedades**

Estos ficheros contienen configuración relativa al funcionamiento del proyecto. Aquí se definen los directorios de almacenamiento de archivos, configuración relativa a la conexión de base de datos e información asociada a los perfiles multimedia.

- **Clase de variables estáticas**

Contiene las constantes de programación comunes a todo el proyecto. Parámetros tales como rutas, nombres y extensiones de los archivos de propiedades, listas blancas de formatos válidos para la herramienta de publicación o tamaños máximos permitidos para los campos de las tablas de la base de datos.

4.4. Perfiles multimedia

La herramienta de conversión requiere de la definición previa de los perfiles multimedia. Cada uno de ellos define los parámetros técnicos asociados a un conjunto de dispositivos con unas características determinadas.

Se han predefinido un total de seis perfiles, basados en los principales conjuntos de referencia. En la versión actual los perfiles y sus parámetros están establecidos como fijos o estáticos, pudiendo ser cambiados únicamente por el administrador a nivel interno. Para una posible versión futura se plantea la posibilidad de añadir la gestión de perfiles a la interfaz, para una libre configuración del usuario.

Los perfiles predefinidos se pueden desglosar en dos. En la Tabla 5 se recogen los tres primeros, orientados tanto para *Android* como para *iOS*. La Tabla 6 recoge tres más, esta vez dedicados a diferentes conjuntos de dispositivos *Android*, de menor a mayor calidad.

		Perfil 1	Perfil 2	Perfil 3
Nombre		General	Alta calidad	Limitados
Formato		mp4	mp4	mp4
Video	Codec	H264	H264	H264
	Profile	Baseline	High	Baseline
	Level	3.0	3.0	3.0
	Tasa de Cuadro	30 fps	30 fps	30 fps
	Resolución	480 x 320	1280 x 720	176 x 144
	Tasa Binaria	1500 kbps	5000 kbps	192 kbps
Audio	Codec	AAC	AAC	AAC
	Tasa de Muestreo	44100 Hz	48000 Hz	16000 Hz
	Canales	2	2	2
	Tasa Binaria	128 kbps	160 kbps	24 kbps

Tabla 5. Perfiles multimedia (I).

		Perfil 4	Perfil 5	Perfil 6
Nombre		Android SD (Low Quality)	Android SD (High Quality)	Android HD
Formato		mp4	mp4	mp4
Video	Codec	H264	H264	H264
	Profile	Baseline	Main	High
	Level	3.0	3.1	4.0
	Tasa de Cuadro	12 fps	30 fps	30 fps
	Resolución	176 x 144	480 x 360	1280 x 720
	Tasa Binaria	56 kbps	500 kbps	2 Mbps
Audio	Codec	AAC	AAC	AAC
	Tasa de Muestreo	16000 Hz	16000 Hz	48000 Hz
	Canales	1	2	2
	Tasa Binaria	24 kbps	128 kbps	192 kbps

Tabla 6. Perfiles predefinidos (II).

El criterio escogido para la definición de cada perfil es reflejar, mediante parámetros técnicos, el índice de calidad que exige el conjunto de dispositivos a los que el perfil está orientado. Teniendo en cuenta, además, qué parámetros son más óptimos para la plataforma a la que están dedicados.

Se podrían presentar infinidad de perfiles, ya que las combinaciones de parámetros son ilimitadas. Incluso dedicar perfiles exclusivos a otras plataformas como *iOS*, pero la atención se centra más en *Android* por ser la plataforma de los dispositivos utilizados para llevar a cabo la experimentación.

Detalles de implementación

Cada uno de los perfiles queda definido en un archivo de propiedades dentro del paquete '*repositorio.config*'. El nombre del archivo viene determinado por el patrón *perfil_n*, siendo *n* un número entero, identificador correspondiente al perfil.

En la Figura 7 se muestra el perfil 1, con sus correspondientes atributos, definido en como archivo '*perfil_1.properties*'. Los atributos, definidos como parejas '*nombre=valor*', serán usados tanto por la herramienta de conversión como por el servicio *REST* que solicite dichos valores para mostrarlos en la interfaz de usuario.

El parámetro *profile* hace referencia al subconjunto de técnicas de codificación disponibles en el códec *H264*. Esto es útil para enfocarse en decodificadores con capacidades de cómputo diferentes (memoria o potencia de procesamiento). El parámetro *level*, o nivel de codificación, está relacionado a cuestiones de ancho de banda, resolución máxima y memoria por parte del decodificador [23].

No debe confundirse el *profile*, o perfil de codificación, con el propio perfil multimedia que representa el archivo de propiedades.

Cabe la posibilidad de definir el perfil y nivel de codificación tanto para video como para audio, pero en este trabajo se especifica únicamente para video. Es por ello que para audio estén declarados, pero no definidos.

```
1  ##
2  # Perfil de video general para iOS y Android
3  # -----
4
5  id=1
6  nombre=General
7  descripcion=Perfil general orientado a dispositivos iOS y Android
8  formatoSalida=mp4
9  sobrescribir=true
10 forzarFormato=true
11 soloAudio=false
12
13 # Parámetros de video
14 profile_video=baseline
15 profileLevel_video=3.0
16 codec_video=h264
17 fotogramas_video=30
18 resolucion_video=480x320
19 bitrate_video=1500k
20
21 # Parámetros de audio
22 profile_audio=
23 profileLevel_audio=
24 codec_audio=aac
25 freqMuestreo_audio=44100
26 canales_audio=2
27 bitrate_audio=128k
28
```

Figura 7. Archivo de propiedades correspondiente al perfil multimedia 1.


4.5. Herramienta de publicación de video

Esta es la primera gran pieza que compone el repositorio multimedia. En este apartado, se describen las diferentes elementos de las que consta (tanto en *back-end* como en *front-end*), y los correspondientes detalles de implementación.

4.5.1 Control de acceso

La herramienta de publicación está provista de un control de acceso tanto por motivos de seguridad como para registrar la autoría de los medios publicados en el repositorio.

El método de autenticación propuesto es mediante usuario y contraseña. Solo aquellos usuarios registrados podrán acceder al servicio de publicación de medios. En la Figura 8 se muestra el formulario de inicio de sesión donde se solicitan las credenciales de acceso.



Repositorio Multimedia

Inicio de sesión

Usuario

Password

Acceder

¿Olvidó las credenciales? [Click aquí](#)

¿Nuevo usuario? [Registro](#)

Máster en Ingeniería de Telecomunicación
Proyecto Fin de Máster

Figura 8. Vista web del portal de acceso a la herramienta de publicación.

Cuando un nuevo usuario no posee credenciales tiene la opción de registrarse en el sistema. El acceso al formulario de registro se realiza haciendo clic, o bien en el icono verde situado en el menú superior, o bien en el enlace ‘Registro’ de la parte inferior (Ver Figura 8).

El formulario de registro, mostrado en la Figura 9, solicita información asociada al usuario. Está compuesto por campos obligatorios, identificados por asteriscos, y por otros campos adicionales.

El nombre de usuario y el correo electrónico están considerados obligatorios porque deben ser únicos, distintos al del resto de usuarios del sistema.

Tanto en la vista de inicio de sesión como en el registro, se muestra la opción de recuperar la contraseña. Al hacer clic, se solicita al usuario que contacte directamente con el administrador para recuperar el acceso. La implementación de un método de recuperación de contraseña automático está pospuesta para una posible versión futura. Cabe recordar que el sistema de control de acceso excede los objetivos del proyecto.

The image shows a web registration form titled "Repositorio Multimedia" with a sub-header "Registro". The form includes the following fields and elements:

- Home icon (house) and Refresh icon (circular arrow).
- Form fields: Usuario*, Contraseña*, Repita la contraseña*, Correo electrónico*, Nombre, Apellidos, Teléfono, and Fecha de nacimiento* (with a dd/mm/aaaa placeholder).
- Gender selection: Género with radio buttons for Hombre and Mujer.
- Registrar button.
- Links: ¿Olvidó las credenciales? [Click aquí](#) and ¿Ya tiene una cuenta? [Login](#).
- Footer: Máster en Ingeniería de Telecomunicación, Proyecto Fin de Máster.

Figura 9. Vista web del formulario de registro de usuarios.

Detalles de implementación

La gestión de la sesión está basada en *HttpSession* [24] y en un filtro web [25] para proteger la herramienta de publicación de accesos no autorizados a través de *URL*. No es la mejor implementación desde el punto de vista de seguridad, ya que no se ha utilizado cifrado en el envío de las credenciales, pero atiende a las necesidades básicas de la aplicación. La protección de la herramienta de publicación mediante un control de acceso es un puente para registrar la autoría de las publicaciones, pero la aplicación de fuertes medidas de seguridad excede los objetivos de este trabajo.

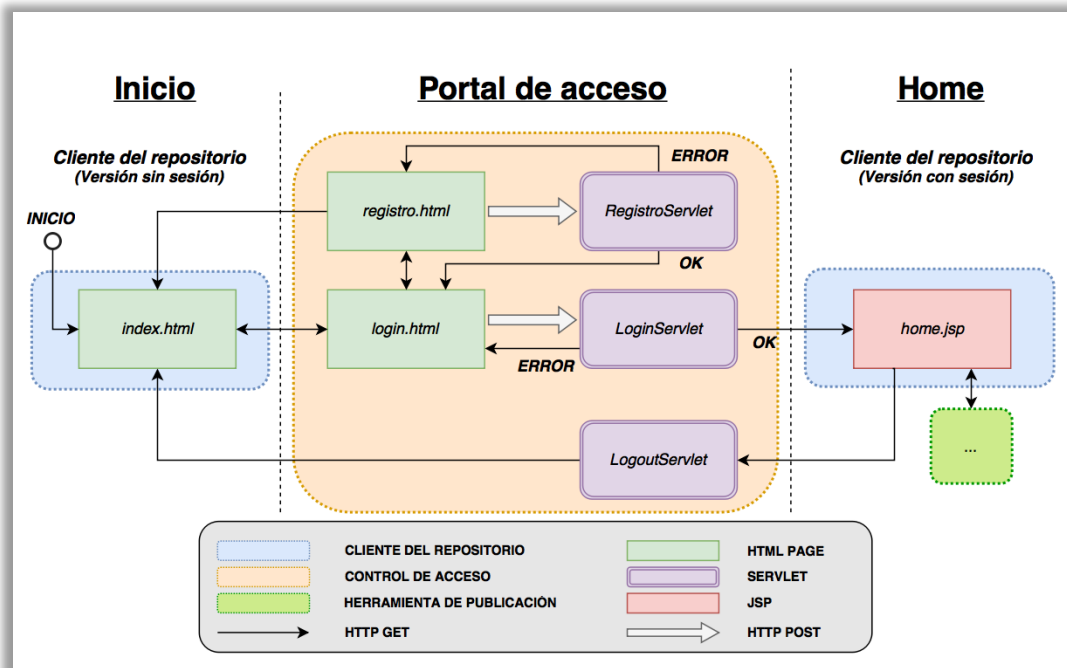


Figura 10. Diagrama de flujo del portal de acceso.

La Figura 10 muestra el diagrama de flujo correspondiente al funcionamiento del control de acceso. A continuación, se describen cada uno de los elementos que lo componen y la relación entre ellos.

- ***index.html***

Esta es la página de inicio donde se muestra el listado de medios y perfiles para los usuarios sin sesión.

- ***login.html***

Contiene la vista web donde el usuario introduce sus credenciales (Ver Figura 8). Los campos del formulario se validan con el *plugin jQuery Form Validator* [26] para comprobar que no estén vacíos o cumplen el tamaño adecuado. Si la validación es correcta los datos se envían a *LoginServlet* vía *POST*.

- ***LoginServlet***

Servlet encargado de validar en base de datos las credenciales recibidas. Si el par de valores usuario y contraseña existe, se crea la sesión *HTTP* y se añade el usuario como variable. A continuación, se redirecciona a *home.jsp*. En cambio, si la validación no es correcta, se redirecciona a *login.html?error*. El parámetro *error*

activa en la interfaz un mensaje, comunicando al usuario que las credenciales no son válidas.

- ***home.jsp***

Variante de *index.html* para usuarios con sesión. Además de mostrar el contenido del repositorio, contiene un enlace para acceder a la herramienta de publicación.

- ***LogoutServlet***

Servlet responsable de cerrar la sesión. Al recibir una petición, se elimina la variable que contiene al usuario y se invalida la sesión actual. Después, se redirecciona a la página de inicio: *index.html*.

- ***registro.html***

Página web contenedora del formulario de registro de usuarios. Al igual que en *login.html*, los datos son validados con el *plugin jQuery Form Validator* [26]. Si la validación es correcta, los datos del formulario se envían a *RegistroServlet* vía *POST*.

- ***RegistroServlet***

Servlet encargado de crear nuevos usuarios. Realiza una validación de los datos recibidos para comprobar que el usuario y el *e-mail* recibido no existan ya en la base de datos local. Solo si la validación es correcta, se procede con el registro. En cualquier caso, el *servlet* genera una respuesta mostrando al usuario el resultado de la operación.

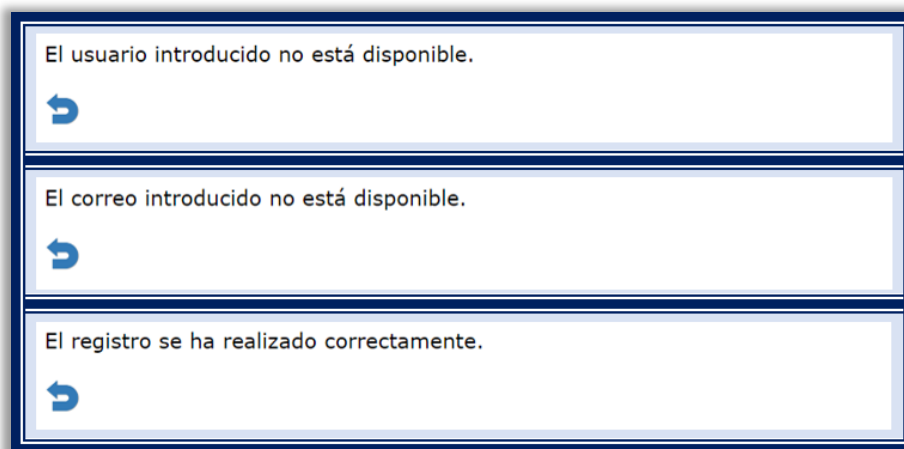


Figura 11. Posibles respuestas del servlet de registro.

En la Figura 11 se muestran las posibles respuestas del *servlet*. Las dos primeras corresponden a errores en la validación, y la última a la confirmación del registro. La flecha azul permite redireccionar a la página de registro cuando se trata de un mensaje de error, o a la página de *login* cuando el registro haya sido correcto.

▪ **FiltroAcceso**

En cada petición *HTTP* al servidor se ejecuta este filtro, basado en *Java Filter*. En la Figura 12 se muestra el comportamiento del filtro a través de un diagrama de flujo.

Tiene dos funciones. La primera y más importante es denegar el acceso a los recursos privados, redireccionando al portal de inicio de sesión. La segunda es redireccionar a la vista con sesión del repositorio, en caso de que un usuario logueado intente acceder mediante *URL* a la página de bienvenida o a la página de inicio de la aplicación. Es decir, evita que un usuario con sesión acceda a la versión sin sesión del repositorio.

Se entiende como página de inicio a *index.html*, y como página de bienvenida a la *URL* raíz del proyecto: '*192.168.1.10:1010/RepositorioWeb/*'. De hecho, la página de bienvenida, definida en el descriptor de despliegue '*web.xml*', coincide con la página de inicio, pero a efectos de *URL* es necesario diferenciar ambos casos.

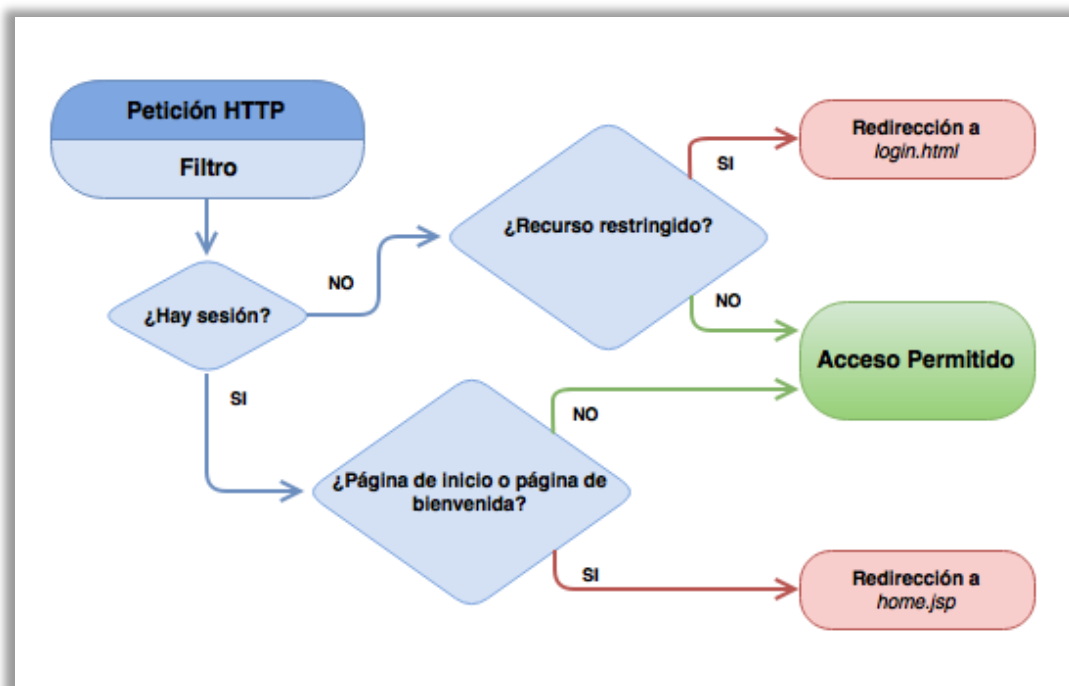


Figura 12. Diagrama de flujo del filtro de control de acceso.

4.5.2 Subida de medios al servidor

Con la creación del portal de acceso ya se dan las condiciones para acceder a la herramienta de publicación. En este apartado se describe el proceso de carga de los medios al servidor, paso previo a la posterior conversión y publicación.

Comenzando desde el lado del usuario (*front-end*). Se dispone de una vista web definida en el archivo '*upload.jsp*' (Ver Figura 13), donde a través de un formulario se permite definir el título del medio, seleccionar un archivo del disco, y accionar un botón para proceder con la petición de subida. Los datos del formulario son enviados vía *POST* al servlet '*UploadServlet*' (*back-end*), para su posterior procesamiento.

The screenshot shows a web interface for uploading media. At the top, there is a dark blue header with the text 'Repositorio Multimedia'. Below this is a light blue navigation bar with a home icon on the left, the title 'Subir Medio' in the center, and a back arrow icon on the right. The main content area is light blue and contains a form with the following elements: a text input field labeled 'Titulo del medio*', a button labeled 'Seleccionar archivo' followed by the text 'Ningún archivo seleccionado', a list of valid file formats: '.avi .mp4 | .wav .mp3 .wma (Ver todos)', and a blue button labeled 'Subir'. At the bottom of the page, there is a dark blue footer with the text 'Máster en Ingeniería de Telecomunicación' and 'Proyecto Fin de Máster'.

Figura 13. Vista web del formulario de subida de archivos.

Existe un filtro de seguridad que comprueba la extensión del archivo subido. Para ello, se ha definido una lista de extensiones válidas. Si la extensión del archivo no está en la lista, se detiene el proceso de conversión y el *servlet* genera una respuesta indicando el motivo al usuario (Ver Figura 14). La flecha contiene un enlace para redirigir a la vista anterior. En una posible versión futura, este mensaje podría mostrarse directamente en la propia interfaz web sin necesidad de redirección. Esto supone aumentar considerablemente la complejidad de la interfaz, lo que excede los objetivos del proyecto.

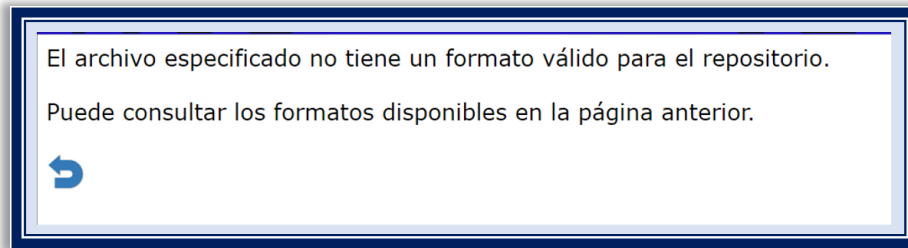


Figura 14. Respuesta del servlet al subir un archivo no válido.

El criterio seguido para definir el filtro de seguridad atiende a los formatos y sus correspondientes extensiones que es capaz de decodificar el núcleo de la herramienta de conversión; el *framework FFMPEG* [13], descrito en detalle en los próximos apartados.

Hay que tener en cuenta que este filtro es una medida preventiva, pero no es la mejor práctica recomendada para evitar amenazas graves de seguridad [27]. Existen otras medidas complementarias como podrían ser:

- Definir un archivo *.htaccess* donde incluir la lista de extensiones permitidas.
- Ubicar el *.htaccess* en un directorio diferente al destinado a los archivos subidos.
- Prevenir ataques de doble extensión.
- Subir los archivos a un directorio diferente del directorio raíz del servidor.
- Limitar la sobrescritura de archivos existentes para evitar modificaciones del fichero *.htaccess*.
- Crear una lista de *mime-types* permitidos y verificar las cabeceras de los archivos.
- Generar el nombre de archivo aleatorio, para añadir posteriormente la extensión original.
- No limitar la validación solo en *front-end*. Lo ideal es mantener la validación en ambos: cliente y servidor.

Detalles de implementación

La vista utiliza un formato *JSP* en lugar de *HTML* para evitar que el navegador guarde la página en caché. Esto obliga a ejecutar el filtro de acceso en cada petición garantizando la redirección cuando no se cumplan los requisitos de admisión.

Se realiza validación de los datos tanto en *front-end* como en *back-end*. En el cliente, se hace uso del *plugin jQuery Form Validator* [26] para exigir que los dos campos requeridos no estén en blanco. En el servidor, tal como muestra la Figura 15, se realiza

en serie tanto la validación de campos como el filtrado por extensión. Solo si se cumplen los requisitos se procede a recuperar el archivo solicitado. En caso contrario se muestra el correspondiente mensaje de error.

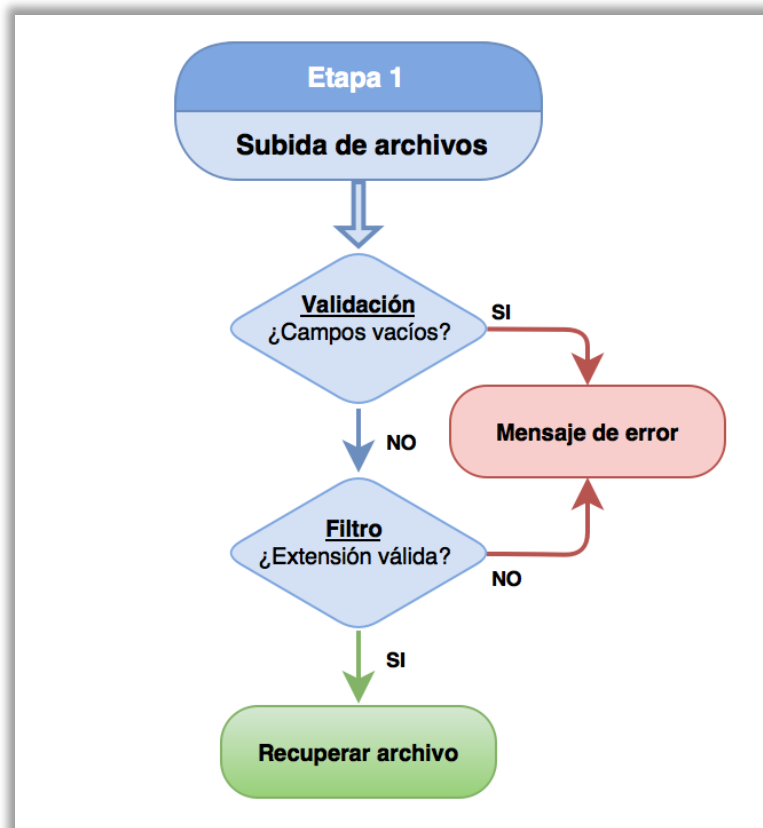


Figura 15. Diagrama de flujo del proceso de subida de archivos.

El proceso de recuperación del archivo está basado en el ejemplo propuesto en *Java EE 6 Tutorial* [28]. Se establece un *buffer* de datos de 1024 *bytes* sobre el que se va cargando el archivo seleccionado en la interfaz, para quedar guardado finalmente en un directorio temporal.

El archivo queda representado por un objeto de tipo *MedioMultimedia*. Esta clase contiene los elementos característicos de un medio, como son: el título, el usuario, el estado actual, la ruta del archivo, y los mecanismos necesarios para mantener la persistencia de datos.

Tras crear la instancia con los valores citados y definir el estado actual como 'PROCESANDO', se guarda en la tabla 'medios' de la base de datos local. A continuación, se inicia la siguiente etapa.

4.5.3 Conversión de medios

La conversión de los medios a sus correspondientes perfiles es la última etapa del proceso de publicación de medios. A diferencia de las etapas anteriores, este proceso se realiza íntegramente en el servidor, siendo transparente al usuario. Sin embargo, se ofrece una monitorización del estado de la conversión en la interfaz, como veremos en la sección dedicada al repositorio.

La Figura 16 muestra un diagrama que ayuda a esclarecer el, a priori, complejo sistema de conversión. A continuación, se procede a profundizar en cada uno de los elementos que intervienen en el proceso.

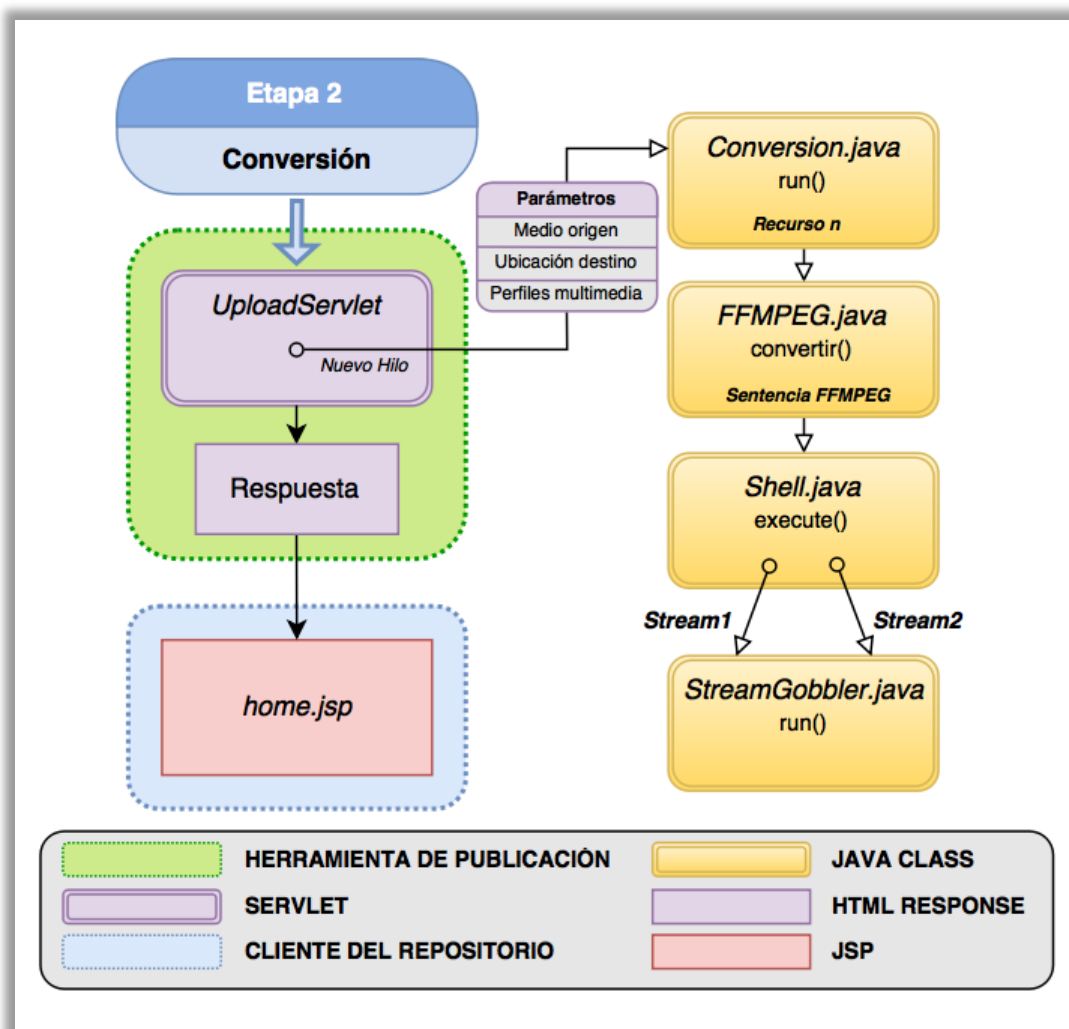


Figura 16. Proceso de conversión de medios.

El punto de partida es el final de la etapa uno, donde se dispone del medio multimedia original almacenado en un directorio temporal. El *servlet* lanza el proceso de conversión

en un hilo de ejecución independiente, permitiendo la conversión en segundo plano. A continuación, genera una respuesta *HTML* con el contenido mostrado en la Figura 17. Aquí el usuario vuelve a disponer de un enlace para redireccionar, en este caso, al repositorio, donde podrá ver el medio publicado y seguir el estado de la conversión.

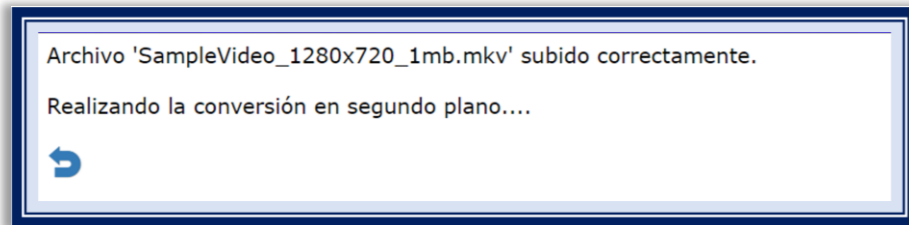


Figura 17. Respuesta del servlet tras lanzar el proceso de conversión.

Volviendo al proceso de conversión, los elementos implicados son los siguientes:

- ***Conversion.java***

Clase que implementa la interfaz *Runnable* para permitir ejecutar el método *run* de la clase como hilo independiente. Recibe como parámetros del *servlet*, el medio multimedia de origen, la ubicación de destino y el listado de perfiles multimedia.

El código se estructura en tres partes. En la primera se prepara la conversión. Para cada perfil multimedia se instancia un nuevo objeto *RecursoMultimedia*. Este se asocia a la ubicación de destino, creando el directorio si es necesario. Seguidamente se define el estado actual como '*EN_COLA*', y se inserta en base de datos.

En la segunda parte se procede a convertir cada *RecursoMultimedia*. Antes de iniciar cada conversión individual se cambia el estado actual a '*PROCESANDO*' y se actualiza en base de datos. A continuación, se llama al método *convertir()* de la clase '*FFMPEG.java*', pasándole el recurso como parámetro.

En la tercera parte, cuando ya se han lanzado todas las conversiones, se analizan los resultados. Si todas las conversiones son correctas, se establece el valor '*OK*' al estado del medio multimedia al que pertenecen los recursos. En cambio, si ha habido algún error, el estado pasa a ser '*ERROR*'.

El último paso es borrar el archivo asociado al medio multimedia original, almacenado en el directorio temporal.

- ***FFMPEG.java***

Esta clase consta de dos métodos principales. El primero, *toFFMPEG()*, construye una sentencia de ejecución acorde a la documentación del *framework FFmpeg* [13]. Los principales valores son el archivo de origen, definido por el medio multimedia; la ruta de destino, definido por el recurso multimedia; y los parámetros técnicos de conversión, definidos por el perfil multimedia.

El segundo método, '*convertir()*', se encarga de ejecutar la sentencia *FFmpeg* en la línea de comandos. Para ello hace uso del siguiente eslabón del proceso, la clase '*Shell.java*'. De ella recibe un código de error para determinar el estado de la ejecución.

- ***Shell.java***

Clase dedicada a ejecutar instrucciones directamente en la línea de comandos. Está orientado a ejecutar comandos sobre una plataforma *Windows*, siendo *Windows 10* el sistema operativo utilizado en este trabajo.

La clase consta de un único método, '*execute()*'. Recibe la sentencia *FFmpeg* de la clase anterior y la ejecuta como se haría en la propia línea de comandos. Al finalizar, devuelve el código de error mencionado previamente para proceder de un modo u otro.

- ***StreamGobbler.java***

Esta última clase va de la mano de la anterior. Es útil para leer la salida generada por la línea de comandos, pudiendo escribir, tanto en el *log* como en la salida del *IDE*. Así, es posible realizar un seguimiento a la ejecución de los comandos realizados por '*execute()*' de la clase *Shell*. En este caso, se ha hecho llamar a dos ejecuciones en hilos independientes, para dos flujos de datos. En el primer *stream* está la salida 'normal' o informativa de la consola. En el segundo *stream* se muestran los errores de ejecución.

Es importante mencionar que toda la salida de datos generada por el *framework FFmpeg* se realiza por el *stream* de errores. No debiendo interpretar esta salida como tal [29]. En la Figura 18 se muestra como ejemplo, un fragmento de la salida del flujo de datos generada durante la conversión de un medio multimedia.

```
Execing cmd.exe /C ffmpeg -i "C:\Users\Juan Carlos\Desktop\TFM_0\NetBeansProjects\RepositorioWeb\k
OUTPUT>ffmpeg version N-81729-g7d17d31 Copyright (c) 2000-2016 the FFmpeg developers
OUTPUT> built with gcc 5.4.0 (GCC)
OUTPUT> configuration: --enable-gpl --enable-version3 --disable-w32threads --enable-dxva2 --enabl
OUTPUT> libavutil      55. 30.100 / 55. 30.100
OUTPUT> libavcodec     57. 57.101 / 57. 57.101
OUTPUT> libavformat     57. 50.100 / 57. 50.100
OUTPUT> libavdevice     57.  0.102 / 57.  0.102
OUTPUT> libavfilter      6. 62.100 /  6. 62.100
OUTPUT> libswscale       4.  1.100 /  4.  1.100
OUTPUT> libswresample    2.  1.100 /  2.  1.100
OUTPUT> libpostproc     54.  0.100 / 54.  0.100
OUTPUT>Input #0, avi, from 'C:\Users\Juan Carlos\Desktop\TFM_0\NetBeansProjects\RepositorioWeb\bu
OUTPUT> Duration: 00:00:06.25, start: 0.000000, bitrate: 1071 kb/s
OUTPUT>   Stream #0:0: Video: indeo4 (IV41 / 0x31345649), yuv410p, 256x240, 1065 kb/s, 40 fps, 40
OUTPUT>[libx264 @ 0000000002508b80] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX
OUTPUT>[libx264 @ 0000000002508b80] profile Main, level 3.1
OUTPUT>[libx264 @ 0000000002508b80] 264 - core 148 r2705 3f5ed56 - H.264/MPEG-4 AVC codec - Copyle
OUTPUT>Output #0, mp4, to 'C:\media\Android_SD (High-quality)\ID12-cbw3.mp4':
OUTPUT> Metadata:
OUTPUT>   encoder           : Lavf57.50.100
OUTPUT>   Stream #0:0: Video: h264 (libx264) ([33][0][0][0] / 0x0021), yuv420p, 480x360, q=-1--1,
OUTPUT> Metadata:
OUTPUT>   encoder           : Lavc57.57.101 libx264
OUTPUT> Side data:
```

Figura 18. Ejemplo de flujo de datos generado por la ejecución de *ffmpeg*.

4.6. Repositorio multimedia

El repositorio multimedia es la piedra angular sobre la que gira el resto de componentes del proyecto. Constituye el portal donde se almacenan, se monitorizan y se reproducen los medios subidos a través de la herramienta de publicación.

El término repositorio abarca los contenidos almacenados, la interfaz gráfica donde se visualizan y los servicios que permiten realizar la comunicación entre ambos.

En primer lugar, se va a hablar de los servicios que ofrecen los datos almacenados por los diferentes métodos de persistencia, para después describir en detalle la interfaz gráfica que los consume, separando así la parte servidor de la parte cliente.

4.6.1. *Back-end*: Servicios *REST*

Uno de los requisitos para el diseño de este proyecto es utilizar tecnología *REST* [30] como sistema de comunicación entre cliente y servidor. A continuación, se procede a describir esta tecnología y a justificar su uso [31].

Principios de la arquitectura REST

El primero de estos principios es saber que todo lo que se mueve a través de las comunicaciones web es un recurso. La idea es que los datos se representan con el formato específico que tienen y no como un archivo físico.

Cada recurso disponible en la red tiene un formato en particular que se describe por el tipo de contenido: *jpeg* para imágenes, *mpeg* para video, *xml*, *html*, entre otros. Los cuales van a estar referidos en los protocolos de comunicación como: *image/jpeg*, *video/mpeg*, *text/html*, *text/xml*, etc.

El segundo principio a tener en cuenta es que cada uno de estos recursos debe tener un identificador único, definido por su *URL*. Ya que hay una infinidad de recursos en la web, deben estar accesibles e identificables.

El tercer principio alude a la comunicación. Este protocolo de transmisión de datos debe utilizar los verbos estándares de *HTTP*, que están definidos en el protocolo nativo. Cada uno de estos verbos significa una acción diferente. Hay definidas ocho acciones principales: *GET*, *POST*, *PUT*, *DELETE*, *HEAD*, *OPTIONS*, *TRACE* y *CONNECT*.

El cuarto principio y clave en este tipo de arquitectura es que cada recurso puede tener múltiples representaciones, independientemente de cómo esté almacenado. Un ejemplo sencillo para ilustrar este principio sería tener un recurso en formato *XML* y poder solicitarlo en *JSON*.

El quinto principio es clave para entender las comunicaciones cliente-servidor. Se trata de comunicaciones que se denominan sin estado (*STATELESS*), lo que significa que cada petición al servidor es tratada de manera totalmente independiente.

Ventajas de la arquitectura REST

Algunas de las ventajas que ofrece esta arquitectura son:

- **Separación de un recurso de su representación**

Un recurso es solo un conjunto de información y como se ha definido anteriormente, puede tener múltiples representaciones.



El recurso no tiene estado. Corresponde a quien lo solicita especificar en el encabezado de la petición *HTTP* el tipo de contenido que se desea obtener.

Una vez recibido este requerimiento, será la aplicación del servidor la encargada de manejar la representación y devolver el estado *HTTP* apropiado:

- *HTTP 200*: código de confirmación *OK*.
- *HTTP 404*: implica que un recurso no está disponible.
- *HTTP 500*: denota un error interno del lado del servidor.

Es necesario, por tanto, enviar al servidor, en la cabecera *HTTP*, qué es lo que se espera recibir. Por ejemplo:

- *text/xml*
- *application/json*
- *image/jpeg*
- *video/mpeg*

▪ **Visibilidad**

REST está diseñado para ser visible y simple, lo que significa que cada aspecto del servicio debe ser auto descriptivo siguiendo las normas *HTTP*, comentadas anteriormente.

▪ **Seguridad.**

Al utilizar *REST* se garantiza que los métodos *HTTP* son seguros. El hecho de solicitar un recurso no modifica o causa ningún tipo de cambio en su estado.

▪ **Escalabilidad**

Si el aumento de la demanda exige aumentar el número de servidores, esto puede hacerse sin comprometer la sincronización entre los mismos, ya que no afecta al estado de los recursos.

▪ **Rendimiento**

La escalabilidad no debe ser confundida con el rendimiento. El rendimiento se mide por el tiempo necesario para que una única petición sea procesada, mientras que

la escalabilidad depende del número total de peticiones que la aplicación puede manejar.

Al tratarse de pequeños servicios, el tiempo de respuesta es muy bajo, ya que éstos no deberían manejar un gran volumen de lógica de negocio. Con esta arquitectura se aprovecha el poder de procesamiento de las máquinas cliente.

Una vez introducida la tecnología, se procede a presentar los servicios *REST* creados para satisfacer las necesidades del sistema.

La Tabla 7 muestra los métodos y sus correspondientes *URIs*, o “*Identificadores de Recursos Uniformes*”, para cada uno de los servicios creados. La base de los servicios, representada en la tabla por los tres puntos suspensivos iniciales, corresponde a <http://192.168.1.10:1010/RepositorioWeb/>. Siendo *192.168.1.10:8084*, la dirección *IP* y el puerto donde está configurado el servidor web que aloja el servicio. Y *RepositorioWeb* el nombre de la aplicación web. El direccionamiento se justifica en el capítulo dedicado a la experimentación.

	Método	URI
MedioService	<i>getListadoMedios()</i>	.../webresources/ medio /
	<i>getMedioById()</i>	... /webresources/ medio / {medio_id}
RecursoService	<i>getRecursoById()</i>	... /webresources/ recurso / {recurso_id}
	<i>getRecursosByMedio()</i>	... /webresources/ recursos / {medio_id}
	<i>getRecursoByMedioAndPerfil()</i>	... /webresources/ medio / {medio_id} / {perfil_id}
PerfilService	<i>getPerfilesFromProperties()</i>	... /webresources/ perfil /
	<i>getPerfilByIdFromProperties()</i>	... /webresources/ perfil / {perfil_id}
UsuarioService	<i>getUsuarios()</i>	... /webresources/ usuario /
	<i>getUsuarioById()</i>	... /webresources/ usuario / {usuario_id}

Tabla 7. Servicios REST y sus correspondientes métodos y URIs.

Los distintos métodos están dedicados, o bien para devolver un *array* completo de recursos, o bien para obtener un único elemento, dependiendo de las necesidades del cliente. Es decir, todos implementan el método *GET* del protocolo *HTTP* para recuperación de datos. Se pospone la implementación de otros de métodos *HTTP* como *PUT* o *DELETE* para una versión futura, donde la interfaz gráfica permita la modificación y borrado de los contenidos.

El principio de funcionamiento es el mismo para todos los servicios. En primer lugar, se accede a los datos solicitados utilizando el método de persistencia empleado. De los cuatro servicios, el único que obtiene los valores de un archivo de propiedades es *PerfilService*, por ser el método elegido para implementar los perfiles multimedia. El resto obtienen los datos de la base de datos local, utilizando las tablas: *medios*, *recursos*, y *usuarios* respectivamente.

Una vez obtenidos los datos de origen, el siguiente paso es tratarlos para conseguir el formato de intercambio deseado. En este trabajo se ha optado por *JSON* [32], entre otros por compatibilidad y ligereza [33]. Puede ser leído por cualquier lenguaje de programación, permitiendo la comunicación entre distintas tecnologías. La ligereza radica en que los datos son representados como texto, en una estructura más simple que *XML*, por ejemplo.

Para la representación de los objetos *Java* a *JSON* se han contemplado dos alternativas. O bien utilizar un *parser* o analizador sintáctico como *GSON* [34], o bien realizar una implementación manual, construyendo a conciencia la cadena de texto representativa del objeto. El uso de un *parser* ofrece la ventaja de generar el *JSON* directamente, haciendo uso de las herramientas que ofrece la librería. Para la versión final de este trabajo se ha escogido la implementación manual. Pese a ser más costoso, ofrece una mayor libertad respecto a que campos añadir, quitar o formatear al realizar la representación.

En el proceso de desarrollo es interesante disponer de una herramienta para verificar el correcto funcionamiento de los servicios. En este trabajo se ha utilizado la extensión de *Chrome*, *Restlet Client* [35]. En la Figura 19 se muestra un ejemplo de uso al solicitar el *JSON* asociado al listado de medios. La petición permite definir la *URI* del recurso, el

protocolo a emplear (*HTTP* o *HTTPS*), el método de solicitud (*GET*, *POST*, *PUT*, *DELETE*, etc.), o modificar las cabeceras.

El código de respuesta nos indica el estado del recurso, comprobando si está disponible o no, o si hay un error interno en el servidor que requiera su correspondiente depuración y corrección. El cuerpo devuelve, si el estado es *OK*, el recurso en el formato solicitado. Aquí podemos comprobar si los campos y sus valores son los esperados.

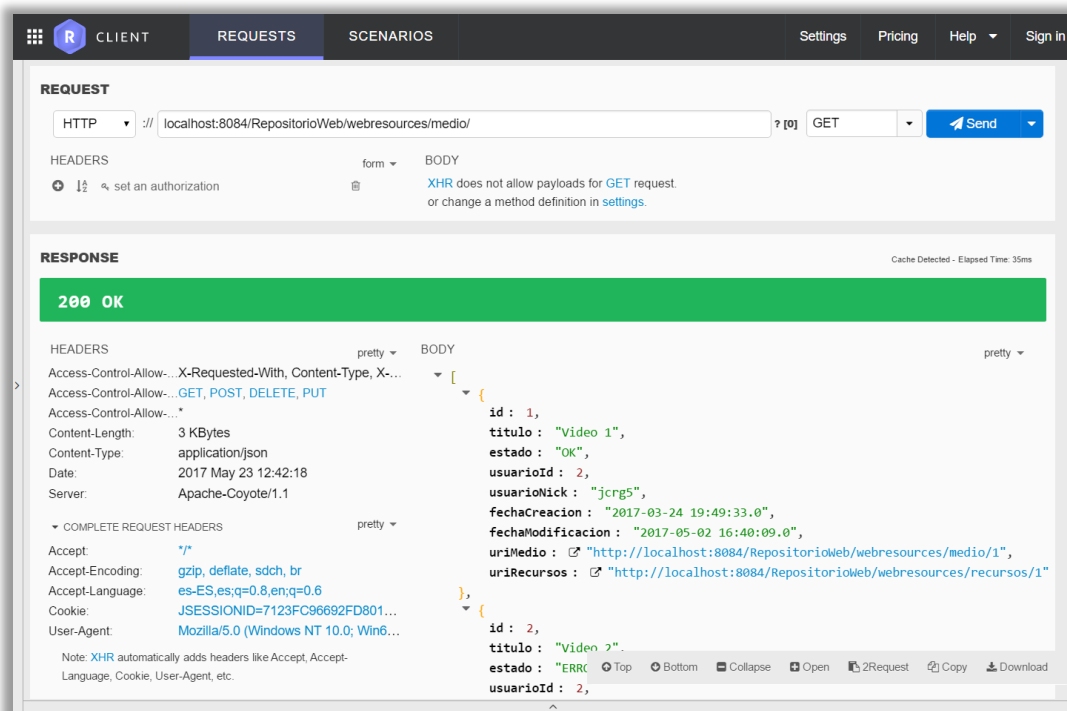


Figura 19. Restlet Client: Herramienta de testeo de los servicios REST.

Detalles de implementación

Para el uso de los servicios *REST* en *Java* son necesarias las librerías *JAX-RS* y *Jersey* [36]. *JAX-RS* proporciona soporte en la creación de los servicios web acorde al estilo *REST*, definiendo además un conjunto de anotaciones para simplificar el desarrollo y despliegue de los servicios. *Jersey* por su parte es la implementación de referencia de la especificación *JSR 311*. Permite desplegar servicios web en un contenedor de *servlets*.

Para implementar de forma manual la conversión del objeto a *JSON* se ha sobrescrito el método *toString()* de las clases a representar.



Se diferencian las clases destinadas a la representación de los objetos en formato *JSON*, de las que realizan la lógica de negocio. A su vez, se destinan clases diferentes para los listados y para los elementos individuales. Por ejemplo, para representar un medio multimedia tenemos:

- *MedioMultimedia.java*

Clase principal, definida en el paquete “*repositorio.modelo.logica*”. Esta clase no está orientada a la representación de los medios como recurso *REST*, sino a manejar la lógica más profunda, orientada a la gestión de los ficheros asociados al medio multimedia, por ejemplo.

- *MedioListItem.java*

Elemento de lista, definido en el paquete “*repositorio.modelo.logica.listitems*”. A diferencia del anterior, sus atributos son tipos de datos simples para crear una representación *JSON* lo más plana o básica posible. Aquí es donde se implementa el método *toString()*, utilizado para representar el elemento en formato *JSON*.

- *MediosList.java*

Lista de medios definido en el paquete “*repositorio.modelo.logica.lists*”. Contiene elementos del tipo *MedioListItem*. Su representación en *JSON* pasa por llamar al método *toString()* de la propia lista, generando automáticamente el listado donde cada elemento corresponde a la implementación manual del método sobrescrito en la clase anterior.

Siguiendo estas pautas, la relación entre clases para cada objeto asociado a los servicios *REST* queda reflejada en la Tabla 8.

	Clase Principal	Elemento de lista	Listado
Medio	<i>MedioMultimedia.java</i>	<i>MedioListItem.java</i>	<i>MediosList.java</i>
Recurso	<i>RecursoMultimedia.java</i>	<i>RecursoListItem.java</i>	<i>RecursosList.java</i>
Perfil	<i>PerfilMultimedia.java</i>	<i>PerfilListItem.java</i>	<i>PerfilesList.java</i>
Usuario	<i>Usuario.java</i>	<i>UsuarioListItem.java</i>	<i>UsuariosList.java</i>

Tabla 8. Clases java involucradas en la representación de los recursos *REST*.

Hay que tener en cuenta que al crear los servicios *REST* se crea automáticamente la clase *ApplicationConfig*, en el paquete “*org.netbeans.rest.application.config*”. En ella es posible definir, entre otros, el nombre base a los servicios. Por defecto: *webresources* (Ver Tabla 7).

En esta clase de configuración de los servicios se especifica además el uso de un filtro *CORS*, necesario para atender peticiones de dominio cruzado [37]. Este filtro modifica las cabeceras de las peticiones para permitir el acceso desde dominios externos. Esto va a permitir que un cliente móvil, situado en un dominio diferente, pueda acceder a los servicios *REST*. Sin la modificación explícita de las cabeceras, las peticiones externas serían rechazadas por motivos de seguridad.

El filtro *CORS*, definido por la clase *CORSResponseFilter* queda estructurado en el paquete “*repositorio.controlador.filtros*”, junto al filtro de acceso.

4.6.2. Back-end: Servicio de video streaming

Mientras que la transmisión de datos basados en *JSON* está implementada con servicios *REST*, no es así para los contenidos multimedia que se deseen consumir en el repositorio. Con tecnología *REST* no es posible enviar un medio para su reproducción en vivo. Es necesario descargar el archivo antes de poder visualizarlo.

Como alternativa, se ha optado por utilizar un *servlet* como servidor de los contenidos en *streaming*. En la Tabla 9 se muestra la *URI* correspondiente para solicitar un recurso, definido por el parámetro *recurso_id*. Como ya se ha comentado en el apartado anterior, los puntos suspensivos corresponden al contexto de la aplicación. (Ver Tabla 7).

Se ha optado por incluir el servicio dentro de *webresources*, la *URI base* destinada a los servicios *REST*. Esta inclusión permite generar las *URIs* de cada recurso dinámicamente a partir del contexto (*UriInfo*).

	Método	URI
PasarelaServlet	<i>doGet()</i>	<i>.../webresources/media?recurso={recurso_id}</i>

Tabla 9. URI asociado al servicio de video streaming.



Detalles de implementación

Cuando el *servlet*, denominado *PasarelaServlet*, recibe una petición *GET*, espera como parámetro el identificador del recurso multimedia que se desea reproducir. La primera tarea que lleva a cabo el *servlet* es comprobar que el recurso existe en base de datos y que su estado está *OK*, es decir, que la conversión resultó exitosa en el momento de su publicación. Se obtiene entonces de base de datos el *path* o ruta asociada al archivo, almacenada en el sistema de archivos local. La siguiente comprobación es asegurar que el archivo realmente existe. Si cualquiera de las comprobaciones no resulta satisfactoria, se cancela el proceso y se genera un código de respuesta *404*, asociado a recurso no encontrado (*NOT FOUND*).

Si el fichero existe, se procede a abrir un *InputStream*, o flujo de datos, para leer el archivo en binario. Este flujo, o *stream*, se agrega a la respuesta del *servlet* utilizando un *buffer* de lectura, permitiendo un envío progresivo del contenido del archivo.

En la respuesta se especifican además algunos parámetros, como son el *mime-type*, el tamaño total del medio, y el nombre del archivo.

Cabe mencionar que si algún perfil multimedia define sus archivos con algún formato adaptativo como *HLS (HTTP Live Streaming)* o *DASH (Dynamic Adaptive Streaming over HTTP)*, el *servlet* realizará la transmisión de datos acorde a estos protocolos.

4.6.3. Front-end: Cliente Web

Una vez implementados los servicios que dan acceso a los contenidos del repositorio, ya se dan las condiciones para desarrollar el cliente.

Uno de los requisitos a tener en cuenta para la elaboración del cliente es el uso exclusivo de lenguajes *front-end*, como *HTML*, *JavaScript*, o *CSS*. Estos tienen la característica de ejecutarse íntegramente en el lado del cliente, es decir, en el propio dispositivo donde se visualiza la interfaz gráfica con la que interactúa el usuario.

Aunque a priori este requisito parezca una restricción, los beneficios que se obtienen son considerables. Utilizar exclusivamente esta tecnología va a permitir reutilizar el

código para desarrollar el cliente móvil multiplataforma; última pieza clave de este trabajo.

Con lo que al cliente respecta, a partir de este momento se hace una distinción entre el cliente web y el cliente móvil. Aunque ambos estén basados en web, presentan variaciones de aspecto y código. El cliente web, descrito en este apartado, forma parte de *RepositorioWeb*, la aplicación que compone todo el sistema completo. Es decir, el proyecto que incluye todos los elementos descritos hasta ahora: control de acceso, herramienta de publicación, *servlets*, filtros, servicios *REST*, etc. El cliente móvil, en cambio, puede considerarse una aplicación independiente, derivada únicamente de los elementos que componen el *front-end* del cliente web.

El cliente web, como ya se ha comentado previamente a lo largo del capítulo, cuenta con dos interfaces diferentes. La primera es la página de bienvenida, dedicada a usuarios sin sesión. La segunda, para usuarios logueados, proporciona acceso a la herramienta de publicación. Ambas, implementadas en archivos diferentes, representan el *front-end* del repositorio multimedia.

La diferencia visual de las interfaces solo se aprecia en el encabezado de la vista web. En la Figura 20 se muestra la versión sin sesión. En ella aparecen tres iconos, cuyas funcionalidades son: la casa, para mostrar el listado de medios; el icono rojo con la tuerca, para mostrar el listado de perfiles multimedia; y la puerta con la flecha azul, para iniciar sesión.

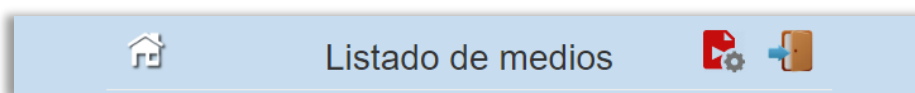


Figura 20. Encabezado de página para la versión sin sesión del repositorio.

En la Figura 21 se muestra el encabezado de la versión con sesión. Aquí cambia la distribución de los iconos. El icono para mostrar los perfiles se sitúa junto a la casa y aparece ahora un icono azul con forma de nube para redirigir a la herramienta de publicación. El icono de la puerta, con la flecha roja en este caso, permite cerrar sesión. Al hacerlo, la vista cambiaría a la versión anterior.

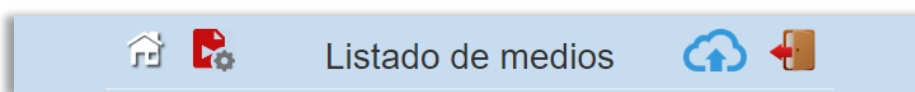


Figura 21. Encabezado de página para la versión con sesión del repositorio.

En la Figura 22 se muestra como estas dos versiones interactúan del mismo modo con el *back-end* correspondiente al repositorio.

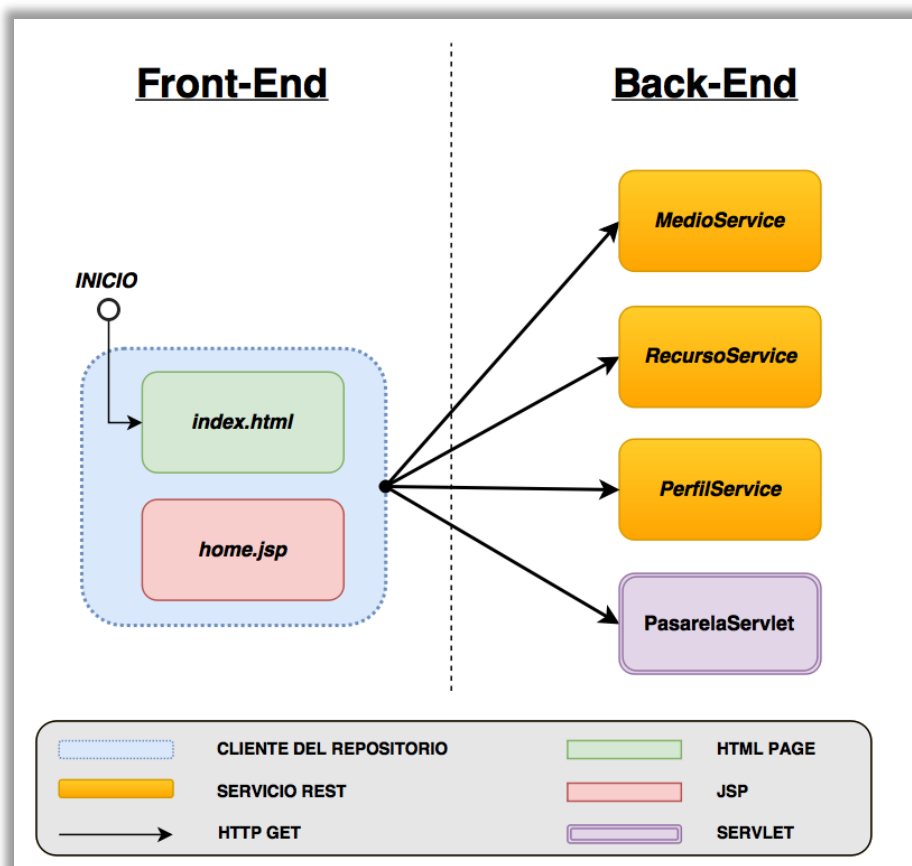


Figura 22. Repositorio multimedia: Relación cliente-servidor.

El uso de diferentes formatos para cada versión atiende al mismo criterio aplicado en la herramienta de publicación (Ver detalles de implementación del apartado 4.4.2. Subida de medios al servidor)

Los servicios que consume el cliente son los descritos en los apartados anteriores, a excepción de *UsuarioService*. Este servicio en concreto está destinado a la gestión de usuarios, cuya interfaz web dentro de la aplicación excede los objetivos del proyecto, proponiéndose como ampliación de un posible trabajo futuro.

A continuación, se procede a mostrar y describir la funcionalidad común a las dos versiones del cliente del repositorio web. En la Figura 23 aparece la página de bienvenida de la aplicación: el repositorio donde se listan los medios publicados. Cada medio se identifica por su título, estado, usuario y fecha en que se realizó la publicación.

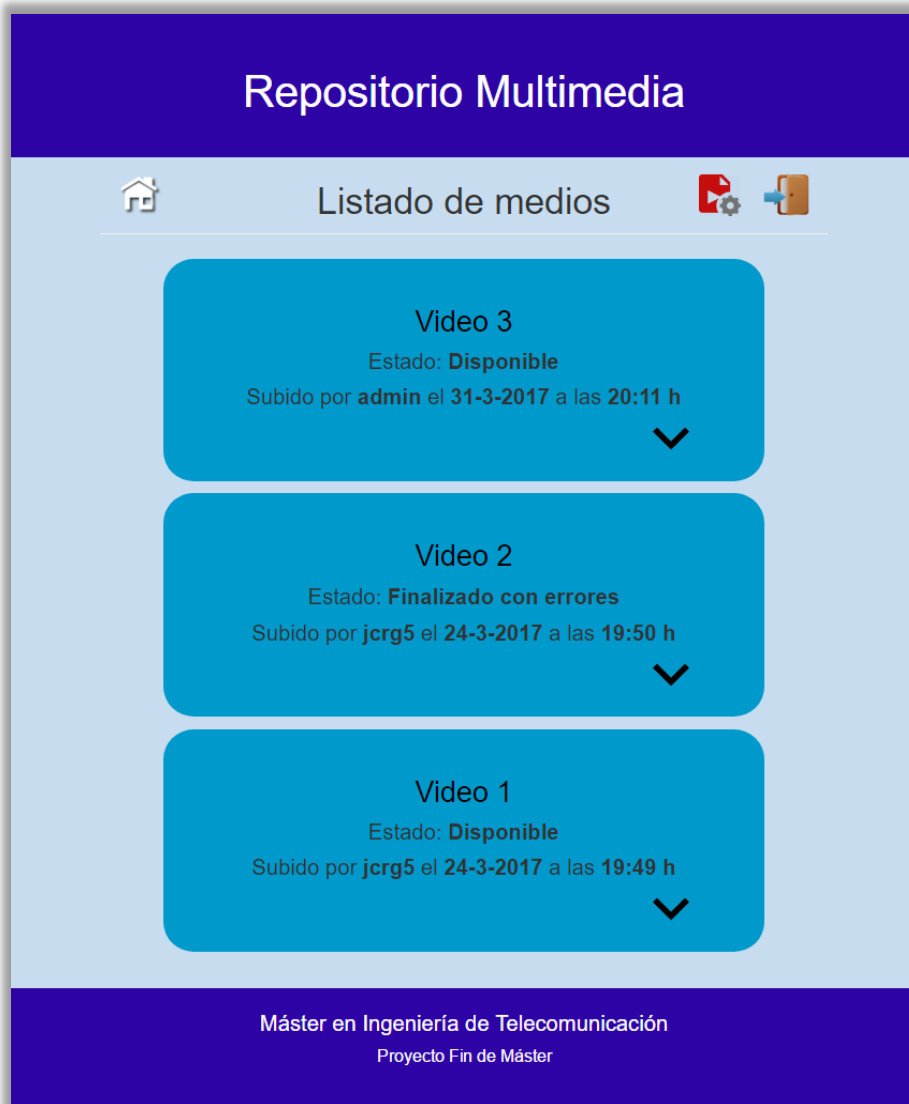






Figura 23. Vista web del repositorio multimedia: listado de medios.

Cada medio contiene, en una lista desplegable, los recursos multimedia asociados. Dependiendo del estado en el que se encuentre el recurso, cambiará el icono del mismo, presentándose los siguientes casos:

- Esperando conversión 
- Procesando 
- Disponible 
- Error en la conversión 



Con los estados y su correspondiente icono es posible monitorizar el estado de la conversión en tiempo real, como se describirá en detalle más adelante.

En la Figura 24 se muestra, como ejemplo, la lista de recursos asociada a un medio cualquiera. Como es de esperar, existe un recurso multimedia para cada uno de los perfiles definidos en el sistema.



Figura 24. Lista de recursos asociada a un determinado medio.

Solo cuando el estado del recurso sea 'Disponible', el icono asociado contará con la funcionalidad para reproducir el contenido.

Al solicitar la reproducción de un recurso, aparece el reproductor multimedia tras su correspondiente efecto de entrada, quedando en una capa emergente, sobrepuesta al listado de medios. El reproductor, mostrado en la Figura 25, está implementado con el complemento *Html5 LightBox Free* [38]. Dispone de funcionalidad básica como reproducir, pausa, control de volumen, ver en pantalla completa y descarga del archivo. Para finalizar la reproducción basta con hacer clic sobre la equis superior derecha, o sobre cualquier punto del área sombreada externa.

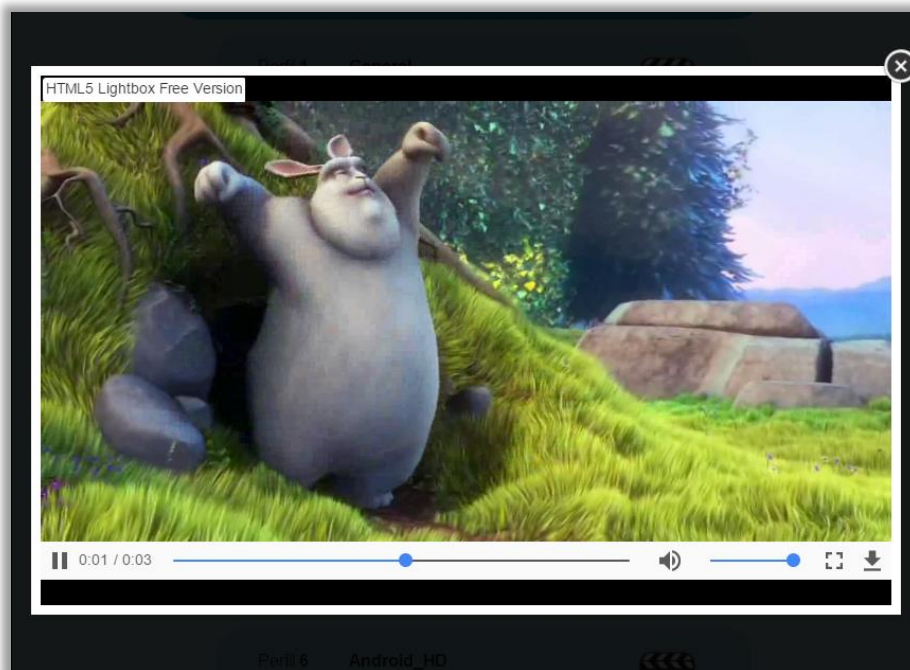


Figura 25. Aspecto del reproductor multimedia.

Además de visualizar los contenidos, el repositorio permite ver los diferentes perfiles multimedia definidos para la conversión. Para mostrar la lista de perfiles basta con hacer clic sobre el icono rojo del encabezado de página. Como ya se ha explicado previamente, su ubicación depende de la versión de la interfaz. La Figura 26, correspondiente a la versión sin sesión, muestra el aspecto inicial del listado.

Se presenta en formato comprimido, pudiendo ampliar cada elemento haciendo clic sobre el icono con el signo más (+). Al hacerlo, la vista alcanza el tamaño mostrado en la Figura 27. Ahora aparecen, junto a una breve descripción, todos los parámetros técnicos asociados al perfil.



Figura 26. Vista web del listado de perfiles multimedia.



Figura 27. Vista web de un perfil multimedia extendido.

Detalles de implementación

Una vez presentado el aspecto y funcionalidad que ofrece la interfaz web, es interesante conocer qué elementos intervienen en la implementación.

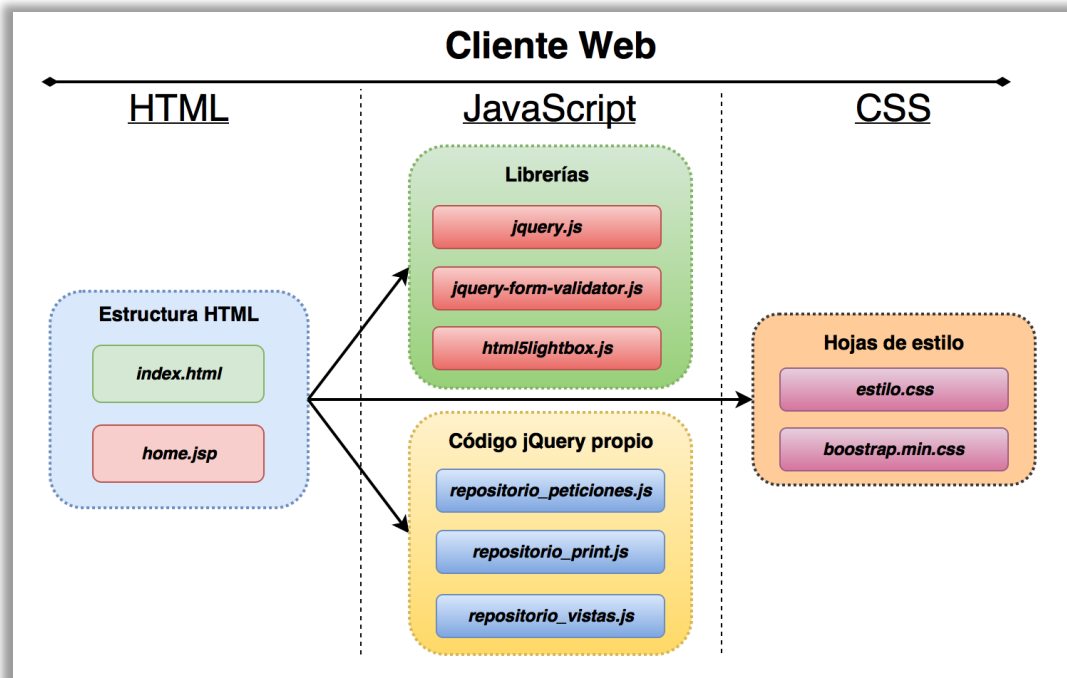


Figura 28. Principales archivos implicados en la implementación del cliente web.

La Figura 28 muestra los principales archivos implicados en el cliente web. Se distinguen, en primer lugar, las dos versiones del repositorio. Pese a presentar formatos diferentes, ambos contienen código *HTML* que define la estructura de la página. En segundo lugar, se distinguen los archivos *JavaScript*. Son los responsables de generar contenido dinámico, ofrecer efectos visuales, así como realizar la comunicación con los servicios *REST*, entre otros. Se distinguen dos grupos de archivos *JavaScript*: arriba las librerías, archivos de fuentes externas como, necesarias para el funcionamiento de algunos elementos agregados a la aplicación; abajo, los archivos propios, desarrollados por el autor de este trabajo. En último lugar están las hojas de estilo. Con ellas se consigue dotar a la página de la apariencia deseada, además de un comportamiento adaptativo para cualquier tamaño de pantalla gracias al uso del *framework Bootstrap* [39].

Una de las librerías JavaScript utilizadas es *jQuery* [40]. Este *framework* permite simplificar la manera de interactuar con los documentos *HTML*, manipular el árbol *DOM* (*Document Object Model*), manejar eventos, desarrollar animaciones y agregar



interacción con la técnica *AJAX (Asynchronous JavaScript And XML)* a páginas web [41]. Todos los complementos utilizados en el desarrollo de la interfaz web, como *jQuery Form Validator*, *Html5 Light Box* o *Bootstrap* requieren de esta librería para funcionar. Los archivos JavaScript propios también utilizan esta librería y su sintaxis. A continuación, se describe la funcionalidad básica de cada uno de ellos:

- *repositorio_peticones.js*

Este archivo contiene las funciones que realizan las peticiones *REST* al servidor. Gestiona el modo de actuar dependiendo del estado de las peticiones. Si los datos se reciben correctamente, delega en las funciones de *repositorio_print.js* la tarea de generar el contenido, para mostrarlos cuando estén disponibles.

Además, incorpora el algoritmo de actualización dinámica de la vista web para monitorizar el proceso de conversión, descrito más adelante.

- *repositorio_print.js*

Incluye las funciones necesarias para generar y agregar contenido dinámico a la vista, a partir de los datos obtenidos del servidor. Otra de sus funcionalidades es la adaptación de los valores recibidos, a los valores mostrados en la interfaz, ya que por conveniencia pueden no ser iguales. Por ejemplo, cuando el estado de un medio en base de datos es 'OK', en la interfaz se muestra como 'Disponible'.

- *repositorio_vistas.js*

En este archivo se incluyen las funciones necesarias para mostrar u ocultar contenido dinámico. Por ejemplo, la lista de recursos de un determinado medio, o los detalles de un determinado perfil. Cabe mencionar que cada vez que se solicita una vista, no se muestra el contenido oculto, sino que se vuelven a solicitar los datos al servidor, generando de nuevo el contenido para disponer de información actualizada.

Monitorización de la conversión en tiempo real

El cliente incorpora un mecanismo de actualización de la interfaz web sin necesidad de recargar la página para monitorizar la conversión de un medio recién subido.

Los datos que se desean monitorizar son los estados de conversión de cada recurso multimedia, así como el estado del medio en conjunto. El estado es el modo de saber si un determinado recurso está disponible o no para su reproducción.

En la Figura 29 se representa el diagrama de flujo correspondiente al algoritmo de monitorización de los estados. Se distinguen dos ramas, puesto que la monitorización de medios y recursos multimedia se realiza de forma independiente. Aunque la relación entre los estados de ambas entidades está claramente ligada, el modo en que se muestran los datos en la interfaz requiere que sea así. Un usuario puede realizar el seguimiento de la conversión con o sin la lista de recursos multimedia visible, necesitando de la rama I, o de ambas respectivamente.

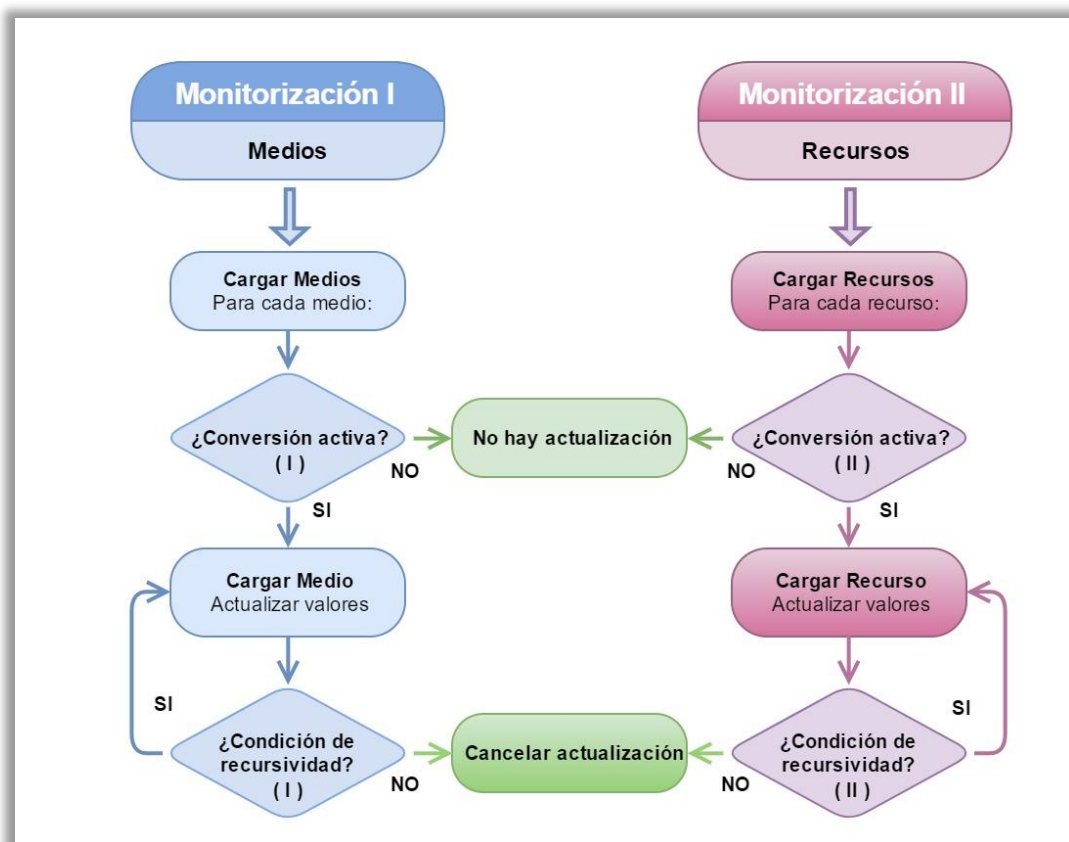


Figura 29. Diagrama de flujo del algoritmo de monitorización.

Ambos ramas o procesos de actualización de valores siguen el mismo patrón. Cuando se recibe el listado de medios o recursos del servidor, se comprueba si alguno de ellos presenta un estado considerado *activo*. Es aquí donde se produce la primera diferencia de criterio para cada rama. Se considera un medio activo aquel que tenga el estado ‘*PROCESANDO*’. Sin embargo, un recurso puede estar activo, no solo cuando se cumple la condición anterior, sino cuando se encuentra esperando, es decir, con el estado ‘*EN_COLA*’. Si se cumplen estas condiciones se inicia el proceso de actualización periódica.

El proceso de actualización en ambas ramas es una función recursiva que solicita datos periódicamente al servidor, actualizando los valores de la interfaz si es necesario. Las condiciones de recursividad para continuar con el chequeo periódico varían en cada rama. Sin embargo, existe una condición común: no se deben superar un número máximo de peticiones. Esta medida de seguridad previene que la llamada recursiva se convierta en un bucle infinito en caso de una mala consistencia de datos, provocada por un fallo del servidor al realizar la conversión, por ejemplo.

Las condiciones de recursividad para continuar actualizando un medio son:

- El listado de medios debe mantenerse visible. En el momento en que se actualiza manualmente el mismo, o se navega hacia otro listado, se incumple esta condición. Para ello, se comparan las fechas exactas de cada petición.
- No se debe superar el número máximo de peticiones preestablecido.

En cambio, las condiciones para continuar actualizando un recurso son:

- La lista de recursos debe estar visible. En el momento que se contrae la lista se cancela el proceso.
- No debe haber una petición de medios o de perfiles más reciente a la petición actual de recursos. Esto implica que no se haya actualizado manualmente el listado de medios o se haya navegado al listado de perfiles.
- No se debe superar el número máximo de peticiones de seguridad.

En las siguientes figuras se muestra el aspecto que presenta el repositorio durante un hipotético proceso de conversión.

Tras subir un medio válido se recibe un mensaje de confirmación (Ver Figura 17), junto a un enlace que redirige al repositorio. En la Figura 30 se muestra el momento justo en que comienza la conversión, donde se está procesando el primer recurso multimedia mientras el resto esperan su turno. Como ya se ha comentado anteriormente, el icono asociado al recurso refleja el estado del mismo. Cuando un recurso está siendo procesado, el icono gira periódicamente en sentido horario. El resto permanecen estáticos, ya que la conversión se realiza en serie.



Figura 30. Monitorización de la conversión (I).

Durante todo el proceso de conversión se aprecia que el estado global del medio es 'Procesando'. Así será hasta que finalice la conversión, independientemente del resultado.

La Figura 31 muestra una evolución en el tiempo de la figura anterior, donde el primer recurso ya ha sido procesado satisfactoriamente, el segundo ha presentado un error durante su conversión y el tercero se está procesando.



Figura 31. Monitorización de la conversión (II).

En la Figura 32 se muestra el aspecto del repositorio al finalizar la conversión global del medio, en el hipotético caso de que se haya producido un error en la conversión individual de uno de sus recursos. Aunque cinco de los seis medios estén disponibles y listos para su reproducción, el estado global del medio en la interfaz es ‘Finalizado con errores’.

En cambio, si todas las conversiones individuales son correctas, el resultado sería el mostrado en la Figura 33. Donde, ahora sí, el estado global es ‘Disponible’.



Figura 32. Monitorización de la conversión (III).



Figura 33. Monitorización de la conversión (IV).

4.7. Cliente móvil

Llegados a este punto solo queda desarrollar la aplicación que permita a un dispositivo móvil acceder al repositorio multimedia. Pero antes, se van a exponer los diferentes tipos de aplicaciones móviles, para después elegir e implementar la alternativa que mejor se adapte a los requisitos del sistema.

4.7.1. Tipos de aplicaciones móviles

Actualmente existen tres tipos de aplicaciones móviles. Cada una de ellas posee un diferente nivel de integración con el dispositivo, así como diferente grado de complejidad en cuanto a desarrollo y alcance multiplataforma.

A continuación, se enumeran estos tres modelos con sus correspondientes características, ventajas y desventajas [42] [43].

- **Aplicaciones nativas.**

Una aplicación nativa es la que se desarrolla de forma específica para un determinado sistema operativo o SDK (*Software Development Kit*). Cada una de las plataformas como *Android*, *iOS* o *Windows Phone*, tienen un sistema diferente, por lo que es necesario desarrollar una aplicación exclusiva para cada entorno. La Tabla 10 resume las ventajas y desventajas de este primer tipo.

Ventajas	Desventajas
<ul style="list-style-type: none">• Acceso completo al dispositivo• Mejor experiencia de usuario• Pueden ser publicadas en tiendas para su distribución• No requieren conexión a Internet para funcionar (Siempre que la App no consuma servicios externos)	<ul style="list-style-type: none">• Exclusivas para el sistema que han sido desarrolladas• El código del cliente no es reutilizable entre las diferentes plataformas• Tienden a ser más costosas de desarrollar• Requieren de instalación• Necesitan aprobación de las tiendas para ser publicadas

Tabla 10. Ventajas y desventajas de las aplicaciones nativas.

La principal ventaja con respecto a los otros dos tipos, es la posibilidad de acceder a todas las características del hardware del móvil: cámara, *GPS*, agenda, medio de almacenamiento, lista de contactos, linterna, etc. Esto dota a las aplicaciones de un mayor potencial, ofreciendo una mejor experiencia al usuario. En definitiva, consiguen un funcionamiento muy estable y fluido.

Como principal desventaja destaca el mayor coste de desarrollo, debido precisamente a la necesidad de utilizar un lenguaje de programación diferente para cada plataforma a la que se quiera dar servicio.

▪ **Aplicaciones web.**

Este tipo de aplicaciones, conocidas como *Web Apps*, no son más que una adaptación de una web tradicional a formato móvil. Suelen estar basadas en plantillas adaptativas, utilizando lenguajes web universales como *HTML*, *Javascript* y *CSS*. De este modo, el mismo código base es válido para diferentes dispositivos sin necesidad de crear varias aplicaciones.

Estas aplicaciones se ejecutan dentro del propio navegador web del dispositivo, por lo que no requieren instalación. Por ello pueden no ser consideradas como aplicaciones al uso, reduciendo el tamaño que ocupan en el dispositivo a un simple acceso directo con la *URL* donde estén alojadas. En la Tabla 11 se resumen las ventajas y desventajas este tipo de aplicaciones.

Ventajas	Desventajas
<ul style="list-style-type: none"> • El código base es reutilizable para múltiples plataformas • Coste de creación mínimo • No es necesaria una aprobación externa para su publicación • Pueden reutilizarse plantillas adaptativas prediseñadas • Mayor número de plataformas soportadas 	<ul style="list-style-type: none"> • Requiere de conexión a Internet • Acceso muy limitado al hardware del dispositivo • Menor rendimiento y peor experiencia de usuario comparado con las aplicaciones nativas • Requiere de mayor esfuerzo en promoción y visibilidad

Tabla 11. Ventajas y desventajas de las aplicaciones web.

▪ **Aplicaciones híbridas.**

Una aplicación híbrida es una combinación de las dos anteriores. Se desarrollan con lenguajes propios de las aplicaciones web, lo que permite un uso multiplataforma. Además, tienen acceso a gran parte del hardware del dispositivo, ya que se ejecutan dentro de un componente nativo: el *WebView*.

A diferencia de las aplicaciones web, pueden ser compiladas para cada plataforma, generando un archivo de instalación independiente para cada una. Esto permite publicar la aplicación en las tiendas de aplicaciones y distribuirlas como nativas.

En la Tabla 12 se muestra un resumen de las ventajas y desventajas de este tipo de aplicación.

Ventajas	Desventajas
<ul style="list-style-type: none">• Mismo código base para múltiples plataformas• Instalación nativa, basada en código <i>HTML</i>, <i>CSS</i> y <i>JavaScript</i>• Acceso a parte del hardware del dispositivo• Menor coste de desarrollo que una aplicación nativa• Es posible distribuirlas en las tiendas de aplicaciones	<ul style="list-style-type: none">• Experiencia de usuario más propia de aplicación web que de aplicación nativa• No cuenta con todas las funcionalidades nativas• Requiere de instalación• Diseño visual no siempre relacionado con la plataforma en la que se instala

Tabla 12. Ventajas y desventajas de las aplicaciones híbridas.

Algunos de los *framework* más utilizados para desarrollar aplicaciones híbridas son *PhoneGap* [44] o *Cordova* [45].

4.7.2. Implementación del cliente móvil

Una vez realizada la comparativa entre los diferentes tipos de aplicaciones móviles, se ha optado por implementar los dos tipos de aplicaciones basadas en web.

Aplicación Web

Aprovechando que el cliente web utiliza el *framework Bootstrap*, se han optimizado todas las vistas de la aplicación para poder ser gestionada a través de cualquier dispositivo móvil de forma adaptativa. De este modo, se puede hacer uso de la aplicación completa accediendo mediante *URL* desde el navegador del dispositivo. Sin contar el esfuerzo realizado para conseguir un comportamiento adaptativo, la creación de la aplicación web en el dispositivo se reduce a seguir los pasos mostrados en la Figura 34.

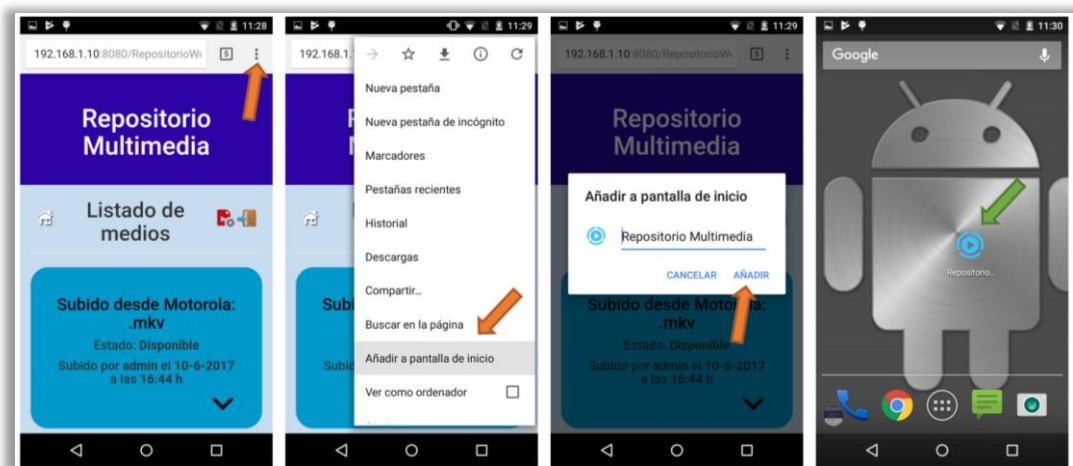


Figura 34. Creación de la aplicación web en un dispositivo Android.

En primer lugar, se accede a la dirección web de la aplicación en el navegador. Posteriormente se añade, mediante una de las opciones del menú, el acceso directo a la pantalla de inicio con el nombre deseado. Realizado esto, ya se dispone del acceso directo en la pantalla de inicio para acceder al repositorio a través del navegador.

Esta versión de cliente ofrece todas las funcionalidades de la aplicación, ya que, en realidad es el propio cliente web, optimizado para adaptarse al formato móvil. Por consiguiente, además de navegar y reproducir contenidos, con esta aplicación es posible iniciar sesión o darse de alta, para posteriormente poder subir medios al repositorio.

Aplicación híbrida

Además de la aplicación web, se ha desarrollado una aplicación híbrida. Este tipo de aplicación es el que mejor se adapta a los requisitos de este trabajo, pues ofrece mejor experiencia al usuario, mayor integración con el dispositivo, servicio multiplataforma y algunas de las ventajas de las aplicaciones nativas como el acceso a buena parte del hardware. Todo ello reutilizando el código del cliente web original, lo que reduce considerablemente el coste de implementación.

El proceso de maquetación de la aplicación web original en la aplicación híbrida consta de dos pasos.

1) Adaptación del código de origen.

La maquetación requiere partir de una aplicación basado únicamente en *HTML*, *CSS* y *JavaScript*. Para ello, se crea en el *IDE* un nuevo proyecto, esta vez de tipo *HTML5/JS Application*, con nombre *RepositorioWeb_Basic*. En él, se incluye una copia de los archivos *front-end* asociados al repositorio. Es necesario excluir el *plugin HTML5 LightBox Free*, dedicado a la reproducción de video, ya que no es compatible con el *framework*. De este modo, la estructura del nuevo proyecto queda reflejada en la Figura 35.

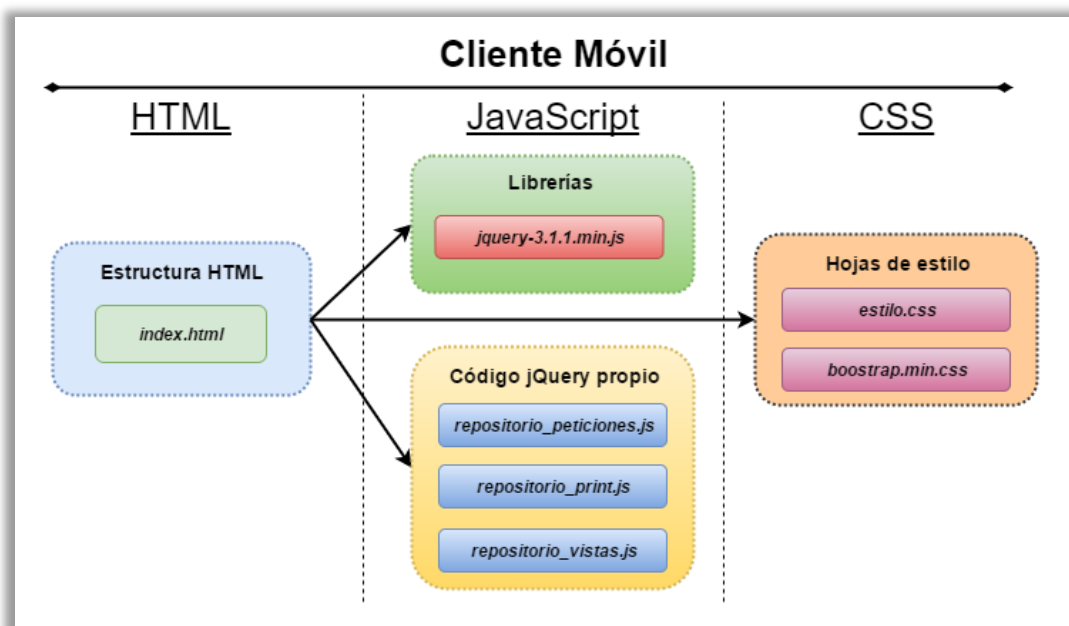


Figura 35. Archivos reutilizados para la implementación del cliente móvil.

La no incorporación de todo el código cliente del proyecto original queda justificada por los requisitos del proyecto. Como se ha establecido desde un principio, al cliente móvil no se le exige la funcionalidad plena del cliente web, sino que se limite a mostrar y reproducir los contenidos del repositorio. Esto implica que la aplicación híbrida carece de la herramienta de publicación. De este modo, solo es necesaria una vista: la correspondiente a la versión sin sesión del repositorio, definida en *index.html*.

La nueva vista precisa, en principio, de una adaptación mínima. El único cambio necesario en *index.html* es el encabezado de la página. Desaparece el icono asociado al inicio de sesión, siendo el resultado el mostrado en la Figura 36. Ahora solo están disponibles los iconos para visualizar el listado de medios y el de perfiles, a diferencia del original, mostrado en la Figura 20.

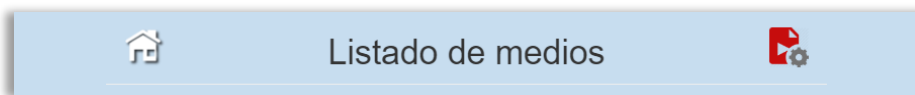


Figura 36. Encabezado de página del cliente móvil.

El siguiente cambio necesario es a nivel interno. Al separar el cliente a una aplicación independiente, es necesario configurar la dirección a la que se realizan las peticiones, ya que los servicios *REST* forman parte de una aplicación externa.

Es necesario definir, por tanto, en el archivo *repositorio_peticiones.js* la nueva ubicación del servicio (Dirección IP y puerto). Más adelante, en el capítulo dedicado a experimentación se describe la arquitectura y el direccionamiento empleado para probar el sistema en su conjunto.

Por último, se eliminan las referencias al reproductor *HTML5 LightBox Free*, delegando la reproducción de los contenidos al propio dispositivo.

Cuando se han realizado todos estos cambios y se comprueba en el IDE que la nueva aplicación funciona, ya se dan las condiciones para realizar la maquetación.

2) Maquetación con un *framework* multiplataforma.

De entre los *frameworks* disponibles para crear la aplicación híbrida, se ha elegido utilizar tanto *Phonegap* como *Cordova*.

PhoneGap dispone de una aplicación de escritorio, denominada *PhoneGap Desktop* para gestionar y simular proyectos directamente en el dispositivo, y una herramienta online para compilar el código, generando los archivos instalables para cada plataforma. Como limitación, decir que *PhoneGap Desktop* no permite visualizar ni editar los archivos del proyecto, lo que exige un editor externo.

Cordova, en cambio, no dispone de entorno gráfico, pero va a permitir agregar mediante consola los *plugins* necesarios para ampliar la funcionalidad de la aplicación.

Comenzando con la maquetación, el primer paso es crear un proyecto *PhoneGap* en blanco en su interfaz gráfica, denominado *RepositorioAPP*. Después, se copia el código mencionado en la Figura 35 a este nuevo proyecto. Tal como muestra la Figura 37, la nueva aplicación está almacenada en el path: “C:\PhoneGapProjects\RepositorioAPP”, y puede testearse en la dirección “<http://192.168.1.10:3000>”.

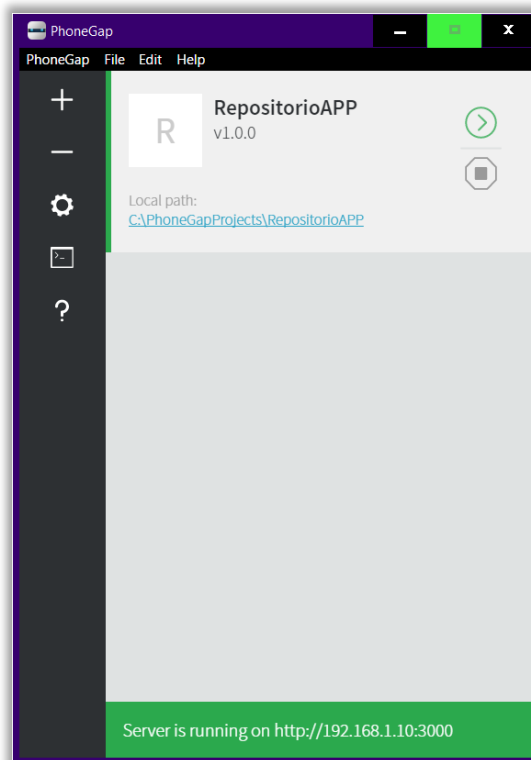
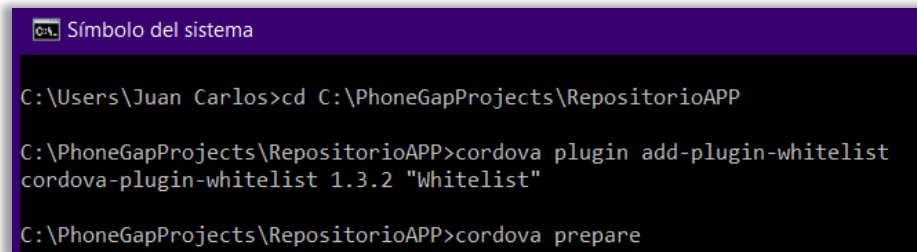


Figura 37. Aplicación híbrida creada en el *framework* *PhoneGap Desktop*.

Con la configuración por defecto, la aplicación no es capaz de realizar peticiones HTTP. Es en este punto donde es necesario instalar el complemento *cordova-plugin-whitelist*, mediante los comandos definidos en la Figura 38. Cabe destacar que es necesario tener instalado *Apache Cordova* y ejecutar el comando sobre la ubicación del proyecto *PhoneGap*.



```

C:\Users\Juan Carlos>cd C:\PhoneGapProjects\RepositorioAPP

C:\PhoneGapProjects\RepositorioAPP>cordova plugin add-plugin-whitelist
cordova-plugin-whitelist 1.3.2 "Whitelist"

C:\PhoneGapProjects\RepositorioAPP>cordova prepare
  
```

Figura 38. Comandos necesarios para instalar el plugin *whitelist*.

El plugin *whitelist* actúa de filtro. Define a qué *URLs* puede tener acceso la aplicación. Tras su instalación, permite acceder a cualquier *URL*, pero por motivos de seguridad es conveniente definir únicamente el dominio al que la aplicación necesita acceder, en este caso, la dirección web de la aplicación en el servidor local. Para cambiar la configuración hay que editar el archivo *config.xml*, ubicado en el directorio raíz del proyecto [46]. En la Figura 39 se muestra cómo queda finalmente este archivo tras realizar los cambios.



```

1 <?xml version='1.0' encoding='utf-8'?>
2 <widget id="com.tfm.RepositorioAPP" version="1.0.0" xmlns="http://www.w3.org/ns/widgets"
3   <name>RepositorioAPP</name>
4   <description>
5     Repositorio Multimedia para aplicaciones móviles.
6   </description>
7   <author email="jcr55@alu.ua.es" href="http://eps.ua.es">
8     Juan Carlos Rubio
9   </author>
10  <content src="index.html" />
11  <access origin="http://192.168.1.10:8080/RepositorioWeb/*" />
12  <plugin name="cordova-plugin-whitelist" spec="~1.3.2" />
13 </widget>
14
  
```

Figura 39. Archivo de configuración del proyecto *PhoneGap*.

Como se aprecia en la línea 11, la *URL* permitida queda definida en el atributo *origin* de la etiqueta *access*.

El siguiente paso para la obtención de la aplicación híbrida es compilar el código. Esta tarea se realiza con *Phonegap Build*, uno de los servicios en la nube de *Adobe*. Puede compilar la aplicación para *Android*, *iOS* y *Windows Phone* a partir del código generado en la fase anterior. Solo requiere tener una cuenta activa y subir el código en formato *ZIP*. En el caso de *iOS*, requiere además de un certificado de desarrollador, concedido por *Apple.inc* [47].

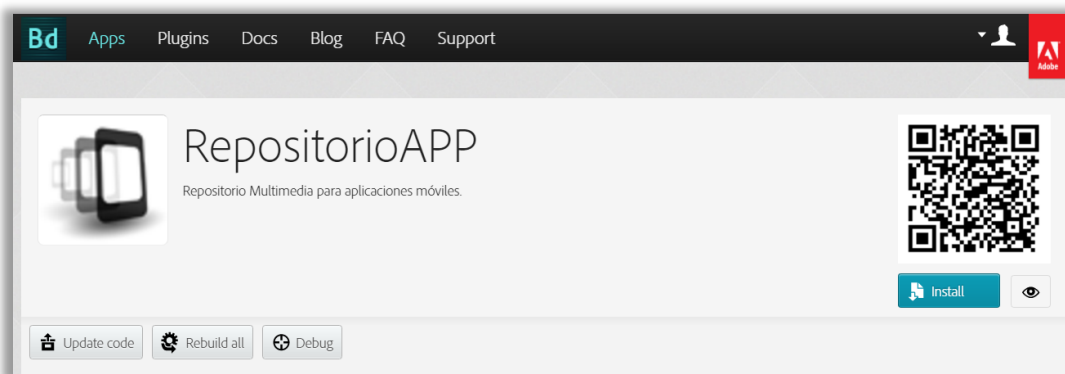


Figura 40. Interfaz web del servicio Adobe PhoneGap build.

Si la compilación es correcta, el servicio permite la descarga de los archivos de instalación para cada plataforma. Se proporciona un código QR para facilitar la descarga en el propio dispositivo, ahorrando el proceso de descarga y transferencia mediante USB.

En la Figura 41 se muestra un resumen del proceso de instalación, así como el aspecto final de la aplicación en un dispositivo Android.



Figura 41. Instalación de la aplicación híbrida en Android.

En el próximo capítulo, destinado a experimentación, se mostrará con mayor detalle el funcionamiento de la aplicación híbrida en diferentes dispositivos.



Capítulo 5. Experimentación

Una vez implementado el sistema, llega el momento de realizar las pruebas necesarias para evaluar su funcionamiento. En este capítulo se describe el proceso para poner en funcionamiento el repositorio, así como los resultados obtenidos al utilizarlo en los diferentes entornos donde se ha probado.

5.1 Despliegue de la aplicación

Durante el proceso de desarrollo, se ha llevado a cabo el testeo y depuración tanto de los servicios como del cliente web en la propia máquina donde está instalado el *IDE*. Esto es posible porque el *IDE*, *NetBeans* en este caso, cuenta con su propio servidor web. Al ejecutar la aplicación, se reconstruye automáticamente a medida que se realizan cambios. Esto permite depurar y corregir el código de forma ágil, ya que los cambios se reflejan inmediatamente.

En cambio, cuando el proceso de desarrollo finaliza, es el momento de trasladar la aplicación de servidor para que su alcance no se limite a la máquina local donde se ejecuta. Entre las alternativas planteadas para ubicar la aplicación están, o bien Internet o bien en red local. Las ventajas que ofrece Internet en cuanto a disponibilidad y acceso desde cualquier lugar, suponen a la vez un riesgo de seguridad y cierto grado de incompatibilidad para esta aplicación en cuestión. La publicación de videos no está limitada, y el almacenamiento de ficheros se realiza en el propio servidor, por lo que es un riesgo dar acceso global a esta máquina. Respecto a la incompatibilidad; si la aplicación se aloja en un servidor externo, éste debe cumplir con una serie de requisitos. Debe ser un entorno *Windows* y tener instalada la herramienta *FFmpeg*, además de soportar aplicaciones *Java* y el servicio de base de datos *MySQL*. Por lo general, estas condiciones no se cumplen para un alojamiento básico o económico, ya que la mayoría de estos servidores en Internet están basados en *Linux*.

El despliegue de la aplicación en una red local es la opción más sencilla y segura para llevar a cabo la experimentación, pese a contar con el inconveniente de un alcance limitado.

En la Figura 42 se muestra el diagrama de red utilizado para llevar a cabo la experimentación. A continuación, se procede a explicar la topología y los elementos que la componen.

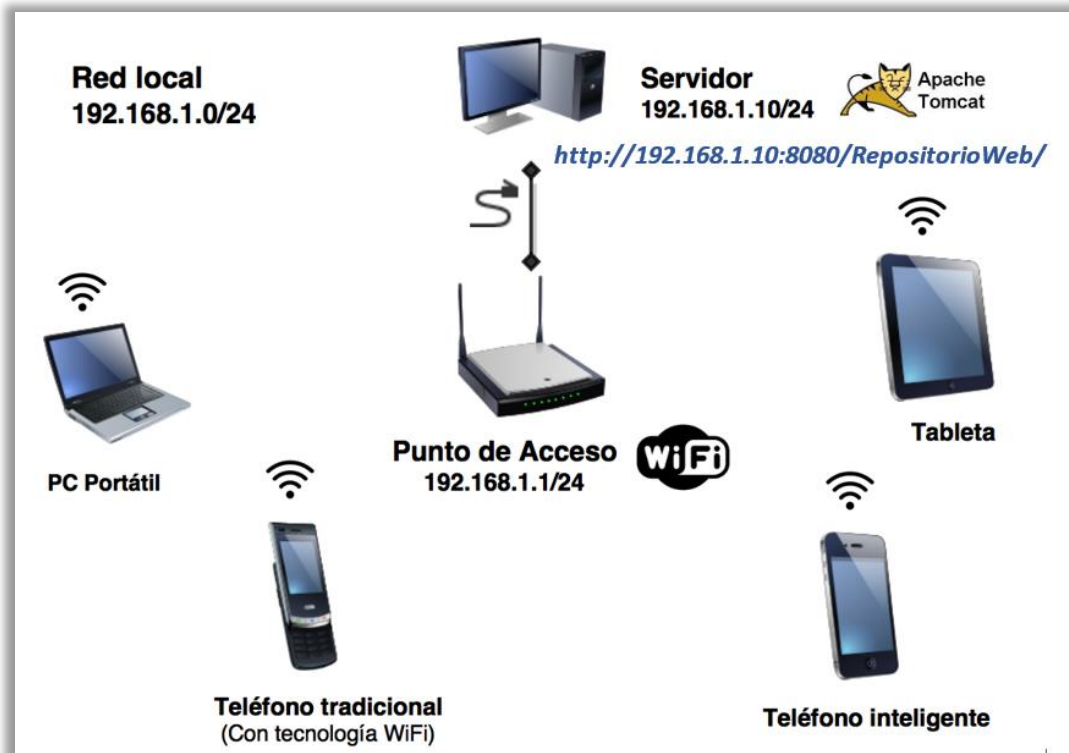


Figura 42. Diagrama de red utilizado para llevar a cabo la experimentación.

En primer lugar, se ha utilizado un *router* doméstico con tecnología *Wi-Fi* para crear la red. Se ha empleado el direccionamiento privado 192.168.1.0/24, correspondiente a una máscara 255.255.255.0. Esto permite un total de 254 equipos en la red, incluyendo el propio servidor.

Desde la interfaz web de configuración del *router* se ha configurado el *DHCP* (*Dynamic Host Configuration Protocol*) para que reserve, mediante asociación *MAC* (*Media Access Control*), la IP 192.168.1.10 al servidor. De este modo, se garantiza que el equipo disponga de esta IP de forma permanente, y que no sea cedida a ningún otro dispositivo en la red.

Se ha instalado en el servidor la aplicación *Xampp*. Además de la herramienta de gestión de la base de datos *PhpMyAdmin*, incluye Apache Tomcat, el contenedor web donde se va a alojar la aplicación *Java* [16].



Tomcat requiere configurar el direccionamiento, ya que por defecto la dirección del conector es *localhost*. Esto se consigue editando el fichero *server.xml*, ubicado dentro de la carpeta *config* donde está instalado *Tomcat*. Cuando este cambio surte efecto, el servidor ya está accesible a través de cualquier dispositivo de la red local. En ocasiones, puede ser necesario configurar el cortafuegos para permitir la comunicación.

El último paso es desplegar la aplicación. Para ello, se ha utilizado la web de gestión de *Tomcat*, ubicada en la dirección y puerto definidos por el conector en *server.xml*: <http://192.168.1.10:8080>. El archivo *.war* a desplegar está ubicado en el directorio *dist* del proyecto *Netbeans*, correspondiente a la última compilación del proyecto *RepositorioWeb*. Al realizar el despliegue, la aplicación queda disponible a través de la URL: <http://192.168.1.10:8080/RepositorioWeb/>.

5.2 Cliente web

En este apartado se describe cuál ha sido la experiencia al utilizar la aplicación en diferentes navegadores de un dispositivo cliente.

Partiendo de la base de que el navegador de referencia sobre el que se ha realizado la implementación ha sido *Google Chrome*, hay que analizar además el comportamiento en otros navegadores para evaluar e intentar corregir posibles problemas de compatibilidad.

- ***Google Chrome***

Es el navegador utilizado en el proceso de desarrollo. El funcionamiento es correcto y la apariencia es la deseada.

- ***Mozilla Firefox***

El complemento *HTML5 Lightbox Free* encargado de la reproducción de medios no se comporta correctamente. Al hacer clic sobre un medio para su reproducción, aparece la ventana contenedora, pero en lugar de ocupar el tamaño que le corresponde, se estrecha verticalmente hasta el punto de que el video se muestra en miniatura.

Pese a intentar corregir este comportamiento, no se ha podido solucionar el problema. Queda pendiente hacer hincapié en este aspecto o plantearse el cambio del *plugin* dedicado a la reproducción para un futuro.

▪ **Internet Explorer**

Este navegador es el que más problemas presenta:

- El plugin *HTML5 Lightbox Free* no funciona correctamente. La vista emergente aparece, pero en lugar de reproducir el contenido en su interior, se muestra un mensaje por parte del navegador preguntando qué hacer con el contenido. Permite *Abrir*, *Guardar*, o *Cancelar*. Al seleccionar la opción *Abrir*, se reproduce el medio con el reproductor por defecto del sistema operativo.
- No muestra bien las fechas. La conversión de formato con el método *parse* de la clase *Date* en *JavaScript* no es correcta, genera valores nulos. Como resultado, se muestran el valor '*NaN*' en lugar de los diferentes campos en los que está compuesta la fecha de publicación de los medios.
- Los efectos y animaciones de la interfaz dejan mucho que desear. Al expandir los perfiles, por ejemplo, aparecen unos contornos irregulares durante la transición.
- Al intentar publicar medios, se ha producido un error al cargar el archivo. Se ha detectado que, al subir el archivo, el navegador incluye la ruta completa en el campo *filename* de la cabecera. En consecuencia, en *back-end*, al intentar guardar el archivo en el directorio temporal, se produce un error. Para solucionar este problema se evalúa el *filename* del archivo en *back-end* antes de utilizarlo, reduciendo la ruta completa al nombre del archivo si es necesario.
- Es necesario recargar la página para actualizar el listado de medios. Todo indica a que utiliza el *json* de la última petición, almacenado en *caché*, como fuente para mostrar los contenidos. Sin embargo, al realizar la monitorización de la conversión, sí actualiza los valores del listado.

▪ **Microsoft Edge**

Se ha detectado el mismo problema que en *Internet Explorer* al intentar publicar un medio. Ambos navegadores agregan la ruta completa del archivo en el campo *filename* de la cabecera. Por lo demás, se obtienen unos resultados muy parecidos a Chrome, tanto en funcionalidad, comportamiento y estilo.

- **Safari para Windows**

Este navegador, propiedad de Apple, dispone de una versión para Windows, en la que se ha realizado la experimentación. Presenta un par de problemas, comunes a *Internet Explorer*. No permite reproducir el archivo en el propio navegador. Ofrece descargarlo para visualizarlo posteriormente en un reproductor externo. Además, las fechas no se muestran bien, indicativo de que el *Date Parser* utilizado en *JavaScript* no es compatible.

- **Opera**

En este navegador no se ha encontrado ninguna incompatibilidad. Todos los módulos funcionan correctamente.

Comparativa

En la Tabla 13 se muestra un resumen de los resultados de compatibilidad tras realizar la experimentación con el cliente web.

	Reproducción embebida	Publicación	<i>Date Parser</i>	Cabecera Campo <i>filename</i>	Caché
Chrome	✓	✓	✓	<i>filename</i>	✓
Firefox	~	✓	✓	<i>filename</i>	✓
Explorer	×	✓	×	<i>fullpath</i>	×
Edge	✓	✓	✓	<i>fullpath</i>	✓
Safari	×	✓	×	<i>filename</i>	✓
Opera	✓	✓	✓	<i>filename</i>	✓

Tabla 13. Compatibilidad del repositorio con los principales navegadores web.

De las incompatibilidades, la reproducción es la más importante. Se plantea para una versión futura, buscar una alternativa al *plugin HTML5 Light Free*, dedicado a la reproducción de medios. El problema con el *Date Parser* se puede atajar, por ejemplo, enviando la fecha formateada directamente en el *json*. Por su parte, el problema de las cabeceras ya se ha solucionado, admitiendo tanto el *filename* como el *fullpath*. En cuanto al mal uso de la caché por parte de *Internet Explorer*, no se considera un problema grave al que dedicar tiempo en este trabajo.

5.3 Cliente móvil

En este apartado se evalúa el funcionamiento del repositorio al ser consumido por diferentes tipos y modelos de dispositivos móviles. Para cada uno se comentan los resultados obtenidos al realizar la experimentación con los dos tipos de aplicaciones móviles: la web y la híbrida.

En la Tabla 14 se presentan los dispositivos empleados para la experimentación junto a sus principales características.

	Tipo	Plataforma	Pantalla
BQ Aquaris M5	<i>Smartphone</i>	<i>Android 6.0.1</i>	<i>1080 x 1920 – 5" – 440 ppi</i>
Motorola Moto G 1ª Generación	<i>Smartphone</i>	<i>Android 5.1</i>	<i>720 x 1280 – 4.5" – 329 ppi</i>
AIRIS PhonePAD 7BG	<i>Tablet</i>	<i>Android 4.2.2</i>	<i>1024 x 600 – 7" – 170 ppi</i>
LG Optimus L7	<i>Smartphone</i>	<i>Android 4.1.2</i>	<i>480 x 800 – 4.3" – 217 ppi</i>
Samsung Wave GT-S8500	<i>Smartphone</i>	<i>Bada 2.0</i>	<i>480 x 800 – 3.3" – 283 ppi</i>

Tabla 14. Dispositivos móviles empleados en la experimentación.

A continuación, se describe el comportamiento del repositorio con cada cliente.

- **Bq Aquaris M5**

De entre todos los dispositivos utilizados es el que mayores prestaciones presenta. Por sus características, está orientado a los perfiles de mayor calidad.

- Aplicación web

Utilizando la aplicación desde el navegador *Google Chrome*, el comportamiento es el esperado. Funcionan todos los componentes del repositorio del mismo modo que en una computadora portátil, pero con el correspondiente ajuste de la vista. La única diferencia está en la reproducción de los contenidos. No se inician automáticamente. Es necesario hacer clic sobre el botón *play* para comenzar la visualización. Este comportamiento está advertido en la propia web del *plugin* de reproducción para las plataformas *Android* e *iOS* [38].

➤ Aplicación híbrida

El comportamiento de la aplicación es estable y fluido. En contraposición a la aplicación móvil, no permite hacer zoom. En la Figura 43 se muestran algunas capturas de la vista. Se puede apreciar de izquierda a derecha: la lista de medios, la lista de recursos de un medio en concreto, y el aspecto de un perfil desplegado mostrando todos sus parámetros.

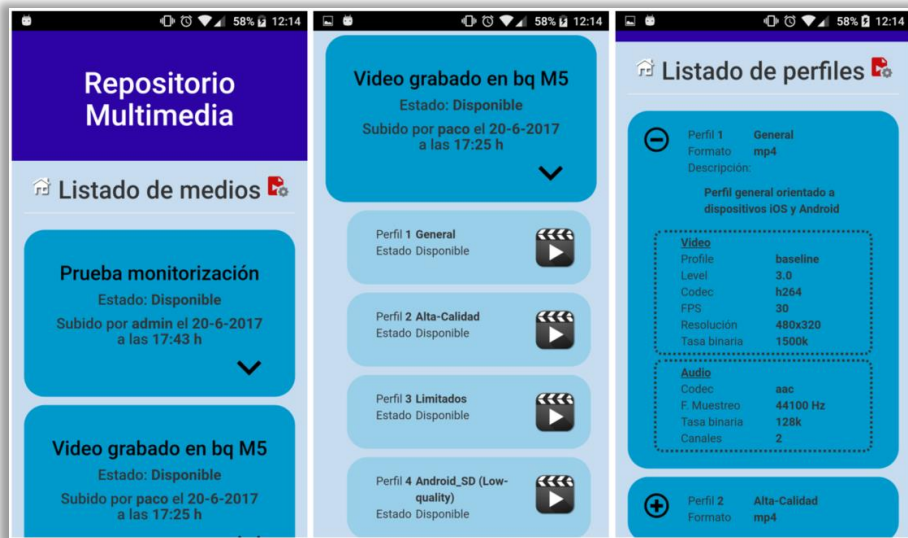


Figura 43. Aplicación híbrida en bq Aquaris M5 (Android).

La reproducción de video, a diferencia de la aplicación web, se realiza directamente en el navegador, tal como muestra la Figura 44. Éste se abre al hacer clic sobre el icono de reproducción de cualquier recurso disponible.

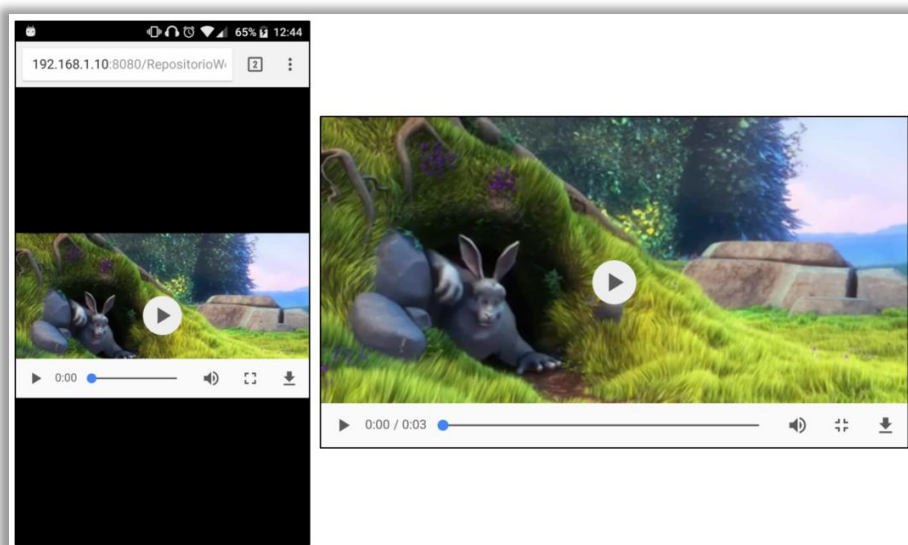


Figura 44. Reproducción en el navegador web de bq M5. Portrait y fullscreen.

- **Motorola Moto G**

En este dispositivo se obtienen unos resultados favorables, al igual que en el *BQ M5*. No existe una diferencia de comportamiento en ninguna de las dos aplicaciones.

- **AIRIS PhonePAD 7BG**

Esta *tablet* presenta mayores limitaciones de hardware y rendimiento respecto a los dispositivos anteriores. Como consecuencia, se ha observado, tanto en la aplicación web como en la híbrida, que no es capaz de reproducir contenidos asociados a perfiles de mayor calidad. Al intentarlo, se produce un error en el servidor. (Ver Figura 45).

```

c:\xampp\catalina_start.bat
Cerrando InputStream...
jun 23, 2017 1:02:13 PM repositorio.controlador.servlets.PasarelaServlet doGet
GRAVE: null
ClientAbortException: java.net.SocketException: Connection reset by peer: socket write error
    at org.apache.catalina.connector.OutputBuffer.realWriteBytes(OutputBuffer.java:407)
    at org.apache.tomcat.util.buf.ByteChunk.append(ByteChunk.java:371)
    at org.apache.catalina.connector.OutputBuffer.writeBytes(OutputBuffer.java:432)
    at org.apache.catalina.connector.OutputBuffer.write(OutputBuffer.java:420)
    at org.apache.catalina.connector.CoyoteOutputStream.write(CoyoteOutputStream.java:91)
    at org.apache.catalina.connector.CoyoteOutputStream.write(CoyoteOutputStream.java:84)
    at repositorio.controlador.servlets.PasarelaServlet.doGet(PasarelaServlet.java:99)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:620)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:727)

```

Figura 45. Error en el servidor al reproducir un perfil no soportado.

- **LG Optimus L7**

Este es un dispositivo muy limitado. Presenta problemas de rendimiento y fluidez en todas las aplicaciones en general. Aun así, se ha conseguido reproducir los perfiles limitados, pero con los siguientes inconvenientes.

- Aplicación web

El *plugin* para la reproducción embebida no funciona. Es necesario abrir el enlace al video en una nueva pestaña para que sea el propio navegador el que lo gestione. Cuando el perfil no está soportado, el propio navegador muestra un mensaje notificando la incompatibilidad.

- Aplicación híbrida

Ha sido necesario dotar a la aplicación híbrida de un tiempo adicional de respuesta para que no se quede colgada mientras realiza las peticiones y genera los contenidos dinámicos.



Cuando este contratiempo ha sido solventado, se ha detectado otro problema adicional. Al intentar reproducir un video, se abre el navegador, pero inmediatamente después se vuelve a la aplicación móvil. Es necesario abrir manualmente el navegador para encontrar la pestaña abierta anteriormente, y poder así realizar la reproducción.

- ***Samsung Wave GT-S8500***

De todos los dispositivos utilizados, éste es el *smarthpone* más antiguo. Se diferencia del resto por utilizar la plataforma *Bada*, un sistema obsoleto actualmente. Como esta plataforma esta en desuso, no entra dentro de la cobertura multiplataforma de *PhoneGap*, de este modo, no cuenta con la aplicación híbrida. Sin embargo, se ha considerado interesante incluir teléfonos antiguos en el estudio, para conocer mejor el grado de compatibilidad de la aplicación.

- Aplicación web

El *plugin* de reproducción no funciona correctamente. Sin embargo, al abrir los contenidos mediante *URL* en una nueva pestaña, se ha observado que el dispositivo es capaz de reproducir los perfiles de gama media y baja. Los de gama alta en cambio, no son soportados, quedando el reproductor en espera constante.

- ***Dispositivos con plataforma iOS y Windows Phone***

El *framework* utilizado para generar la aplicación híbrida ofrece cobertura a estas dos plataformas, pero en el momento de realizar la experimentación no se dispone de ningún dispositivo de este tipo. Queda pendiente para una versión futura la experimentación en dispositivos de estas plataformas.

- ***Dispositivos no inteligentes***

No ha sido posible realizar pruebas con dispositivos tradicionales por no disponer de ninguno con tecnología *Wi-Fi* capaz de conectarse a la red local. Otra alternativa podría ser la de acceder al repositorio a través de una tecnología inalámbrica distinta, como la red de datos móviles. Esto requiere, sin embargo, que el repositorio esté accesible a través de Internet. Se pospone, pues, a un posible trabajo futuro.

Comparativa

Para finalizar, se muestra un resumen de los resultados obtenidos al experimentar con las dos aplicaciones móviles: la aplicación web (Tabla 15), y la aplicación híbrida (Tabla 16).

➤ Aplicación web

	Reproducción embebida	Perfiles permitidos	Publicación	Date Parser	Monitorización
BQ Aquaris M5	✓	Todos	✓	✓	✓
Motorola Moto G	✓	Todos	✓	✓	✓
AIRIS PhonePAD	✓	Limitados	✓	✓	✓
LG Optimus L7	✓	Limitados	✓	✓	✓
Samsung Wave GT-S8500	✗	General y limitados	✗	✗	✗

Tabla 15. Compatibilidad de la aplicación web en cada dispositivo.

Los resultados muestran que, para los dispositivos más modernos, la funcionalidad es completa. Solo ha habido problemas en un dispositivo antiguo, con un sistema operativo obsoleto como es el Samsung Wave, lo que no supone un gran inconveniente. Se demuestra, además, que la compatibilidad de los perfiles es acorde a la gama del dispositivo.

➤ Aplicación híbrida

	Reproducción cómoda	Perfiles permitidos	Date Parser	Monitorización
BQ Aquaris M5	✓	Todos	✓	✓
Motorola Moto G	✓	Todos	✓	✓
AIRIS PhonePAD	✓	Limitados	✓	✓
LG Optimus L7	✗	Limitados	✓	✓

Tabla 16. Compatibilidad de la aplicación híbrida en cada dispositivo.

El comportamiento es parejo al de la aplicación web al tratarse del mismo código base. Sin embargo, a la hora de reproducir videos se ha encontrado, en el caso del LG, un comportamiento “incómodo” a la hora de reproducir los contenidos.

Capítulo 6. Conclusiones y trabajo futuro

6.1. Conclusiones

Llegados a este punto se puede hacer una valoración de los resultados obtenidos. Teniendo en cuenta cuáles eran los objetivos marcados desde un principio, es posible analizar en qué medida se han cumplido.

Desde un punto de vista global, se ha conseguido desarrollar un sistema de publicación de contenidos multimedia con capacidad de transcodificación. Un sistema que ofrece un perfil de codificación adaptado a cada conjunto de dispositivos, reduciendo los problemas de compatibilidad a la hora de reproducir ciertos contenidos.

Analizando los objetivos concretos, se ha propuesto una solución modular para cada una de las tareas que desempeña el sistema. Se ha conseguido implementar una herramienta de publicación con capacidades añadidas, como las de control de acceso y seguridad. Para la conversión, al hacer uso del potente framework *FFmpeg*, se garantiza una gran capacidad de transcodificación, compatible con la mayoría de formatos.

Se han diseñado los correspondientes métodos de persistencia para que los contenidos publicados queden almacenados en el repositorio. Gracias a la tecnología *REST*, se ha creado un conjunto de servicios de alta disponibilidad, con el que gestionar la comunicación cliente-servidor.

Para el desarrollo del cliente, se han utilizado tecnologías basadas exclusivamente en web, ofreciendo una interfaz dinámica, adaptativa y multiplataforma. Al mismo tiempo, esto ha permitido exportar la aplicación mediante un *framework* para generar una aplicación móvil multiplataforma con la que reproducir los contenidos del repositorio.

Una amplia experimentación ha validado la aplicación en la mayoría de los navegadores. Se ha demostrado la compatibilidad de la aplicación móvil en *Android*, sin embargo, queda pendiente la experimentación de la aplicación móvil en plataformas como *iOS* o *Windows Phone*.

6.2. Trabajo futuro

A lo largo del texto se han ido haciendo propuestas de mejora, tanto globales como enfocadas a lo diferentes módulos que componen el sistema. En esta sección se hace una recopilación de las mismas, organizándolas bajo diferentes líneas de trabajo.

1) Migración de la aplicación local a la nube.

Se contempla la posibilidad de abrir los horizontes de la aplicación más allá de la red local. Siguiendo las tendencias de hoy en día, se pueden descentralizar los módulos del repositorio para alojarlos en la nube. Las diferentes propuestas pasan por:

- Dar acceso al servidor para que sea accesible a través de Internet.
- Alojarse el servidor directamente en la nube.
- Utilizar un servicio de base de datos en la nube.
- Utilizar un servicio de almacenamiento de archivos en la nube.

2) Mejoras de diseño.

Entre las diferentes mejoras de diseño planteadas destacan:

- Los mensajes generados por los *servlets* podrían mostrarse en la propia interfaz de la página del cliente que solicita la petición. Las peticiones podrían realizarse mediante tecnología *AJAX*, sin necesidad de abandonar la página a la espera de respuesta.
- Implementación de los métodos *PUT* y *DELETE* en los servicios *REST*.

En una versión futura sería interesante disponer de estos métodos en todos los servicios para poder gestionar completamente los contenidos.

En cuanto al método *DELETE*, cabe mencionar que en una versión más reciente ya se ha implementado el método para poder borrar los medios y recursos del repositorio. Sin embargo, no se ha llegado a integrar en la interfaz web. Se consumen directamente con la herramienta *Restlet-Client*.

- Implementar un *pool* para gestionar las conexiones con base de datos.



Con el modelo de conexión a base de datos actual, las conexiones se abren y se cierran en cada petición. Un *pool* de conexiones aportaría más estabilidad y escalabilidad al servidor.

- Optimizar el proceso de monitorización de la conversión.

En la versión actual, el cliente solicita periódicamente información al servidor cuando se produce la monitorización. Con el objetivo de optimizar este proceso y ganar en eficiencia, sería conveniente sustituir el sistema de monitorización actual por otro en el que sea el servidor el que notifica al cliente cuando ha habido algún cambio. Por ejemplo, utilizando notificaciones *Push*.

- Sería conveniente cambiar el método de persistencia empleado para definir los perfiles multimedia. Se propone integrar los perfiles en la base de datos, para posteriormente añadir un módulo de gestión en el cliente, que permita la creación y edición de los mismos.

3) Ampliación de funcionalidades.

Se proponen, además, algunas ideas para que el repositorio crezca en funcionalidad:

- Incorporar un método de recuperación de contraseña automático.

Esta funcionalidad ha quedado pendiente en la versión actual. Se método de recuperación basado en la creación de contraseñas aleatorias, enviadas al usuario a través de un servidor de correo electrónico.

- Gestión de usuarios en la interfaz web.

En una versión futura podría añadirse un módulo de gestión de cuentas para que los usuarios puedan modificar sus datos de registro.

4) Experimentación.

Ha quedado pendiente, por falta de recursos, la maquetación del cliente móvil a otras plataformas distintas de *Android*. Se plantea como trabajo futuro compilar la aplicación, tanto para *iOS* como para *Windows Phone*, y experimentar con el sistema para verificar el funcionamiento multiplataforma avalado por el *framework* Adobe *PhoneGap*.

Capítulo 7. Bibliografía y referencias

- [1] Ditrendia, «Informe Mobile en España y en el mundo» 2016.
- [2] Depto. Ciencia de la Computación e IA, «Servicios Multimedia para Dispositivos Móviles» 2013.
- [3] «Webopedia: On-premises» [En línea]. <http://www.webopedia.com/TERM/O/on-premises.html>. [Último acceso: Mayo 2017].
- [4] «Streamingmedia. Video: The future of cloud transcoding». [En línea]. <http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/Video-The-Future-of-Cloud-Transcoding-118467.aspx>. [Último acceso: Mayo 2017].
- [5] J. Ozer, «A Cloud Encoding Pricing Comparison» 2016. [En línea]. http://docs.hybrik.com/repo/cloud_pricing_comparison.pdf.
- [6] «Zencoder» [En línea]. <https://zencoder.com/es/>. [Último acceso: Mayo 2017].
- [7] «encoding.com» [En línea]. <https://www.encoding.com/resources/>. [Último acceso: Mayo 2017].
- [8] «Wowza Media System» [En línea]. <https://www.wowza.com/products#>. [Último acceso: Junio 2017].
- [9] «Videocloud» [En línea]. <https://videocloud.com/about>. [Último acceso: Junio 2017].
- [10] «Rightscale» [En línea]. <http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2017-state-cloud-survey>. [Último acceso: Junio 2017].
- [11] «Wikipedia, Java (Lenguaje de programación)» [En línea]. [https://es.wikipedia.org/wiki/Java_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)). [Último acceso: Junio 2017].
- [12] «MySQL». [En línea]. <https://www.mysql.com/>. [Último acceso: Junio 2017].

- [13] «ffmpeg» [En línea]. <https://www.ffmpeg.org/ffmpeg.html>.
[Último acceso: Mayo 2017].
- [14] «Wikipedia, JAX-RS» [En línea]. <https://es.wikipedia.org/wiki/JAX-RS>.
[Último acceso: Juno 2017].
- [15] «Apache Tomcat» [En línea]. <https://tomcat.apache.org/>.
[Último acceso: Juno 2017].
- [16] «Xampp» [En línea]. <https://www.apachefriends.org/es/index.html>.
[Último acceso: 2016 Junio].
- [17] «phpMyAdmin» [En línea]. <https://www.phpmyadmin.net/>.
[Último acceso: Junio 2017].
- [18] «Restlet Client» [En línea]. <https://restlet.com/modules/client/>.
[Último acceso: Junio 2017].
- [19] «PhoneGap Desktop» [En línea]. <http://docs.phonegap.com/references/desktop-app/>. [Último acceso: Junio 2017].
- [20] «PhoneGap Build» [En línea]. <https://build.phonegap.com/>.
[Último acceso: Junio 2017].
- [21] «Wikipedia, Modelo-Vista-Controlador» [En línea].
<https://es.wikipedia.org/wiki/Modelo-vista-controlador>.
[Último acceso: Mayo 2017].
- [22] M. Flower, «martinflower.com» [En línea].
<https://martinfowler.com/eaDev/uiArchs.html>. [Último acceso: Mayo 2017].
- [23] «MediaCoder» 28 Abril 2008. [En línea]. <http://blog.mediacoderhq.com/h264-profiles-and-levels/>. [Último acceso: Mayo 2017].
- [24] «javaTpoint» [En línea]. <https://www.javatpoint.com/servlet-http-session-login-and-logout-example>. [Último acceso: Mayo 2017].
- [25] «javaTpoint» [En línea]. <https://www.javatpoint.com/servlet-filter>.
[Último acceso: Mayo 2017].



- [26] «formvalidator» [En línea]. <http://www.formvalidator.net/>.
[Último acceso: Mayo 2017].
- [27] «Acunetix» [En línea]. <https://www.acunetix.com/websitesecurity/upload-forms-threat/>. [Último acceso: Mayo 2017].
- [28] Oracle, «The Java EE 6 Tutorial» 2013. [En línea].
<https://docs.oracle.com/javaee/6/tutorial/doc/glraq.html>.
[Último acceso: Mayo 2017].
- [29] «Stackoverflow» Julio 2013. [En línea].
<http://stackoverflow.com/questions/17402900/ffmpeg-process-saying-it-is-erroring>. [Último acceso: Mayo 2017].
- [30] «Wikipedia: REST» 25 Abril 2017. [En línea].
https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional.
[Último acceso: Mayo 2017].
- [31] «latamdigital.softtek.co» Septiembre 2015. [En línea].
<http://latamdigital.softtek.co/rest-restful-ventajas-y-diferencias>.
[Último acceso: Mayo 2017].
- [32] «Wikipedia: JSON» [En línea]. <https://es.wikipedia.org/wiki/JSON>.
[Último acceso: 2017 Mayo].
- [33] «W3schools: JSON» [En línea]. https://www.w3schools.com/js/js_json_intro.asp.
[Último acceso: Mayo 2017].
- [34] «Wikipedia: GSON» [En línea]. <https://es.wikipedia.org/wiki/Gson>.
[Último acceso: Mayo 2017].
- [35] «Restlet» [En línea]. <https://restlet.com/modules/client/>.
[Último acceso: Mayo 2017].
- [36] Lars Vogel, «Voguella» [En línea].
<http://www.voguella.com/tutorials/REST/article.html>.
[Último acceso: 2017 Mayo].

- [37] «Arquitecturajava: CORS» Enero 2015. [En línea].
<http://www.arquitecturajava.com/que-es-cors/>.
[Último acceso: Mayo 2017].
- [38] «html5box.com» [En línea]. <https://html5box.com/html5lightbox/>.
[Último acceso: Mayo 2017].
- [39] «Bootstrap» [En línea]. <http://getbootstrap.com/>.
[Último acceso: Mayo 2017].
- [40] «jQuery» [En línea]. <https://jquery.com/>.
[Último acceso: Mayo 2017].
- [41] «Wikipedia: jQuery» [En línea]. <https://es.wikipedia.org/wiki/JQuery>.
[Último acceso: Mayo 2017].
- [42] «Lancetalent» Febrero 2014. [En línea]. <https://www.lancetalent.com/blog/tipos-de-aplicaciones-moviles-ventajas-inconvenientes/>.
[Último acceso: Mayo 2017].
- [43] «Appyourself» Noviembre 2016. [En línea].
<https://appyourself.net/es/2016/11/10/tipos-de-aplicaciones-moviles/>.
[Último acceso: Mayo 2017].
- [44] «Adobe PhoneGap» [En línea]. <http://phonegap.com/>.
[Último acceso: Mayo 2017].
- [45] «Apache Cordova» [En línea]. <https://cordova.apache.org/>.
[Último acceso: Mayo 2017].
- [46] «Apache Cordova Documentation» [En línea].
<https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-whitelist/>.
[Último acceso: Junio 2017].
- [47] «Apple Developer Support» [En línea].
<https://developer.apple.com/support/certificates/>.
[Último acceso: Junio 2017].