Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Sistema de Gestión de Sesiones de Entrenamiento con Monitorización Mediante Streaming de Vídeo

Autor: Pablo Gil Pereira

Tutor: Mª Teresa Ariza Gómez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Sistema de Gestión de Sesiones de Entrenamiento con Monitorización Mediante Streaming de Vídeo

Autor:

Pablo Gil Pereira

Tutor:

Mª Teresa Ariza Gómez

Profesor Titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado: Sistema de Gestión de Sesiones de Entrenamiento con Monitorización Mediante Streaming de Vídeo

Autor: Pablo Gil Pereira
Tutor: Mª Teresa Ariza Gómez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Resumen

La tecnología actual nos permite disponer de multitud de dispositivos conectados a Internet, ya sean potentes smartphones y tablets o computadores personales. Esto, unido a la evolución de las redes de telecomunicación, nos ofrece un amplio marco para el desarrollo de aplicaciones basadas en la transmisión de vídeo, mediante las cuales poder monitorizar a un grupo de usuarios.

En este proyecto se ha desarrollado la infraestructura necesaria para que un especialista, desde cualquier ordenador con conexión a internet y la máquina virtual de Java instalada, pueda monitorizar a ciertos usuarios mientras hacen ejercicio, recibiendo el vídeo capturado por la cámara de los dispositivos móviles de los mismos.

El objetivo principal del proyecto es el desarrollo de una aplicación Java en la cual el monitor pueda crear sesiones de ejercicios, así como recibir y reproducir en tiempo real el vídeo de los usuarios realizando dicha sesión.

Abstract

Nowadays lots of powerful smartphones and tablets, or flexible personal computers, are available due to technological improvements. In addition, communication networks have gone through a big evolution in the last few years. As a result, we have a wide range of options to develop apps based on video streaming, in order to monitor users.

In this project we have developed the infrastructure which is needed by experts, in every computer with Internet connection and a Java Virtual Machine on it, to monitor some users doing excercises, to receive video caught by the camera on theirs mobile devices.

The main target of the project is to achieve a Java app where a monitor can create different workouts, as well as receive and play video streaming from the users who are doing these workouts.

Índice

Índice de Figuras

Índice de Tablas

1 Introducción

Lo que se pretende con este proyecto es desarrollar la infraestructura necesaria para que un grupo de usuarios pueda ser monitorizado por un especialista a través de streaming de vídeo mientras realizan ejercicio. Para ello nos apoyaremos en las tecnologías de comunicación y los grandes avances que en ellas se han producido en los últimos años, los cuales nos brindan un marco perfecto para desarrollar este tipo de aplicaciones.

1.1 Motivaciones

En la sociedad actual el ritmo de vida de la mayoría de las personas es muy acelerado, lo que implica que el tiempo disponible para el ocio se ve disminuido. Esto puede implicar no llevar unos hábitos de vida saludables, descuidando la alimentación y la actividad física. El objetivo de esta aplicación es que las personas sin tiempo para ir al gimnasio puedan realizar ejercicio físico de calidad, mientras son monitorizados por un especialista, evitando así posibles errores a la hora de realizar el entrenamiento. Todo ello sin la necesidad de salir de casa y gracias a los avances tecnológicos, que nos permiten comunicar al entrenador con el entrenado estando éstos en localizaciones distintas.

1.2 Escenario inicial

Para la realización de este proyecto partimos de varios proyectos realizados con anterioridad. En el primero de ellos [1], el usuario puede acceder, a través de un servidor, a diferentes rutinas de ejercicios almacenadas en una base de datos. Para ello se desarrollaron un servidor PHP, una base de datos MySQL y una aplicación Android, siendo la arquitectura la que se muestra en la figura (1.1). En nuestro caso, se reutilizarán el servidor y la base de datos, realizando en ellos pequeñas modificaciones que se comentarán más adelante.

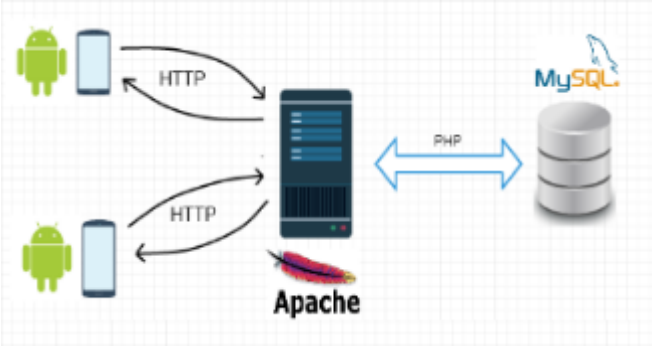

Figura 1.1 Arquitectura proyecto Antonio José Díaz Lora.

En el segundo de los proyectos [2], el monitor recibe streaming de vídeo procedente del usuario mientras éste realiza los ejercicios indicados. Para conseguir estas funcionalidades se implementaron una aplicación Android para el streaming de vídeo y un servidor HTML para la recepción de dicho vídeo y la comunicación

con el usuario por parte del monitor. Para la recepción del vídeo en este servidor se utilizó un servidor de vídeo Wowza [3], siendo la arquitectura la que observamos en la figura (1.2). El usuario envía el vídeo al servidor Wowza, mientras que el monitor lo recibe procedente de dicho servidor. En este proyecto nos encontramos con el problema de que el retardo en la reproducción del vídeo en el servidor del monitor es mayor del deseado (unos 4 ó 5 segundos). En este caso, se va a reutilizar la aplicación Android, realizando también en ella varias modificaciones, las cuales se tratarán posteriormente.

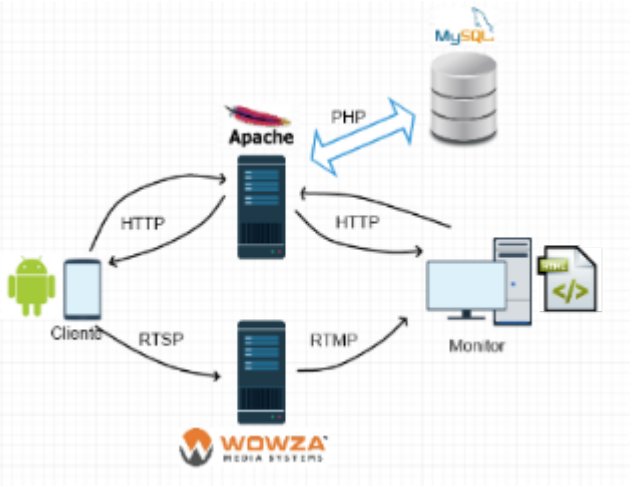

Figura 1.2 Arquitectura proyecto Antonio Clavaín Mateo.

1.3 Objetivos

Partiendo del trabajo realizado anteriormente por los compañeros, se proponen diferentes objetivos a la hora de realizar este proyecto. El objetivo principal es la implementación de una aplicación en el lenguaje de programación Java que, utilizando el servidor PHP, la base de datos MySQL y la aplicación Android previamente desarrolladas, permita la gestión de sesiones de entrenamiento, así como la recepción y reproducción del streaming de vídeo procedente de los distintos usuarios. Además, toda la información debe ser almacenada en la base de datos, evitando así el envío de información de las sesiones entre el usuario y el monitor. Por lo tanto, se almacenarán en la base de datos los ejercicios asignados a cada sesión y los usuarios inscritos a las mismas.

A este objetivo principal se le unen varios objetivos secundarios, pero no por ello menos importantes. En primer lugar, se deberá reducir el retardo en la reproducción del streaming de vídeo, demasiado elevado en el proyecto anterior [2], poniendo como objetivo un retardo máximo de 2 segundos. Además, deberá desaparecer del sistema el servidor Wowza, pasando a ser la transmisión del vídeo directamente entre el monitor y el usuario. Con esto se pretende la disminución de los agentes externos a las aplicaciones, evitando así el mantenimiento por parte de terceros del servidor de vídeo.

1.4 Agentes del sistema

En esta sección introduciremos los diferentes elementos que componen nuestro sistema, que serán de tres tipos: servidor, aplicación Java y aplicación Android.

- **Servidor:** Será el servidor del gimnasio y en él se alojarán tanto la base de datos, donde se almacenará toda la información persistente relativa a los usuarios, ejercicios y sesiones del sistema; como el servidor PHP, el cual se encargará de realizar las consultas a la base de datos para ofrecer la información allí almacenada al resto de aplicaciones mediante peticiones HTTP, así como de insertar en la base de datos la información procedente de dichas aplicaciones. El servidor del gimnasio se explicará con más detalle en el capítulo (3).
- **Aplicación Java:** Denominaremos así a la aplicación Java desde la cual el monitor podrá crear sesiones para los clientes, así como monitorizarlos a través de streaming de vídeo mientras hacen ejercicio. El

monitor deberá iniciar sesión en el sistema y, una vez dentro, se le mostrará el menú de las sesiones. Desde dicho menú se pueden crear nuevas sesiones, así como acceder y eliminar cualquiera de las sesiones del monitor. Toda la información de las nuevas sesiones se almacena en la base de datos del servidor del gimnasio. Una vez accedemos a una sesión, se mostrará una lista con los ejercicios de la misma, así como otra lista con el resto de ejercicios que no han sido incluídos en la sesión, los cuales podrán ser añadidos en cualquier momento. Además, en esta pantalla también se encuentran 4 reproductores de vídeo donde, una vez comenzada la sesión, se podrá reproducir el vídeo de cualquiera de los clientes inscritos en la misma. Mientras se está visualizando a un usuario, el monitor tendrá la posibilidad de enviarle mensajes de texto para comunicarse con él en caso de que fuera necesario. Esta aplicación será tratada con mayor profundidad en el capítulo (4).

- **Aplicación Android:** Será la aplicación Android que utilizarán los clientes para realizar los ejercicios desde casa. Ésta será compatible con los dispositivos móviles que utilicen una versión de Android igual o superior a la 4.0, debido a los requisitos de las librerías utilizadas. Desde esta aplicación, el usuario podrá iniciar sesión en el sistema, tras lo cual se le mostrarán todas las sesiones disponibles. Una vez seleccionada una sesión, el usuario quedará inscrito en la misma y se le mostrará la lista de ejercicios que la componen. Cuando la sesión haya comenzado, podrá seleccionar alguno de estos ejercicios para su ejecución. Tras haber sido seleccionado un ejercicio, la aplicación se quedará a la espera de la petición del monitor para comenzar a transmitir el vídeo. Además, el usuario podrá recibir mensajes del monitor mientras realiza el ejercicio. Esta aplicación se explica más extensamente en el capítulo (5).

1.5 Arquitectura del sistema

Como se observa en la figura (1.3), nuestro sistema consta de 3 elementos: el servidor del gimnasio, la aplicación Android y la aplicación Java. El servidor del gimnasio, donde se almacenará toda la información relativa a usuarios, ejercicios, sesiones, etc., es único y está formado por un servidor Apache y una base de datos MySQL. Por otra parte se encuentra la aplicación Android, de las que podrá haber tantas como clientes haya en el sistema. El último elemento es la aplicación del monitor, que es una aplicación Java que podrá ser instalada tanto en los equipos del gimnasio como en los ordenadores personales de cada uno de los monitores; y de la cual podrá haber tantas instancias como sea necesario.

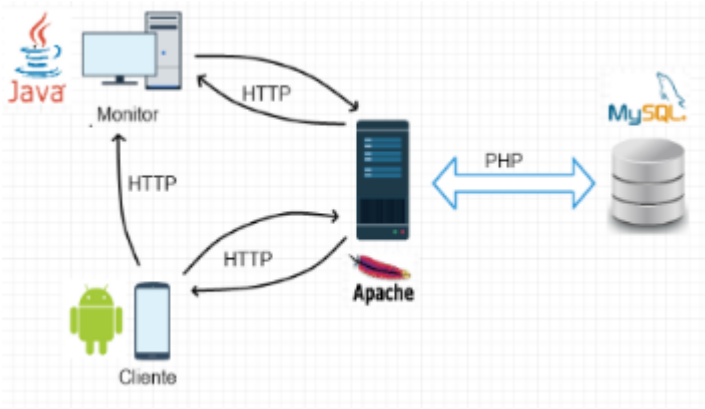

Figura 1.3 Arquitectura del sistema.

La comunicación entre el servidor y la base de datos será igual que en los proyectos anteriores; es decir, el servidor PHP se alojará en el servidor Apache, que se encargará de hacer las consultas a la base de datos. De igual manera, al reutilizarse la aplicación Android, la comunicación entre la aplicación del cliente y el servidor del gimnasio para obtener información de la base de datos seguirá siendo mediante peticiones HTTP. Por otro lado, la aplicación Java para el monitor también utilizará el protocolo HTTP para las peticiones al servidor del gimnasio.

El gran cambio en cuanto a la arquitectura podemos observarlo en el envío del streaming de vídeo. Se ha optado por el envío directo entre cliente y monitor, ahorrándonos así la necesidad de pasar por un servidor

intermedio. En este caso, se implementará en la aplicación Android un servidor que estará a la espera de conexiones para enviar el vídeo. Una vez el monitor envíe la petición para ver a uno de los clientes, la aplicación Android transmitirá el vídeo en formato MJPEG sobre HTTP.

A continuación veremos varios ejemplos de la comunicación entre los 3 elementos durante la ejecución de los mismos, viendo de forma gráfica la diferencia con las anteriores arquitecturas, centrándonos sobre todo en la parte del monitor, ya que es el objetivo principal de nuestro proyecto.

Inicialmente el monitor introducirá sus credenciales, las cuales se enviarán al servidor para comprobar si son correctas. Después de entrar en el sistema, se recibirán todas las sesiones del monitor procedentes del servidor. Este proceso podemos verlo en la figura (1.4).

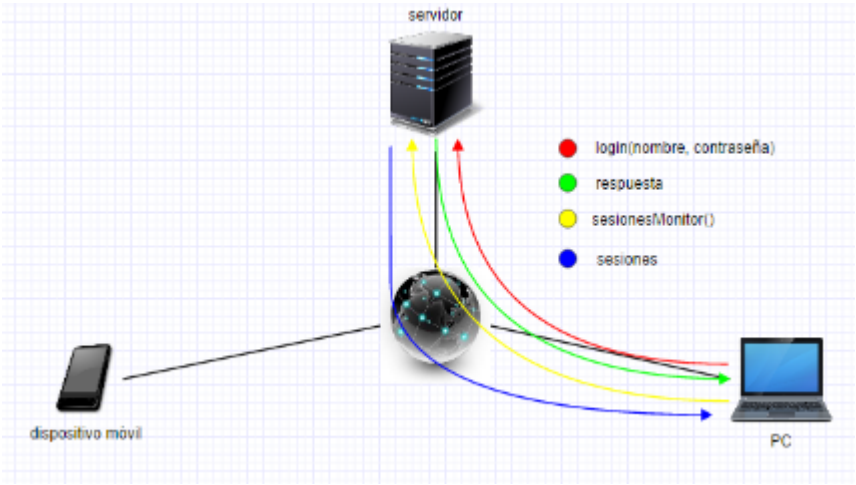

Figura 1.4 Monitor iniciando sesión en el sistema.

Una vez dentro del sistema, el monitor podrá crear nuevas sesiones. Como observamos en la figura (1.5), se envían los datos de la sesión al monitor y, una vez llegado el mensaje de confirmación, añadimos los ejercicios correspondientes.

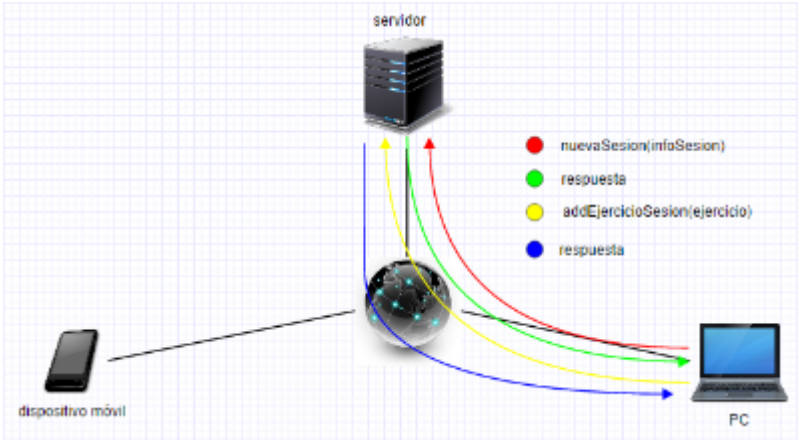

Figura 1.5 Monitor creando una nueva sesión.

Ya desde el lado del cliente, el login se hará de la misma manera que en el monitor, por lo que obviaremos este paso. La diferencia en cuanto a otras arquitecturas aparece a la hora de recibir las sesiones. El dispositivo solicitará todas las sesiones al servidor nada más entrar en el sistema. Una vez elegida alguna de las sesiones, se realizará la petición de los ejercicios que la componen, la cual se enviará al servidor y no al monitor como ocurría anteriormente [2]. Todo este proceso es el que se muestra en la figura (1.6)

Una vez recibidos los ejercicios, el cliente podrá comenzar a realizarlos. Antes de ésto, como vemos en la figura (1.7), deberá registrarse en la sesión mediante un mensaje al monitor correspondiente. Para ello se solicita al servidor la IP del equipo del monitor, tras lo que enviará el mensaje de registro. Una vez hecho ésto,

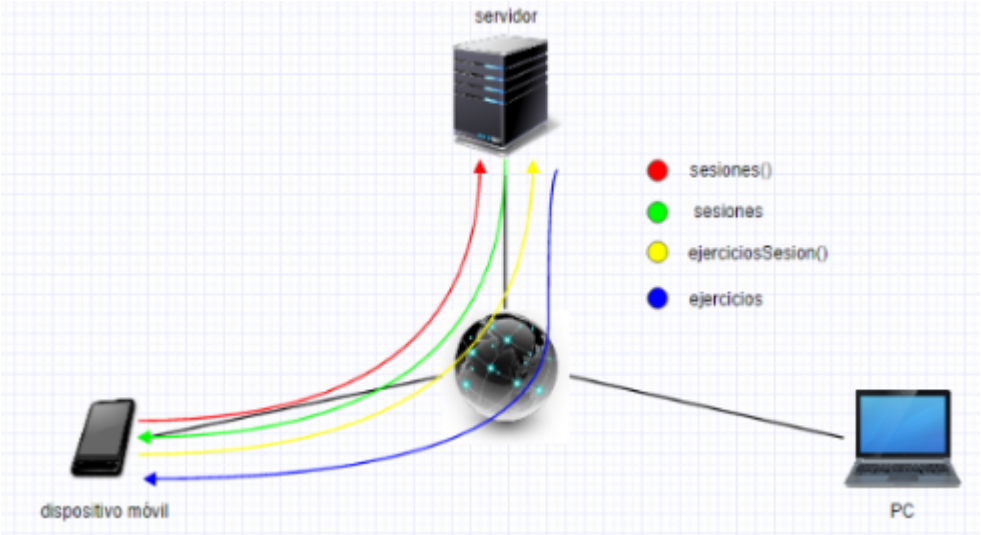

Figura 1.6 Recepción de sesiones en el dispositivo móvil.

si el monitor quiere ver a este usuario enviará una petición de vídeo directamente a la aplicación Android y ésta le responderá enviando el vídeo. Como vemos, la comunicación para la solicitud y entrega del vídeo es directamente entre las dos aplicaciones.

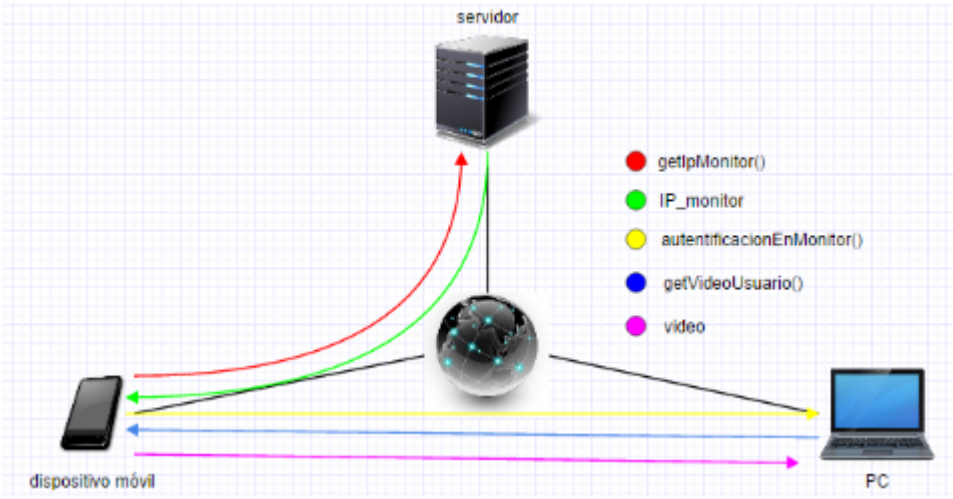

Figura 1.7 Petición y recepción de vídeo.

1.6 Estructura de la memoria

La memoria del proyecto se estructura en capítulos y el esquema que se va a seguir es el siguiente:

- **Capítulo 1**. Introducción: En este capítulo se exponen las motivaciones del proyecto, así como la situación inicial desde la que partimos y los objetivos a los que se pretende llegar. También se describe a nivel general la arquitectura de nuestro sistema y se realiza una breve descripción de los elementos que componen el mismo.
- **Capítulo 2**. Tecnología: En esta sección se realizará una introducción a las diferentes tecnologías utilizadas durante la elaboración del proyecto y cómo contribuye cada una de ellas al mismo.
- **Capítulo 3**. Servidor: Descripción detallada del servidor del gimnasio. Se expondrán tanto las tablas de la base de datos como las funciones del servidor PHP.

- **Capítulo 4**. Aplicación Java: Muestra las funcionalidades que se han implementado en la aplicación del monitor y describe detalladamente las clases que la componen.
- **Capítulo 5**. Aplicación Android: En este capítulo se profundizará en la aplicación Android y las distintas clases que la forman. También se justificarán las modificaciones efectuadas para cumplir con los objetivos marcados al comienzo del proyecto.
- **Capítulo 6**. Funcionamiento del sistema: Capítulo en el que se incluirán diferentes diagramas UML explicando la interacción entre los elementos que componen el sistema.
- **Capítulo 7**. Conclusiones: Se indicarán los objetivos alcanzados durante la elaboración del proyecto y las posibles líneas de mejora del mismo. También se expondrán las conclusiones personales del proyecto.
- **Bibliografía**. Compendio de todas las fuentes consultadas durante la elaboración del proyecto.
- **Anexos**. En los diferentes anexos se adjuntarán los códigos fuente del servidor PHP y de las aplicaciones Java y Android. También se proporcionará un manual de instalación para poner en funcionamiento el sistema.

2 Tecnología

La elección de las tecnologías utilizadas en un proyecto es una tarea complicada y que debe realizarse cautelosamente, ya que esta decisión afectará directamente a la dificultad del proyecto, e incluso al fracaso o el éxito en la consecución de los objetivos inicialmente marcados. Para ello será necesario realizar un estudio de las tecnologías disponibles y seleccionar las que mejor se ajusten a las necesidades de nuestro proyecto.

Al partir de proyectos anteriores nos encontramos con que en este proyecto parte de la tecnología está predefinida antes de comenzar. A la hora de elegir el resto de opciones, se ha considerado positivamente el uso de software libre. A continuación se introducirán brevemente las diferentes tecnologías utilizadas.

Como el servidor del gimnasio está basado en una estructura PHP y base de datos MySQL, se ha optado por la utilización de la arquitectura AMP (Apache+MySQL+PHP) sobre un entorno Windows para el despliegue de este servidor. En concreto hemos elegido el entorno XAMPP [4], con el que conseguimos instalar de una forma sencilla Apache, MySQL y PHP, además de otras funcionalidades que no serán tenidas en cuenta en este proyecto.

También hemos utilizado JSON [9], formato de serialización de datos, para la comunicación del servidor tanto con la aplicación de los clientes como con la de los monitores.

Por último, para el desarrollo de la aplicación del monitor hemos empleado el lenguaje de programación Java [7], lo cual nos permite la ejecución de esta aplicación en cualquier equipo con la JVM[1] instalada. Esto nos ofrece una gran versatilidad, ya que así no dependeremos del sistema operativo en el que se ejecute la aplicación.

2.1 Apache

Apache es un servidor HTTP de código abierto, modular y multiplataforma desarrollado por Apache Software Fundation [5]. Las ventajas de este servidor es que es altamente configurable, gracias a la gran cantidad de módulos que posee; y seguro, con módulos específicos para la seguridad del sevidor; lo cual ha hecho que sea el servidor web más extendido en la actualidad. Según Netcraft[2], el 51 % de los servidores activos en Abril del 2015 utilizaban Apache.

En nuestro sistema se desplegará un servidor Apache, que albergará el servidor PHP de consultas a la base de datos, por lo que será necesaria la instalación del módulo de PHP para Apache. Este servidor estará a la escucha de las peticiones de las aplicaciones Java y Android, respondiendo a las mismas con la información solicitada.

2.2 PHP

PHP (PHP Hypertext Pre-processor) [8], es un lenguaje de programación interpretado de alto nivel del lado del servidor que puede ser incrustado en páginas HTML. A diferencia de otros lenguajes como JavaScript, el código PHP se ejecuta en el servidor generando un archivo HTML que será enviado al cliente.

[1] Java Virtual Machine

[2] http://news.netcraft.com/archives/2015/04/20/april-2015-web-server-survey.html

Este lenguaje puede emplearse en la mayoría de sistemas operativos y en servidores como Apache o IIS, lo cual nos otorga la libertad de elegir el que más se ajuste a nuestras necesidades. Por otro lado, PHP no solo genera HTML, sino que también tiene la capacidad de generar imágenes, ficheros PDF o películas Flash; así como cualquier tipo de fichero XML.

Pero lo que hace más interesante PHP para su inclusión en nuestro proyecto es su soporte para una gran cantidad de bases de datos diferentes, ofreciendo extensiones para muchas de ellas, como por ejemplo la extensión para MySQL, que será la base de datos utilizada en este caso. Esta gran versatilidad nos permitiría, en caso de que fuera necesario en un futuro, cambiar MySQL por cualquier otra base de datos que admita la extensión ODBC[3], abriéndonos un gran abanico de posibilidades en este aspecto.

2.3 MySQL

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario, desarrollado por MySQL AB como software libre. Algunas de las características principales de este sistema gestor, que está disponible para la mayoría de los sistemas operativos más famosos, son: arquitectura cliente/servidor, totalmente compatible con SQL, soporta replicación o, lo que es lo mismo, permite copiar el contenido de una base de datos a múltiples ordenadores; permite realizar transacciones, que son un conjunto de sentencias SQL realizadas en bloque y que o se ejecutan todas o no se ejecuta ninguna; y además, hay gran variedad de APIs y librerias para MySQL disponibles en diferentes lenguajes de programación.

A pesar de la gran cantidad de características disponibles, no todo son ventajas en MySQL, ya que posee carencias como la imposibilidad de añadir tipos de datos definidos por el usuario, la ausencia de soporte para XML o que no ofrece funcionalidades para aplicaciones en tiempo real.

Nuestro sistema dispondrá de una base de datos MySQL en el servidor del gimnasio. Sobre ella se realizarán consultas para la obtención e inserción de datos, llevándose un registro de los usuarios, ejercicios y sesiones del sistema, así como toda la información relacionada con los mismos.

2.4 JSON

JSON (JavaScript Object Notation) [9] es un formato de serialización de datos construido sobre JavaScript y diseñado por Douglas Crockford. Su aparición se debió a la creciente popularidad de los servicios web, lo que hacía necesario un nuevo formato de codificación de datos que no precisara de tantos bytes como XML, siendo JSON un formato mucho más ligero.

Hemos elegido este lenguaje de codificación de datos porque resulta muy simple no solo para las máquinas, sino también para los humanos, ya que está orientado a las estructuras de datos de lenguajes de programación modernos. En el proyecto únicamente lo utilizaremos en la comunicación del servidor con el resto de las aplicaciones.

2.5 Java

Java [7] es un lenguaje de programación de propósito general, concurrente y orientado a objetos, que fue diseñado en 1995 por Sun Microsystems con el objetivo de conseguir un lenguaje de programación con las menores dependencias de implementación como fuera posible, haciendo que no tuviera que ser recompilado dependiendo de la plataforma donde fuera a ejecutarse. Ésto ha hecho que actualmente sea uno de los lenguajes de programación más extendidos, especialmente en las aplicaciones cliente-servidor web.

Gracias a esta versatilidad en cuanto a los sistemas operativos, encontramos muy variados entornos de funcionamiento de aplicaciones, como pueden ser los dispositivos móviles, navegadores web, servidores y aplicaciones de escritorio. Teniendo en cuenta los objetivos buscado en este proyecto, vamos a desarrollar una aplicación de escritorio, descartando el resto de entornos posibles. Otra gran ventaja que nos proporciona esta versatilidad es que la aplicación a desarrollar podrá ser utilizada con independencia de los dispositivos de los que disponga el gimnasio o los monitores, el único requisito necesario será el entorno de ejecución Java (JRE).

[3] Estándar abierto de Conexión con Base de Datos

Además, Java implementa gran cantidad de funciones de red, fundamentales para conseguir los objetivos marcados al principio del proyecto. Aprovecharemos estas utilidades a la hora de desarrollar nuestra aplicación, que deberá comunicarse con el resto de elementos que componen el sistema a través de la red.

2.6 Eclipse

Eclipse [10] es un entorno de desarrollo multiplataforma de código abierto y que inicialmente fue desarrollado por IBM, pero que ahora es desarrollado por la Fundación Eclipse. Utiliza diferentes módulos o plug-in para ofrecer multitud de funcionalidades al usuario, permitiendo así que Eclipse pueda trabajar con diferentes lenguajes de programación, como C++ o Python; así como aplicaciones de red como Telnet, entre otras.

Esta modularidad de Eclipse es la que nos ha hecho decantarnos por este IDE, ya que no solo tiene una versión específica para el desarrollo de aplicaciones en Java[4], sino que además podríamos añadir diferentes módulos para trabajar con otros lenguajes. Esto es una gran ventaja, teniendo en cuenta que en este proyecto no trabajamos con un único lenguaje de programación.

Por otra parte, como se comentará más adelante (Capítulo 4), Eclipse también nos da la opción de cargar diferentes librerías para Java, extendiendo así las funcionalidades que originalmente nos ofrece este lenguaje de programación y que nos será de utilidad para ciertas partes del proyecto.

2.7 Android Studio

IDE desarrollado por Google para la creación de aplicaciones para el sistema operativo Android. Este entorno de desarrollo nos falicita la implementación de aplicaciones Android con un depurador muy completo que facilita la labor de la programación, reduciendo considerablemente el tiempo empleado en resolver errores y comportamientos no deseados del programa. En nuestro proyecto lo utilizaremos para realizar modificaciones en la aplicación ya desarrollada con anterioridad [2].

2.8 VLC Media Player

Reproductor de vídeo libre y de código abierto desarrollado por VideoLAN Organization. Es un programa multiplataforma con capacidad para reproducir gran cantidad de formatos de vídeo sin la necesidad de la instalación de codecs, a lo que se le suma la funcionalidad de streaming de vídeo mediante diferentes protocolos de transmisión.

El streaming de vídeo es lo que hace que VLC media player sea una aplicación a tener en cuenta para este proyecto. Ésto, unido a las librerías VLCj [13], que nos permiten incrustar este reproductor en nuestra aplicación Java, es lo que hace a esta aplicación ser fundamental para la recepción de streaming en la aplicación del monitor. Se encargará de realizar las peticiones de vídeo a los clientes y reproducir el vídeo recibido.

2.9 MJPEG

MJPEG es un formato de compresión de vídeo en el que cada frame de un vídeo se comprime separadamente como una imagen JPEG. Aunque originalmente se desarrolló para aplicaciones multimedia para el PC, actualmente es utilizado en todo tipo de dispositivos como cámaras IP o webcams. En nuestro proyecto en concreto se va a utilizar en su versión MPEG-4 [14] para la transmisión de vídeo de los clientes al monitor.

2.10 UPnP

UPnP[5] es un protocolo para la conexión entre equipos de diferentes fabricantes, otorgándoles los mecanismos para descubrir otros dispositivos y establecer diferentes servicios de red. Los equipos UPnP, una vez conectados

[4] Eclipse IDE for Java Developers
[5] Universal Plug and Play

a una red, serán capaces de establecer de forma autónoma comunicaciones con otros dispositivos. Este protocolo se basa en estándares como *TCP*, *IP*, *HTTP*, *XML* y *SOAP*.

Cualquier dispositivo que implemente UPnP y sea conectado a una red, será capaz de anunciar sus servicios y descubrir la presencia y servicios del resto de dispositivos en la red. Además, también ofrece otras funcionalidades como independencia de formato, el control de dispositivos mediante interfaz de usuario e independencia del sistema operativo o del lenguaje de programación, entre otras.

No es el objetivo en esta sección detallar al completo el funcionamiento de este protocolo, ya que entre todas sus funcionalidades solo vamos a emplear la de mapeo de puertos en el NAT[6]. Como ya sabemos, las direcciones IPv4 se pueden dividir en dos rangos diferentes: direcciones IP públicas y direcciones IP privadas. Un problema recurrente en este proyecto será que tendremos que comunicar las distintas aplicaciones, las cuales pueden estar instaladas en equipos con direcciones privadas y en redes diferentes. Para solventar este problema, se ha optado por tener cierto control de la tabla de traducción del NAT desde las aplicaciones, para así poder conocer en todo momento los conjuntos IP-puerto asociados a nuestro equipo. De esta forma podremos comunicar las diferentes instancias de nuestras aplicaciones con cualquier equipo, con independencia de que éstos estén en diferentes redes privadas.

Por ésto es por lo que utilizamos el protocolo UPnP, ya que nos permitirá configurar mapeo de puertos desde nuestra aplicación. Lo que haremos será que, si el equipo está conectado a un NAT, se pondrá en contacto con el mismo, solicitando cierto puerto y reservándolo para nuestro equipo. Así conseguimos que todas las peticiones que le lleguen al NAT con puerto destino coincidente con el seleccionado, automáticamente serán reenviadas a nuestra aplicación, con la ventaja de que en el origen de las peticiones solo será preciso conocer la dirección pública del NAT y el puerto seleccionado.

[6] Network Address Translation

3 Servidor

Éste será el servidor que se encuentre en el gimnasio y en él se desplegarán el servidor PHP y la base de datos. El servidor PHP se encargará de responder todas las peticiones de las aplicaciones Java y Android, tanto las consultas a la base de datos, respondiendo con la información solicitada, como las peticiones que pretendan insertar cierta información en la base de datos, confirmando en la respuesta si la solicitud ha sido exitosa o no.

Por otro lado, la base de datos almacenará toda la información persistente del sistema, que solo podrá ser accedida a través del servidor PHP. La información que almacenará será la relacionada con las sesiones de ejercicios, los ejercicios disponibles para realizar y los usuarios registrados en el sistema.

La comunicación con el servidor por parte del resto de aplicaciones se realizará mediante peticiones HTTP, respondiendo el servidor con la información codificada en JSON.

Aunque nos basaremos en los proyectos anteriores ([1] y [2]), realizaremos varias modificaciones tanto en el código PHP, con la implementación de nuevas funciones para la obtención de datos que no estaban previamente implementadas y que son necesarias en este proyecto; como en la base de datos, con la unificación de las tablas de rutinas y sesiones.

3.1 Base de datos

Partiendo de la base de datos del proyecto [2] y tras ciertas modificaciones, llegamos a la base de datos mostrada en la figura (3.1), en la cual observamos su modelo entidad-relación.

La base de datos deberá cumplir ciertos requisitos (Tabla 3.1) para el correcto funcionamiento del sistema. Algunos de estos requisitos ya están cubiertos por la base de datos de la que partimos, pero para conseguir otros se han realizado ciertas modificaciones que se comentarán más adelante.

Tabla 3.1 Requisitos de la base de datos.

Código	Descripción
R01	Usuarios e información de usuario
R02	Ejercicios e información detallada de los ejercicios
R03	Historial de sesiones
R04	Ejercicios de una sesión
R05	Usuarios inscritos en las sesiones

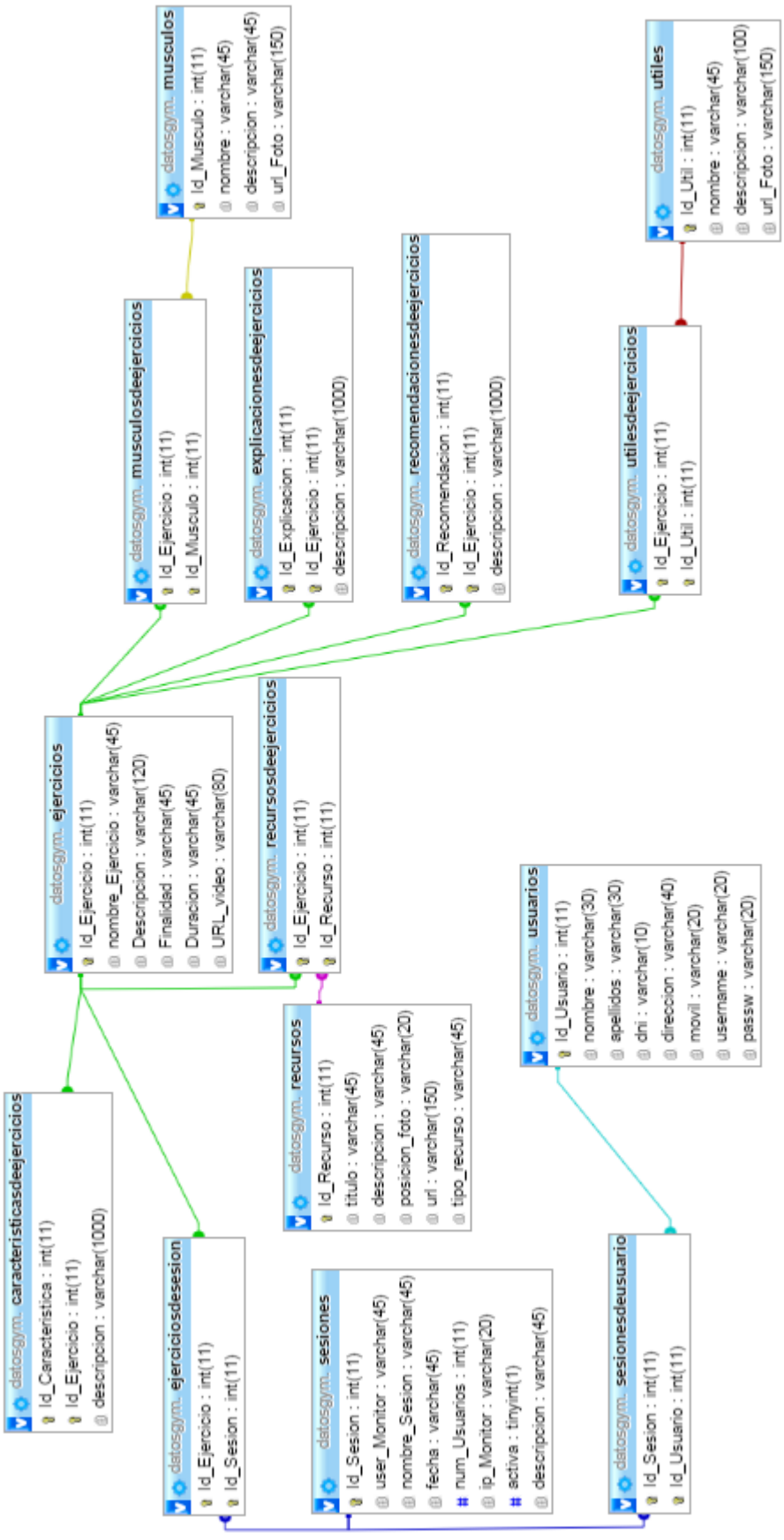

Figura 3.1 Base de datos: modelo entidad-relacion.

Para cumplir los dos primeros requisitos (R01 y R02) se utilizarán las tablas *usuarios* y *ejercicios*, que tenemos representadas en la figura (3.1). Estas dos tablas son heredadas de la base de datos anterior [2] y, mientras la tabla *usuarios* no ha sufrido ninguna modificación, en la tabla *ejercicios* hemos añadido el campo URL_video. Los vídeos de los ejercicios que utiliza la aplicación Android (Capítulo 5), tenían una URL fija y se encontraban en el servidor del gimnasio. Sin embargo, con este cambio se pretende que la URL pueda ser dinámica, por lo que el vídeo se podrá almacenar en cualquier otra localización, disminuyendo así la carga del servidor en el gimnasio.

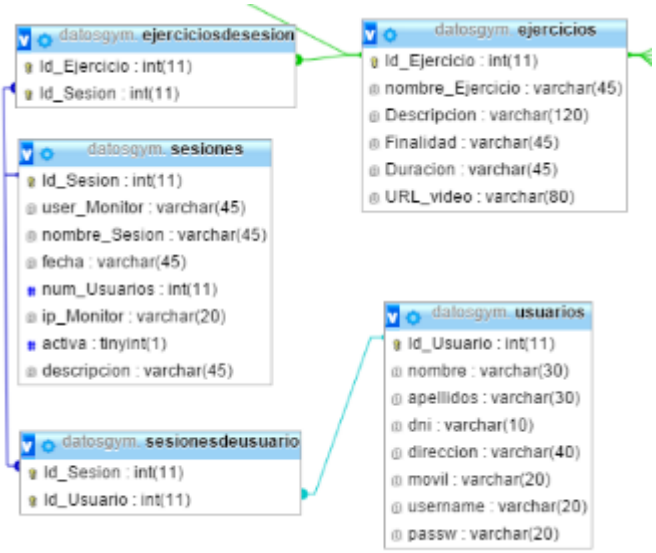

Figura 3.2 Tablas para cumplir los nuevos requisitos.

Los cambios más importantes en la base de datos se producen a la hora de conseguir el resto de los requisitos. En el proyecto anterior [2] nos encontrábamos con las tablas *rutinas* y *sesiones*, las cuales se muestran en la figura (3.3). La tabla *sesiones* llevaba un historial de todas las sesiones creadas, mientras que los usuarios se suscribían a las diferentes rutinas, las cuales incluían los ejercicios que debían hacer.

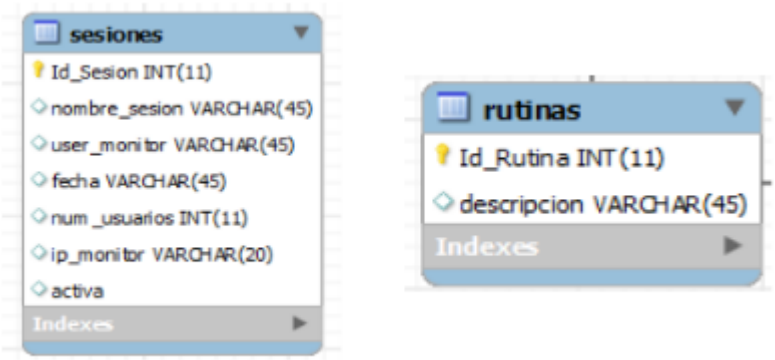

Figura 3.3 Tablas del proyecto de Antonio Clavain Mateo.

En este proyecto se ha optado por unificar ambas tablas en una sola tabla llamada *sesiones*, como observamos en la figura (3.1). Esta tabla se utilizará como historial de todas las sesiones creadas en el sistema, cumpliendo con el requisito R03. Así mismo, como podemos observar con más detalle en la figura (3.2), estará relacionada con las tablas *ejerciciosdesesion*, almacenándose así todos los ejercicios que forman parte de una sesión (R04) y *sesionesdeusuario*, que servirá para tener un registro con todos los usuarios que se inscriben a cada sesión (R05). Terminando así con todos los requisitos que debe cumplir la base de datos en nuestro proyecto.

3.2 Servidor PHP

Pese a que vamos a reutilizar gran parte del servidor previamente desarrollado [1], la modificación en la base de datos nos obliga a marcarnos nuevos requisitos en el servidor (Tabla 3.2), lo que supondrá el desarrollo de nuevos ficheros PHP para así implementar estas nuevas funcionalidades. Por otra parte, también eliminaremos los ficheros que hayan quedado obsoletos tras la modificación de la base de datos. El contenido de todos los ficheros del anterior proyecto que hayan sufrido alguna modificación, así como los nuevos que hayamos creado, podremos encontrarlo en los Anexos.

Tabla 3.2 Requisitos del servidor PHP.

Código	Descripción
R06	Obtener los datos de una sesión
R07	Insertar nuevas sesiones en la base de datos
R08	Asociar ejercicios a una sesión
R09	Eliminar ejercicios de una sesión
R10	Obtener todos los ejercicios de una sesión
R11	Obtener todos los ejercicios no asociados a una sesión
R12	Inscribir usuarios en una sesión
R13	Modificar la IP del monitor asociada a una sesión
R14	Obtener la información de un ejercicio

Por otro lado, también queremos implementar una clase gestor de la base de datos, que será en la que se implementen todos los métodos de consulta a la base de datos, evitando así que se realicen consultas desde otras clases del servidor, lo cual ocurría en el servidor inicial (para más información consultar los anexos del proyecto [1]). Para desarrollar este gestor partiremos del fichero *funciones_bd.php* de dicho proyecto, que pasará a denominarse *gestor_bd.php* y en el cual añadiremos los métodos que sean necesarios. Ésto no se considerará como un nuevo requisito al no estar añadiendo ninguna nueva funcionalidad al sistema.

En cuanto a las nuevas funcionalidades, deberá ser posible obtener toda la información de una sesión dada, para lo cual se utilizará el archivo *getSesion.php* que a partir del identificador de una sesión nos proporcionará el valor de todos sus campos. Por otro lado, para añadir nuevas sesiones a la base de datos tendremos *addSesion.php*, que nos permitirá insertar una sesión, cumpliéndose así los requisitos R06 y R07.

También será necesario añadir usuarios y ejercicios a las sesiones, por lo que se crearán los ficheros *addEjercicioSesion.php* y *addUsesSesion.php*, con los que cumplimos con los requisitos R08 y R12. Al igual que podemos añadir ejercicios, se deberá poder retirar estos ejercicios de una sesión (R09), para lo que utilizaremos *eliminaEjercicioSesion.php*, que eliminará un ejercicio de la sesión indicada.

Como ya veremos más adelante (Capítulo 4), necesitaremos obtener una lista con los ejercicios incluídos en una sesión, así como otra con el resto de ejercicios que no estén incluídos en dicha sesión (R10 y R11). *listaEjerciciosNoSesion.php* y *listaEjerciciosSesion.php* serán los ficheros PHP que implementen estas funcionalidades.

Además, debe ser posible modificar la IP del monitor asociada a una sesión (R13), ya que el monitor puede iniciar sesión en diferentes PCs. Para esto se implementamos *setIpMonitorServidor.php*, que tomará la dirección IP origen del paquete que llega al servidor, la cual será la dirección que almacenamos en la base de datos. Esto lo hacemos así porque el PC del monitor puede tener una dirección IP privada, por lo que tomando la IP del paquete que llega al servidor nos aseguramos que esta sea la dirección pública del monitor. Además, también necesitaremos conocer el puerto en el que la aplicación del monitor se mantendrá a la escucha, por lo que también se tomará el puerto que es enviado como parámetro en la petición POST por el monitor. Para conocer con más detalle por qué hacemos ésto, consultar los capítulos (4 y 6).

Por último, en ocasiones será necesario obtener toda la información de un ejercicio (R14), para lo que utilizaremos *getEjercicio.php*. También, cuando queramos dar una sesión por terminada, deberá pasar al estado de no activa, para lo que hemos creado la clase *'setSesionNoActiva'*, que cambiará el campo *activa* de una sesión y lo pondrá a 0.

4 Aplicación Java

La aplicación desarrollada en Java para el monitor era el objetivo principal que nos marcamos al comienzo del proyecto. Con ella el monitor podrá diseñar multitud de sesiones de entrenamiento y, una vez comenzada una sesión, podrá observar cómo realizan los ejercicios los diferentes usuarios inscritos en ella mediante streaming de vídeo, pudiendo comunicarse con ellos en caso de que fuera necesario, con lo que se pretende que el usuario cometa el menor número de errores posibles mientras realiza un ejercicio gracias a las correcciones del monitor.

Recordando el resto de objetivos marcados al inicio del proyecto, esta aplicación deberá comunicarse con el resto de elementos del sistema. Con el servidor del gimnasio se comunicará para acceder a la información de la base de datos, así como para almacenar toda la información relativa a las sesiones que el monitor vaya a crear. Por otro lado, la comunicación con el cliente Android es necesaria debido a la eliminación del servidor de vídeo intermedio, por lo que el streaming de vídeo se realizará directamente entre cliente y monitor. Por último, otro objetivo importante es el retardo del vídeo, que debe ser el menor posible, ya que esto afectará directamente a la monitorización del usuario. Si el retardo es elevado, esto impedirá al monitor hacer correctamente su trabajo.

En cuanto al diseño visual de la aplicación, se ha optado por seguir con el diseño de la aplicación Android, ya que considero que es un diseño atractivo y, por otro lado, unificamos los diseños de cara al cliente, con lo cual podrá identificar a simple vista que está utilizando una de nuestras aplicaciones.

4.1 Librerías

Para la elaboración de este programa ha sido necesaria la utilización de diferentes librerías, las cuales nos han proporcionado nuevas funcionalidades imprescindibles para nuestro proyecto.

En la comunicación con el servidor, la información contenida en las respuestas vendrá codificada en JSON, por lo que tendremos que descodificarlas. Para ello utilizaremos la librería *'java-json'*, la cual nos ofrece diferentes clases para la manipulación de información codificada en JSON.

Por otro lado, como ya hemos mencionado anteriormente, para la reproducción del streaming de vídeo vamos a utilizar VLC media player. Para ello haremos uso de la API de VLCj [13], para lo que tendremos que añadir a nuestro proyecto tres librerías: *'vlcj-2.4.1'*, para las clases de VLCj; *'jna-3.5.2'* y *'platform-3.5.2'*, ambas para acceder a archivos propios de Windows mediante invocación nativa de Java, lo que nos permitirá acceder a los archivos DLL de la aplicación VLC.

También será necesario el manejo de fechas en la aplicación, para lo que incluiremos las librerías de la clase *jCalendar*. Estas librerías son: *'jcalendar-1.4'*, *'jgoodies-common-1.2.0'*, *'jgoodies-looks-2.4.1'* y *'junit-4.6'*. Emplearemos estas librerías para la introducción de la fecha de inicio de una sesión.

Por último, para utilizar el protocolo UPnP en nuestra aplicación vamos a utilizar la librería *Weupnp* [16], que nos ofrece distintas clases para realizar el mapeo de puerto o *port mapping* necesario para la comunicación entre aplicaciones. Para añadir estas librerías se ha creado un nuevo paquete en el proyecto, llamado *'weupnp'*, en el cual se han introducido directamente las clases que se van a utilizar.

4.2 Desarrollo

Para explicar el desarrollo de la aplicación se describirán las distintas clases implementadas, así como las ventanas de la aplicación según el orden en el que se le muestran al monitor. También comentaremos las dificultades encontradas durante este desarrollo y la forma en que las hemos solventado. Inicialmente nos marcamos como objetivo que la aplicación cumpla con los requisitos mostrados en la tabla (4.1).

Tabla 4.1 Requisitos de la aplicación Java.

Código	Descripción
R15	Enviar peticiones al servidor
R16	Iniciar sesión en el sistema
R17	Crear nuevas sesiones de ejercicios
R18	Eliminar sesiones
R19	Mostrar información de sesiones
R20	Mostrar historial de sesiones
R21	Añadir ejercicios a una sesión
R22	Eliminar ejercicios de una sesión
R23	Mostrar información de un ejercicio
R24	Enviar petición de vídeo a la aplicación del usuario
R25	Reproducir vídeo del usuario
R26	Terminar reproducción de vídeo del usuario
R27	Enviar mensajes al usuario

Para tener una visión general de la estructura de la aplicación, vamos a comenzar introduciendo las diferentes clases que la componen, las cuales podemos observar en la figura (4.1). Esta figura consiste en un diagrama de clases UML, con el cual podemos ver de un solo vistazo las clases más importantes de nuestro programa.

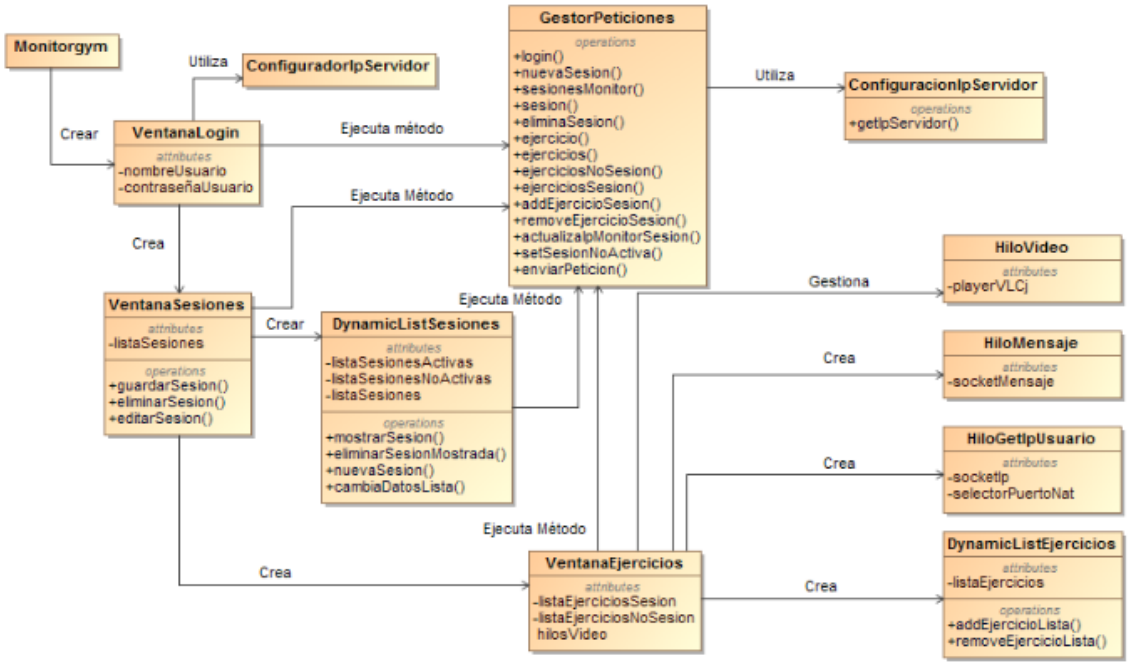

Figura 4.1 Diagrama de clases UML.

La clase principal será *Monitorgym*, que lanzará la primera ventana de usuario de la aplicación, *VentanaLogin*, que como su nombre indica será la ventana de inicio de sesión. Desde esta ventana además podremos configurar la IP del servidor del gimnasio con la clase *ConfiguradorIpServidor*. Tras iniciar sesión se lanzará

VentanaSesiones, clase desde la que gestionaremos todo lo relacionado con las sesiones creadas por el usuario, las cuales se mostrarán en una *DynamicListSesiones*. Una vez seleccionada una sesión, tendremos la clase *VentanaSesiones* para poder tratarla más en detalle y configurar todos los ejercicios que la componen, los cuales serán mostrados en una *DynamicListEjercicios*. Además, también nos permitirá gestionar todo lo relacionado con los reproductores de vídeo, que serán distintas instancias de la clase *HiloVideo*. Para finalizar con esta clase, en ella se lanzarán otros dos hilos: el primero de ellos será para el registro de usuarios en una sesión, llamado *HiloGetIpUsuario*; mientras que para enviar mensajes a estos usuarios utilizaremos *HiloMensaje*. Por último, podemos ver que todas las diferentes ventanas utilizan la clase *GestorPeticiones*, que se encargará de realizar todas las peticiones al servidor, cuya dirección obtendrá de la clase *ConfiguracionIpServidor*. Para poder obtener la IP del servidor primero tendremos que configurarla haciendo uso de la calse *ConfiguradorIpServidor*. Ahora entraremos en más detalle en las clases para explicar de forma precisa qué realiza cada una de ellas, comenzando por la clase encargada de las peticiones al servidor.

Como ya sabemos, la aplicación deberá solicitar información al servidor, para lo cual se implementa la clase *GestorPeticiones*, que se encargará de realizar todas estas peticiones, que serán del tipo HTTP POST, cumpliendo así con el requisito R15. Esta clase tendrá tantos métodos como peticiones diferentes debamos realizar, los cuales recibirán como parámetros los datos a enviar en la petición POST y devolverán la respuesta del servidor almacenada en un *JSONArray*. Cada uno de estos métodos tendrá asociado la URL correspondiente al fichero PHP al que debe acceder en el servidor y llamarán al método encargado de enviar las peticiones. Este último método formará el mensaje HTTP y lo enviará al servidor. De aquí en adelante, cada vez que mencionemos que esta aplicación realiza una petición al servidor se supondrá, a no ser que se especifique lo contrario, que lo hace utilizando una instancia de esta clase.

Para oder enviar las peticiones al servidor necesitaremos conocer su IP. Para evitar que se almacene de forma estática en el código, vamos a hacer uso del fichero *configuracion.properties*, que no es más que una posibilidad que nos otorga Java para almacenar parámetros de configuración de la aplicación. Para la gestión de este fichero tendremos dos clases diferentes, la primera de ellas será *ConfiguradorIpServidor*, que la utilizaremos para almacenar la dirección en el fichero; mientras que la segunda será *ConfiguracionIpServidor*, que servirá para obtener la dirección previamente almacenada.

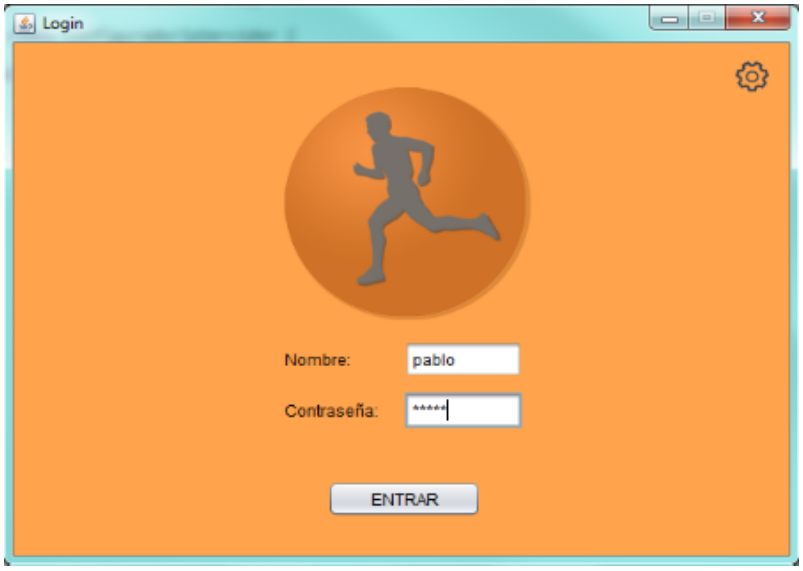

Figura 4.2 Formulario de inicio de sesión aplicación Java.

Centrándonos ya en la ejecución del programa, ésta comenzará en la clase principal, *Monitorgym*. En ella se define la apariencia de la GUI[1], decantándonos por el 'look and feel'*Nimbus* de entre todos los disponibles; y creará un nuevo objeto del tipo *VentanaLogin*, que mostará la primera ventana al usuario. Con esta clase cumplimos con el requisito R16 ya que, como podemos observar en la figura (4.2), consistirá en un formulario de inicio de sesión, donde el monitor deberá introducir sus credenciales. Estas credenciales serán enviadas al servidor para comprobar que son correctas y, en caso afirmativo, accederemos al sistema.

[1] Grafic User Interface

En caso contrario, mostramos un mensaje de error por pantalla y esperamos a que el usuario introduzca de nuevo sus credenciales. Si por cualquier motivo no ha sido posible contactar con el servidor, también se le mostrará un mensaje informativo al usuario, indicándole que espere unos instantes para comprobar de nuevo si la conexión se ha reinstaurado sin problemas.

Para poder enviar las credenciales al servidor, primero deberemos configurar la dirección IP de éste. En la esquina superior derecha de la pantalla (figura 4.2) hay un botón de configuración que, tras hacer click sobre él, nos mostrará un mensaje como el que vemos en la figura (4.3), utilizando un *JOptionPane*, para introducir la IP del servidor. Una vez introducida, utilizaremos la clase *ConfiguradorIpServidor* para almacenarla de forma persistente. Para ello, *ConfiguradorIpServidor* hará uso de la clase *Properties*, que nos ofrecerá la posibilidad de crear un fichero de configuración llamado *configuracion.properties*, donde se almacenará la IP.

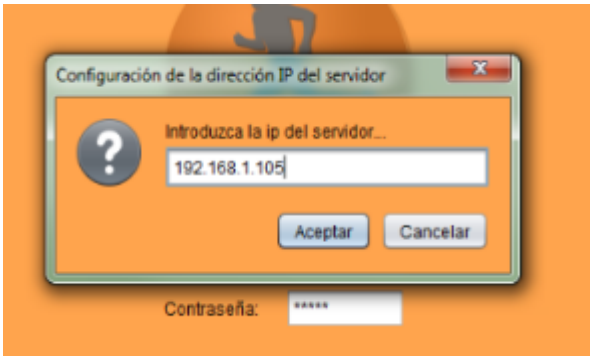

Figura 4.3 Campo a rellenar para configurar la IP del servidor.

Para acceder al fichero de configuración antes mencionado se utilizará la clase *ConfiguracionIpServidor*, que tiene un método que devuelve la IP almacenada en el fichero. Esta clase también emplea la clase *Properties* para el acceso al fichero de configuración y será utilizada por *GestorPeticiones* cada vez que quiera enviar una petición al servidor, para comprobar en todo momento a qué dirección tiene que enviarla.

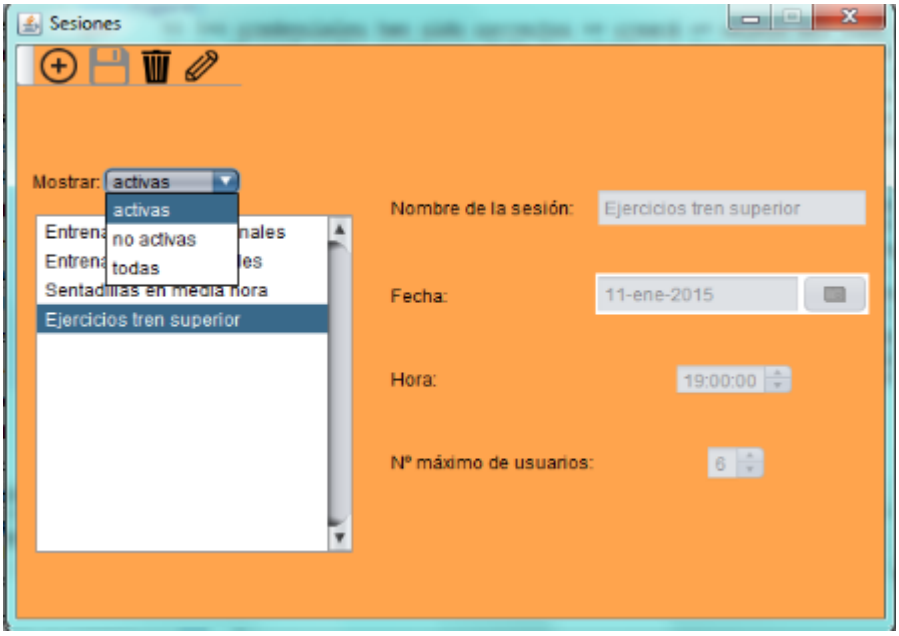

Figura 4.4 Ventana de sesiones con una sesión seleccionada.

Si las credenciales han sido correctas, se creará un objeto del tipo *VentanaSesion*, que nos mostrará un menú desde el cual poder gestionar todo lo relativo a las sesiones de ejercicios. Como vemos en la figura (4.4), en la ventana habrá una lista con todas las sesiones activas que haya creado este monitor; sobre ella, en la esquina superior izquierda de la pantalla, aparecerá un menú con cada una de las opciones habilitadas en

cada momento; y por último, a la derecha de la pantalla aparecen los diferentes campos de los que consta una sesión. Estos campos aparecerán inicialmente inactivos y tendrán dos usos diferentes: cuando vayamos a crear una nueva sesión, pasarán a estar activos y podremos introducir los parámetros que debe poseer la sesión (figura 4.5); y cuando seleccionamos una sesión de la lista, en estos campos se mostrará toda su información, permaneciendo los campos inactivos, ya que esta información será inalterable desde el momento en que se registre la sesión en el sistema (figura 4.4).

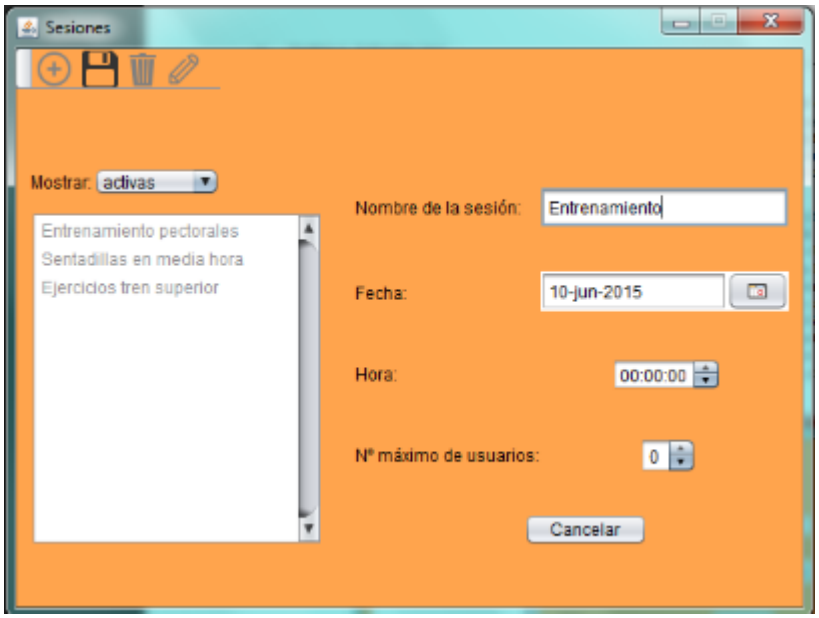

Figura 4.5 Ventana de las sesiones en el momento de introducción de una nueva sesión.

Para facilitar la introducción de fechas en la aplicación, hemos utilizado la librería *JCalendar*, gracias a la cual podemos emplear la clase *JDateChooser*. Esta clase le muestra al usuario un desplegable como el que vemos en la figura (4.6), con el cual elegir la fecha que desee para la sesión simplemente haciendo click sobre ella.

Figura 4.6 Panel desplegable de selección de fecha de inicio de la sesión.

Las sesiones pueden tener dos estados diferentes, activas o no activas, dependiendo de si han finalizado o no. Se pretende poder filtrar las sesiones mostradas según este criterio, pudiendo también mostrar tanto unas como otras a la vez, cumpliendo así el requisito R20. Para ello vamos a utilizar un objeto del tipo *DynamicListSesiones*, que extenderá el tipo *JList<String>* para mostrar la lista por la pantalla. En este objeto tendremos tres *ArrayList<String>* con cada una de las posibles listas de sesiones a mostrar (activas, no activas o todas las sesiones), y que se actualizarán cada vez que se cree una nueva sesión o alguna sesión cambie de estado. La lista que debe mostrarse dependerá de la opción seleccionada en el *JComboBox* que vemos en la figura (4.4). Además, se detectará que se ha hecho doble click sobre una sesión gracias al *MouseListener* asociado a la lista. Tras el doble click, la sesión quedará seleccionada y se mostrarán su

información por pantalla, rellenando los campos que inicialmente se encontraban vacíos, como podemos observar en la figura (4.4), cumpliendo así con el requisito R19.

También disponemos de un completo menú desde el que poder realizar las distintas acciones sobre las sesiones, que serán, de izquierda a derecha, crear nueva sesión (R17), guardar nueva sesión en el sistema, eliminar la sesión seleccionada (R18) y editar la sesión seleccionada. Estas acciones se irán activando según en qué momento podemos utilizarlas, por lo que inicialmente solo podremos crear nuevas sesiones, permaneciendo las demás opciones desactivadas. Cuando hacemos click en la opción de nueva sesión, se habilitarán todos los campos situados a la derecha de la pantalla (figura 4.4), que deberán rellenarse con la información de la nueva sesión, la cual almacenaremos en el sistema utilizando el botón de guardar sesión, que ya se aparecería activo.

Por otro lado, las opciones de borrar y editar sesión solo aparecerán activas tras seleccionar una sesión de la lista haciendo doble click sobre ella. Antes de borrar una sesión, al usuario se le mostrará un mensaje de confirmación mediante un *JOptionPane* (figura 4.7) y, en caso de confirmar la eliminación, la sesión será eliminada en la base de datos del servidor. Por último, si el usuario decide editar la sesión, se creará un nuevo objeto del tipo *VentanaEjercicios*, que nos mostrará la siguiente ventana de la aplicación (figura 4.8).

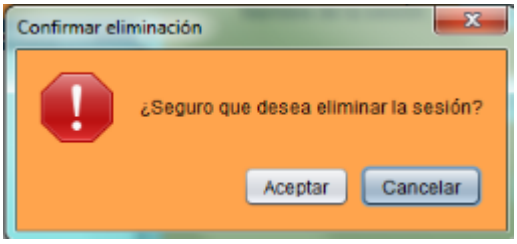

Figura 4.7 Mensaje de confirmación para borrar una sesión.

Desde esta ventana gestionaremos todo lo referente a una sesión; es decir, podremos añadir y eliminar ejercicios (R21 y R22), así como visualizar a los usuarios mientras realizan estos ejercicios (R25). En cuanto a la disposición de la misma, como vemos en la figura (4.8), en la esquina superior izquierda se nos presentan dos listas distintas, una con los ejercicios que ya están incluidos en la sesión, que será la situada más a la izquierda, y otra con los que no, dispuesta a la derecha de la anterior. Entre estas dos listas tendremos dos botones que se utilizarán para incluir o eliminar ejercicios de la sesión respectivamente. Bajo las listas se dispone un panel en el cual se mostrará toda la información relativa a un ejercicio seleccionado. Y por último, en la zona derecha de la pantalla habrá cuatro reporductores de vídeo, en los cuales podremos ver simultaneamente hasta cuatro usuarios diferentes mientras realizan sus ejercicios y sobre los cuales habrá un botón con el cual dar por terminada la sesión.

Centrémonos primero en el desarrollo de la zona izquierda de esta ventana. Como en el caso de las sesiones, es necesaria la implementación de unas listas, pero en esta ocasión para mostrar ejercicios. Basándonos en lo realizado anteriormente, implemetaremos una clase que extienda a *JList<String>*, que será *DynamicListEjercicios*. Como ya hemos mencionado, tendremos dos listas diferentes, una con los ejercicios de la sesión y otra con el resto de ejercicios. El contenido de estas listas variará según los ejercicios que el monitor decida que deben realizarse en esta sesión: si un ejercicio pasa a formar parte de la sesión, éste deberá desaparecer de la lista del resto de ejercicios y pasar a formar parte de la de ejercicios incluídos en la sesión; y al contrario si se elimina un ejercicio de la sesión.

Además, *DynamicListeEjercicios* tendrá asociado un *MouseListener* para detectar el doble click sobre uno de los ejercicios, tras lo cual el ejercicio quedará seleccionado. En el momento que seleccionemos un ejercicio, se mostrará, como vemos en la figura (4.8), su duración y descripción en el recuadro de color marrón. Con esto pretendemos que el monitor disponga de toda la información del ejercicio (R23) y, basándose en ésta, pueda decidir si es adecuado o no para el entrenamiento que está configurando. La clase *TextBubbleBorder* únicamente la utilizaremos para conseguir los bordes redondeados del recuadro de información de ejercicio.

Por otro lado, tenemos la zona de reproductores, en la que el monitor podrá ver el streaming de vídeo de cualquiera de los usuarios inscritos en la sesión. A continuación expondremos las dificultades encontradas a la hora de implementar esta funcionalidad y cómo se han abordado para su correcta resolución.

Ya mencionamos anteriormente que en este proyecto el envío de vídeo será directamente entre la aplicación Android y la del monitor. Para que esto sea posible, debe existir una comunicación entre las dos aplicaciones,

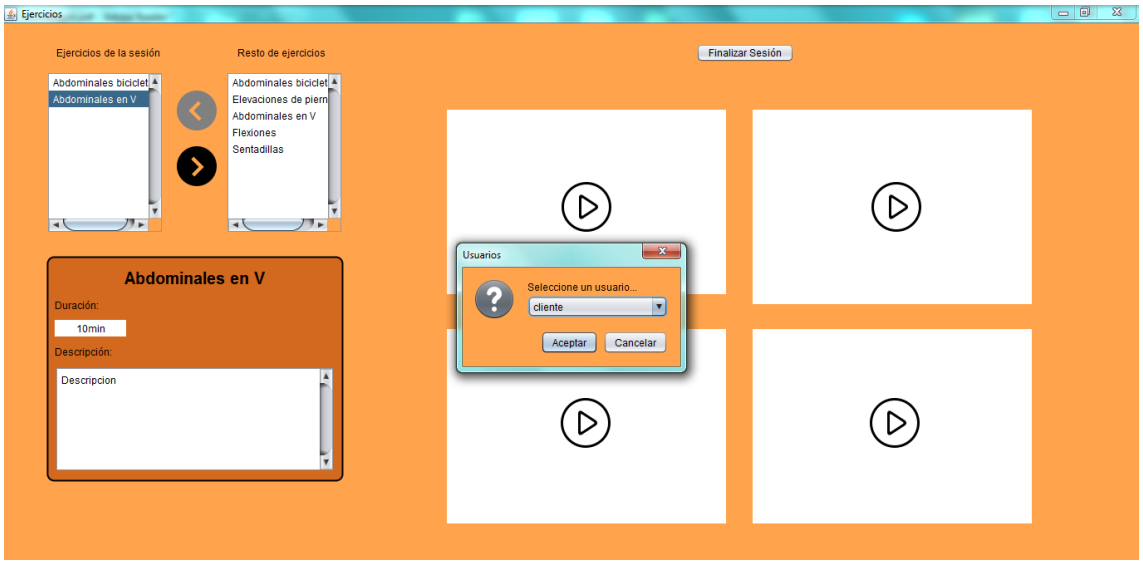

Figura 4.8 Ventana de gestión de una sesión.

que transcurrirá de la siguiente manera: primero, cuando el monitor esté preparado para comenzar una sesión, la aplicación quedará a la espera de los usuarios; a partir de este momento el usuario podrá comenzar a realizar la sesión. Cuando la aplicación Android esté preparada para el envío de vídeo, contactará con la aplicación Java, consiguiendo así la dirección IP de cada uno de los usuarios que van a hacer una sesión; y se quedará a la espera de la petición de vídeo por parte del monitor; cuando el monitor desee ver a cierto usuario, realizará dicha petición a la dirección del usuario que anteriormente ha almacenado; y, por último, la aplicación Android comenzará a enviar el vídeo, que se reproducirá en nuestra aplicación Java, como podemos ver en la figura (4.9).

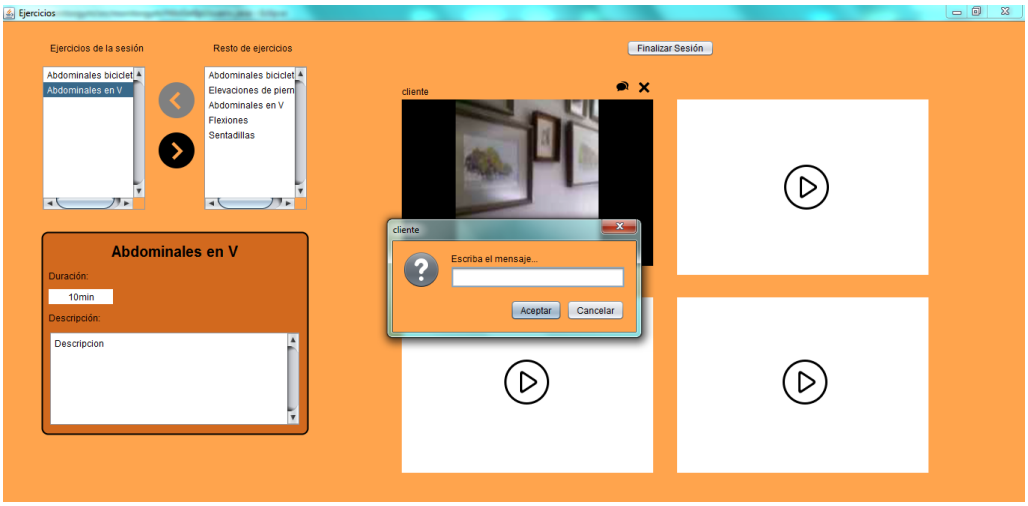

Figura 4.9 Reproducción vídeo del usuario.

El primer problema que nos encontramos es que el monitor debe conocer la dirección IP del dispositivo que está utilizando el usuario para poder realizar la petición del vídeo, por lo que almacenaremos en un listado las direcciones de todos los usuarios que vayan a realizar la sesión de ejercicios. Llegados a este punto, podríamos encontrarnos con dos situaciones diferentes: el usuario de la aplicación Android utiliza su conexión de datos móviles, por lo que el dispositivo tendría asociada una dirección IP pública; o por el contrario, el usuario está conectado a un punto de acceso WiFi, por lo que en este caso dispondría de una dirección IP privada. El verdadero problema reside en el segundo caso, ya que en el primero de ellos bastaría con que la aplicación de usuario enviara al monitor la IP del dispositivo en el que se esté ejecutando.

Para solucionar esta situación vamos a implementar un sistema de registro de usuarios. En el monitor se

ejecutará una instancia de la clase *HiloGetIpUsuario* que, como su nombre indica, extenderá la clase *Thread* de Java. Hemos optado por la utilización de hilos porque no sabemos en qué momento llegarán los mensajes del usuario, con lo que conseguimos que la aplicación no quede a la espera de las mismas, sino que el monitor podrá seguir trabajando con independencia de los usuarios. Lo primero que hará esta clase será utilizar el protocolo UPnP gracias a la librería *Weupnp*, haciendo uso de la clase *SelectorPuertoNat.* Este protocolo lo utilizamos para poder asignar un puerto del NAT[2] a nuestro equipo utilizando lo que se conoce como mapeo de puertos. En nuestra aplicación está configurada por defecto para utilizar el puerto 54321, pero en caso de que éste ya estuviera asociado a otro equipo, intentará buscar un puerto libre entre los 10 siguientes.

Ésto solo tendría sentido si el equipo estuviera conectado a un NAT, por lo que en caso contrario continuaríamos con la ejecución por los siguientes pasos. Esto lo conseguimos gracias a que en la clase *SelectorPuertoNat* se comprueba que nuestro equipo esté conectado a un NAT. Si no estamos conectados a ninguno, no se realizará la petición del puerto.

El objetivo que buscamos con todo esto es que se entregen a esta aplicación todos los mensajes que lleguen a nuestro router con puerto destino el que hayamos elegido. Para ello necesitamos un socket esperando las conexiones en ese mismo puerto, por lo que *SelectorPuertoRouter* deberá devolver de alguna forma el puerto que finalmente ha seleccionado. Una vez ya no necesitemos más este puerto, utilizaremos el método de *SelectorPuertoNat* con el que lo liberamos en el NAT, para que pueda volver a utilizarlo con normalidad. Este método se ejecutará cuando pulsemos sobre el botón de fin de sesión (figura 4.9). Además, tras pulsar este botón el estado de la sesión pasará a ser no activa.

El socket que creamos quedará a la espera de la conexión por parte de la aplicación Android. Cada vez que el usuario vaya a comenzar una sesión, conectará automáticamente con este socket, enviándole el nombre del usuario que va a hacer los ejercicios. En el monitor se recibirá este mensaje, del cual se obtendrá el nombre y la dirección IP y puerto origen, extrayéndola del propio paquete IP. Esta dupla dirección-puerto corresponderá con la dirección del dispositivo, si éste dispone de una dirección pública, o con la IP y puerto que el NAT del punto de acceso WiFi haya asociado al equipo del cliente en caso de que la dirección del dispositivo del usuario sea privada. Podemos ver este proceso en el diagrama de paso de mensajes de la figura (4.10). Con estos dos pasos hemos solucionado el problema de las direcciones IP privadas tanto en el lado del monitor como en el del cliente, ya que los equipos serán totalmente accesibles desde cualquier subred.

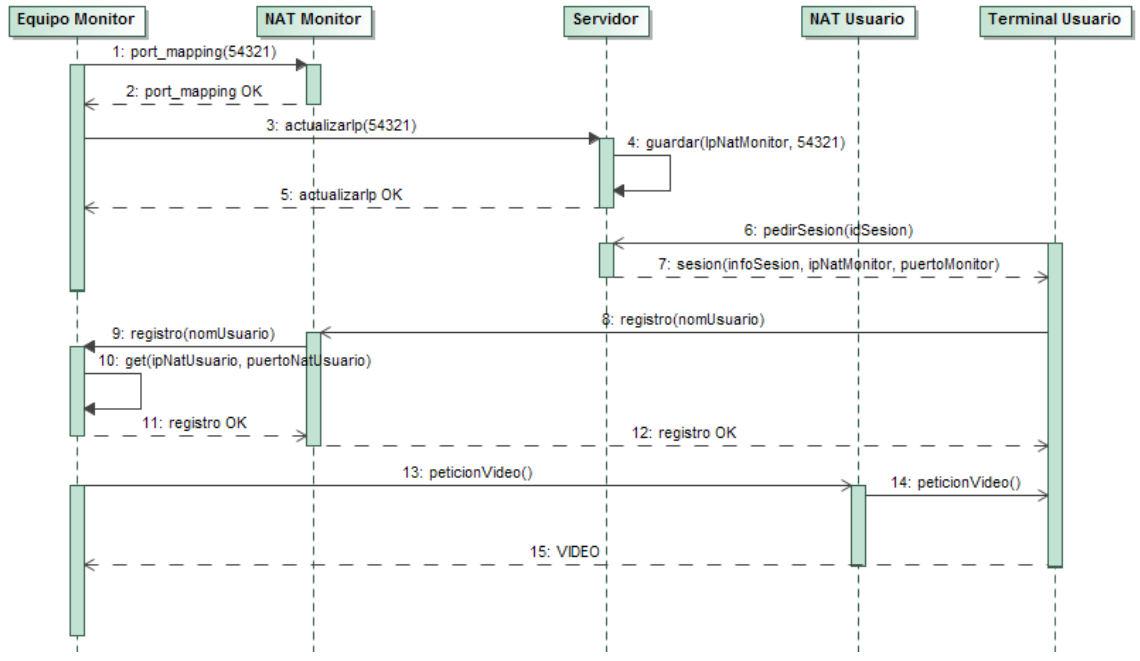

Figura 4.10 Paso de mensajes durante la petición de vídeo.

Para entender mejor la solución a este problema primero debemos comprender el funcionamiento de un NAT, para lo que se ha incluído la figura (4.11). Por otro lado, la figura (4.10) precisa de una explicación

[2] NetWork Address Translation

más detallada para comprender la solución elegida. Como podemos ver, tanto el monitor como el usuario se encuentran detrás de un NAT, por lo que todos los mensajes, tanto de uno como del otro, sufrirán las traducciones de direcciones correspondientes cada vez que pasen por el NAT. Lo primero que hará la aplicación Java será el mapeo de puerto, tras lo que enviará un mensaje al servidor indicándole el puerto elegido. Al llegar este mensaje al servidor, se tomará la IP origen del paquete, que corresponderá con la dirección del NAT del monitor, y el puerto que la aplicación ha reservado en el NAT. Ya tendríamos el problema de las direcciones privadas resuelto en el lado del monitor, porque esta dirección almacenada será enviada al usuario cuando decida realizar una sesión creada por el monitor, por lo que ya tendría la IP y el puerto a los cuales conectarse para realizar el registro en la sesión.

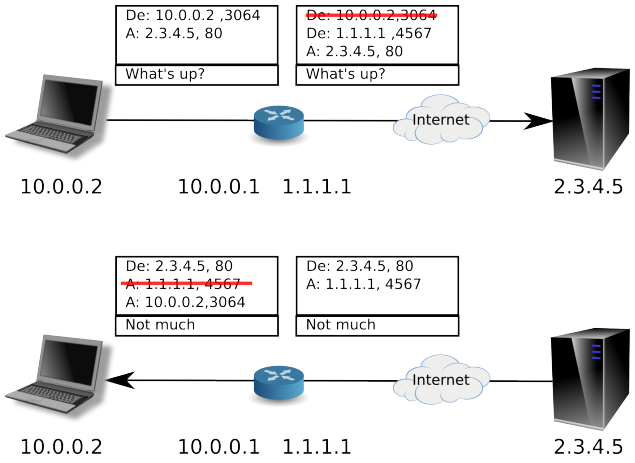

Figura 4.11 Imagen aclaratoria de la traducción de direcciones en un NAT.

Ya tenemos conexión en el lado del monitor, pero como ya hemos mencionado también deberemos acceder al usuario para solicitar el vídeo. La aplicación Android reberá realizar un registro en la sesión que vaya a realizar mediante un mensaje al monitor. En el momento que el mensaje de registro del usuario pasa por el NAT, la dirección IP y el puerto sufrirán la traducción. Cuando el paquete llega al monitor, se toman de el la IP y puerto origen, que corresponderán a los que debemos acceder en el router para que el paquete llegue al usuario. Para que esto sea posible, la petición de vídeo debe llegarle al usuario en el puerto 8080, por lo que tendremos que forzar a que el mensaje de registro salga por este puerto, para que en el NAT se haga la traducción inversa cuando llegue una petición de vídeo por parte del monitor, entregándose así en el puerto destino correcto. Así hemos conseguido que la conexión sea posible en las dos direcciones.

Aunque con esta implementación hemos solucionado el problema, cabe mencionar que no siempre funcionará. Esto es debido a que, aunque la mayoría de routers soportan el protocolo UPnP hoy en día, dependeremos de que esté configurada la opción de este protocolo. Algunos routers traen esta opción configurada por defecto, pero como no podemos englobar el 100 % de los casos, se considerarán diferentes opciones como líneas de mejoras futuras para solucionar este problema, las cuales podemos ver en el capítulo (7).

Una vez que ya conocemos la dirección de los clientes, ya podemos realizar las peticiones de vídeo, para lo que utilizaremos la clase *HiloVideo*, que implementará la interfaz *Runnable*. Esto se debe a que, como será posible reproducir a la vez el vídeo de 4 usuarios diferentes, vamos a utilizar para cada uno de ellos un hilo. Así conseguimos que la reproducción de cada vídeo sea independiente al hilo principal del programa, pudiendo así continuar con la ejecución normal de la aplicación independientemente de cuántos vídeos estén reproduciéndose a la vez y de si ha habido un error en la reproducción de alguno de ellos. Además, *HiloVideo* también extenderá la clase *WindowsCanvas* de la librería *VLCj*. Esta clase representa el área rectangular donde se reproducirá el vídeo y recibirá como parámetro la URL a la que acceder para obtener el vídeo y realizará la petición a dicha dirección (R24). Por otro lado, *HiloVideo* se utilizará como parámetro a la hora de crear una nueva instancia de la clase *Thread*, consiguiendo así la reproducción en paralelo de todos los vídeos. Cuando cada hilo comience a ejecutarse, comenzará la reproducción del vídeo y se realizarán ciertos cambios en la pantalla, como podemos apreciar en la figura (4.12), los cuales comentaremos a continuación.

Observando las diferencias entre la figura (4.12) y la figura (4.8) podemos ver que ahora se muestra sobre el reproductor el nombre del usuario que estamos viendo. Por otro lado, sobre la esquina superior derecha del reproductor, han aparecido dos pequeños botones. El primero de ellos, el que encontramos más a la derecha,

Figura 4.12 Sección de pantalla con reproductor de vídeo activo.

servirá para dejar de reproducir el vídeo actual (R26), mientras que el segundo lo utilizaremos en caso de querer comunicarnos con el usuario (R27). Si hacemos click sobre el botón de comunicación con el usuario, se nos mostrará un *JOptionPane*, el cual podemos ver también en la figura (4.12). En este cuadro de diálogo se habilita un espacio en blanco en el que el monitor podrá escribir el mensaje que desea comunicarle al usuario, que será enviado a través de la red hasta llegar a él. Para el envío de estos mensaje utilizaremos la clase *HiloMensaje*, que también extenderá la clase *Thread* y que enviará a través de un socket conectado con el cliente el mensaje que haya escrito el monitor.

Por otro lado, cuando pulsamos el botón de cerrar vídeo dejará de mostrarse a dicho usuario, interrumpiendo la ejecución del hilo. Lo mismo ocurrirá en caso de detectarse que la aplicación Android ha dejado de transmitir vídeo. Una vez interrumpido el hilo, se deja de mostrar el vídeo por pantalla y se vuelve al estado inicial desde el que partimos (figura 4.8).

5 Aplicación Android

En este capítulo se expondrá la aplicación *'ProjectGym'* y profundizaremos en los cambios realizados en la misma con respecto a otros proyectos [2]. Ésta será la aplicación que utilicen los usuarios del sistema para realizar las sesiones de ejercicios creadas por el monitor mientras los supervisa en tiempo real. La función principal de la aplicación es realizar streaming de vídeo desde la cámara del dispositivo hacia la aplicación del monitor.

Para el streaming de vídeo no utilizaremos las mismas librerías que utilizaba la aplicación original. En este caso, nos basaremos en la aplicación de código abierto *peepers* [15], que funciona como servidor a la espera de peticiones para enviar el vídeo, lo cual es perfecto de acuerdo a la estructura que nos marcamos como objetivo al comienzo de nuestro proyecto (Capítulo 1). Básicamente lo que haremos será eliminar las instancias de otras librerías de vídeo y acoplar *peepers* en nuestra aplicación.

En cuanto a nuevos requisitos del sistema, no vamos a marcar ninguno, ya que pretendemos que las funcionalidades de la aplicación sigan intactas, con la única diferencia de que el streaming de vídeo no será contra ningún servidor, sino contra el propio monitor. A continuación pasamos a comentar el funcionamiento de la aplicación, así como los cambios realizados.

5.1 Desarrollo

Como hemos hecho con la aplicación Java, vamos a ir comentando las clases y layouts que intervienen en la aplicación en el orden en que se muestran cuando el usuario realiza una ejecución normal de la aplicación. La actividad *InicioSesion* se lanzará al ejecutar la sesión. En ella se muestra un formulario de inicio de sesión (figura 5.1) que deberá rellenar el usuario con sus credenciales, las cuales serán enviados al servidor para comprobar si son correctas. Si no son correctas se le mostrará un mensaje de error al usuario.

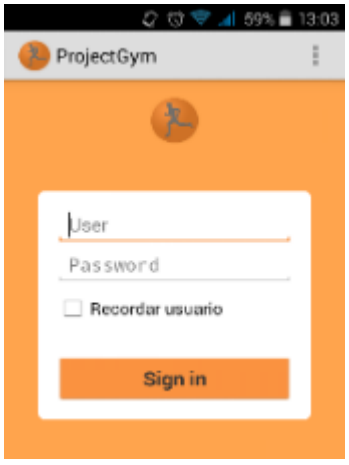

Figura 5.1 Formulario de inicio de sesión.

En la actividad *ListaSesiones* se obtienen del servidor todas las sesiones activas y se crea un listado con ellas, el cual podemos ver en la figura (5.2). Además, se añade un detector de eventos para detectar que el listado se estira hacia abajo. Cuando esto ocurre, se realiza de nuevo la petición de sesiones y se recarga la lista. Los elementos de la lista tendrán asociado un *listener* para que, cuando el usuario pulse sobre alguno de ellos, lanzar la actividad *ListaEjercicios*. Aquí hay una pequeña modificación con respecto a la aplicación inicial, ya que como realizaba peticiones de información al servidor del monitor, necesitaba conocer su dirección IP, la cual solicitaba al servidor del gimnasio en este instante. Como en este proyecto se pretende que toda la información quede almacenada únicamente en el servidor del gimnasio, no se realizará dicha petición al servidor.

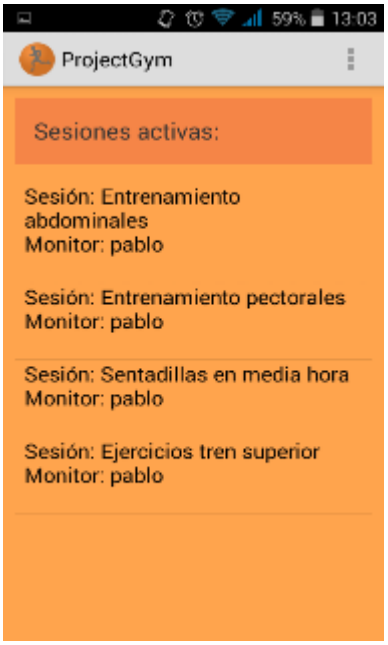

Figura 5.2 Lista de sesiones activas.

Cuando se lance *ListaEjercicios*, se realizará una petición al servidor para que éste nos devuelva todos los ejercicios que componen la sesión. En el proyecto anterior [2] esta petición se hacía contra el servidor del monitor pero, como hemos mencionado antes, modificamos la aplicación para que esta petición sea contra el servidor del gimnasio. Además, otra nueva modificación será la solicitud de toda la información relativa a la sesión, ya que necistaremos la fecha de comienzo de la misma y la dirección IP del monitor. Por otro lado, al usuario se le mostrará una lista con los ejercicios (figura 5.3), en la que cada ejercicio tendrá asociado un *listener* que, tras comprobar que la sesión haya comenzado, lanzará la actividad *Ejercicio*. Para considerar que la sesión haya comenzado deben darse dos circunstancias: que nos encontremos en un instante posterior a la fecha y hora de comienzo de la sesión, y que haya sido posible registrarnos en la aplicación del monitor. Explicamos a continuación cómo realizamos estas comprobaciones.

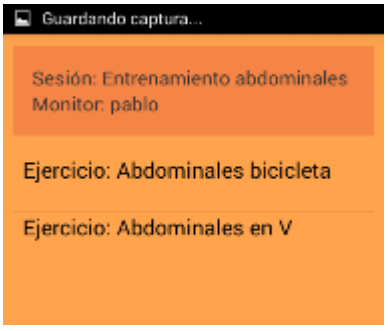

Figura 5.3 Lista de ejercicios de la sesión.

Ya sabemos que no podremos empezar la sesión hasta el día y la hora indicados por el monitor, lo que implica que lo único que podremos hacer antes de que la sesión comience será observar los ejercicios que la componen. Para comparar la fecha de la sesión con la actual se ha creado un método, que devolverá true si la fecha y hora devueltas por el servidor son anteriores a las del equipo donde se ejecuta la aplicación.

Por otro lado, la dirección IP del monitor la necesitamos porque, como se ha comentado en el capítulo (4), antes de comenzar la sesión el usuario deberá registrarse enviando un mensaje con su nombre a la aplicación del monitor. Para poder conectar con la aplicación del monitor se ha creado la clase *SocketAsync*, la cual extenderá a clase *AsyncTask* para poder crear un socket con el que contactar con el socket abierto en el monitor. Una vez conectados, el cliente le enviará su nombre de usuario y ya quedará registrado en el monitor. Por el contrario, si no ha sido posible establecer la conexión, eso indica que el monitor aún no ha comenzado la sesión, por lo que nos mantendremos a la espera de que empiece. Esta funcionalidad también es un añadido a la aplicación original, ya que el cliente no precisaba de ningun registro.

Una vez ya comenzada la sesión, al pulsar sobre un ejercicio se lanzará la actividad *Ejercicio*, que realizará una petición al servidor de toda la información del ejercicio que se va a realizar y descargará de una URL asociada al ejercicio el vídeo informativo del mismo, el cual se mostrará por pantalla, como vemos en la figura (5.4). La única modificación con respecto a la aplicación original es que la URL del vídeo no será fija, por lo que tendremos que tomarla de los campos del ejercicio devueltos por el servidor. La URL que recibamos puede ser o completa, en cuyo caso descargaremos directamente el vídeo de donde se nos indique; o solo el directorio donde se encuentra el vídeo sin especificar ninguna dirección IP, lo que significará que el vídeo se encuentra en el servidor, por lo que entonces accederemos al directorio indicado en dicho servidor. Esto lo hacemos así para facilitar el cambio del servidor donde se encuentran los vídeos sin tener que cambiar una a una las URL. Además, también se mostrará, en la esquina inferior derecha del dispositivo, el vídeo capturado por la cámara del terminal, que será lo que se envíe al monitor.

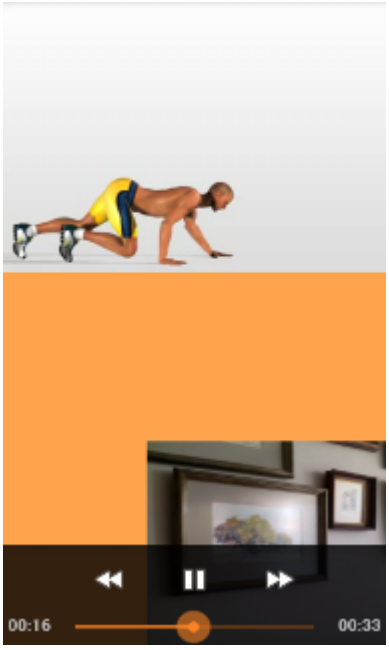

Figura 5.4 Reproducción vídeo del ejercicio.

Uno de los objetivos marcados era que debe existir comunicación entre el monitor y el usuario, ya que no tendría sentido la monitorización sin una realimentación por parte del monitor que nos indique si estamos realizando correctamente los ejercicios. Para ello, en la clase *Ejercicio*, se ejecutará un hilo que quedará a la escucha de mensajes del monitor. Una vez recibido un mensaje, éste se mostrará inmediatamente por pantalla (Figura 5.5). Por otro lado, también debe ser posible ver la información de cierto ejercicio mientras lo hacemos, para lo que se ha implementado un *listener* que detecta las pulsaciones sobre la pantalla. Cuando el usuario toque la pantalla, se mostrará toda la información relativa al ejercicio que se esté realizando en ese momento (figura 5.6).

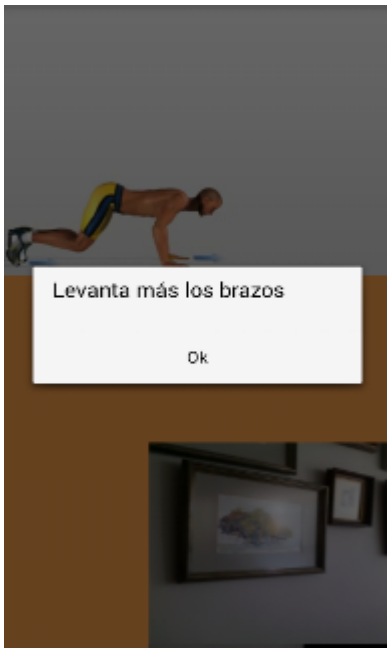

Figura 5.5 Mensaje del monitor.

Merece una mención aparte las modificaciones realizadas para la captura y el envío del vídeo, que sufrirá importantes cambios en este proyecto. Como nos marcamos al principio del proyecto, deberá desaparecer el servidor Wowza utilizado en el proyecto de Antonio Clavain [2], por lo que como solución alternativa se ha optado por implementar en el cliente un servidor de vídeo haciendo uso de la aplicación de código abierto *peepers* [15]. No es objetivo de este proyecto entrar en detalle el funcionamiento de esta aplicación, pero si vamos a comentar qué hacen las clases más importantes que la componen, las cuales van a ser incluídas en nuestra aplicación Android. La clase *StreamCameraActivity* se encarga de cargar las preferencias de la cámara, mostrar lo que se está grabando por pantalla y crear una nueva instancia de la clase *CameraStreamer*. En nuestro proyecto esta clase no será una nueva actividad, sino un fragment que incluiremos en la actividad *Ejercicio*, permaneciendo el resto de las clases exactamente igual a como nos las encontramos en la aplicación *peepers*. *CameraStreamer* será la clase encargada de, tras diferentes comprobaciones, lanzar la clase *MJpegHttpStreamer*, encargada del streaming de vídeo a través de HTTP.

Al poner en funcionamiento la aplicación, observamos que la imagen capturada tiene un giro de 90º, por lo que nos hemos visto obligados a solucionar este problema tanto en el lado del cliente como en el del monitor. En la aplicación del cliente lo que hacemos será girar la imagen mostrada al usuario, añadiendo una línea de código en la clase *CameraStreamer*. Por otro lado, en la aplicación Java lo que hemos hecho ha sido girar igualmente el reproductor VLC, quedando así solventado el problema.

Por último, como recordaremos, uno de los problemas del proyecto anterior se encontraba en el retardo que sufría el vídeo a la hora de reproducirlo en el monitor. Tras las modificaciones realizadas hemos ejecutado las aplicaciones para comprobar si este objetivo se cumple, obteniendo un resultado de unos 1 o 2 segundos de retardo en la mayoría de las ejecuciones realizadas, por lo que entendemos que este objetivo ha sido alcanzado satisfactoriamente.

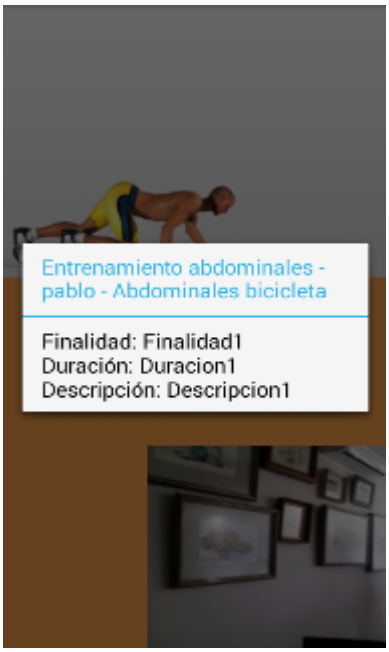

Figura 5.6 Información del ejercicio.

6 Funcionamiento del sistema

En esta sección se mostrarán varios diagramas UML[1], en concreto diagramas de secuencia, en los que se detallará la interacción entre los diferentes elementos que componen nuestro sistema para el correcto funcionamiento de las aplicaciones. El objetivo es aclarar cómo se ha integrado nuestra aplicación en el sistema ya existente, así como que el lector pueda entender el funcionamiento del código implementado de una forma más rápida y sencilla.

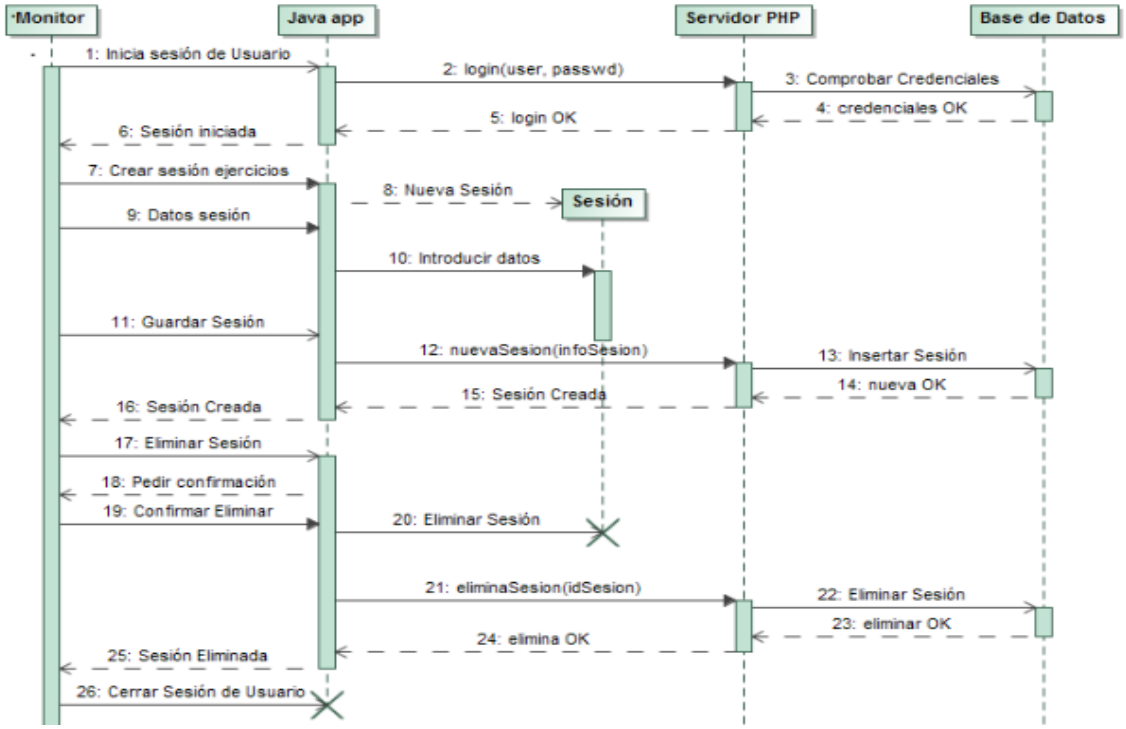

Figura 6.1 Ciclo de vida de una sesión en nuestro sistema.

En el diagrama de la figura (6.1) se representa el ciclo de vida de una sesión de entrenamiento desde que se crea hasta que se destruye. El monitor deberá iniciar sesión en el sistema introduciendo sus credenciales, tras lo cual la aplicación Java las enviará al servidor para comprobar si son correctas. Una vez dentro del sistema, el monitor podrá crear una nueva sesión o eliminar una ya existente. Si el monitor desea crear una nueva sesión, deberá introducir toda la información relativa a la misma, la cual será enviada al servidor para ser almacenada en la base de datos. De igual forma, las sesiones pueden ser eliminadas si así se desea, debiendo confirmar el monitor que quiere eliminarla para evitar borrar una sesión por error. Una vez confirmada la

[1] Unified Modeling Lenguage

eliminación, se enviará una petición al servidor para que elimine de la base de datos toda la información relativa a la sesión seleccionada.

Por otro lado, como ya se mencionó en el capítulo (4), las sesiones también pueden ser editadas. El monitor tiene total libertad a la hora de elegir los ejercicios que componen la sesión, para lo cual la aplicación se comportará como se muestra en la figura (6.2). Cuando el usuario eliga editar una sesión, el sistema se encargará de solicitar todos los ejercicios al servidor, mostrándoselos al usuario, tras lo cual, éste podrá seleccionar aquellos que crea oportunos e incluirlos en la sesión. Para añadir los ejercicios a la sesión se realizará una petición al servidor indicándole que debe realizar las operaciones necesarias para que ésto quede reflejado en la base de datos. Al igual que se pueden añadir ejercicios, el usuario de la aplicación deberá poder eliminarlos. El procedimiento que sigue el programa es similar al anterior, realizando una petición al servidor indicándole que debe retirar un ejercicio concreto de la sesión indicada.

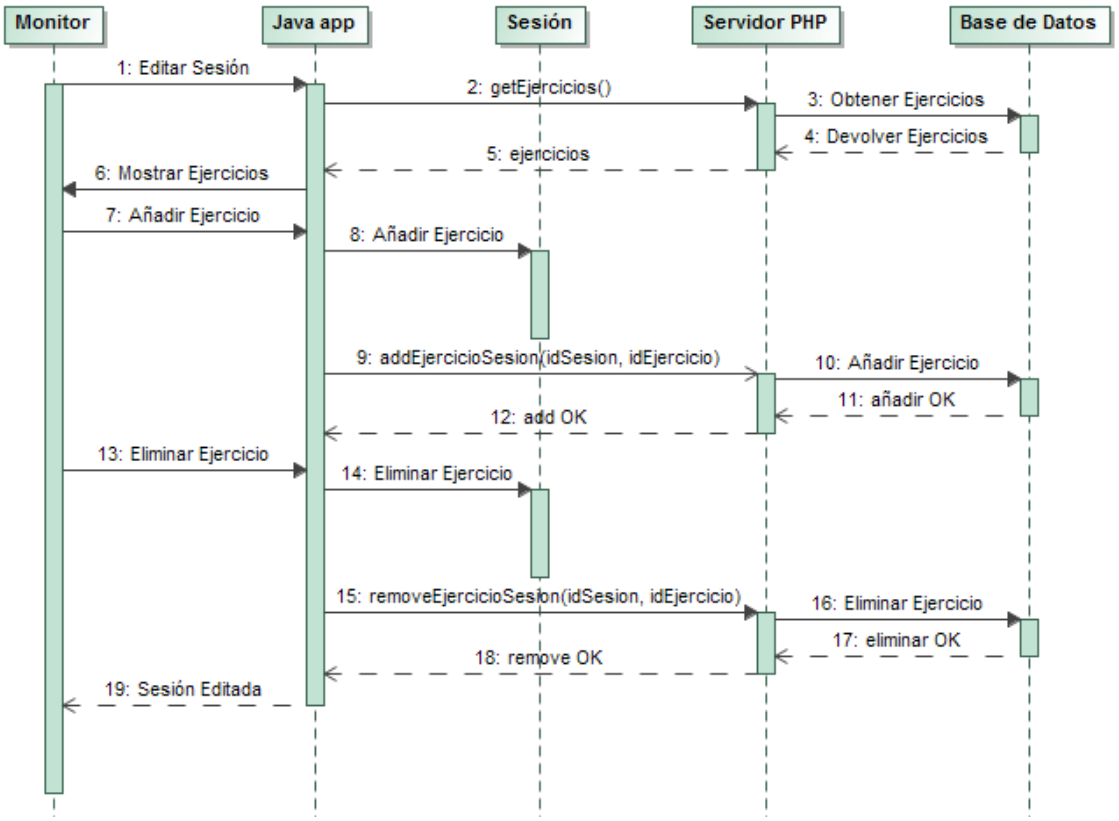

Figura 6.2 Proceso de edición de una sesión.

Nuestra aplicación no solo interactuará con el servidor, ya que el vídeo de los usuarios lo tomará directamente de la aplicación Android. En la figura (6.3) se muestra con todo detalle los mensajes enviados a la hora de reproducir el vídeo en la aplicación Java.

Comenzando por la secuencia en la aplicación Android, el usuario deberá suscribirse a una sesión, para lo cual se enviará la correspondiente petición al servidor. A su vez, la aplicación Android solicitará todos los ejercicios que componen la sesión a la que se acaba de suscribir el usuario para mostrárselos por pantalla. El usuario seleccionará el ejercicio que desea realizar y la aplicación comprobará que la sesión ha comenzado. Si ha comenzado, el usuario deberá registrarse en la aplicación Java, como ya mencionamos en capítulos anteriores (4 y 5). Posteriormente solicitará toda la información del ejercicio y el vídeo del mismo, y también empezará a grabar. Al usuario se le mostrará tanto el vídeo del ejercicio como lo que está grabando con la cámara del dispositivo. Para ver la información del ejercicio, el usuario deberá pulsar sobre la pantalla del dispositivo si así lo desea, mostrándosele así un mensaje con toda la información relevante del ejercicio.

Por otro lado, con respecto a la aplicación Java, cuando el monitor quiera reproducir el vídeo de algún usuario, se le mostrarán por pantalla todos los que se hayan registrado, pudiendo elegir el que desee ver por

pantalla. Cuando haya seleccionado a un usuario, se realizará la petición a la aplicación Android correspondiente, que siempre será al puerto 8080 del usuario, recibiendo el vídeo como respuesta y mostrándolo por pantalla. Mientras se muestra el vídeo por pantalla, el monitor puede enviar mensajes al usuario, los cuales se mostrarán por pantalla en el destino. Finalmente, el monitor puede cerrar el vídeo que está viendo, terminando así el envío de vídeo.

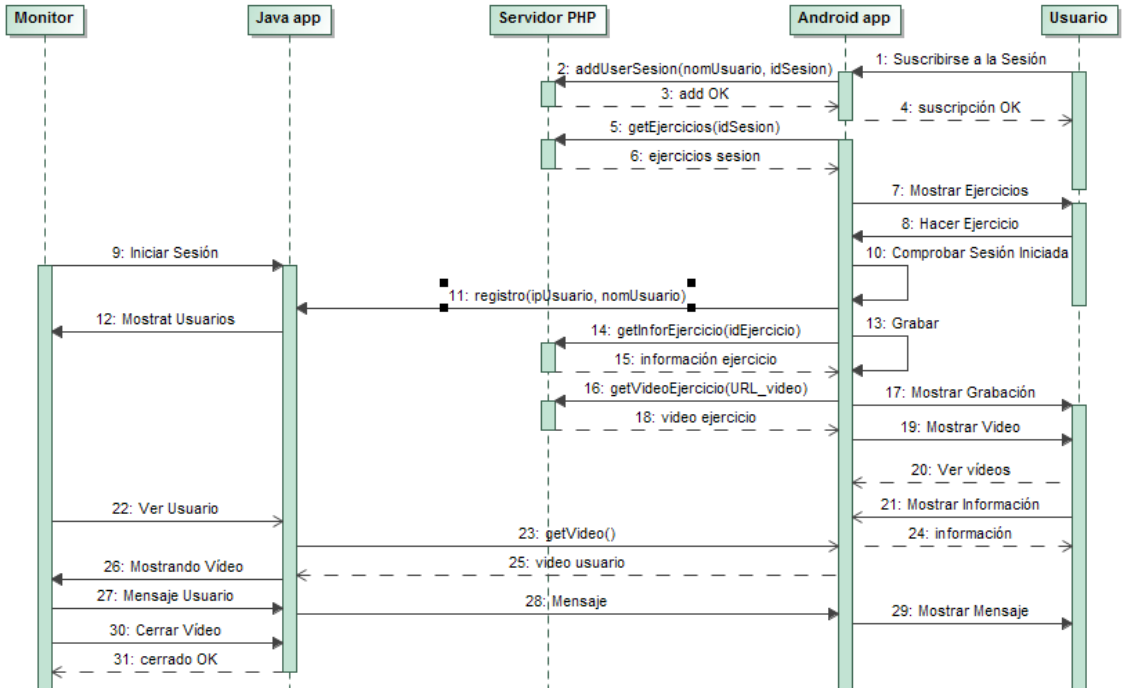

Figura 6.3 Interacción para la reproducción del vídeo de un usuario.

7 Conclusiones

Este proyecto ha supuesto un reto al integrar nuestra aplicación con los proyectos de los que partíamos. Esto ha implicado un estudio de los mismos no solo para conocer su funcionamiento, lo cual ha sido fundamental para elegir las tecnologías que mejor se adapten a ellos, sino también para realizar las modificaciones del código que han sido necesarias, tanto en el servidor web como en la aplicación Android.

Por otro lado, al desarrollo web y de aplicaciones para Android de los anteriores proyectos, en éste incluiremos el desarrollo de aplicaciones Java, combinando así diversos tipo de programación. Además, ésto implica la integración de las diferentes aplicaciones desarrolladas no solo entre ellas, sino con el resto de tecnologías utilizadas en el proyecto, lo que ha supuesto un interesante desafío. Otro escollo a salvar ha sido la reimplementación del envío de vídeo en la aplicación Android, utilizando *MJPEG* sobre *HTTP*, tecnología completamente desconocida antes de la realización del proyecto, en vez del envío de vídeo sobre *RTSP* que se utilizaba originalmente. Por último, las direcciones privadas han supuesto un gran problema en este proyecto, ya que desde un inicio pretendimos que la comunicación entre las aplicaciones fuera totalmente transparente para el usuario, ya que éste no tiene por qué tener ningún conocimiento de telecomunicaciones. Para ésto hemos utilizado el protocolo *UPnP*, el cual nos ha permitido comunicarnos con el NAT desde la aplicación Java, otorgándonos cierto control sobre las traducciones de direcciones IP.

7.1 Objetivos logrados

Una vez finalizado el proyecto, vamos a comprobar si cumplimos los objetivos que nos marcamos al comienzo del mismo. Los objetivos que hemos cumplido son:

- Desarrollo de una aplicación Java para el monitor.
- El monitor puede gestionar las sesiones desde dicha aplicación.
- La aplicación permite la recepción y reproducción del vídeo de los usuarios.
- El monitor puede enviar mensajes de texto al usuario.
- Se ha eliminado el flujo de información entre el usuario y el monitor, por lo que ahora se almacena en la base de datos un historial con los usuarios inscritos a una sesión, así como los ejercicios que componen cada una de las sesiones.
- Se ha disminuido el retardo de vídeo, pasando de los aproximadamente 5 segundos del proyecto anterior a estar en un rango de 1-2 segundos.
- Desaparece el servidor de vídeo del sistema, pasando a ser el envío de vídeo directamente del usuario al monitor.

Por lo tanto, podemos afirmar que se han alcanzado los objetivos marcados al comienzo del proyecto.

7.2 Líneas de mejora

Que se hayan cumplido todos los objetivos no implica que la aplicación no sea susceptible a mejoras, por lo que a continuación se muestran diferentes líneas de mejoras del mismo:

- Implementar un buscador de ejercicios para facilitar el trabajo de búsqueda del monitor y para, en caso de que hubiera una gran cantidad de ejercicios almacenados en la base de datos, no se muestren todos a la vez por pantalla.
- Facilitar la comunicación con el cliente implementando envío de mensajes de voz, siendo más cómodos para el cliente que los mensajes de texto, ya que podría escucharlos mientras sigue haciendo el ejercicio, mientras que para leer los mensajes debería parar de hacer el ejercicio.
- Implementar comunicación en la dirección del cliente hacia el monitor, por si éste desea preguntar alguna duda sobre el ejercicio.
- Otorgar al monitor el control sobre la sesión, indicando en cada momento qué ejercicio deben estar realizando los usuarios.
- Volver a añadir a la base de datos la tabla llamda *rutina*, pero con una funcionalidad diferente. En este caso debería agrupar un conjunto de ejercicios, permitiendo así al monitor trabajar con un subconjunto de ellos, evitando que debe añadirlos en las sesiones de uno en uno. Las sesiones por lo tanto pasarían a ser conjuntos de rutinas.
- Mejorar la seguridad del sistema, codificando la información confidencial sobre los usuarios almacenada en la base de datos, sobre todo con las contraseñas de los mismos, evitando que éstas viajen en claro en los mensajes intercambiados entre alguna de las aplicaciones con el servidor.
- Crear una aplicación para la gestión de todo lo relacionado con el servidor del gimnasio, permitiendo añadir ejercicios y usuarios al sistema, así como la libre manipulación de estos datos de una forma sencilla, haciendo transparente la base de datos al usuario.
- Desarrollar una opción alternativa en caso de que el router no tenga activada la opción de UPnP, conviviendo así con la solución aquí propuesta para el problema de las direcciones privadas. Suponiendo que el número de monitores que tuvieran problemas con el protocolo UPnP sería pequeño, se deberá implementar una solución alternativa. Lo que se propone es que, al igual que hemos hecho en el caso del cliente Android, forzar al NAT a traducir el puerto deseado, almacenando posteriormente la traducción en el servidor. Para ello deberá habilitarse un socket en el servidor que quede a la escucha de las peticiones de los monitores con problemas, para así atender sus peticiones y almacenar la traducción del puerto en el campo de las sesiones correspondientes.

Bibliografía

[1] Díaz Lora, Antonio José *Aplicación Android para la rehabilitación física de usuario usando MySQL, PHP y codificación JSON*, Sevilla, España, 2014

[2] Clavain Mateo, Antonio *Monitorización mediante streaming de vídeo para entrenamiento personal*, Sevilla, España, 2014

[3] Wowza Media Systems *http://www.wowza.com/*

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

[5] Apache Server *http://httpd.apache.org/*

[6] MySQL *https://www.mysql.com/*

[7] Java *https://www.java.com/es/*

[8] PHP *http://php.net/*

[9] JSON *http://json.org/*

[10] Eclipse *https://eclipse.org/*

[11] Android Studio *https://developer.android.com/sdk/index.html*

[12] VLC media player *http://www.videolan.org/vlc/*

[13] API VLCj *http://caprica.github.io/vlcj/javadoc/2.1.0/*

[14] ISO *Information Technology – Coding of audio-visual objects*. ISO/IEC 14496

[15] Peepers *https://github.com/foxdog-studios/peepers*

[16] Weupnp *https://github.com/bitletorg/weupnp*

Anexos

En esta sección incluiremos todos los ficheros generados durante la elaboración de este proyecto, así como los ficheros de anteriores proyectos que hayan sufrido alguna modificación o aquellos pertenecientes a alguna librería y que se hayan visto alterados para adaptarlos a nuestras necesidades.

1 Código

1.1 Aplicación Java

Monitorgym.java

"Monitorgym.java"

```
1  package monitorgym;
2
3  import java.awt.Color;
4
5  import javax.swing.UIManager;
6  import javax.swing.UIManager.LookAndFeelInfo;
7
8  /*
9   * Clase principal de nuestra aplicacion, en la que se define
10  * el aspecto de la misma y se lanza la primera ventana de
11  * usuario
12  */
13 public class Monitorgym {
14         public static void main(String[] args) {
15                 // TODO code application logic here
16                 try {
17                         // Definimos el aspecto de nuestra aplicacion
18                         for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
19                                 if ("Nimbus".equals(info.getName())) {
20                                         UIManager.setLookAndFeel(info.getClassName());
21                                         UIManager.getLookAndFeelDefaults().put("Panel.background", new Color(0xFFA44D));
22                                         break;
23                                 }
24                         }
25                 } catch (Exception e) {
26                         // If Nimbus is not available, fall back to cross-platform
27                         try {
28                                 UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
29                         } catch (Exception ex) {
30                                 ex.printStackTrace();
31                         }
32                 }
33
34                 // Lanzamos la primera ventana de usuario
35                 VentanaLogin ventana = new VentanaLogin();
36                 ventana.setVisible(true);
37         }
38 }
```

GestorPeticiones.java

"GestorPeticiones.java"

```
1  package monitorgym;
2
3  import java.io.BufferedReader;
4  import java.io.DataOutputStream;
5  import java.io.InputStreamReader;
```

```
6  import java.net.HttpURLConnection;
7  import java.net.URL;
8
9  import org.json.JSONArray;
10
11 /*
12  * Clase que gestionara todas las conexiones con el servidor.
13  * Cada vez que se realice una peticion al servidor, debera
14  * utilizar un metodo de esta clase.
15  */
16 public class GestorPeticiones {
17
18         private String SERVER_PATH;
19         private static final String USER_AGENT = "Mozilla/5.0";
20
21         private JSONArray peticion(String url,String params) throws Exception{
22
23                 StringBuffer response = null;
24                 JSONArray object = null;
25
26                 try{
27                         // Actualizamos la IP del servidor por si ha habido algun cambio
28                         actualizaIp();
29                         // Establecemos conexion con el servidor
30                         String urlString = SERVER_PATH + url;
31                         URL urlObj = new URL(urlString);
32                         HttpURLConnection con = (HttpURLConnection) urlObj.openConnection();
33
34                         //Ponemos la cabecera
35                         con.setRequestMethod("POST");
36                         con.setRequestProperty("User-Agent", USER_AGENT);
37
38                         // Enviamos la peticion por POST
39             con.setDoOutput(true);
40             DataOutputStream wr = new DataOutputStream(con.getOutputStream());
41             wr.writeBytes(params);
42                     wr.flush();
43                     wr.close();
44
45                     //Capturamos la respuesta del servidor
46             int responseCode = con.getResponseCode();
47             System.out.println("\nSending 'POST' request to URL : " + url);
48             System.out.println("Response Code : " + responseCode);
49
50             BufferedReader in = new BufferedReader(
51                     new InputStreamReader(con.getInputStream(), "UTF-8"));
52             String inputLine;
53             response = new StringBuffer();
54
55             while ((inputLine = in.readLine()) != null) {
56                 response.append(inputLine);
57             }
58
59             // Quitamos el BOM UTF-8 a la respuesta
60             String responseWithoutBOM= quitarBOM(response);
61             object = new JSONArray(responseWithoutBOM);
62             //Mostramos la respuesta del servidor por consola
63             if(object.length()>0){
64                     System.out.println("Respuesta del servidor: "+ object.getJSONObject(0).toString());
65                 System.out.println();
66             }
67
68             //cerramos la conexion
69             in.close();
70
71                 } catch (Exception e){
72                         throw e;
73                 }
74
75                 return object;
76         }
77
78         /*
79          * Metodo para quitar el BOM debido a la codificacion UTF-8 en el servidor.
80          * Este metodo es necesario debido a un bug de Java, que no retira el BOM
81          * Completo al recibir un paquete.
82          */
83         private static String quitarBOM(StringBuffer bom){
84                 return bom.substring(1);
85         }
86
87         /*
88          * Metodo para actualizar la IP del servidor segun la configurada por el usuario
89          */
90         private void actualizaIp(){
```

```
91                  ConfiguracionIpServidor config = new ConfiguracionIpServidor();
92                  SERVER_PATH = "http://"+config.getIpServer()+"/";
93          }
94
95          /*
96           * Metodo para enviar las credenciales de usuario al servidor. este nos
97           * indicara en la respuestas si son validas o no.
98           */
99          public String login(String usuario, String passwd) throws Exception{
100                 String url = "gymserver/acces.php";
101                 String params = "usuario="+usuario+"&password="+passwd;
102                 String codigo = "logstatus";
103                 return this.peticion(url, params).getJSONObject(0).getString(codigo);
104         }
105
106         /*
107          * Metodo para enviar al servidor la informacion relativa a una sesion.
108          */
109         public String nuevaSesion(String nomSesion, String monitor, String fecha, String numUsuarios) throws Exception{
110                 String url = "gymserver/addSesion.php";
111                 String params = "nombre_Sesion="+nomSesion+
112                                                 "&user_Monitor="+monitor+
113                                                 "&fecha="+fecha+
114                                                 "&num_Usuarios="+numUsuarios+
115                                                 "&activa=1";
116                 String codigo = "nuevaSesion";
117                 return this.peticion(url, params).getJSONObject(0).getString(codigo);
118         }
119
120         /*
121          * Metodo para solicitar al servidor todas las sesiones creadas por
122          * cierto monitor.
123          */
124         public JSONArray sesionesMonitor(String monitor) throws Exception{
125                 String url = "gymserver/listasesionesmonitor.php";
126                 String params = "monitor="+monitor;
127                 return this.peticion(url, params);
128         }
129
130         /*
131          * Metodo que solicita al servidor toda la informacion de una sesion
132          */
133         public JSONArray sesion(String id) throws Exception{
134                 String url = "gymserver/getSesion.php";
135                 String params = "Id_Sesion="+id;
136                 return this.peticion(url, params);
137         }
138
139         /*
140          * Metodo para indicarle al servidor que debe eliminar cierta sesion.
141          */
142         public JSONArray eliminaSesion(String id) throws Exception{
143                 String url = "gymserver/eliminaSesion.php";
144                 String params = "Id_Sesion="+id;
145                 return this.peticion(url, params);
146         }
147
148         /*
149          * Metodo para obtener del servidor toda la informacion de cierto ejercicio
150          */
151         public JSONArray ejercicio(String idEjercicio) throws Exception{
152                 String url = "gymserver/ejercicio.php";
153                 String params = "id_ejercicio="+idEjercicio;
154                 return this.peticion(url, params);
155         }
156
157         /*
158          * Metodo para obtener todos los ejercicios almacenados en la base de datos.
159          */
160         public JSONArray ejercicios() throws Exception{
161                 String url = "gymserver/listaEjercicios.php";
162                 String params = "";
163                 return this.peticion(url, params);
164         }
165
166         /*
167          * Metodo para solicitar al servidor todos los ejercicios que no han sido
168          * incluidos en una sesion.
169          */
170         public JSONArray ejerciciosNoSesion(String idSesion) throws Exception{
171                 String url = "gymserver/listaEjerciciosNoSesion.php";
172                 String params = "Id_Sesion="+idSesion;
173                 return this.peticion(url, params);
174         }
175
```

```
176         /*
177          * Metodo para solicitar al servidor todos los ejercicios que forman
178          * parte de una sesion.
179          */
180         public JSONArray ejerciciosSesion(String idSesion) throws Exception{
181                 String url = "gymserver/listaEjerciciosSesion.php";
182                 String params = "Id_Sesion="+idSesion;
183                 return this.peticion(url, params);
184         }
185 
186         /*
187          * Metodo para introducir un ejercicio en una sesion.
188          */
189         public JSONArray addEjercicioSesion(String idSesion, String idEjercicio) throws Exception{
190                 String url = "gymserver/addEjercicioSesion.php";
191                 String params = "Id_Ejercicio="+idEjercicio+
192                                                 "&Id_Sesion="+idSesion;
193                 return this.peticion(url, params);
194         }
195 
196         /*
197          * Metodo para eliminar un ejercicio de una sesion.
198          */
199         public JSONArray removeEjercicioSesion(String idSesion, String idEjercicio) throws Exception{
200                 String url = "gymserver/eliminaEjercicioSesion.php";
201                 String params = "Id_Ejercicio="+idEjercicio+
202                                                 "&Id_Sesion="+idSesion;
203                 return this.peticion(url, params);
204         }
205 
206         /*
207          * Metodo utilizado para actualizar la IP del monitor asociada a una sesion.
208          * Con este metodo evitamos problemas si el monitor quiere  monitorizar a los
209          * usuarios desde un PC distinto desde el que creo la sesion.
210          */
211         public JSONArray actualizaIpMonitorSesion(String idSesion, int puerto) throws Exception{
212                 String url = "gymserver/setIpMonitorSesion.php";
213                 String params = "Id_Sesion="+idSesion+
214                                                 "&Puerto="+puerto;
215                 return this.peticion(url, params);
216         }
217 
218         /*
219          * Metodo para indicar al servidor que una sesion ha finalizado,
220          * pasando a estar en el estado de NO activa
221          */
222         public JSONArray setSesionNoActiva(String idSesion) throws Exception{
223                 String url = "gymserver/setSesionNoActiva.php";
224                 String params = "Id_Sesion="+idSesion;
225                 return this.peticion(url, params);
226         }
227 }
```

ConfiguradorIpServidor

"ConfiguradorIpServidor.java"

```
1  package monitorgym;
2  
3  import java.io.FileOutputStream;
4  import java.io.IOException;
5  import java.io.OutputStream;
6  import java.util.Properties;
7  
8  public class ConfiguradorIpServidor {
9  
10         public ConfiguradorIpServidor(String ip){
11                 Properties propiedades = new Properties();
12             OutputStream salida = null;
13 
14             try {
15                 salida = new FileOutputStream("configuracion.properties");
16 
17                 // asignamos los valores a las propiedades
18                 propiedades.setProperty("ip_server", ip);
19 
20                 // guardamos el archivo de propiedades en la carpeta de aplicacion
21                 propiedades.store(salida, null);
22             } catch (IOException io) {
23                 io.printStackTrace();
24             } finally {
25                 if (salida != null) {
26                     try {
27                         salida.close();
28                     } catch (IOException e) {
```

```
                        e.printStackTrace();
                    }
                }

            }
        }
}
```

ConfiguracionIpServidor

"ConfiguracionIpServidor.java"

```
package monitorgym;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class ConfiguracionIpServidor {

        String ipServer;

        public ConfiguracionIpServidor(){
                Properties propiedades = new Properties();
            InputStream entrada = null;

            try {

                entrada = new FileInputStream("configuracion.properties");

                // cargamos el archivo de propiedades
                propiedades.load(entrada);

                // obtenemos las propiedades y la almacenamos
                ipServer = propiedades.getProperty("ip_server");

            } catch (IOException ex) {
                ex.printStackTrace();
            } finally {
                if (entrada != null) {
                    try {
                        entrada.close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }

        public String getIpServer(){
                return ipServer;
        }
}
```

VentanaLogin.java

"VentanaLogin.java"

```
package monitorgym;


import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import javax.swing.JButton;
import javax.swing.SwingConstants;
import javax.swing.UIManager;

import java.awt.Color;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import javax.swing.JLabel;
import javax.swing.ImageIcon;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

/*
 * Ventana de usuario en la que se mostrara un formulario
 * de inicio de sesion en el sistema.
 */
```

```
28 public class VentanaLogin extends JFrame {
29
30         private JPanel contentPane;
31         private JTextField txtUsuario;
32         private JPasswordField pwdPassword;
33         private JLabel label;
34         private JButton btnEntrar;
35         private GestorPeticiones gestor = new GestorPeticiones();
36         private String usuario, passwd;
37
38         /**
39          * Create the frame.
40          */
41         public VentanaLogin() {
42                 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
43                 setBounds(100, 100, 570, 420);
44                 setLocationRelativeTo(null); // Ventana centrada
45         setTitle("Login");
46         setResizable(false);
47
48                 contentPane = new JPanel();
49                 contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
50                 setContentPane(contentPane);
51                 contentPane.setLayout(null);
52
53                 // Campo para introducir el nombre de usuario
54                 txtUsuario = new JTextField();
55                 txtUsuario.setText("usuario");
56                 txtUsuario.setBounds(284, 228, 86, 28);
57                 contentPane.add(txtUsuario);
58                 txtUsuario.setColumns(10);
59
60                 // Campo para introducir el password del usuario
61                 pwdPassword = new JPasswordField();
62                 pwdPassword.setText("password");
63                 pwdPassword.setBounds(284, 267, 86, 28);
64                 contentPane.add(pwdPassword);
65
66                 label = new JLabel(" ");
67                 label.setBounds(81, 304, 424, 20);
68                 label.setHorizontalAlignment(SwingConstants.CENTER);
69                 contentPane.add(label);
70
71                 // Boton para acceder al sistema tras proporcionar las credenciales
72                 btnEntrar = new JButton("ENTRAR");
73                 btnEntrar.addActionListener(new ActionListener() {
74                         public void actionPerformed(ActionEvent e) {
75                                 try{
76                                         // Obtenemos las credenciales introducidas por el usuario
77                                         usuario = txtUsuario.getText();
78                                         passwd = new String(pwdPassword.getPassword());
79                                         // Consulta al servidor para comprobar si las credenciales
80                                         // del usuario son correctas
81                             if (gestor.login(usuario, passwd).equals("1")){
82                                 // Lanzamos la siguiente ventana
83                                 VentanaSesiones ventana = new VentanaSesiones(usuario);
84                                 ventana.setVisible(true);
85                                 dispose();
86                             } else{
87                                 // Le mostramos al usuario que ha introducido mal las
88                                     // credenciales
89                                 label.setText("Usuario o contrasena incorrectos");
90                             }
91                                 } catch(Exception ex){
92                                         // Si no es posible conectar con el servidor se lo
93                                         // indicamos al usuario
94                                         System.err.println("Obtenido el siguiente error: " +
95                                                                                 ex.getMessage());
96                                         label.setText("Problemas con el servidor. Vuelva a "
97                                                                 + "intentarlo en unos minutos");
98                                 }
99                         }
100                });
101                btnEntrar.setBounds(228, 335, 112, 28);
102                contentPane.add(btnEntrar);
103
104                JLabel lblNombre = new JLabel("Nombre:");
105                lblNombre.setBounds(197, 235, 60, 14);
106                contentPane.add(lblNombre);
107
108                JLabel lblContrasea = new JLabel("Contrase\u00F1a:");
109                lblContrasea.setBounds(197, 274, 76, 14);
110                contentPane.add(lblContrasea);
111
112                JLabel label_1 = new JLabel("");
```

```
113                 label_1.setIcon(new ImageIcon(VentanaLogin.class.getResource("/icons/logo.png")));
114                 label_1.setBounds(189, 28, 192, 189);
115                 contentPane.add(label_1);
116 
117                 JLabel buttonConfig = new JLabel("");
118                 buttonConfig.addMouseListener(new MouseAdapter() {
119                         @Override
120                         public void mouseClicked(MouseEvent e) {
121                                 UIManager.put("OptionPane.background", new Color(0xFFA44D));
122                                 String cartel = "Introduzca la ip del servidor...";
123                                 String titulo = "Configuracion de la direccion IP del servidor";
124                                 String ip = JOptionPane.showInputDialog(
125                                                 null, cartel, titulo, JOptionPane.QUESTION_MESSAGE);
126                                 if(ip != null){
127                                         new ConfiguradorIpServidor(ip);
128                                 }
129                         }
130                 });
131                 buttonConfig.setIcon(new ImageIcon(
132                                 VentanaLogin.class.getResource("/icons/config_icon-16x16.png")));
133                 buttonConfig.setBounds(525, 11, 29, 28);
134                 contentPane.add(buttonConfig);
135         }
136 }
```

VentanaSesiones.java

"VentanaSesiones.java"

```
1  package monitorgym;
2  
3  import javax.swing.JFrame;
4  import javax.swing.JPanel;
5  import javax.swing.border.EmptyBorder;
6  import javax.swing.JLabel;
7  import javax.swing.JOptionPane;
8  import javax.swing.JScrollPane;
9  import javax.swing.JTextField;
10 import javax.swing.JSpinner;
11 import javax.swing.SpinnerDateModel;
12 import javax.swing.UIManager;
13 
14 import com.toedter.calendar.JDateChooser;
15 
16 import javax.swing.JButton;
17 
18 import java.awt.event.ActionListener;
19 import java.awt.event.ActionEvent;
20 
21 import javax.swing.JComboBox;
22 import javax.swing.JToolBar;
23 import javax.swing.ImageIcon;
24 
25 import java.awt.Color;
26 import java.text.SimpleDateFormat;
27 import java.util.Calendar;
28 
29 /*
30  * Clase que muestra una ventana al monitor desde la que
31  * gestionar todo lo relativo a las sesiones que haya creado.
32  */
33 public class VentanaSesiones extends JFrame {
34 
35         /**
36          *
37          */
38         private static final long serialVersionUID = 1L;
39 
40         private JPanel contentPane;
41         private JTextField textFieldNombre;
42         private JSpinner spinnerNumUser;
43         private JDateChooser dateChooser;
44         private JComboBox<String> comboBox;
45         private JSpinner timeSpinner;
46         private DynamicListSesiones listaSesiones;
47         private GestorPeticiones gestor = new GestorPeticiones();
48 
49         private JButton buttonAdd;
50         private JButton buttonSave;
51         private JButton buttonDelete;
52         private JButton buttonEdit;
53         private JButton buttonCancelar;
54 
55         private String mMonitor;
56 
```

```
57
58          /*
59           * Recibiremos el nombre del monitor que ha iniciado la sesion
60           * para poder obtener del servidor todas las sesiones que haya
61           * creado
62           */
63          public VentanaSesiones(final String monitor) throws Exception {
64
65                  mMonitor = monitor;
66
67                  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
68                  setBounds(100, 100, 570, 420);
69                  setLocationRelativeTo(null); // Ventana centrada
70          setTitle("Sesiones");
71          setResizable(false);
72
73                  contentPane = new JPanel();
74                  contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
75                  setContentPane(contentPane);
76                  contentPane.setLayout(null);
77
78                  JLabel labelNombreSesion = new JLabel("Nombre de la sesi\u00F3n:");
79                  labelNombreSesion.setBounds(243, 99, 125, 25);
80                  contentPane.add(labelNombreSesion);
81
82                  JLabel labelFecha = new JLabel("Fecha:");
83                  labelFecha.setBounds(243, 159, 63, 25);
84                  contentPane.add(labelFecha);
85
86                  JLabel labelHora = new JLabel("Hora:");
87                  labelHora.setBounds(243, 216, 46, 25);
88                  contentPane.add(labelHora);
89
90                  JLabel labelMaximoUsuarios = new JLabel("N\u00BA m\u00E1ximo de usuarios:");
91                  labelMaximoUsuarios.setBounds(243, 273, 151, 25);
92                  contentPane.add(labelMaximoUsuarios);
93
94                  // Campo para introducir el nombre de una nueva sesion
95                  textFieldNombre = new JTextField();
96                  textFieldNombre.setBounds(376, 97, 178, 28);
97                  contentPane.add(textFieldNombre);
98                  textFieldNombre.setColumns(10);
99                  textFieldNombre.setEnabled(false);
100
101                 // Campo para introducir el numero maximo de usuarios de una sesion
102                 spinnerNumUser = new JSpinner();
103                 spinnerNumUser.setBounds(447, 273, 41, 25);
104                 contentPane.add(spinnerNumUser);
105                 spinnerNumUser.setEnabled(false);
106
107                 // Campo para seleccionar la fecha de una sesion
108                 dateChooser = new JDateChooser();
109                 dateChooser.setBackground(Color.white);
110                 dateChooser.setBounds(376, 156, 178, 28);
111                 contentPane.add(dateChooser);
112                 dateChooser.setEnabled(false);
113
114                 // Campo para seleccionar las sesiones a mostrar.
115                 // Podremos mostrar las activas, no activas o todas a la vez.
116                 comboBox = new JComboBox<String>();
117                 comboBox.setBounds(55, 83, 91, 20);
118                 comboBox.setAlignmentY(CENTER_ALIGNMENT);
119                 comboBox.addItem("activas");
120                 comboBox.addItem("no activas");
121                 comboBox.addItem("todas");
122                 comboBox.addActionListener(new ActionListener(){
123                         public void actionPerformed(ActionEvent e) {
124                                 listaSesiones.cambiaDatosLista(comboBox.getSelectedIndex());
125                         }
126                 });
127                 contentPane.add(comboBox);
128
129                 JLabel labelMostrar = new JLabel("Mostrar:");
130                 labelMostrar.setBounds(10, 86, 46, 14);
131                 contentPane.add(labelMostrar);
132
133                 // Lista con todas las sesiones creadas por el monitor
134                 listaSesiones = new DynamicListSesiones(gestor.sesionesMonitor(monitor), this);
135                 JScrollPane scrollPaneSesiones = new JScrollPane(listaSesiones);
136                 scrollPaneSesiones.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
137                 scrollPaneSesiones.setBounds(10, 114, 210, 235);
138                 contentPane.add(scrollPaneSesiones);
139
140                 // Spinner para seleccionar la hora de comienzo de una sesion
141                 timeSpinner = new JSpinner(new SpinnerDateModel());
```

```
142             JSpinner.DateEditor timeEditor = new JSpinner.DateEditor(timeSpinner, "HH:mm:ss");
143             timeSpinner.setEditor(timeEditor);
144             timeSpinner.setBounds(427, 217, 79, 23);
145             contentPane.add(timeSpinner);
146             timeSpinner.setEnabled(false);
147
148             JToolBar toolBar = new JToolBar();
149             toolBar.setBounds(0, 0, 146, 28);
150             contentPane.add(toolBar);
151
152             // Boton para cancelar la creacion de una nueva sesion
153             buttonCancelar = new JButton("Cancelar");
154             buttonCancelar.setBounds(364, 326, 89, 23);
155             contentPane.add(buttonCancelar);
156             buttonCancelar.setVisible(false);
157             buttonCancelar.addActionListener(new ActionListener(){
158                     public void actionPerformed(ActionEvent arg0){
159                             //Desactivamos los campos
160                             desactivaCampos();
161                             //Mostramos solo los botones que deben estar activos en este momento
162                             buttonCancelar.setVisible(false);
163                             buttonSave.setEnabled(false);
164                             buttonAdd.setEnabled(true);
165                             listaSesiones.setEnabled(true);
166                     }
167             });
168
169             // Boton para crear una nueva sesion
170             buttonAdd = new JButton("");
171             buttonAdd.addActionListener(new ActionListener() {
172                     public void actionPerformed(ActionEvent arg0) {
173                             //Vaciamos todos los campos necesairos para introducir nuevas sesiones...
174                             vaciaCampos();
175                             //...y los activamos
176                             activaCampos();
177                             //Mostramos solo los botones que deben estar activos en este momento
178                             buttonSave.setEnabled(true);
179                             buttonAdd.setEnabled(false);
180                             buttonDelete.setEnabled(false);
181                             buttonEdit.setEnabled(false);
182                             listaSesiones.setEnabled(false);
183                             buttonCancelar.setVisible(true);
184
185                     }
186             });
187             buttonAdd.setIcon(new ImageIcon(VentanaSesiones.class.getResource("/icons/add_icon-24.png")));
188             toolBar.add(buttonAdd);
189
190             // Boton para guardar una nueva sesion
191             buttonSave = new JButton("");
192             buttonSave.addActionListener(new ActionListener() {
193                     public void actionPerformed(ActionEvent e) {
194                             guardaSesion();
195                     }
196             });
197             buttonSave.setIcon(new ImageIcon(VentanaSesiones.class.getResource("/icons/save_icon-26.png")));
198             toolBar.add(buttonSave);
199             //Solo estara activo cuando vayamos a crear una nueva sesion
200             buttonSave.setEnabled(false);
201
202             // Boton para eliminar una sesion seleccionada
203             buttonDelete = new JButton("");
204             buttonDelete.addActionListener(new ActionListener() {
205                     public void actionPerformed(ActionEvent arg0) {
206                             UIManager.put("OptionPane.background", new Color(0xFFA44D));
207
208                             eliminarSesion();
209                     }
210             });
211             buttonDelete.setIcon(new ImageIcon(VentanaSesiones.class.getResource("/icons/bin_icon-24.png")));
212             toolBar.add(buttonDelete);
213             //Solo estara activo despues de seleccionar una sesion
214             buttonDelete.setEnabled(false);
215
216             // Boton para editar una sesion seleccionada
217             buttonEdit = new JButton("");
218             buttonEdit.addActionListener(new ActionListener() {
219                     public void actionPerformed(ActionEvent arg0) {
220                             editarSesion();
221                     }
222             });
223             buttonEdit.setIcon(new ImageIcon(VentanaSesiones.class.getResource("/icons/pencil_icon-24.png")));
224             toolBar.add(buttonEdit);
225             buttonEdit.setEnabled(false);
226     }
```

```
227
228         /*
229          * Metodo para obtener el campo de texto del nombre de la sesion.
230          */
231         public JTextField getTextFieldNombre(){
232                 return this.textFieldNombre;
233         }
234
235         /*
236          * Metodo para obtener el spinner para el numero maximo de usuarios
237          * de la sesion.
238          */
239         public JSpinner getSpinnerNumUser(){
240                 return this.spinnerNumUser;
241         }
242
243         /*
244          * Metodo para obtener el selector de fecha de la sesion
245          */
246         public JDateChooser getDateChooser(){
247                 return this.dateChooser;
248         }
249
250         /*
251          * Metodo para obtener el JComboBox de seleccion del tipo de
252          * sesiones a mostrar.
253          */
254         public JComboBox<String> getComboBox(){
255                 return this.comboBox;
256         }
257
258         /*
259          * Metodo para obtener el spinner de seleccion de hora de
260          * inicio de una sesion.
261          */
262         public JSpinner getTimeSpinner(){
263                 return this.timeSpinner;
264         }
265
266         /*
267          * Metodo para obtener el boton de eliminacion de sesion
268          */
269         public JButton getButtonDelete(){
270                 return this.buttonDelete;
271         }
272
273         /*
274          * Metodo para obtener el boton de edicion de sesion
275          */
276         public JButton getButtonEdit(){
277                 return this.buttonEdit;
278         }
279
280         /*
281          * Metodo para vaciar todos los campos en los que el monitor
282          * introduce la informacion de la sesion.
283          */
284         private void vaciaCampos(){
285                 textFieldNombre.setText("");
286                 spinnerNumUser.setValue(0);
287                 dateChooser.setDate(null);
288                 Calendar calendar = Calendar.getInstance();
289                 calendar.set(Calendar.HOUR_OF_DAY, 24);
290                 calendar.set(Calendar.MINUTE, 0);
291                 calendar.set(Calendar.SECOND, 0);
292                 timeSpinner.getModel().setValue(calendar.getTime());
293         }
294
295         /*
296          * Metodo para desactivar todos los campos de introduccion de
297          * informacion de la sesion.
298          */
299         private void desactivaCampos(){
300                 textFieldNombre.setEnabled(false);
301                 spinnerNumUser.setEnabled(false);
302                 dateChooser.setEnabled(false);
303                 timeSpinner.setEnabled(false);
304         }
305
306         /*
307          * Metodo para activar todos los campos de introduccion de
308          * informacion de la sesion.
309          */
310         private void activaCampos(){
311                 textFieldNombre.setEnabled(true);
```

```
312                 spinnerNumUser.setEnabled(true);
313                 dateChooser.setEnabled(true);
314                 timeSpinner.setEnabled(true);
315         }
316 
317         /*
318          * Metodo para guardar una nueva sesion
319          */
320         private void guardaSesion(){
321                 try{
322                         // Obtenemos los valores introducidos por el usuario en todos los campos
323                         String nomSesion = textFieldNombre.getText();
324                         String numUsuarios = spinnerNumUser.getValue().toString();
325                         SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
326                         String stringFecha = dateFormat.format(dateChooser.getDate());
327                         String rawHora = timeSpinner.getValue().toString();
328                         int index = rawHora.indexOf(":");
329                         String stringHora = rawHora.substring(index-2, index+6);
330 
331                         // Creamos la nueva sesion
332             if(listaSesiones.nuevaSesion(nomSesion, mMonitor, stringFecha+" "+stringHora, numUsuarios)) {
333                     // Actualizamos la lista con la nueva sesion
334                 listaSesiones.cambiaDatosLista(comboBox.getSelectedIndex());
335                 //Vaciamos todos los campos
336                 vaciaCampos();
337                                 //Y los desactivamos
338                 desactivaCampos();
339                 //Mostramos solo los botones que deben estar activos en este momento
340                                 buttonSave.setEnabled(false);
341                             buttonAdd.setEnabled(true);
342                                 listaSesiones.setEnabled(true);
343                                 buttonCancelar.setVisible(false);
344             } else {
345                 System.out.println("Ha habido algun problema creando la sesion");
346             }
347         } catch(Exception ex){
348                 UIManager.put("OptionPane.background", new Color(0xFFA44D));
349             System.err.println("Obtenido el siguiente error: " + ex.getMessage());
350             String mensaje = "Complete todos los campos para crear la sesion";
351             JOptionPane.showMessageDialog(null, mensaje);
352         }
353         }
354 
355         /*
356          * Metodo para eliminar la sesion seleccionada
357          */
358         private void eliminarSesion(){
359                 // Mostramos al usuario el mensaje de confirmacion
360                 String[] buttons = {"Aceptar", "Cancelar"};
361                 String mensaje = "Seguro que desea eliminar la sesion?";
362                 String titulo = "Confirmar eliminacion";
363                 int confirmar = JOptionPane.showOptionDialog(null, mensaje, titulo, JOptionPane.WARNING_MESSAGE,
364                                 0, null, buttons, buttons[1]);
365                 if(confirmar == 0){
366                         try{
367                                 // Si el usuario lo confirma, eliminamos la sesion
368                                 if(listaSesiones.eliminarSesionMostrada(comboBox.getSelectedIndex())){
369                                         // Actualizamos la lista de sesiones
370                                         listaSesiones.cambiaDatosLista(comboBox.getSelectedIndex());
371                                         //Mostramos solo los botones que deben estar activos en este momento
372                                         buttonDelete.setEnabled(false);
373                                         buttonEdit.setEnabled(false);
374                                 }
375                         } catch(Exception e){
376                                 e.printStackTrace();
377                         }
378                 }
379         }
380 
381         /*
382          * Metodo para editar la sesion seleccionada
383          */
384         private void editarSesion(){
385                 // Si el monitor desea editar la sesion, se le mostrara una nueva ventana desde
386                 // la que editar todo lo relativo a la sesion
387                 VentanaEjercicios ventana = new VentanaEjercicios(listaSesiones.getIdSesionMostrada());
388                 ventana.setVisible(true);
389         dispose();
390         }
391 }
```

DynamicListSesiones.java

"DynamicListSesiones.java"

```java
package monitorgym;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;

import javax.swing.DefaultListModel;
import javax.swing.JList;

import org.json.JSONArray;

/*
 * Clase desde la que se gestionara todo lo relativo a mostrar
 * las sesiones del monitor por pantalla.
 */
public class DynamicListSesiones extends JList<String> {

	private ArrayList<String> idSesiones;
	private ArrayList<String> idSesionesActivas;
	private ArrayList<String> idSesionesNoActivas;
	private ArrayList<String> nomSesiones;
	private ArrayList<String> nomSesionesActivas;
	private ArrayList<String> nomSesionesNoActivas;
	private DefaultListModel<String> model;
	private GestorPeticiones gestor;
	private VentanaSesiones ventanaSesiones;
	private String idSesionMostrada;
	private int posSesionMostrada;

	public DynamicListSesiones(JSONArray sesiones, VentanaSesiones ventana) throws Exception{
		String nombre, id;
		int activa;
		//Inicializamos las variables
		gestor = new GestorPeticiones();
		ventanaSesiones=ventana;
		idSesiones = new ArrayList<String>();
		idSesionesActivas = new ArrayList<String>();
		idSesionesNoActivas = new ArrayList<String>();
		nomSesiones = new ArrayList<String>();
		nomSesionesActivas = new ArrayList<String>();
		nomSesionesNoActivas = new ArrayList<String>();
		model = new DefaultListModel<String>();
		this.setModel(model);
		this.setBounds(10, 36, 76, 175);
		try{
			//Llenamos la lista con las sesiones obtenidas del servidor
			for(int i=0; i<sesiones.length(); i++){
				id = sesiones.getJSONObject(i).getString("Id_Sesion");
				nombre = sesiones.getJSONObject(i).getString("nombre_Sesion");
				nomSesiones.add(nombre);
				idSesiones.add(id);
				activa = sesiones.getJSONObject(i).getInt("activa");
				// Dependiendo de si la sesion es activa o no ira a una lista diferente
				if(activa==0){
					nomSesionesNoActivas.add(nombre);
					idSesionesNoActivas.add(id);
				} else if(activa==1){
					nomSesionesActivas.add(nombre);
					idSesionesActivas.add(id);
					model.addElement(nombre);
				}
			}
		} catch (Exception e){
			throw e;
		}
		this.addMouseListener(this.nuevoMouseListener());
	}

	/*
	 * Listener que atendera al doble click sobre una sesion.
	 */
	private MouseListener nuevoMouseListener() throws Exception{
		MouseListener mouseListener = new MouseAdapter() {
		    public void mouseClicked(MouseEvent e) {
			mostrarSesion(e);
		    }
		};

		return mouseListener;
	}

```

```
86
87          /*
88           * Metodo para cambiar las sesiones que se muestran en la lista
89           * dependiendo de la seleccion del usuario
90           */
91          public void cambiaDatosLista(int option){
92                  int i;
93                  // Actualizamos todos los elementos que forman parte de las listas, ya que
94                  // ha habido una modificacion en alguna de ellas.
95                  model.removeAllElements();
96                  if(option==0 && nomSesionesActivas.size()>0){
97                          for(i=0; i<nomSesionesActivas.size(); i++){
98                                  model.addElement(nomSesionesActivas.get(i).toString());
99                          }
100                 } else if(option==1 && nomSesionesNoActivas.size()>0){
101                         for(i=0; i<nomSesionesNoActivas.size(); i++){
102                                 model.addElement(nomSesionesNoActivas.get(i).toString());
103                         }
104                 } else if (option==2 && nomSesiones.size()>0){
105                         for(i=0; i<nomSesiones.size(); i++){
106                                 model.addElement(nomSesiones.get(i).toString());
107                         }
108                 }
109         }
110
111         /*
112          * Metodo para introducir una nueva sesion tanto en la base de datos como en la lista
113          */
114         public boolean nuevaSesion(
115                         String nomSesion, String monitor, String fecha, String numUsuarios) throws Exception{
116
117                 boolean resul = false;
118
119                 try{
120                         // Crea la nueva sesion
121                         if(gestor.nuevaSesion(nomSesion, monitor, fecha, numUsuarios).equals("1")){
122                                 // Si no ha habido problemas para crear la sesion...
123                                 resul = true;
124                                 //... introducimos la sesion creada en las listas
125                                 JSONArray sesiones = gestor.sesionesMonitor(monitor);
126                                 idSesiones.add(sesiones.getJSONObject(
127                                                 sesiones.length()-1).getString("Id_Sesion"));
128                                 nomSesiones.add(sesiones.getJSONObject(
129                                                 sesiones.length()-1).getString("nombre_Sesion"));
130                                 idSesionesActivas.add(sesiones.getJSONObject(
131                                                 sesiones.length()-1).getString("Id_Sesion"));
132                                 nomSesionesActivas.add(sesiones.getJSONObject(
133                                                 sesiones.length()-1).getString("nombre_Sesion"));
134                         }
135                 } catch(Exception e){
136                         throw e;
137                 }
138
139                 return resul;
140         }
141
142
143         /*
144          * Metodo para eliminar la sesion mostrada por pantalla
145          */
146         public boolean eliminarSesionMostrada(int option) throws Exception{
147                 JSONArray consulta;
148                 boolean resul = false;
149
150                 try{
151                         // Eliminamos la sesion en la base de datos
152                         consulta = gestor.eliminaSesion(idSesionMostrada);
153                         if(consulta.getJSONObject(0).getString("eliminaSesion").equals("1")){
154                                 resul = true;
155                                 // Si no ha habido ningun problema limpiamos los campos
156                                 // mostrados por pantalla
157                                 ventanaSesiones.getTextFieldNombre().setText("");
158                                 ventanaSesiones.getSpinnerNumUser().setValue(0);
159                                 ventanaSesiones.getDateChooser().setDate(null);
160                                 Calendar calendar = Calendar.getInstance();
161                                 calendar.set(Calendar.HOUR_OF_DAY, 24);
162                                 calendar.set(Calendar.MINUTE, 0);
163                                 calendar.set(Calendar.SECOND, 0);
164                                 ventanaSesiones.getTimeSpinner().getModel().setValue(
165                                                 calendar.getTime());
166                                 //Eliminamos la sesion de los array
167                                 if(option==0 && nomSesionesActivas.size()>0){
168                                         idSesionesActivas.remove(posSesionMostrada);
169                                         nomSesionesActivas.remove(posSesionMostrada);
170                                         int index = idSesiones.indexOf(idSesionMostrada);
```

```
171                                     idSesiones.remove(index);
172                                     nomSesiones.remove(index);
173                             } else if (option==2 && nomSesiones.size()>0){
174                                     idSesiones.remove(posSesionMostrada);
175                                     nomSesiones.remove(posSesionMostrada);
176                                     int index = idSesionesActivas.indexOf(idSesionMostrada);
177                                     idSesionesActivas.remove(index);
178                                     nomSesionesActivas.remove(index);
179                             }
180                     }
181             } catch(Exception e){
182                     throw e;
183             }
184
185             return resul;
186
187     }
188
189
190     /*
191      * Metodo que devuelve el ID de la sesion que se esta mostrando por pantalla
192      */
193     public String getIdSesionMostrada(){
194             return this.idSesionMostrada;
195     }
196
197     /*
198      * Metodo que se asociara al listener de la lista para mostar
199      * la sesion seleccionada por pantalla
200      */
201     private void mostrarSesion(MouseEvent e){
202             JSONArray sesion;
203         JList<?> list = (JList<?>)e.getSource();
204         if (e.getClickCount() == 2) {
205                 try{
206                         // Obtenemos la sesion sobre la que se ha hecho click
207                         posSesionMostrada = list.locationToIndex(e.getPoint());
208                         switch(ventanaSesiones.getComboBox().getSelectedIndex()){
209                         case 0: idSesionMostrada = idSesionesActivas.get(posSesionMostrada);
210                                         sesion = gestor.sesion(idSesionMostrada);
211                                         break;
212                         case 1: idSesionMostrada = idSesionesNoActivas.get(posSesionMostrada);
213                                         sesion = gestor.sesion(idSesionMostrada);
214                                         break;
215                         default: idSesionMostrada =  idSesiones.get(posSesionMostrada);
216                                         sesion = gestor.sesion(idSesionMostrada);
217                                         break;
218                         }
219                         // Modificamos los campos de la ventana para mostrarle al usuario
220                         // toda la informacion de la sesion.
221                         ventanaSesiones.getTextFieldNombre().setEnabled(false);
222                         ventanaSesiones.getSpinnerNumUser().setEnabled(false);
223                         ventanaSesiones.getDateChooser().setEnabled(false);
224                         ventanaSesiones.getTimeSpinner().setEnabled(false);
225                         ventanaSesiones.getTextFieldNombre().setText(sesion.getJSONObject(0)
226                                         .getString("nombre_Sesion"));
227                         String fecha = sesion.getJSONObject(0).getString("fecha");
228                         int index = fecha.indexOf(" ");
229                         Date date = new SimpleDateFormat("yyyy-mm-dd").parse(
230                                         String.valueOf(fecha.substring(0, index)));
231                         ventanaSesiones.getDateChooser().setDate(date);
232                         Calendar calendar = Calendar.getInstance();
233                         calendar.set(
234                                         Calendar.HOUR_OF_DAY, Integer.parseInt(
235                                                         fecha.substring(index+1, index+3)));
236                         calendar.set(
237                                         Calendar.MINUTE, Integer.parseInt(
238                                                         fecha.substring(index+4, index+6)));
239                         calendar.set(
240                                         Calendar.SECOND, Integer.parseInt(
241                                                         fecha.substring(index+7, index+9)));
242                         ventanaSesiones.getTimeSpinner().setValue(calendar.getTime());
243                         ventanaSesiones.getSpinnerNumUser().setValue(Integer.parseInt(
244                                         sesion.getJSONObject(0).getString("num_Usuarios")));
245                         if(sesion.getJSONObject(0).getString("activa").equals("1")){
246                                 //Si la sesion seleccionada esta activa podremos eliminarla
247                                 ventanaSesiones.getButtonDelete().setEnabled(true);
248                                 ventanaSesiones.getButtonEdit().setEnabled(true);
249                         } else{
250                                 // En caso contrario, no se activara el boton
251                                 // para eliminar la sesion
252                                 ventanaSesiones.getButtonDelete().setEnabled(false);
253                                 ventanaSesiones.getButtonEdit().setEnabled(false);
254                         }
255                 } catch (Exception ex){
```

```
256                         ex.printStackTrace();
257                     }
258          }
259         }
260
261 }
```

VentanaEjercicios.java

"VentanaEjercicios.java"

```
1  package monitorgym;
2
3  import java.awt.Color;
4  import java.awt.Container;
5  import java.awt.Font;
6  import java.awt.Frame;
7  import java.awt.event.MouseAdapter;
8  import java.awt.event.MouseEvent;
9
10 import javax.swing.BorderFactory;
11 import javax.swing.JFrame;
12 import javax.swing.JOptionPane;
13 import javax.swing.JPanel;
14 import javax.swing.JScrollPane;
15 import javax.swing.SwingConstants;
16 import javax.swing.UIManager;
17 import javax.swing.border.AbstractBorder;
18 import javax.swing.border.Border;
19 import javax.swing.ImageIcon;
20 import javax.swing.JLabel;
21
22 import uk.co.caprica.vlcj.discovery.NativeDiscovery;
23
24 import java.awt.SystemColor;
25
26 import javax.swing.JButton;
27
28 import java.awt.event.ActionListener;
29 import java.awt.event.ActionEvent;
30
31 @SuppressWarnings({ "serial" })
32 public class VentanaEjercicios extends JFrame {
33
34         private Container contentPane;
35         private JLabel buttonLeftArrow;
36         private JLabel buttonRightArrow;
37
38         private String idSesion;
39         private DynamicListEjercicios listaEjercicios, listaEjerciciosSesion;
40         private GestorPeticiones gestor = new GestorPeticiones();
41         private JLabel lblDescripcion;
42         private JLabel lblDuracion;
43         private JLabel labelNombreEjercicio;
44         private JLabel labelDescripcionEjercicio;
45         private JLabel labelDuracionEjercicio;
46
47         private HiloGetIpUsuario server;
48
49         private int[] numUsuarioMostrado;
50
51         private static final int PUERTO_SERVER = 12345;
52
53         private JLabel labelBubbles1;
54         private JLabel labelBubbles2;
55         private JLabel labelBubbles3;
56         private JLabel labelBubbles4;
57
58         private JLabel labelCross1;
59         private JLabel labelCross2;
60         private JLabel labelCross3;
61         private JLabel labelCross4;
62
63
64         /**
65          * Create the frame.
66          * @throws Exception
67          */
68         public VentanaEjercicios(String sesion) {
69
70                 try{
71                         idSesion = sesion;
72                         numUsuarioMostrado = new int[4];
73
74                         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
75                      setTitle("Ejercicios");
76                      // Fijamos la ventana a pantalla completa
77                      setExtendedState(Frame.MAXIMIZED_BOTH);
78
79                      contentPane = getContentPane();
80                      contentPane.setLayout(null);
81                      setBounds(0,0,1300,700);
82
83                      // Lista con los ejercicios que forman parte de la sesion
84                      listaEjerciciosSesion = new DynamicListEjercicios(
85                                      gestor.ejerciciosSesion(idSesion), this, true);
86                      //Insertamos la lista en un JScrollPane
87                      JScrollPane scrollPaneSesion = new JScrollPane(listaEjerciciosSesion);
88                      scrollPaneSesion.setVerticalScrollBarPolicy(
89                                      JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
90                      scrollPaneSesion.setBounds(51, 58, 141, 195);
91                      contentPane.add(scrollPaneSesion);
92
93                      // Lista con todos los ejercicios que NO forman parte de la sesion
94                      listaEjercicios = new DynamicListEjercicios(
95                                      gestor.ejerciciosNoSesion(idSesion), this, false);
96                      //Insertamos la lista en un JScrollPane
97                      JScrollPane scrollPaneEjercicios = new JScrollPane(listaEjercicios);
98                      scrollPaneEjercicios.setVerticalScrollBarPolicy(
99                                      JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
100                     scrollPaneEjercicios.setBounds(267, 58, 141, 195);
101                     contentPane.add(scrollPaneEjercicios);
102
103                     //Boton para introducir un ejercicio en la sesion
104                     buttonLeftArrow = new JLabel("");
105                     buttonLeftArrow.setIcon(
106                                     new ImageIcon(
107                                                     VentanaEjercicios.class.getResource(
108                                                                     "/icons/left_arrow_icon-48.png")));
109                     buttonLeftArrow.setBounds(208, 81, 49, 48);
110                     buttonLeftArrow.setEnabled(false);
111                     buttonLeftArrow.addMouseListener(new MouseAdapter(){
112                             public void mouseClicked(MouseEvent e){
113                                     try{
114                                             // Introducimos el ejercicio en la
115                                             // lista correspondiente y actualizamos
116                                             listaEjerciciosSesion.addEjercicioLista(
117                                                             listaEjercicios.getIdEjercicioSeleccionado());
118                                             listaEjercicios.removeEjercicioLista();
119                                             // Peticion al servidor para introducimos
120                                             // el ejercicio en la sesion correspondiente
121                                             gestor.addEjercicioSesion(
122                                                             idSesion,
123                                                             listaEjercicios.getIdEjercicioSeleccionado());
124                                             buttonLeftArrow.setEnabled(false);
125                                     } catch(Exception ex){
126                                             ex.printStackTrace();
127                                     }
128                             }
129                     });
130                     contentPane.add(buttonLeftArrow);
131
132                     //Boton para eliminar un ejercicio de la lista de
133                     // ejercicios de la sesion
134                     buttonRightArrow = new JLabel("");
135                     buttonRightArrow.setIcon(
136                                     new ImageIcon(
137                                                     VentanaEjercicios.class.getResource(
138                                                                     "/icons/right_arrow-48.png")));
139                     buttonRightArrow.setBounds(208, 141, 49, 63);
140                     buttonRightArrow.setEnabled(false);
141                     buttonRightArrow.addMouseListener(new MouseAdapter(){
142                             public void mouseClicked(MouseEvent e){
143                                     try{
144                                             // Introducimos el ejercicio en la lista
145                                             // correspondiente y actualizamos
146                                             listaEjercicios.addEjercicioLista(
147                                                             listaEjerciciosSesion.getIdEjercicioSeleccionado());
148                                             listaEjerciciosSesion.removeEjercicioLista();
149                                             // Peticion al servidor para eliminar el ejercicio de la sesion
150                                             gestor.removeEjercicioSesion(
151                                                             idSesion,
152                                                             listaEjerciciosSesion.getIdEjercicioSeleccionado());
153                                             buttonRightArrow.setEnabled(false);
154                                     } catch(Exception ex){
155                                             ex.printStackTrace();
156                                     }
157                             }
158                     });
159                     contentPane.add(buttonRightArrow);
```

```
160
161                     AbstractBorder border = new TextBubbleBorder(Color.BLACK,2,16,16,false);;
162
163                     // Seccion de la pantalla en la que mostraremos toda la informacion relativa
164                     // al ejercicio seleccinado
165                     JPanel panelEjercicio = new JPanel();
166                     panelEjercicio.setBorder(border);
167                     panelEjercicio.setBackground(new Color(210, 105, 30));
168                     panelEjercicio.setBounds(51, 280, 357, 287);
169                     getContentPane().setLayout(null);
170                     contentPane.add(panelEjercicio);
171                     panelEjercicio.setLayout(null);
172
173                     lblDescripcion = new JLabel("Descripci\u00F3n:");
174                     lblDescripcion.setBounds(10, 108, 86, 14);
175                     panelEjercicio.add(lblDescripcion);
176
177                     lblDuracion = new JLabel("Duraci\u00F3n:");
178                     lblDuracion.setBounds(10, 52, 64, 14);
179                     panelEjercicio.add(lblDuracion);
180
181                     labelNombreEjercicio = new JLabel("");
182                     //Texto centrado
183                     labelNombreEjercicio.setHorizontalAlignment(SwingConstants.CENTER);
184                     //Texto en negrita y mas grande que el definido por defecto
185                     Font font = labelNombreEjercicio.getFont();
186                     Font boldFont = new Font(font.getFontName(), Font.BOLD, 20);
187                     labelNombreEjercicio.setFont(boldFont);
188                     labelNombreEjercicio.setBounds(10, 11, 337, 30);
189                     panelEjercicio.add(labelNombreEjercicio);
190
191                     labelDescripcionEjercicio = new JLabel("");
192                     Border descripcionBorder = BorderFactory.createEmptyBorder(5,5,5,5);
193                     labelDescripcionEjercicio.setBorder(descripcionBorder);
194                     labelDescripcionEjercicio.setBackground(SystemColor.window);
195                     labelDescripcionEjercicio.setOpaque(true);
196                     labelDescripcionEjercicio.setVerticalAlignment(SwingConstants.TOP);
197                     JScrollPane scrollPaneDescripcion = new JScrollPane(
198                                     labelDescripcionEjercicio);
199                     scrollPaneDescripcion.setVerticalScrollBarPolicy(
200                                     JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
201                     scrollPaneDescripcion.setBounds(10, 133, 337, 127);
202                     panelEjercicio.add(scrollPaneDescripcion);
203
204                     labelDuracionEjercicio = new JLabel("");
205                     labelDuracionEjercicio.setHorizontalAlignment(SwingConstants.CENTER);
206                     labelDuracionEjercicio.setBackground(SystemColor.window);
207                     labelDuracionEjercicio.setBounds(10, 77, 86, 20);
208                     labelDuracionEjercicio.setOpaque(true);
209                     panelEjercicio.add(labelDuracionEjercicio);
210
211
212                     // Nuevo hilo para conectar con la aplicacion del entrenado
213                     server = new HiloGetIpUsuario(idSesion);
214                     server.start();
215
216
217                     // Preparamos el espacio donde se reproduciran los videos
218                     // Habra cuatro paneles de video
219                     JPanel panelVideo1 = new JPanel();
220                     panelVideo1.setBackground(Color.WHITE);
221                     panelVideo1.setBounds(533, 104, 336, 222);
222                     panelVideo1.setLayout(null);
223                     contentPane.add(panelVideo1);
224
225                     JPanel panelVideo2 = new JPanel();
226                     panelVideo2.setBackground(Color.WHITE);
227                     panelVideo2.setBounds(901, 104, 336, 234);
228                     contentPane.add(panelVideo2);
229                     panelVideo2.setLayout(null);
230
231                     JPanel panelVideo3 = new JPanel();
232                     panelVideo3.setBackground(Color.WHITE);
233                     panelVideo3.setBounds(533, 368, 336, 234);
234                     contentPane.add(panelVideo3);
235                     panelVideo3.setLayout(null);
236
237                     JPanel panelVideo4 = new JPanel();
238                     panelVideo4.setBackground(Color.WHITE);
239                     panelVideo4.setBounds(901, 368, 336, 234);
240                     contentPane.add(panelVideo4);
241                     panelVideo4.setLayout(null);
242
243
244                     // Etiquetas para el nombre de usuario que se muestra en cada reproductor
```

```
245                     JLabel labelNombreVideo1 = new JLabel("");
246                     labelNombreVideo1.setBounds(533, 81, 262, 24);
247                     getContentPane().add(labelNombreVideo1);
248
249                     JLabel labelNombreVideo2 = new JLabel("");
250                     labelNombreVideo2.setBounds(901, 81, 262, 24);
251                     getContentPane().add(labelNombreVideo2);
252
253                     JLabel labelNombreVideo3 = new JLabel("");
254                     labelNombreVideo3.setBounds(533, 602, 262, 24);
255                     getContentPane().add(labelNombreVideo3);
256
257                     JLabel labelNombreVideo4 = new JLabel("");
258                     labelNombreVideo4.setBounds(901, 602, 262, 24);
259                     getContentPane().add(labelNombreVideo4);
260
261
262                     // Botones para enviar mensaje al cliente
263                     labelBubbles1 = new JLabel("");
264                     labelBubbles1.setIcon(
265                                     new ImageIcon(
266                                                     VentanaEjercicios.class.getResource(
267                                                                     "/icons/bubbles_icon-16.png")));
268                     labelBubbles1.setBounds(820, 79, 21, 14);
269                     labelBubbles1.addMouseListener(new ClickMensaje(0));
270                     getContentPane().add(labelBubbles1);
271
272                     labelBubbles2 = new JLabel("");
273                     labelBubbles2.setIcon(
274                                     new ImageIcon(
275                                                     VentanaEjercicios.class.getResource(
276                                                                     "/icons/bubbles_icon-16.png")));
277                     labelBubbles2.setBounds(1185, 79, 21, 14);
278                     labelBubbles2.addMouseListener(new ClickMensaje(1));
279                     getContentPane().add(labelBubbles2);
280
281                     labelBubbles3 = new JLabel("");
282                     labelBubbles3.setIcon(
283                                     new ImageIcon(
284                                                     VentanaEjercicios.class.getResource(
285                                                                     "/icons/bubbles_icon-16.png")));
286                     labelBubbles3.setBounds(820, 612, 21, 14);
287                     labelBubbles3.addMouseListener(new ClickMensaje(2));
288                     getContentPane().add(labelBubbles3);
289
290                     labelBubbles4 = new JLabel("");
291                     labelBubbles4.setIcon(
292                                     new ImageIcon(
293                                                     VentanaEjercicios.class.getResource(
294                                                                     "/icons/bubbles_icon-16.png")));
295                     labelBubbles4.setBounds(1185, 612, 21, 14);
296                     labelBubbles4.addMouseListener(new ClickMensaje(3));
297                     getContentPane().add(labelBubbles4);
298
299
300                     // Botones para dejar de reproducir el video de un cliente
301                     labelCross1 = new JLabel("");
302                     labelCross1.setIcon(
303                                     new ImageIcon(
304                                                     VentanaEjercicios.class.getResource(
305                                                                     "/icons/cross_icon-16.png")));
306                     labelCross1.setBounds(849, 81, 21, 14);
307                     getContentPane().add(labelCross1);
308
309                     labelCross2 = new JLabel("");
310                     labelCross2.setIcon(
311                                     new ImageIcon(
312                                                     VentanaEjercicios.class.getResource(
313                                                                     "/icons/cross_icon-16.png")));
314                     labelCross2.setBounds(1216, 79, 21, 14);
315                     getContentPane().add(labelCross2);
316
317                     labelCross3 = new JLabel("");
318                     labelCross3.setIcon(
319                                     new ImageIcon(
320                                                     VentanaEjercicios.class.getResource(
321                                                                     "/icons/cross_icon-16.png")));
322                     labelCross3.setBounds(849, 612, 21, 14);
323                     getContentPane().add(labelCross3);
324
325                     labelCross4 = new JLabel("");
326                     labelCross4.setIcon(
327                                     new ImageIcon(
328                                                     VentanaEjercicios.class.getResource(
329                                                                     "/icons/cross_icon-16.png")));
```

```
330                     labelCross4.setBounds(1216, 613, 21, 14);
331                     getContentPane().add(labelCross4);
332
333
334                     // Iconos para comenzar a reproducir el video
335                     JLabel labelPlay1 = new JLabel("");
336                     labelPlay1.setBounds(136, 85, 64, 64);
337                     panelVideo1.add(labelPlay1);
338                     labelPlay1.setIcon(
339                                     new ImageIcon(
340                                                     VentanaEjercicios.class.getResource(
341                                                                     "/icons/play_icon-64.png")));
342                     labelPlay1.addMouseListener(
343                                     new ClickVideo(
344                                                     labelPlay1, labelNombreVideo1,
345                                                     panelVideo1, 0, labelCross1, this));
346
347                     JLabel labelPlay2 = new JLabel("");
348                     labelPlay2.setBounds(141, 86, 64, 63);
349                     panelVideo2.add(labelPlay2);
350                     labelPlay2.setIcon(
351                                     new ImageIcon(
352                                                     VentanaEjercicios.class.getResource(
353                                                                     "/icons/play_icon-64.png")));
354                     labelPlay2.addMouseListener(
355                                     new ClickVideo(
356                                                     labelPlay2, labelNombreVideo2,
357                                                     panelVideo2, 1, labelCross2, this));
358
359                     JLabel labelPlay3 = new JLabel("");
360                     labelPlay3.setBounds(135, 82, 64, 63);
361                     panelVideo3.add(labelPlay3);
362                     labelPlay3.setIcon(
363                                     new ImageIcon(
364                                                     VentanaEjercicios.class.getResource(
365                                                                     "/icons/play_icon-64.png")));
366                     labelPlay3.addMouseListener(
367                                     new ClickVideo(
368                                                     labelPlay3, labelNombreVideo3,
369                                                     panelVideo3, 2, labelCross3, this));
370
371                     JLabel labelPlay4 = new JLabel("");
372                     labelPlay4.setBounds(139, 82, 64, 63);
373                     panelVideo4.add(labelPlay4);
374                     labelPlay4.setIcon(
375                                     new ImageIcon(
376                                                     VentanaEjercicios.class.getResource(
377                                                                     "/icons/play_icon-64.png")));
378                     labelPlay4.addMouseListener(
379                                     new ClickVideo(
380                                                     labelPlay4, labelNombreVideo4,
381                                                     panelVideo4, 3, labelCross4, this));
382
383                     JLabel lblEjerciciosDeLa = new JLabel("Ejercicios de la sesi\u00F3n");
384                     lblEjerciciosDeLa.setBounds(63, 23, 129, 24);
385                     getContentPane().add(lblEjerciciosDeLa);
386
387                     JLabel lblNewLabel = new JLabel("Resto de ejercicios");
388                     lblNewLabel.setBounds(281, 23, 106, 24);
389                     getContentPane().add(lblNewLabel);
390
391                     // Boton para dar por finalizada una sesion
392                     JButton btnFinalizarSesin = new JButton("Finalizar Sesi\u00F3n");
393                     btnFinalizarSesin.addActionListener(new ActionListener() {
394                             public void actionPerformed(ActionEvent e) {
395                                     try {
396                                             // Liberamos el puerto solicitado por UPnP
397                                             server.releasePort();
398                                             // La sesion pasa a no estar activa
399                                             gestor.setSesionNoActiva(idSesion);
400                                     } catch (Exception e1) {
401                                             e1.printStackTrace();
402                                     }
403                             }
404                     });
405                     btnFinalizarSesin.setBounds(833, 24, 118, 23);
406                     getContentPane().add(btnFinalizarSesin);
407
408                     // Botones de chat y cerrar video inicialmente desactivados
409                     desactivarBotones();
410
411
412                     // Cargamos las librerias necesarias para el funcionamiento de VLCj
413                     new NativeDiscovery().discover();
414
```

```
415                 } catch (Exception e){
416                         System.out.println("Excepcion VentanaEjercicios");
417                         e.printStackTrace();
418                 }
419         }
420
421         /*
422          * Metodo para obtener la etiqueta para el nombre del ejercicio seleccionado
423          */
424         public JLabel getLabelNombreEjercicio(){
425                 return this.labelNombreEjercicio;
426         }
427
428         /*
429          * Metodo para obtener la etiqueta para la descripcion del ejercicio seleccionado
430          */
431         public JLabel getLabelDescripcionEjercicio(){
432                 return this.labelDescripcionEjercicio;
433         }
434
435         /*
436          * Metodo para obtener la etiqueta para la duracion del ejercicio seleccionado
437          */
438         public JLabel getLabelDuracionEjercicio(){
439                 return this.labelDuracionEjercicio;
440         }
441
442         /*
443          * Metodo para obtener el boton de introduccion de ejercicio en la sesion
444          */
445         public JLabel getButtonLeftArrow(){
446                 return this.buttonLeftArrow;
447         }
448
449         /*
450          * Metodo para obtener el boton de eliminacion de ejercicio de la sesion
451          */
452         public JLabel getButtonRightArrow(){
453                 return this.buttonRightArrow;
454         }
455
456         /*
457          * Metodo para activar los botones de chat y cerrar video dependiendo de
458          * que reproductor se ha activado
459          */
460         public void activarBotones(int i){
461                 switch(i){
462                         case 0:
463                                 labelBubbles1.setVisible(true);
464                                 labelCross1.setVisible(true);
465                                 break;
466                         case 1:
467                                 labelBubbles2.setVisible(true);
468                                 labelCross2.setVisible(true);
469                                 break;
470                         case 2:
471                                 labelBubbles3.setVisible(true);
472                                 labelCross3.setVisible(true);
473                                 break;
474                         case 3:
475                                 labelBubbles4.setVisible(true);
476                                 labelCross4.setVisible(true);
477                                 break;
478                 }
479         }
480
481         /*
482          * Metodo para desactivar los botones de chat y cerrar video de uno de los
483          * reproductores de video
484          */
485         public void desactivarBotones(int i){
486                 switch(i){
487                         case 0:
488                                 labelBubbles1.setVisible(false);
489                                 labelCross1.setVisible(false);
490                                 break;
491                         case 1:
492                                 labelBubbles2.setVisible(false);
493                                 labelCross2.setVisible(false);
494                                 break;
495                         case 2:
496                                 labelBubbles3.setVisible(false);
497                                 labelCross3.setVisible(false);
498                                 break;
499                         case 3:
```

```
500                             labelBubbles4.setVisible(false);
501                             labelCross4.setVisible(false);
502                             break;
503                     }
504             }
505
506             /*
507              * Metodo para desactivar todos los botones de chat y cerrar video
508              */
509             public void desactivarBotones(){
510                     labelBubbles1.setVisible(false);
511                     labelBubbles2.setVisible(false);
512                     labelBubbles3.setVisible(false);
513                     labelBubbles4.setVisible(false);
514                     labelCross1.setVisible(false);
515                     labelCross2.setVisible(false);
516                     labelCross3.setVisible(false);
517                     labelCross4.setVisible(false);
518             }
519
520             /*
521              * Listener para detectar el click sobre el boton de cerrar video
522              */
523             public class ClickCierraVideo extends MouseAdapter{
524
525                     private Thread mThread;
526                     private int mNumeroBoton;
527
528                     public ClickCierraVideo(Thread thread, int numeroBoton){
529                             mThread = thread;
530                             mNumeroBoton = numeroBoton;
531                     }
532
533                     public void mouseClicked(MouseEvent e){
534                             System.out.println("Cerrando video");
535                             // Para el hilo del video
536                             mThread.interrupt();
537                             // Desactiva los botones del reproductor
538                             desactivarBotones(mNumeroBoton);
539                     }
540             }
541
542             /*
543              * Listener para detectar click sobre el boton de chat
544              */
545             public class ClickMensaje extends MouseAdapter{
546
547                     private int mNumeroBoton;
548                     private String cartel = "Escriba el mensaje...";
549                     private String titulo;
550
551                     public ClickMensaje(int numeroBoton){
552                             mNumeroBoton = numeroBoton;
553                     }
554
555                     public void mouseClicked(MouseEvent e){
556                             titulo = server.getNomUsuario(numUsuarioMostrado[mNumeroBoton]);
557                             // Muestra el cuadro de introduccion del mensaje
558                             String mensaje = JOptionPane.showInputDialog(
559                                             null, cartel, titulo, JOptionPane.QUESTION_MESSAGE);
560                             if(mensaje != null){
561                                     // Crea un nuevo hilo...
562                                     HiloMensaje hiloMensaje =
563                                                     new HiloMensaje(server.getIpUsuario(numUsuarioMostrado[mNumeroBoton]),
564                                                                     PUERTO_SERVER, mensaje);
565                                     // ... y envia el mensaje
566                                     hiloMensaje.start();
567                             }
568                     }
569             }
570
571             /*
572              * Listener para detectar el click sobre el boton de reproduccion de video
573              */
574             public class ClickVideo extends MouseAdapter{
575
576                     private JLabel mLabelButton;
577                     private JLabel mLabelNombre;
578                     private JLabel mLabelCross;
579                     private JPanel mPanel;
580                     private HiloVideo mHilo;
581                     private Thread mThread;
582                     private int mNumeroBoton;
583                     private VentanaEjercicios mVentana;
584
```

```
585                 public ClickVideo(JLabel labelButton, JLabel labelNombre, JPanel panel,
586                                 int numeroBoton, JLabel labelCross, VentanaEjercicios ventana){
587                         mLabelButton = labelButton;
588                         mLabelNombre = labelNombre;
589                         mPanel = panel;
590                         mNumeroBoton = numeroBoton;
591                         mLabelCross = labelCross;
592                         mVentana = ventana;
593                 }
594
595                 public void mouseClicked(MouseEvent e){
596                         String titulo = "Usuarios";
597                         String mensaje = "Seleccione un usuario...";
598                         String[] array = new String[server.getUsuarios().size()];
599                         array = server.getUsuarios().toArray(array);
600
601                         // Se le muestra al usuario un mensaje para que seleccione al usuario que desea ver
602                         UIManager.put("OptionPane.background", new Color(0xFFA44D));
603                         String input = (String) JOptionPane.showInputDialog(null, mensaje, titulo,
604                                         JOptionPane.QUESTION_MESSAGE, null, array, server.getUsuarios().toString());
605                         if(input != null){
606                                 mLabelButton.setVisible(false);
607
608                                 // Realiza todas las operaciones necesarias para
609                                 // comenzar con la reproduccion del video
610                                 new NativeDiscovery().discover();
611                                 mHilo = new HiloVideo(mLabelButton, mLabelNombre, mVentana, mNumeroBoton);
612                                 mHilo.setBounds(0, 0, 336, 234);
613                                 mPanel.add(mHilo);
614                                 mPanel.revalidate();
615                                 mPanel.repaint();
616                                 int indexUser = server.getUsuarios().indexOf(input);
617                                 numUsuarioMostrado[mNumeroBoton] = indexUser;
618                                 // Fijamos la direccion del cliente que queremos reproducir
619                                 mHilo.setUrl("http:/"+server.getIpUsuario(indexUser));
620                                 // Reproduce el video
621                                 mThread = new Thread(mHilo);
622                                 mThread.start();
623                                 mLabelCross.addMouseListener(new ClickCierraVideo(mThread, mNumeroBoton));
624                                 mLabelNombre.setText(server.getNomUsuario(indexUser));
625                                 activarBotones(mNumeroBoton);
626                         }
627                 }
628         }
629 }
```

DynamicListEjercicios.java

"DynamicListEjercicios.java"

```
1  package monitorgym;
2
3  import java.awt.event.MouseAdapter;
4  import java.awt.event.MouseEvent;
5  import java.awt.event.MouseListener;
6  import java.util.ArrayList;
7
8  import javax.swing.DefaultListModel;
9  import javax.swing.JList;
10
11 import org.json.JSONArray;
12
13 /*
14  * Clase desde la que se gestionara todo lo relativo a mostrar
15  * un conjunto de ejercicios por pantalla.
16  */
17 public class DynamicListEjercicios extends JList<String> {
18
19         private ArrayList<String> idEjercicios;
20         private ArrayList<String> nomEjercicios;
21         private DefaultListModel<String> model;
22         private String idEjercicioSeleccionado;
23         private int posicionClick;
24         private GestorPeticiones gestor;
25         private VentanaEjercicios ventanaParent;
26         private boolean isSesion;
27
28         public DynamicListEjercicios(
29                         JSONArray ejercicios, VentanaEjercicios parent, boolean bool) throws Exception{
30
31                 String nombre;
32                 idEjercicios = new ArrayList<String>();
33                 nomEjercicios = new ArrayList<String>();
34                 model = new DefaultListModel<String>();
35                 gestor = new GestorPeticiones();
```

```
36                  ventanaParent = parent;
37                  isSesion = bool;
38                  this.setModel(model);
39                  this.setBounds(10, 36, 76, 175);
40                  try{
41                          // Introducimos todos los ejercicios obtenidos del servidor en la lista
42                          for(int i=0; i<ejercicios.length(); i++){
43                                  idEjercicios.add(ejercicios.getJSONObject(i).getString("Id_Ejercicio"));
44                                  nombre = ejercicios.getJSONObject(i).getString("nombre_Ejercicio");
45                                  nomEjercicios.add(nombre);
46                                  model.addElement(nombre);
47                          }
48                  } catch (Exception e){
49                          throw e;
50                  }
51                  this.addMouseListener(this.nuevoMouseListener());
52          }
53
54
55          /*
56           * Listener para detectar el ejercicio sobre el que se hace click.
57           * Almacena la ID de este ejercicio en idEjercicioSeleccionado
58           */
59          private MouseListener nuevoMouseListener() throws Exception{
60                  MouseListener mouseListener = new MouseAdapter(){
61                          public void mouseClicked(MouseEvent e){
62                                  JList<?> list = (JList<?>) e.getSource();
63                                  if (e.getClickCount() == 2){
64                                          posicionClick = list.locationToIndex(e.getPoint());
65                                          idEjercicioSeleccionado = idEjercicios.get(posicionClick);
66                                          try{
67                                                  JSONArray consulta = gestor.ejercicio(idEjercicioSeleccionado);
68                                                  if(consulta!= null){
69                                                          // Mostramos la informacion del ejercicio
70                                                          // seleccionado por pantalla
71                                                          ventanaParent.getLabelNombreEjercicio().setText(
72                                                                          consulta.getJSONObject(0).getString(
73                                                                                          "nombre_Ejercicio"));
74                                                          ventanaParent.getLabelDuracionEjercicio().setText(
75                                                                          consulta.getJSONObject(0).getString(
76                                                                                          "Duracion"));
77                                                          ventanaParent.getLabelDescripcionEjercicio().setText(
78                                                                          consulta.getJSONObject(0).getString(
79                                                                                          "Descripcion"));
80                                                          if(isSesion == true){
81                                                                  ventanaParent.getButtonRightArrow()
82                                                                          .setEnabled(true);
83                                                                  ventanaParent.getButtonLeftArrow()
84                                                                          .setEnabled(false);
85                                                          } else{
86                                                                  ventanaParent.getButtonLeftArrow()
87                                                                          .setEnabled(true);
88                                                                  ventanaParent.getButtonRightArrow()
89                                                                          .setEnabled(false);
90                                                          }
91                                                  }
92                                          } catch(Exception ex){
93                                                  ex.printStackTrace();
94                                          }
95                                  }
96                          }
97                  };
98
99                  return mouseListener;
100         }
101
102
103         /*
104          * Metodo que devuelve la ID del ejercicio seleccionado
105          */
106         public String getIdEjercicioSeleccionado(){
107                 return this.idEjercicioSeleccionado;
108         }
109
110
111         /*
112          * Metodo para introducir un nuevo ejercicio en la lista
113          */
114         public void addEjercicioLista(String idEjercicio) throws Exception{
115                 JSONArray consulta = gestor.ejercicio(idEjercicio);
116                 String nombre;
117                 for(int i=0; i<consulta.length(); i++){
118                         idEjercicios.add(consulta.getJSONObject(i).getString("Id_Ejercicio"));
119                         nombre = consulta.getJSONObject(i).getString("nombre_Ejercicio");
120                         nomEjercicios.add(nombre);
```

```
121                         model.addElement(nombre);
122                 }
123         }
124
125
126         /*
127          * Metodo para eliminar un ejercicio previamente seleccionado
128          */
129         public void removeEjercicioLista() throws Exception{
130                 idEjercicios.remove(posicionClick);
131                 nomEjercicios.remove(posicionClick);
132                 model.remove(posicionClick);
133         }
134
135
136         /*
137          * Metodo para cambiar los ejercicios que se muestran en la lista
138          * Introduce el ultimo elemento en la lista nomEjercicios, que
139          * sera el que no aparezca en la lista
140          */
141         public void cambiaDatosLista(){
142                 model.addElement(nomEjercicios.get(nomEjercicios.size()-1));
143         }
144
145
146         /*
147          * Metodo que devuelve el array con la id de los ejercicios
148          */
149         public ArrayList<String> getIdEjercicios(){
150                 return this.idEjercicios;
151         }
152
153
154         /*
155          * Metodo que devuelve el array con los nombres de los ejercicios
156          */
157         public ArrayList<String> getNomEjercicios(){
158                 return this.nomEjercicios;
159         }
160 }
```

HiloGetIpUsuario.java

"HiloGetIpUsuario.java"

```
1  package monitorgym;
2
3  import java.io.ObjectInputStream;
4  import java.net.ServerSocket;
5  import java.net.Socket;
6  import java.util.ArrayList;
7  import java.lang.String;
8
9  import weupnp.SelectorPuertoNat;
10
11
12 /*
13  * Clase para conseguir la ip del usuario a la cual
14  * realizar las peticiones de video
15  */
16 public class HiloGetIpUsuario extends Thread {
17
18         private static final int PORT = 54321;
19         private ArrayList<String> nomUsuarios;
20         private ArrayList<String> ipUsuarios;
21         private ServerSocket server;
22         private String ip_puerto, nombreUsuario, mIdSesion;
23         private SelectorPuertoNat upnp;
24         private GestorPeticiones gestor;
25
26         public HiloGetIpUsuario(String idSesion){
27                 mIdSesion = idSesion;
28                 nomUsuarios = new ArrayList<String>();
29                 ipUsuarios = new ArrayList<String>();
30                 gestor = new GestorPeticiones();
31         }
32
33         public void run(){
34                 try{
35                         // Hacemos mapeo del puerto en el NAT
36                         upnp = new SelectorPuertoNat(PORT);
37
38                         // Actualizamos la IP de la sesion almacenada en la base de datos
39                         gestor.actualizaIpMonitorSesion(mIdSesion, upnp.getPortUsed());
40                         // Creamos el socket a la espera de los usuarios
```

```
41                          server = new ServerSocket(upnp.getPortUsed());
42                          while(true){
43                                  System.out.println("Esperando conexiones usuarios...");
44                                  Socket socket = server.accept();
45                                  System.out.println("Conexion establecida");
46                                  if(socket.isConnected()){
47                                          ObjectInputStream input = new ObjectInputStream(socket.getInputStream());
48                                          ip_puerto = socket.getRemoteSocketAddress().toString();
49                                          // Almacenamos la IP del usuario y su nombre en dos listas
50                                          ipUsuarios.add(ip_puerto);
51                                          nombreUsuario = input.readObject().toString();
52                                          if(nomUsuarios.indexOf(nombreUsuario) == -1){
53                                                  nomUsuarios.add(nombreUsuario);
54                                          }
55                                          socket.close();
56                                  }
57                          }
58                  } catch(Exception e){
59                          e.printStackTrace();
60                  }
61          }
62
63          /*
64           * Metodo para obtener la IP de un usuario
65           */
66          public String getIpUsuario(int i){
67                  return ipUsuarios.get(i);
68          }
69
70          /*
71           * Metodo para obtener el nombre de un usuario
72           */
73          public String getNomUsuario(int i){
74                  return nomUsuarios.get(i);
75          }
76
77          /*
78           * Metodo para obtener una lista con todos los nombres
79           * de los usuarios
80           */
81          public ArrayList<String> getUsuarios(){
82                  return nomUsuarios;
83          }
84
85          /*
86           * Metodo para liberar el puerto mapeado anteriormente
87           */
88          public void releasePort() throws Exception{
89                  upnp.releasePort();
90          }
91  }
```

HiloMensaje.java

"HiloMensaje.java"

```
1   package monitorgym;
2
3   import java.io.ObjectOutputStream;
4   import java.net.Socket;
5
6   public class HiloMensaje extends Thread {
7
8           private String SERVER_IP, mMensaje;
9           private int SERVER_PORT;
10
11          public HiloMensaje(String ip, int port, String mensaje){
12                  System.out.println("IP mensaje: "+ip.substring(1,ip.indexOf(":")));
13                  SERVER_IP = ip.substring(1,ip.indexOf(":"));
14                  SERVER_PORT = port;
15                  mMensaje = mensaje;
16          }
17
18          public void run(){
19                  try{
20                          // Conectamos con el socket abierto en la aplicacion del usuario
21                          Socket socket = new Socket(SERVER_IP, SERVER_PORT);
22                          if(socket.isConnected()){
23                                  // Enviamos el mensaje
24                                  ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());
25                                  output.writeObject(mMensaje);
26                                  socket.close();
27                          }
28                  } catch(Exception e){
29                          e.printStackTrace();
```

```
30                  }
31          }
32  }
```

HiloVideo.java

"HiloVideo.java"

```
1   package monitorgym;
2
3   import javax.swing.JLabel;
4
5   import uk.co.caprica.vlcj.player.MediaPlayerFactory;
6   import uk.co.caprica.vlcj.player.embedded.EmbeddedMediaPlayer;
7   import uk.co.caprica.vlcj.player.embedded.videosurface.CanvasVideoSurface;
8   import uk.co.caprica.vlcj.runtime.windows.WindowsCanvas;
9
10  @SuppressWarnings({ "deprecation", "serial" })
11
12  public class HiloVideo extends WindowsCanvas implements Runnable {
13
14          private MediaPlayerFactory mediaPlayerFactory;
15          private EmbeddedMediaPlayer mediaPlayer;
16          private String url;
17          private JLabel mLabelButton;
18          private JLabel mLabelNombre;
19          private VentanaEjercicios mVentana;
20          private int mNumeroHilo;
21
22          public HiloVideo(JLabel labelButton, JLabel labelNombre, VentanaEjercicios ventana, int numeroHilo){
23                  mediaPlayerFactory = new MediaPlayerFactory("--video-filter=transform", "--transform-type=90");
24          mediaPlayer = mediaPlayerFactory.newEmbeddedMediaPlayer();
25          CanvasVideoSurface videoSurface = mediaPlayerFactory.newVideoSurface(this);
26          mediaPlayer.setVideoSurface(videoSurface);
27          mLabelButton = labelButton;
28          mLabelNombre = labelNombre;
29          mVentana = ventana;
30          mNumeroHilo = numeroHilo;
31          }
32
33          public void run(){
34                  while(!Thread.interrupted()){
35                          // Peticion para recibir el video y reproducirlo
36                          mediaPlayer.playMedia(url);
37                          try{
38                                  Thread.sleep(5000);
39                          } catch(Exception ex){
40                                  Thread.currentThread().interrupt();
41                          }
42                          while(!Thread.currentThread().isInterrupted()){
43                                  if(!mediaPlayer.isPlaying()){
44                                          // En el momento en que no recibimos mas video interrumpimos el hilo
45                                          System.out.println("No recibimos mas video");
46                                          Thread.currentThread().interrupt();
47                                  }
48                          }
49                  }
50                  System.out.println("Hilo interrumpido");
51                  this.setVisible(false);
52                  mLabelButton.setVisible(true);
53                  mLabelNombre.setText("");
54                  mVentana.desactivarBotones(mNumeroHilo);
55                  // Liberamos todos los recursos utilizamos por el reproductor
56                  mediaPlayer.release();
57          }
58
59          /*
60           * Metodo para fijar la IP a la que solicitar el video
61           */
62          public void setUrl(String string){
63                  url = string;
64          }
65  }
```

TextBubbleBorder.java

"TextBubbleBorder.java"

```
1   package monitorgym;
2
3   import java.awt.BasicStroke;
4   import java.awt.Color;
5   import java.awt.Component;
6   import java.awt.Graphics;
7   import java.awt.Graphics2D;
8   import java.awt.Insets;
```

```
9  import java.awt.Rectangle;
10 import java.awt.RenderingHints;
11 import java.awt.geom.Area;
12 import java.awt.geom.RoundRectangle2D;
13
14 import javax.swing.border.AbstractBorder;
15
16 @SuppressWarnings("serial")
17
18 class TextBubbleBorder extends AbstractBorder {
19
20     private Color color;
21     private int thickness = 4;
22     private int radii = 8;
23     private int pointerSize = 7;
24     private Insets insets = null;
25     private BasicStroke stroke = null;
26     private int strokePad;
27     RenderingHints hints;
28
29     TextBubbleBorder(
30             Color color) {
31         new TextBubbleBorder(color, 4, 8, 7);
32     }
33
34     TextBubbleBorder(
35             Color color, int thickness, int radii, int pointerSize) {
36         this.thickness = thickness;
37         this.radii = radii;
38         this.pointerSize = pointerSize;
39         this.color = color;
40
41         stroke = new BasicStroke(thickness);
42         strokePad = thickness / 2;
43
44         hints = new RenderingHints(
45                 RenderingHints.KEY_ANTIALIASING,
46                 RenderingHints.VALUE_ANTIALIAS_ON);
47
48         int pad = radii + strokePad;
49         int bottomPad = pad + pointerSize + strokePad;
50         insets = new Insets(pad, pad, bottomPad, pad);
51     }
52
53     TextBubbleBorder(
54             Color color, int thickness, int radii, int pointerSize, boolean left) {
55         this(color, thickness, radii, pointerSize);
56     }
57
58     @Override
59     public Insets getBorderInsets(Component c) {
60         return insets;
61     }
62
63     @Override
64     public Insets getBorderInsets(Component c, Insets insets) {
65         return getBorderInsets(c);
66     }
67
68     @Override
69     public void paintBorder(
70             Component c,
71             Graphics g,
72             int x, int y,
73             int width, int height) {
74
75         Graphics2D g2 = (Graphics2D) g;
76
77         int bottomLineY = height - thickness - pointerSize;
78
79         RoundRectangle2D.Double bubble = new RoundRectangle2D.Double(
80                 0 + strokePad,
81                 0 + strokePad,
82                 width - thickness,
83                 bottomLineY,
84                 radii,
85                 radii);
86
87         Area area = new Area(bubble);
88
89         g2.setRenderingHints(hints);
90
91         // Paint the BG color of the parent, everywhere outside the clip
92         // of the text bubble.
93         Component parent  = c.getParent();
```

```
94          if (parent!=null) {
95              Color bg = parent.getBackground();
96              Rectangle rect = new Rectangle(0,0,width, height);
97              Area borderRegion = new Area(rect);
98              borderRegion.subtract(area);
99              g2.setClip(borderRegion);
100             g2.setColor(bg);
101             g2.fillRect(0, 0, width, height);
102             g2.setClip(null);
103         }
104
105         g2.setColor(color);
106         g2.setStroke(stroke);
107         g2.draw(area);
108     }
109 }
```

SelectorPuertoNat.java

"SelectorPuertoNat.java"

```
1  package weupnp;
2
3  import java.net.InetAddress;
4  import java.text.DateFormat;
5  import java.util.Date;
6  import java.util.Map;
7
8  /*
9   * Clase que utiliza la libreria Weupnp para
10  * hacer el port mapping
11  */
12 public class SelectorPuertoNat {
13
14         private int SAMPLE_PORT;
15         private static boolean LIST_ALL_MAPPINGS = false;
16         private static GatewayDevice activeGW;
17
18         public SelectorPuertoNat(int puerto) throws Exception{
19
20                 SAMPLE_PORT = puerto;
21
22                 addLogLine("Starting weupnp");
23
24                 GatewayDiscover gatewayDiscover = new GatewayDiscover();
25                 addLogLine("Looking for Gateway Devices...");
26
27                 Map<InetAddress, GatewayDevice> gateways = gatewayDiscover.discover();
28
29                 if (gateways.isEmpty()) {
30                         addLogLine("No gateways found");
31                         addLogLine("Stopping weupnp");
32                         return;
33                 }
34                 addLogLine(gateways.size()+" gateway(s) found\n");
35
36                 int counter=0;
37                 for (GatewayDevice gw: gateways.values()) {
38                         counter++;
39                         addLogLine("Listing gateway details of device #" + counter+
40                                         "\n\tFriendly name: " + gw.getFriendlyName()+
41                                         "\n\tPresentation URL: " + gw.getPresentationURL()+
42                                         "\n\tModel name: " + gw.getModelName()+
43                                         "\n\tModel number: " + gw.getModelNumber()+
44                                         "\n\tLocal interface address: " +
45                                                 gw.getLocalAddress().getHostAddress()+"\n");
46                 }
47
48                 // choose the first active gateway for the tests
49                 activeGW = gatewayDiscover.getValidGateway();
50
51                 if (null != activeGW) {
52                         addLogLine("Using gateway: " + activeGW.getFriendlyName());
53                 } else {
54                         addLogLine("No active gateway device found");
55                         addLogLine("Stopping weupnp");
56                         return;
57                 }
58
59
60                 // testing PortMappingNumberOfEntries
61                 Integer portMapCount = activeGW.getPortMappingNumberOfEntries();
62                 addLogLine("GetPortMappingNumberOfEntries: " + (
63                                 portMapCount!=null?portMapCount.toString():"(unsupported)"));
64
```

```
65                  // testing getGenericPortMappingEntry
66                  PortMappingEntry portMapping = new PortMappingEntry();
67                  if (LIST_ALL_MAPPINGS) {
68                          int pmCount = 0;
69                          do {
70                                  if (activeGW.getGenericPortMappingEntry(pmCount,portMapping))
71                                          addLogLine("Portmapping #"+pmCount+" successfully retrieved ("+
72                                                          portMapping.getPortMappingDescription()+":"+
73                                                          portMapping.getExternalPort()+")");
74                                  else{
75                                          addLogLine("Portmapping #"+pmCount+" retrieval failed");
76                                          break;
77                                  }
78                                  pmCount++;
79                          } while (portMapping!=null);
80                  } else {
81                          if (activeGW.getGenericPortMappingEntry(0,portMapping))
82                                  addLogLine("Portmapping #0 successfully retrieved ("+
83                                                  portMapping.getPortMappingDescription()+":"+
84                                                  portMapping.getExternalPort()+")");
85                          else
86                                  addLogLine("Portmapping #0 retrival failed");
87                  }
88
89                  InetAddress localAddress = activeGW.getLocalAddress();
90                  addLogLine("Using local address: "+ localAddress.getHostAddress());
91                  String externalIPAddress = activeGW.getExternalIPAddress();
92                  addLogLine("External address: "+ externalIPAddress);
93
94                  addLogLine("Querying device to see if a port mapping already exists for port "+
95                                  SAMPLE_PORT);
96
97                  for(int i = 0; activeGW.getSpecificPortMappingEntry(
98                                  SAMPLE_PORT,"TCP",portMapping) && i<=10; SAMPLE_PORT++){
99                          if(i==10){
100                                 addLogLine("Port "+SAMPLE_PORT+" is already mapped. Aborting test.");
101                                 return;
102                         } else{
103                                 addLogLine("Port "+SAMPLE_PORT+" is already mapped. Traying next port");
104                         }
105                 }
106
107                 addLogLine("Mapping free. Sending port mapping request for port "+SAMPLE_PORT);
108
109                 // test static lease duration mapping
110                 if (activeGW.addPortMapping(
111                                 SAMPLE_PORT,SAMPLE_PORT,localAddress.getHostAddress(),"TCP","test")) {
112                         addLogLine("Mapping SUCCESSFUL.");
113                 }
114
115         }
116
117         private static void addLogLine(String line) {
118
119                 String timeStamp = DateFormat.getTimeInstance().format(new Date());
120                 String logline = timeStamp+": "+line+"\n";
121                 System.out.print(logline);
122         }
123
124         public void releasePort() throws Exception{
125
126                 addLogLine("Stopping weupnp");
127
128                 if (activeGW.deletePortMapping(SAMPLE_PORT,"TCP")) {
129                         addLogLine("Port mapping removed, test SUCCESSFUL");
130         } else {
131                         addLogLine("Port mapping removal FAILED");
132         }
133         }
134
135         public int getPortUsed(){
136                 return SAMPLE_PORT;
137         }
138 }
```

1.2 Aplicación Android

ListaSesiones.java

"ListaSesiones.java"

```
1 package com.proyecto.antonio.projectdesign;
2
3 import android.app.Activity;
4 import android.content.Context;
```

```
5  import android.content.Intent;
6  import android.content.SharedPreferences;
7  import android.os.AsyncTask;
8  import android.os.Bundle;
9  import android.os.Handler;
10 import android.util.Log;
11 import android.view.Menu;
12 import android.view.MenuItem;
13 import android.view.View;
14 import android.widget.AdapterView;
15 import android.widget.ArrayAdapter;
16 import android.widget.ListView;
17 import android.support.v4.widget.SwipeRefreshLayout;
18 import android.support.v4.widget.SwipeRefreshLayout.OnRefreshListener;
19
20 import com.proyecto.antonio.projectdesign.library.Httppostaux;
21
22 import org.apache.http.NameValuePair;
23 import org.apache.http.message.BasicNameValuePair;
24 import org.json.JSONArray;
25 import org.json.JSONException;
26 import org.json.JSONObject;
27
28 import java.util.ArrayList;
29 import java.util.List;
30
31 /**
32  * Created by Antonio on 22/07/2014.
33  */
34 public class ListaSesiones extends Activity {
35
36     SwipeRefreshLayout swipeRefreshLayout;
37     ListView lv;
38     List<String> array_list = new ArrayList<String>();
39     String URL_connect = "";
40     JSONArray jdata;
41
42     protected void onCreate(Bundle savedInstanceState) {
43         super.onCreate(savedInstanceState);
44         setContentView(R.layout.activity_listasesiones);
45         // Carga la IP del servidor almacenada en sharedpreferences
46         SharedPreferences settings = getSharedPreferences("ProjectGym", Context.MODE_PRIVATE);
47         if(settings != null) {
48             URL_connect = settings.getString("IP", "");
49         }
50         // Carga el listview que será modificado cada vez que se realice la consulta al servidor
51         lv = (ListView) findViewById(R.id.listasesiones);
52         // Establece el listener y los colores del swiperefreshlayout
53         swipeRefreshLayout = (SwipeRefreshLayout) findViewById(R.id.swiperefreshlayout);
54         swipeRefreshLayout.setOnRefreshListener(onRefreshListener);
55         swipeRefreshLayout.setColorScheme(R.color.c1, R.color.c2, R.color.c3, R.color.c4);
56
57         // Realiza en segundo plano la consulta para obtener la lista de sesiones activas
58         new asyncsql().execute();
59     }
60
61     @Override
62     public boolean onCreateOptionsMenu(Menu menu) {
63         // Añade las opciones del menú con el archivo my.xml
64         getMenuInflater().inflate(R.menu.my, menu);
65         return true;
66     }
67
68     /*
69      *  Especifica las funciones a realizar en caso de que se pulse una opción u otra
70      */
71     @Override
72     public boolean onOptionsItemSelected(MenuItem item) {
73         switch (item.getItemId()) {
74             case R.id.logout:
75                 SharedPreferences settings = getSharedPreferences("ProjectGym",
76                         Context.MODE_PRIVATE);
77                 SharedPreferences.Editor editor = settings.edit();
78                 editor.remove("pass");
79                 editor.commit();
80                 finish();
81                 return true;
82             default:
83                 return super.onOptionsItemSelected(item);
84         }
85     }
86
87     /*
88      *  Establece las funciones a realizar por el listener del swiperefreshlayout.
89      *  Cuando detecta un evento realiza una nueva peticion al servidor
```

```
90      *  El evento se disparará cuando desplacemos hacia abajo el swiperefreshlayout
91      */
92     OnRefreshListener onRefreshListener = new OnRefreshListener(){
93
94         @Override
95         public void onRefresh() {
96
97             new Handler().postDelayed(new Runnable() {
98
99                 @Override
100                public void run() {
101                    swipeRefreshLayout.setRefreshing(false);
102                    // Realiza de nuevo la consulta para obtener las sesiones activas
103                    new asyncsql().execute();
104                }
105
106            }, 2000);
107        }
108    };
109
110
111     /*                CLASE ASYNCTASK
112      *
113      *  Clase asíncrona que se encarga de lanzar la consulta y de imprimir el resultado en el
114      *  listview de la actividad. Donde se muestran las sesiones activas en el servidor.
115      *  Esta clase debe ser asíncrona para no provocar problemas de inconsistencia en la
116      *  app debido a los tiempos de respuesta del servidor.
117     */
118
119    class asyncsql extends AsyncTask< String, String, String > {
120
121        protected void onPreExecute() {
122        }
123
124        protected String doInBackground(String... params) {
125
126            ArrayList<NameValuePair> postparameters2send= new ArrayList<NameValuePair>();
127            // Realizamos la peticion y como respuesta se obtiene un array JSON
128            Httppostaux post =new Httppostaux();
129            jdata=post.getserverdata(postparameters2send, "http://"+
130                    URL_connect+"/gymserver/listasesiones.php");
131            array_list = new ArrayList<String>();
132            // Si lo que recibimos no es null y no está vacío
133            if (jdata!=null && jdata.length() > 0){
134                String nom_sesion,user_monitor;
135
136                // Bucle para obtener todos los JSONObject contenidos en el JSONArray recibido
137                for(int i=0;i<jdata.length();i++){
138                    try {
139                        // Se crea un objeto JSONObject y se rellena con
140                        // el elemento i del JSONArray recibido
141                        JSONObject json_data = jdata.getJSONObject(i); //Aquí está el fallo
142
143                        // Se obtienen los nombres de la sesion y del monitor
144                        nom_sesion = json_data.getString("nombre_Sesion");
145                        user_monitor = json_data.getString("user_Monitor");
146                        // Se añaden al array de string, que es lo que se mostrará en el listview
147                        array_list.add("Sesión: "+nom_sesion+"\nMonitor: "+user_monitor+"\n");
148                    } catch (JSONException e) {
149                        e.printStackTrace();
150                    }
151                }
152            }else{        // Se devuelve "err", se produjo un error al obtener el JSONArray del servidor
153                return "err";
154            }
155            return "ok";
156
157        }
158
159        /*
160         *  Función que se ejecuta tras doInBackground(), en ella se comprueba el resultado de la
161         *  consulta. Si es positivo se actualiza el listview con el nuevo array de string donde
162         *  están almacenadas las sesiones y los monitores. Si es negativo se muestra un aviso
163         *  de que no existen sesiones activas.
164         */
165        protected void onPostExecute(String result) {
166
167            if (result.equals("ok")) {
168                // Actualiza el listview
169                ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(ListaSesiones.this,
170                        android.R.layout.simple_list_item_1, array_list );
171                lv.setAdapter(arrayAdapter);
172                // Gestiona clicks en la lista de sesiones
173                lv.setClickable(true);
174                lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
```

```
175
176                     /*
177                      *  Detecta la posición en la que se ha hecho click, se accede al objeto JSON
178                      *  que se corresponde con esa posicion en el JSONArray. Obtiene los datos de
179                      *  nombre de la sesion, del monitor y la IP del monitor. Y lanza la actividad
180                      *  ListaEjercicios añadiendo los datos obtenidos de la sesión.
181                      */
182                     @Override
183                     public void onItemClick(AdapterView<?> arg0, View arg1, int position, long arg3) {
184
185                         try {
186                             JSONObject json_data = jdata.getJSONObject(position);
187                             String id_sesion = json_data.getString("Id_Sesion");
188                             String nom_sesion = json_data.getString("nombre_Sesion");
189                             String user_monitor = json_data.getString("user_Monitor");
190                             Intent myIntent = new Intent(ListaSesiones.this, ListaEjercicios.class);
191                             myIntent.putExtra("Id_Sesion", id_sesion);
192                             myIntent.putExtra("nom_Sesion",nom_sesion);
193                             myIntent.putExtra("user_Monitor",user_monitor);
194                             startActivity(myIntent);
195                         }catch (JSONException e) {
196                             e.printStackTrace();
197                         }
198                     }
199                 });
200             }
201             else {
202                 array_list.add("No hay sesiones activas");
203                 ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(ListaSesiones.this,
204                         android.R.layout.simple_list_item_1, array_list );
205                 lv.setAdapter(arrayAdapter);
206             }
207         }
208     }
209 }
```

ListaEjercicios.java

"ListaEjercicios.java"

```
1  package com.proyecto.antonio.projectdesign;
2
3  import android.app.Activity;
4  import android.content.Context;
5  import android.content.Intent;
6  import android.content.SharedPreferences;
7  import android.os.AsyncTask;
8  import android.os.Bundle;
9  import android.os.Handler;
10 import android.support.v4.widget.SwipeRefreshLayout;
11 import android.util.Log;
12 import android.view.View;
13 import android.view.Window;
14 import android.widget.AdapterView;
15 import android.widget.ArrayAdapter;
16 import android.widget.ListView;
17 import android.widget.TextView;
18 import android.widget.Toast;
19
20 import com.proyecto.antonio.projectdesign.library.Httppostaux;
21
22 import org.apache.http.NameValuePair;
23 import org.apache.http.message.BasicNameValuePair;
24 import org.json.JSONArray;
25 import org.json.JSONException;
26 import org.json.JSONObject;
27
28 import java.text.SimpleDateFormat;
29 import java.util.ArrayList;
30 import java.util.Date;
31 import java.util.List;
32 import java.text.DateFormat;
33
34 /**
35  * Created by Antonio on 22/07/2014.
36  */
37 public class ListaEjercicios extends Activity {
38
39     SwipeRefreshLayout swipeRefreshLayout;
40     ListView lv;
41     List<String> array_list = new ArrayList<String>();
42     String URL_connect = "";
43     JSONArray jdataAdd, jdataLista, jdataSesion;
44     String user;
45     String user_monitor;
```

```
46      String id_sesion;
47      SocketAsync socket;
48      boolean sesionInit = false;
49
50      protected void onCreate(Bundle savedInstanceState) {
51          super.onCreate(savedInstanceState);
52          requestWindowFeature(Window.FEATURE_NO_TITLE);
53          setContentView(R.layout.activity_listaejercicios);
54
55          // Carga el usuario que hizo el login, que se encuentra almacenado en sharedpreferences
56          SharedPreferences settings = getSharedPreferences("ProjectGym", Context.MODE_PRIVATE);
57          if(settings != null) {
58              user = settings.getString("user", "");
59              Log.d("usuario: ",user);
60          }
61
62          // Obtiene los datos de la sesion de ejercicios obtenidos en la actividad ListaSesiones
63          Intent myIntent = getIntent();
64          String nom_sesion = myIntent.getStringExtra("nom_Sesion");
65          user_monitor= myIntent.getStringExtra("user_Monitor");
66          URL_connect = settings.getString("IP", "");
67          id_sesion = myIntent.getStringExtra("Id_Sesion");
68
69          // Modifica los textview para informar al usuario de la sesión de ejercicios en la que
70          // se encuentra y del monitor que la creó
71          TextView tv = (TextView) findViewById(R.id.nombresesion);
72          tv.setText("Sesión: "+nom_sesion);
73          tv = (TextView) findViewById(R.id.nombremonitor);
74          tv.setText("Monitor: "+user_monitor);
75
76          // Carga el listview que será modificado cada vez que se realice la consulta al servidor
77          lv = (ListView) findViewById(R.id.listasesiones);
78
79          // Establece el listener y los colores del swiperefreshlayout
80          swipeRefreshLayout = (SwipeRefreshLayout) findViewById(R.id.swiperefreshlayout);
81          swipeRefreshLayout.setOnRefreshListener(onRefreshListener);
82          swipeRefreshLayout.setColorScheme(R.color.c1, R.color.c2, R.color.c3, R.color.c4);
83
84          // Realiza en segundo plano la consulta para obtener la lista de ejercicios de la sesión
85          new asyncgetejercicios().execute();
86      }
87
88      /*
89       *  Cuando se destruye la actividad informamos al servidor del monitor de que nos vamos a
90       *  desconectar de la sesión, se realiza de forma asíncrona.
91       */
92      @Override
93      protected void onDestroy(){
94          super.onDestroy();
95          //new asyncborra().execute();
96      }
97
98      /*
99       *  Establece las funciones a realizar por el listener del swiperefreshlayout.
100      *  Cuando detecta un evento realiza una nueva peticion al servidor
101      *  El evento se disparará cuando desplacemos hacia abajo el swiperefreshlayout
102      */
103     SwipeRefreshLayout.OnRefreshListener onRefreshListener =
104             new SwipeRefreshLayout.OnRefreshListener(){
105
106         @Override
107         public void onRefresh() {
108             new Handler().postDelayed(new Runnable() {
109
110                 @Override
111                 public void run() {
112                     swipeRefreshLayout.setRefreshing(false);
113                     // Realiza de nuevo la consulta para obtener la rutina de ejercicios
114                     new asyncgetejercicios().execute(user);
115                 }
116
117             }, 2000);
118         }
119     };
120
121
122      /*                CLASE ASYNCTASK
123       *
124       *  Clase asíncrona que se encarga de lanzar la consulta y de imprimir el resultado en el
125       *  listview de la actividad. Donde se muestran la lista de ejercicios de la sesión
126       *  Esta clase debe ser asíncrona para no provocar problemas de inconsistencia en la
127       *  app debido a los tiempos de respuesta del servidor.
128      */
129
130     class asyncgetejercicios extends AsyncTask< String, String, String > {
```

```
131
132         protected void onPreExecute() {
133         }
134
135         protected String doInBackground(String... params) {
136
137             SharedPreferences settings = getSharedPreferences("ProjectGym",
138                     Context.MODE_PRIVATE);
139
140              /*
141               *  Creamos un ArrayList del tipo nombre-valor para agregar los datos del
142               *  usuario y enviarlo mediante POST a nuestro sistema para su suscripción
143               *  en la sesión seleccionada
144              */
145             ArrayList<NameValuePair> postparameters2send= new ArrayList<NameValuePair>();
146             postparameters2send.add(new BasicNameValuePair("Id_Sesion", id_sesion));
147             postparameters2send.add(new BasicNameValuePair("nom_Usuario",
148                     settings.getString("user", "")));
149
150             // Realizamos la peticion y como respuesta se obtiene un array JSON
151             Httppostaux post =new Httppostaux();
152             jdataAdd=post.getserverdata(postparameters2send,
153                     "http://"+URL_connect+"/gymserver/addUserSesion.php");
154             if (jdataAdd!=null && jdataAdd.length() > 0){
155                 //Comprobamos que el usuario se ha suscrito correctamente
156                 try{
157                     if(jdataAdd.getJSONObject(0).getString("addUserSesion").equals("1")) {
158
159                         //Se ha suscrito correctamente
160
161                         // Petición para obtener todos los ejercicios de la sesión
162                         jdataLista=post.getserverdata(postparameters2send,
163                                 "http://"+URL_connect+"/gymserver/listaEjerciciosSesion.php");
164
165                         // Si lo que recibimos no es null y no está vacío
166                         if (jdataLista!=null && jdataLista.length() > 0){
167                             String nom_ejercicio;
168                             array_list = new ArrayList<String>();
169
170                             // Bucle para obtener todos los JSONObject contenidos en el JSONArray recibido
171                             for(int i=0;i<jdataLista.length();i++){
172                                 try {
173                                     // Se crea un objeto JSONObject y se rellena con
174                                     // el elemento i del JSONArray recibido
175                                     JSONObject json_data = jdataLista.getJSONObject(i);
176
177                                     // Se obtiene el nombre del ejercicio y se añada al array
178                                     nom_ejercicio = json_data.getString("nombre_Ejercicio");
179                                     array_list.add("Ejercicio: "+nom_ejercicio+"\n");
180
181
182                                     // Realizamos la petición para obtener los datos de la sesión
183                                     jdataSesion= post.getserverdata(postparameters2send,
184                                             "http://"+URL_connect+"/gymserver/getSesion.php");
185
186                                     // Si lo que recibimos no es null y no está vacío
187                                     if(jdataSesion!= null && jdataSesion.length()>0) {
188
189                                         // Comprobamos que la sesión ha comenzado
190                                         if(comparaFecha(jdataSesion.getJSONObject(0).getString("fecha"))){
191
192                                             // Ya ha comenzado la sesion
193                                             sesionInit = true;
194
195                                             // Obtenemos la IP del monitor y conectamos con él
196                                             socket = new SocketAsync(
197                                                     jdataSesion.getJSONObject(0).getString("ip_Monitor"),
198                                                     settings.getString("user", ""));
199                                             socket.execute();
200                                         }
201                                     }
202
203                                 } catch (JSONException e) {
204                                     e.printStackTrace();
205                                 }
206                             }
207                         }else{
208                                 // Se devuelve "err", se produjo un error al obtener el JSONArray del servidor
209                             array_list = new ArrayList<String>();
210                             return "err";
211                         }
212
213                     }
214                 } catch(Exception e){
215                     e.printStackTrace();
```

```
216                 }
217             }
218
219             return "ok";
220         }
221
222         /*
223          *  Función que se ejecuta tras doInBackground(), en ella se comprueba el resultado de la
224          *  consulta. Si es positivo se actualiza el listview con el nuevo array de string donde
225          *  están almacenadas los ejercicios de la sesión. Si es negativo se muestra un aviso
226          *  de que se ha producido un error.
227          */
228         protected void onPostExecute(String result) {
229
230             if (result.equals("ok")) {
231                 // Actualiza el listview
232                 ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(ListaEjercicios.this,
233                         android.R.layout.simple_list_item_1, array_list );
234                 lv.setAdapter(arrayAdapter);
235
236                 // Gestiona clicks en la lista de sesiones
237                 lv.setClickable(true);
238                 lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
239
240                     /*
241                      *  Detecta la posición en la que se ha hecho click, se accede al objeto JSON
242                      *  que se corresponde con esa posicion en el JSONArray. Obtiene los datos del
243                      *  ejercicio y lanza la actividad Ejercicio añadiendo estos datos en el intent.
244                      */
245                     @Override
246                     public void onItemClick(AdapterView<?> arg0, View arg1, int position, long arg3) {
247                         if(sesionInit) {
248                             try {
249                                 JSONObject json_data = jdataLista.getJSONObject(position);
250                                 String id_ejercicio = json_data.getString("Id_Ejercicio");
251                                 String nom_ejercicio = json_data.getString("Descripcion");
252                                 Intent myIntent = getIntent();
253                                 String user_monitor= myIntent.getStringExtra("user_Monitor");
254                                 String nom_sesion = myIntent.getStringExtra("nom_Sesion");
255                                 myIntent = new Intent(ListaEjercicios.this, Ejercicio.class);
256                                 myIntent.putExtra("Id_Ejercicio",id_ejercicio);
257                                 myIntent.putExtra("nom_Sesion",nom_sesion);
258                                 myIntent.putExtra("user_Monitor",user_monitor);
259                                 myIntent.putExtra("ip_Monitor",URL_connect);
260                                 myIntent.putExtra("nom_Ejercicio",nom_ejercicio);
261                                 startActivity(myIntent);
262                             }catch (JSONException e) {
263                                 e.printStackTrace();
264                             }
265                         } else{
266                             Toast.makeText(getApplicationContext(), "La sesión aún no ha comenzado",
267                                     Toast.LENGTH_LONG).show();
268                         }
269                     }
270                 });
271             }
272             else {
273                 array_list.add("Lista no disponible. Recargue en unos segundos");
274                 ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(ListaEjercicios.this,
275                         android.R.layout.simple_list_item_1, array_list );
276                 lv.setAdapter(arrayAdapter);
277             }
278         }
279
280
281         private boolean comparaFecha(String fecha){
282             boolean resultado = false;
283
284             DateFormat dateFormat = new SimpleDateFormat("yyyy-mm-dd HH:mm:ss");
285             Date date = new Date();
286             String currentDate = dateFormat.format(date);
287             if(currentDate.compareTo(fecha) > 0){
288                 resultado = true;
289             }
290
291             return resultado;
292         }
293     }
294
295
296
297      /*                CLASE ASYNCTASK
298       *
299       *  Clase asíncrona que se encarga de notificar al servidor del monitor de que el usuario
300       *  va a salir de la sesión de ejercicios. Se realiza de forma asíncrona para no dejar la
```

```
301      *  aplicación colgada esperando la respuesta.
302     */
303
304     class asyncborra extends AsyncTask< String, String, String > {
305
306         protected void onPreExecute() {
307         }
308
309         protected String doInBackground(String... params) {
310
311             /*
312              *  Creamos un ArrayList del tipo nombre-valor para agregar el nombre del usuario y
313              *  enviarlo mediante POST a nuestro sistema para eliminarlo de la sesión en el
314               *  servidor del monitor.
315              */
316             ArrayList<NameValuePair> postparameters2send= new ArrayList<NameValuePair>();
317             postparameters2send.add(new BasicNameValuePair("user",user));
318
319             // Realizamos la peticion HTTP
320             Httppostaux post =new Httppostaux();
321             post.getserverdata(postparameters2send,
322                     "http://"+URL_connect+"/gymserver/eliminausuario.php");
323
324             return "ok";
325         }
326
327         protected void onPostExecute(String result) {
328         }
329     }
330 }
```

Ejercicio.java

"Ejercicio.java"

```
1  package com.proyecto.antonio.projectdesign;
2
3  import android.content.Context;
4  import android.content.SharedPreferences;
5  import android.support.v4.app.FragmentActivity;
6  import android.app.AlertDialog;
7  import android.content.DialogInterface;
8  import android.content.Intent;
9  import android.media.MediaPlayer;
10 import android.os.AsyncTask;
11 import android.os.Bundle;
12 import android.util.DisplayMetrics;
13 import android.view.View;
14 import android.view.Window;
15 import android.view.WindowManager;
16 import android.widget.FrameLayout;
17 import android.widget.MediaController;
18 import android.widget.VideoView;
19
20 import com.proyecto.antonio.projectdesign.library.Httppostaux;
21
22 import org.apache.http.NameValuePair;
23 import org.apache.http.message.BasicNameValuePair;
24 import org.json.JSONArray;
25 import org.json.JSONException;
26 import org.json.JSONObject;
27
28 import java.net.DatagramSocket;
29 import java.net.Socket;
30 import java.util.ArrayList;
31 import java.net.ServerSocket;
32
33 import java.io.ObjectInputStream;
34
35
36
37
38 public class Ejercicio extends FragmentActivity {
39
40     private String URL_connect = "";
41     private JSONArray jdatainfo, jdatavideo;
42     private String id_ejercicio;
43     private String nom_sesion, user_monitor, nom_ejercicio, finalidad, duracion, descripcion;
44     private String urlVideo = null;
45
46     // Para recepcion de mensajes del monitor
47     private HiloMensajes myDatagramReceiver;
48     private DatagramSocket datagramSocket;
49
50     @Override
```

```
51      protected void onCreate(Bundle savedInstanceState) {
52          super.onCreate(savedInstanceState);
53
54          // Actividad a pantalla completa (sin título ni barra de estado)
55          requestWindowFeature(Window.FEATURE_NO_TITLE);
56          getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
57                  WindowManager.LayoutParams.FLAG_FULLSCREEN);
58          getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
59          setContentView(R.layout.activity_ejercicio);
60
61          // Recoge datos pasados desde la actividad ListaEjercicios
62          Intent myIntent = getIntent();
63          id_ejercicio =  myIntent.getStringExtra("Id_Ejercicio");
64          nom_sesion = myIntent.getStringExtra("nom_Sesion");
65          user_monitor= myIntent.getStringExtra("user_Monitor");
66          URL_connect = myIntent.getStringExtra("ip_Monitor");
67
68          // Crea un hilo nuevo, en el que se escuchará la llegada de mensajes del monitor
69          myDatagramReceiver = new HiloMensajes();
70          myDatagramReceiver.start();
71
72          // Carga el vídeo desde el servidor
73          new asyncvideo().execute();
74
75          // Esperamos para que llegue la respuesta del servidor con la URL del vídeo
76          try{
77              Thread.sleep(1000);
78          } catch (Exception e){
79              e.printStackTrace();
80          }
81          if(urlVideo != null){
82              VideoView videoView = (VideoView) findViewById(R.id.vistaVideo);
83              videoView.setVideoPath(urlVideo);
84              videoView.setMediaController(new MediaController(this));
85              videoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
86                  @Override
87                  public void onPrepared(MediaPlayer mediaPlayer) {
88                      mediaPlayer.setLooping(true);
89                      mediaPlayer.setVolume(0,0);
90                  }
91              });
92
93              //Establece que el vídeo se reproduzca a pantalla completa
94              DisplayMetrics metrics = new DisplayMetrics();
95              getWindowManager().getDefaultDisplay().getMetrics(metrics);
96              FrameLayout.LayoutParams params = (FrameLayout.LayoutParams) videoView.getLayoutParams();
97              params.width = metrics.widthPixels;
98              params.height = metrics.heightPixels;
99              params.leftMargin = 0;
100             params.rightMargin = 0;
101             params.bottomMargin = 0;
102             params.topMargin = 0;
103             videoView.setLayoutParams(params);
104
105             // Comienza la reproducción del vídeo
106             videoView.requestFocus();
107             videoView.start();
108         }
109
110         // Realiza una consulta asíncrona para obtener la información acerca del ejercicio
111         new asyncinfo().execute();
112     }
113
114
115     @Override
116     public void onDestroy() {
117         super.onDestroy();
118         // Cierra el socket y acaba con el hilo
119         datagramSocket.close();
120         myDatagramReceiver.kill();
121     }
122
123
124     /*
125      *  Esta función muestra un mensaje en el que se describe la información acerca del ejercicio:
126      *       · Nombre de la sesión
127      *       · Nombre del monitor
128      *       · Nombre del ejercicio
129      *       · Finalidad del ejercicio
130      *       · Duración
131      *       · Descripción del ejercicio
132      */
133     public void mostrar_Descripcion(View view){
134         AlertDialog.Builder alert = new AlertDialog.Builder(this);
135         // Si se produce algun error al traer la información del ejercicio desde el
```

```
136         // servidor del monitor, se muestra un mensaje informando sobre ello.
137         nom_ejercicio = nom_ejercicio != null ? nom_ejercicio : "no encontrado";
138         finalidad = finalidad != null ? finalidad:"no encontrada";
139         duracion = duracion != null?duracion:"no encontrada";
140         descripcion = descripcion != null ? descripcion :"no encontrada";
141         alert.setTitle(nom_sesion+" - "+user_monitor+" - " + nom_ejercicio);
142         alert.setMessage("Finalidad: " + finalidad + "\nDuración: " + duracion +
143                 "\nDescripción: "+ descripcion);
144         alert.show();
145     }
146
147
148     /*
149      *  Método que crea un AlertDialog y lo muestra por pantalla. Usado tanto para mostrar errores
150      *  como para mostrar los mensajes enviados por el monitor desde su panel de chat.
151      */
152     private void alert(final String msg) {
153         final String error = (msg == null) ? "Error desconocido: " : msg;
154         if(!isFinishing()) {
155             AlertDialog.Builder builder = new AlertDialog.Builder(Ejercicio.this);
156             builder.setMessage(error).setPositiveButton("Ok",
157                     new DialogInterface.OnClickListener() {
158                         public void onClick(DialogInterface dialog, int id) {
159                         }
160                     }
161             );
162             AlertDialog dialog = builder.create();
163             dialog.show();
164         }
165     }
166
167
168     /*                CLASE ASYNCTASK
169      *
170      *  Clase asíncrona que se encarga de realizar una consulta al servidor del monitor
171      *  para obtener información acerca del ejercicio (nombre, finalidad, duracion y descripcion).
172      *  Esta clase debe ser asíncrona para no provocar problemas de inconsistencia en la
173      *  app debido a los tiempos de respuesta del servidor.
174      */
175     class asyncinfo extends AsyncTask<String, String, String> {
176
177         protected void onPreExecute() {
178         }
179
180         protected String doInBackground(String... params) {
181
182             /*
183              *  Creamos un ArrayList del tipo nombre-valor para agregar el id del ejercicio y
184              *  enviarlo mediante POST al servidor del monitor para que le devuelva la información
185              *  acerca del ejercicio.
186              */
187             ArrayList<NameValuePair> postparameters2send= new ArrayList<NameValuePair>();
188             postparameters2send.add(new BasicNameValuePair("id_ejercicio",id_ejercicio));
189
190             // Realizamos la peticion HTTP y obtenemos un JSONArray como respuesta
191             Httppostaux post =new Httppostaux();
192             jdatainfo=post.getserverdata(postparameters2send,
193                     "http://"+URL_connect+"/gymserver/getEjercicio.php");
194
195
196             // Si lo que recibimos no es null y no está vacío
197             if (jdatainfo!=null && jdatainfo.length() > 0){
198                 try{
199                     // Se crea un objeto JSONObject y se rellena con
200                     // el unico elemento del JSONArray recibido
201                     JSONObject json_data = jdatainfo.getJSONObject(0);
202                     // Se almacena la información recibida
203                     nom_ejercicio = json_data.getString("nombre_Ejercicio");
204                     finalidad = json_data.getString("Finalidad");
205                     duracion = json_data.getString("Duracion");
206                     descripcion = json_data.getString("Descripcion");
207                 } catch (JSONException e){
208                     e.printStackTrace();
209                 }
210             } else{
211                 return "err";
212             }
213             return "ok";
214         }
215
216         protected void onPostExecute(String result) {
217         }
218     }
219
220
```

```
221     /*                CLASE ASYNCTASK
222      *
223      *  Clase asíncrona que se encarga de realizar una consulta al servidor del monitor
224      *  para obtener la URL del vídeo del ejercicio.
225      *  Esta clase debe ser asíncrona para no provocar problemas de inconsistencia en la
226      *  app debido a los tiempos de respuesta del servidor.
227      */
228     class asyncvideo extends AsyncTask<String, String, String>{
229         protected void onPreExecute() {
230         }
231 
232         protected String doInBackground(String... params) {
233 
234             /*
235              *  Creamos un ArrayList del tipo nombre-valor para agregar el id del ejercicio y
236              *  enviarlo mediante POST al servidor del monitor para que le devuelva la información
237              *  acerca del ejercicio.
238              */
239             ArrayList<NameValuePair> postparameters2send= new ArrayList<NameValuePair>();
240             postparameters2send.add(new BasicNameValuePair("id_ejercicio",id_ejercicio));
241 
242             // Realizamos la peticion HTTP y obtenemos un JSONArray como respuesta
243             Httppostaux post =new Httppostaux();
244             jdatavideo=post.getserverdata(postparameters2send,
245                     "http://"+URL_connect+"/gymserver/getVideoEjercicio.php");
246 
247             // Si lo que recibimos no es null y no está vacío
248             if (jdatavideo!=null && jdatavideo.length() > 0){
249                 try{
250                     // Se crea un objeto JSONObject y se rellena con
251                     // el unico elemento del JSONArray recibido
252                     JSONObject json_data = jdatavideo.getJSONObject(0);
253                     // Se almacena la información recibida
254                     urlVideo = compruebaUrlVideo(json_data.getString("URL_video"));
255                 } catch (JSONException e){
256                     e.printStackTrace();
257                 }
258             } else{
259                 return "err";
260             }
261             return "ok";
262         }
263 
264         protected void onPostExecute(String result) {
265         }
266 
267         private String compruebaUrlVideo(String url){
268             String resulUrl;
269             SharedPreferences settings = getSharedPreferences("ProjectGym", Context.MODE_PRIVATE);
270 
271             if(url.toLowerCase().contains("http://")){
272                 resulUrl = url;
273             } else{
274                 int index = settings.getString("IP", "").indexOf(":");
275                 String ip = settings.getString("IP", "").substring(0,index);
276                 resulUrl = "http://"+ip+url;
277             }
278 
279             return resulUrl;
280         }
281     }
282 
283 
284     /*
285      *  Función que se ejecuta cuando se recibe una trama por el socket abierto para la recepción
286      *  de mensajes del monitor. En la función se llama a la función alert() vista anteriormente
287      *  pasandole como argumento el mensaje recibido en el socket.
288      */
289     private Runnable updateTextMessage = new Runnable() {
290         public void run() {
291             if (myDatagramReceiver == null) return;
292             alert(myDatagramReceiver.getLastMessage());
293         }
294     };
295 
296 
297     /*
298      *  Clase que hereda de la clase Thread. Nos permite crear otro hilo en nuestra actividad,
299      *  este hilo será el encargado de quedar a la escucha en el puerto 12345 de mensajes
300      *  que envia el monitor desde su panel de chat. Además, ejecutará la función updateTextMessage
301      *  para mostrar en pantalla el mensaje recibido mediante una alerta.
302      */
303     public class HiloMensajes extends Thread{
304 
305         private boolean bKeepRunning = true;
```

```
306         private String lastMessage = "";
307         private static final int PORT = 12345;
308         private ServerSocket server;
309         private Socket socket;
310         private ObjectInputStream input;
311
312         public void run(){
313             try{
314
315                 server = new ServerSocket(PORT);
316                    while(bKeepRunning){
317                         socket = server.accept();
318                         input = new ObjectInputStream(socket.getInputStream());
319                         lastMessage = input.readObject().toString();
320                         runOnUiThread(updateTextMessage);
321                    }
322             } catch (Exception e){
323                 e.printStackTrace();
324             }
325         }
326         // Elimina el hilo que está en continua ejecución, termina el bucle
327         public void kill() {
328             bKeepRunning = false;
329         }
330         // Método getter para acceder a la variable lastMessage desde otra clase
331         public String getLastMessage() {
332             return lastMessage;
333         }
334     }
335 }
```

activity_ejercicio.xml

"activity_ejercicio.xml"

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <FrameLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent"
6      android:onClick="mostrar_Descripcion"
7      android:clickable="true"
8      android:layout_weight="1"
9      android:background="#FFA44D">
10
11     <fragment
12         android:name="com.proyecto.antonio.projectdesign.peepers.StreamCameraFragment"
13         android:id="@+id/camera_fragment"
14         android:layout_width="200dp"
15         android:layout_height="180dp"
16         android:layout_gravity="right|bottom"/>
17
18     <VideoView
19         android:id="@+id/vistaVideo"
20         android:layout_width="match_parent"
21         android:layout_height="match_parent" />
22
23 </FrameLayout>
```

SocketAsync.java

"SocketAsync.java"

```
1  package com.proyecto.antonio.projectdesign;
2
3  import android.os.AsyncTask;
4  import android.util.Log;
5
6  import java.net.Inet4Address;
7  import java.net.InetAddress;
8  import java.net.InetSocketAddress;
9  import java.net.Socket;
10
11 import java.io.ObjectOutputStream;
12 import         java.net.NetworkInterface;
13 import java.util.Enumeration;
14 import java.net.SocketAddress;
15
16 /**
17  * Created by usuario on 29/04/2015.
18  */
19 public class SocketAsync extends AsyncTask<String, String, String>{
20
21     private static final int PORT_USUARIO = 12345;
22     private String SERVER_IP, mUsuario;
```

```
23      private int SERVER_PORT;
24      private boolean connection = false;
25
26      public SocketAsync(String ip, String usuario){
27          mUsuario = usuario;
28          // Para separar la IP del puerto
29          int index = ip.indexOf(":");
30          SERVER_IP = ip.substring(0,index);
31          SERVER_PORT = Integer.parseInt(ip.substring(index+1,ip.length()));
32      }
33
34      protected String doInBackground(String... params){
35          try{
36              Socket socket = new Socket();
37              // Permitimos que otros sockets puedan utilizar la misma ip y puerto
38              socket.setReuseAddress(true);
39              // Fijamos el puerto del socket al 8080
40              socket.bind(new InetSocketAddress(getIP(),8080));
41              while(!connection){
42                  socket.connect(new InetSocketAddress(SERVER_IP, SERVER_PORT), 1000);
43                  if(socket.isConnected()){
44                      ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());
45                      // Enviamos en la petición el nombre del usuario...
46                      output.writeObject(mUsuario);
47                      connection = true;
48                  }
49              }
50          } catch(Exception e){
51              e.printStackTrace();
52          }
53          return "";
54      }
55
56
57      private String getIP(){
58          String res = null;
59          try {
60              String localhost = InetAddress.getLocalHost().getHostAddress();
61              Enumeration<NetworkInterface> e = NetworkInterface.getNetworkInterfaces();
62              while (e.hasMoreElements()) {
63                  NetworkInterface ni = (NetworkInterface) e.nextElement();
64                  if(ni.isLoopback())
65                      continue;
66                  if(ni.isPointToPoint())
67                      continue;
68                  Enumeration<InetAddress> addresses = ni.getInetAddresses();
69                  while(addresses.hasMoreElements()) {
70                      InetAddress address = (InetAddress) addresses.nextElement();
71                      if(address instanceof Inet4Address) {
72                          String ip = address.getHostAddress();
73                          if(!ip.equals(localhost))
74                              System.out.println((res = ip));
75                      }
76                  }
77              }
78          } catch (Exception e) {
79              e.printStackTrace();
80          }
81          return res;
82      }
83
84      public boolean getConnection(){
85          return connection;
86      }
87  }
```

StreamCameraFragment.java

"StreamCameraFragment.java"

```
1   package com.proyecto.antonio.projectdesign.peepers;
2
3   import android.content.Context;
4   import android.os.Bundle;
5   import android.app.Fragment;
6   import android.view.LayoutInflater;
7   import android.view.View;
8   import android.view.ViewGroup;
9
10  import android.content.SharedPreferences;
11  import android.content.pm.PackageManager;
12  import android.os.AsyncTask;
13  import android.os.PowerManager;
14  import android.preference.PreferenceManager;
15  import android.os.Bundle;
```

```
16 import android.view.Menu;
17 import android.view.MenuItem;
18 import android.app.Activity;
19
20 import android.view.SurfaceView;
21 import android.widget.TextView;
22 import android.view.SurfaceHolder;
23 import android.os.PowerManager.WakeLock;
24
25 import org.apache.http.conn.util.InetAddressUtils;
26
27 import java.net.InetAddress;
28 import java.net.NetworkInterface;
29 import java.util.Enumeration;
30
31 import com.proyecto.antonio.projectdesign.R;
32
33
34 public class StreamCameraFragment extends Fragment implements SurfaceHolder.Callback{
35
36     private static final String PREF_CAMERA = "camera";
37     private static final int PREF_CAMERA_INDEX_DEF = 0;
38     private static final String PREF_FLASH_LIGHT = "flash_light";
39     private static final boolean PREF_FLASH_LIGHT_DEF = false;
40     private static final String PREF_PORT = "port";
41     private static final int PREF_PORT_DEF = 8080;
42     private static final String PREF_JPEG_SIZE = "size";
43     private static final String PREF_JPEG_QUALITY = "jpeg_quality";
44     private static final int PREF_JPEG_QUALITY_DEF = 40;
45     // preview sizes will always have at least one element, so this is safe
46     private static final int PREF_PREVIEW_SIZE_INDEX_DEF = 0;
47     private static final String WAKE_LOCK_TAG = "peepers";
48
49     private boolean mRunning = false;
50     private SurfaceHolder mPreviewDisplay = null;
51     private boolean mPreviewDisplayCreated = false;
52     private CameraStreamer mCameraStreamer = null;
53
54     private SharedPreferences mPrefs = null;
55     private int mCameraIndex = PREF_CAMERA_INDEX_DEF;
56     private boolean mUseFlashLight = PREF_FLASH_LIGHT_DEF;
57     private int mPort = PREF_PORT_DEF;
58     private int mJpegQuality = PREF_JPEG_QUALITY_DEF;
59     private int mPrevieSizeIndex = PREF_PREVIEW_SIZE_INDEX_DEF;
60     private String mIpAddress = "";
61     private WakeLock mWakeLock = null;
62
63
64     public StreamCameraFragment(){}
65
66
67     @Override
68     public View onCreateView(LayoutInflater inflater, ViewGroup container,
69                              Bundle savedInstanceState){
70
71         View fragmentView = inflater.inflate(R.layout.fragment_stream_camera, container, false);
72
73         new LoadPreferencesTask().execute();
74
75         mPreviewDisplay = ((SurfaceView) fragmentView.findViewById(R.id.camera)).getHolder();
76         mPreviewDisplay.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
77         mPreviewDisplay.addCallback(this);
78
79         mIpAddress = tryGetIpV4Address();
80         updatePrefCacheAndUi();
81
82         final PowerManager powerManager = (PowerManager) getActivity().getSystemService(Context.POWER_SERVICE);
83         mWakeLock = powerManager.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, WAKE_LOCK_TAG);
84
85         return fragmentView;
86     }
87
88
89     @Override
90     public void onResume(){
91         super.onResume();
92         mRunning = true;
93         if(mPrefs != null){
94             mPrefs.registerOnSharedPreferenceChangeListener(mSharedPreferenceListener);
95         }
96         updatePrefCacheAndUi();
97         tryStartCameraStreamer();
98         mWakeLock.acquire();
99     }
100
```

```
101
102     @Override
103     public void onPause(){
104         super.onPause();
105         mWakeLock.release();
106         mRunning = false;
107         if(mPrefs != null){
108             mPrefs.unregisterOnSharedPreferenceChangeListener(mSharedPreferenceListener);
109         }
110         ensureCameraStreamerStopped();
111     }
112
113
114     @Override
115     public void surfaceChanged(final SurfaceHolder holder, final int format, final int width, final int height){
116         //Lo ignoramos
117     }
118
119
120     @Override
121     public void surfaceCreated(final SurfaceHolder holder){
122         mPreviewDisplayCreated = true;
123         tryStartCameraStreamer();
124     }
125
126
127     @Override
128     public void surfaceDestroyed(final SurfaceHolder holder){
129         mPreviewDisplayCreated = false;
130         ensureCameraStreamerStopped();
131     } // surfaceDestroyed(SurfaceHolder)
132
133
134     private void tryStartCameraStreamer(){
135         if(mRunning && mPreviewDisplayCreated && mPrefs != null){
136             mCameraStreamer = new CameraStreamer(mCameraIndex, mUseFlashLight, mPort,
137                     mPrevieSizeIndex, mJpegQuality, mPreviewDisplay);
138             mCameraStreamer.start();
139         }
140     }
141
142
143     private void ensureCameraStreamerStopped(){
144         if(mCameraStreamer != null){
145             mCameraStreamer.stop();
146             mCameraStreamer = null;
147         }
148     }
149
150
151     private final class LoadPreferencesTask extends AsyncTask<Void, Void, SharedPreferences> {
152
153         // Constructor
154         private LoadPreferencesTask(){
155             super();
156         }
157
158
159         @Override
160         protected SharedPreferences doInBackground(final Void... noParams){
161             return PreferenceManager.getDefaultSharedPreferences(getActivity());
162         }
163
164
165         @Override
166         protected void onPostExecute(final SharedPreferences prefs){
167             StreamCameraFragment.this.mPrefs = prefs;
168             prefs.registerOnSharedPreferenceChangeListener(mSharedPreferenceListener);
169             updatePrefCacheAndUi();
170             tryStartCameraStreamer();
171         }
172     }
173
174
175     private final SharedPreferences.OnSharedPreferenceChangeListener mSharedPreferenceListener =
176             new SharedPreferences.OnSharedPreferenceChangeListener() {
177                 @Override
178                 public void onSharedPreferenceChanged(final SharedPreferences prefs,
179                                                       final String key) {
180                     updatePrefCacheAndUi();
181                 }
182             };
183
184
185     private final int getPrefInt(final String key, final int defValue){
```

```
186
187         try{
188             return Integer.parseInt(mPrefs.getString(key, null));
189         } catch(final NullPointerException e){
190             return defValue;
191         } catch(final NumberFormatException e){
192             return defValue;
193         }
194     }
195
196
197     private final void updatePrefCacheAndUi(){
198         mCameraIndex = getPrefInt(PREF_CAMERA, PREF_CAMERA_INDEX_DEF);
199         if(hasFlashLight()){
200             if(mPrefs != null){
201                 mUseFlashLight = mPrefs.getBoolean(PREF_FLASH_LIGHT, PREF_FLASH_LIGHT_DEF);
202             } else{
203                 mUseFlashLight = PREF_FLASH_LIGHT_DEF;
204             }
205         } else{
206             mUseFlashLight = false;
207         }
208
209         mPort = getPrefInt(PREF_PORT, PREF_PORT_DEF);
210         // El puerto debe estar en el rango [1024 65535]
211         if(mPort < 1024){
212             mPort = 1024;
213         } else if(mPort > 65535){
214             mPort = 65535;
215         }
216
217         mPrevieSizeIndex = getPrefInt(PREF_JPEG_SIZE, PREF_PREVIEW_SIZE_INDEX_DEF);
218         mJpegQuality = getPrefInt(PREF_JPEG_QUALITY, PREF_JPEG_QUALITY_DEF);
219         if(mJpegQuality < 0){
220             mJpegQuality = 0;
221         } else if(mJpegQuality > 100){
222             mJpegQuality = 100;
223         }
224     }
225
226
227     private boolean hasFlashLight(){
228         return getActivity().getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA_FLASH);
229     }
230
231
232     private static String tryGetIpV4Address(){
233         try{
234             final Enumeration<NetworkInterface> en = NetworkInterface.getNetworkInterfaces();
235             while(en.hasMoreElements()){
236                 final NetworkInterface intf = en.nextElement();
237                 final Enumeration<InetAddress> enumIpAddr = intf.getInetAddresses();
238                 while(enumIpAddr.hasMoreElements()){
239                     final InetAddress inetAddress = enumIpAddr.nextElement();
240                     if(!inetAddress.isLoopbackAddress()){
241                         final String addr = inetAddress.getHostAddress().toUpperCase();
242                         if(InetAddressUtils.isIPv4Address(addr)){
243                             return addr;
244                         }
245                     }
246                 }
247             }
248         } catch(final Exception e){
249             //Lo ignoramos
250         }
251         return null;
252     }
253 }
```

fragment_stream_camera.xml

"fragment_stream_camera.xml"

```
1  <FrameLayout
2      xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      tools:context="com.proyecto.antonio.projectdesign.peepers.StreamCameraFragment">
7
8      <SurfaceView
9          android:id="@+id/camera"
10         android:layout_width="fill_parent"
11         android:layout_height="fill_parent" />
12
13 </FrameLayout>
```

CameraStreamer.java

"CameraStreamer.java"

```
1  package com.proyecto.antonio.projectdesign.peepers;
2
3
4  import java.io.IOException;
5  import java.util.List;
6
7  import android.graphics.ImageFormat;
8  import android.graphics.Rect;
9  import android.graphics.YuvImage;
10 import android.hardware.Camera;
11 import android.os.HandlerThread;
12 import android.os.Looper;
13 import android.os.Handler;
14 import android.os.Message;
15 import android.os.Process;
16 import android.os.SystemClock;
17 import android.util.Log;
18 import android.view.SurfaceHolder;
19
20 /**
21  * Created by usuario on 15/04/2015.
22  */
23 public class CameraStreamer extends Object {
24
25     private static final String TAG = CameraStreamer.class.getSimpleName();
26     private static final int MESSAGE_TRY_START_STREAMING = 0;
27     private static final int MESSAGE_SEND_PREVIEW_FRAME = 1;
28     private static final long OPEN_CAMERA_POLL_INTERVAL_MS = 1000L;
29
30     private final Object mLock = new Object();
31     private final MovingAverage mAverageSpf = new MovingAverage(50 /* numValues */);
32
33     private final int mCameraIndex;
34     private final boolean mUseFlashLight;
35     private final int mPort;
36     private final int mPreviewSizeIndex;
37     private final int mJpegQuality;
38     private final SurfaceHolder mPreviewDisplay;
39
40     private boolean mRunning = false;
41     private Looper mLooper = null;
42     private Handler mWorkHandler = null;
43     private MJpegHttpStreamer mMJpegHttpStreamer = null;
44     private Camera mCamera = null;
45     private int mPreviewFormat = Integer.MIN_VALUE;
46     private int mPreviewWidth = Integer.MIN_VALUE;
47     private int mPreviewHeight = Integer.MIN_VALUE;
48     private int mPreviewBufferSize = Integer.MIN_VALUE;
49     private Rect mPreviewRect = null;
50     private MemoryOutputStream mJpegOutputStream = null;
51
52     private long mNumFrames = 0L;
53     private long mLastTimestamp = Long.MIN_VALUE;
54
55
56     CameraStreamer(final int cameraIndex, final boolean useFlashLight, final int port,
57                    final int previewSizeIndex, final int jpegQuality, final SurfaceHolder previewDisplay){
58         super();
59
60         if(previewDisplay == null){
61             throw new IllegalArgumentException("previewDisplay no puede ser null");
62         }
63
64         mCameraIndex = cameraIndex;
65         mUseFlashLight = useFlashLight;
66         mPort = port;
67         mPreviewSizeIndex = previewSizeIndex;
68         mJpegQuality = jpegQuality;
69         mPreviewDisplay = previewDisplay;
70     }
71
72
73     private final class WorkHandler extends Handler{
74
75         private WorkHandler(final Looper looper){
76             super(looper);
77         }
78
79         @Override
80         public void handleMessage(final Message message){
```

```
            switch(message.what){
                case MESSAGE_TRY_START_STREAMING:
                    tryStartStreaming();
                    break;
                case MESSAGE_SEND_PREVIEW_FRAME:
                    final Object[] args = (Object[]) message.obj;
                    sendPreviewFrame((byte[]) args[0], (Camera) args[1], (Long) args[2]);
                    break;
                default:
                    throw new IllegalArgumentException("cannot handle message");
            }
        }
    }


    void start(){
        synchronized (mLock){
            if(mRunning){
                throw new IllegalStateException("CameraStreamer is already running");
            }
            mRunning = true;
        }

        final HandlerThread worker = new HandlerThread(TAG, Process.THREAD_PRIORITY_MORE_FAVORABLE);
        worker.setDaemon(true);
        worker.start();
        mLooper = worker.getLooper();
        mWorkHandler = new WorkHandler(mLooper);
        mWorkHandler.obtainMessage(MESSAGE_TRY_START_STREAMING).sendToTarget();
    }


    void stop(){
        synchronized (mLock){
            if(!mRunning){
                throw new IllegalStateException("CameraStreamer is already stopped");
            }

            mRunning = false;
            if(mMJpegHttpStreamer != null){
                mMJpegHttpStreamer.stop();
            }
            if(mCamera != null){
                mCamera.release();
                mCamera = null;
            }
        }
        mLooper.quit();
    }


    private void tryStartStreaming(){
        try{
            while (true){
                try{
                    startStreamingIfRunning();
                } catch(final RuntimeException openCameraFailed){
                    Log.d(TAG, "Open camera failed, retying in " + OPEN_CAMERA_POLL_INTERVAL_MS
                            + "ms", openCameraFailed);
                    Thread.sleep(OPEN_CAMERA_POLL_INTERVAL_MS);
                    continue;
                }
                break;
            }
        } catch(final Exception startPreviewFailed){
            Log.w(TAG, "Failed to start camera preview", startPreviewFailed);
        }
    }


    private void startStreamingIfRunning() throws IOException{
        // Throws RuntimeException if the camera is currently opened
        // by another application.
        final Camera camera = Camera.open(mCameraIndex);
        final Camera.Parameters params = camera.getParameters();

        final List<Camera.Size> supportedPreviewSizes = params.getSupportedPreviewSizes();
        final Camera.Size selectedPreviewSize = supportedPreviewSizes.get(mPreviewSizeIndex);
        params.setPreviewSize(selectedPreviewSize.width, selectedPreviewSize.height);

        if(mUseFlashLight){
            params.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
        }

        // Set Preview FPS range. The range with the greatest maximum
```

```
166         // is returned first.
167         final List<int[]> supportedPreviewFpsRanges = params.getSupportedPreviewFpsRange();
168         // XXX: However sometimes it returns null. This is a known bug
169         // https://code.google.com/p/android/issues/detail?id=6271
170         // In which case, we just don't set it.
171         if (supportedPreviewFpsRanges != null){
172             final int[] range = supportedPreviewFpsRanges.get(0);
173             params.setPreviewFpsRange(range[Camera.Parameters.PREVIEW_FPS_MIN_INDEX],
174                     range[Camera.Parameters.PREVIEW_FPS_MAX_INDEX]);
175             camera.setParameters(params);
176         }
177 
178         // Set up preview callback
179         mPreviewFormat = params.getPreviewFormat();
180         final Camera.Size previewSize = params.getPreviewSize();
181         mPreviewWidth = previewSize.width;
182         mPreviewHeight = previewSize.height;
183         final int BITS_PER_BYTE = 8;
184         final int bytesPerPixel = ImageFormat.getBitsPerPixel(mPreviewFormat) / BITS_PER_BYTE;
185         // XXX: According to the documentation the buffer size can be
186         // calculated by width * height * bytesPerPixel. However, this
187         // returned an error saying it was too small. It always needed
188         // to be exactly 1.5 times larger.
189         mPreviewBufferSize = mPreviewWidth * mPreviewHeight * bytesPerPixel * 3 / 2 + 1;
190         camera.addCallbackBuffer(new byte[mPreviewBufferSize]);
191         mPreviewRect = new Rect(0, 0, mPreviewWidth, mPreviewHeight);
192         camera.setPreviewCallbackWithBuffer(mPreviewCallback);
193 
194         // We assumed that the compressed image will be no bigger than
195         // the uncompressed image.
196         mJpegOutputStream = new MemoryOutputStream(mPreviewBufferSize);
197 
198         final MJpegHttpStreamer streamer = new MJpegHttpStreamer(mPort, mPreviewBufferSize);
199         streamer.start();
200 
201         synchronized (mLock){
202             if(!mRunning){
203                 streamer.stop();
204                 camera.release();
205                 return;
206             }
207 
208             try{
209                 camera.setPreviewDisplay(mPreviewDisplay);
210             } catch(final IOException e){
211                 streamer.stop();
212                 camera.release();
213                 throw e;
214             }
215 
216             mMJpegHttpStreamer = streamer;
217             camera.startPreview();
218             mCamera = camera;
219             // Para corregir el giro de 90 grados de la imagen mostrada
220             // por pantalla
221             mCamera.setDisplayOrientation(90);
222         }
223     }
224 
225 
226     private final Camera.PreviewCallback mPreviewCallback = new Camera.PreviewCallback(){
227         @Override
228         public void onPreviewFrame(final byte[] data, final Camera camera){
229             final Long timestamp = SystemClock.elapsedRealtime();
230             final Message message = mWorkHandler.obtainMessage();
231             message.what = MESSAGE_SEND_PREVIEW_FRAME;
232             message.obj = new Object[]{ data, camera, timestamp };
233             message.sendToTarget();
234         }
235     };
236 
237 
238     private void sendPreviewFrame(final byte[] data, final Camera camera, final long timestamp){
239         // Calcalute the timestamp
240         final long MILLI_PER_SECOND = 1000L;
241         final long timestampSeconds = timestamp / MILLI_PER_SECOND;
242 
243         // Update and log the frame rate
244         final long LOGS_PER_FRAME = 10L;
245         mNumFrames++;
246         if(mLastTimestamp != Long.MIN_VALUE){
247             mAverageSpf.update(timestampSeconds - mLastTimestamp);
248             if(mNumFrames % LOGS_PER_FRAME == LOGS_PER_FRAME-1){
249                 Log.d(TAG, "FPS: " + 1.0 / mAverageSpf.getAverage());
250             }
```

```
251         }
252
253         mLastTimestamp = timestampSeconds;
254
255         //Create JPEG
256         final YuvImage image = new YuvImage(data, mPreviewFormat, mPreviewWidth, mPreviewHeight,
257                 null /* strides */);
258         image.compressToJpeg(mPreviewRect, mJpegQuality, mJpegOutputStream);
259
260         mMJpegHttpStreamer.streamJpeg(mJpegOutputStream.getBuffer(), mJpegOutputStream.getLength(),
261                 timestamp);
262
263         // Clean up
264         mJpegOutputStream.seek(0);
265         // XXX: I believe that this is thread-safe because we're not
266         // calling methods in other threads. I might be wrong, the
267         // documentation is not clear.
268         camera.addCallbackBuffer(data);
269     }
270 }
```

1.3 Servidor PHP

config.php

"config.php"

```
1 <?php
2
3 define("DB_HOST", "localhost");//Nombre del host
4 define("DB_USER", "root");//Nombre usuario de la base de datos
5 define("DB_PASSWORD", "");//Contraseña de la base de datos
6 define("DB_DATABASE", "datosgym");//Nombre de la base de datos.
7 ?>
```

gestor_bd.php

"gestor_bd.php"

```
1  <?php
2  class gestor_BD {
3          private $db;
4
5          // Constructor
6          function __construct() {
7                  require_once 'connectbd.php';
8                  //Establece conexion a la base de datos
9                  $this->db = new DB_Connect();
10                 $this->db->connect();
11         }
12
13         // Destructor
14         function __destruct() {
15         }
16
17
18         /*
19          * Metodo privado que devuelve el resultado de la consulta sql que recibe como parametro
20          */
21         private function consultaSql($sql){
22                 $consulta = mysql_query($sql);
23                 $resul = array();
24                 if(mysql_num_rows($consulta)){
25                         for ($i=0; $row=mysql_fetch_assoc($consulta); $i++){
26                                 $resul[$i]=$row;
27                         }
28                 }
29                 return $resul;
30         }
31
32         /*
33          * Metodo para el login del usuario
34          */
35         public function login($user,$passw){
36                 $sql = "SELECT COUNT(*) FROM usuarios WHERE username='$user' AND passw='$passw' ";
37                 $consulta = mysql_query($sql);
38                 $count = mysql_fetch_row($consulta);
39                 //usuario unico
40                 if ($count[0]==0){
41                         $resul = "0";
42                 }else{
43                         $resul = "1";
44                 }
45                 return $resul;
46         }
```

```
47
48
49          /*
50           * Metodo para comprobar que el usuario existe
51           */
52          public function existeUsuario($user){
53                  $result = mysql_query("SELECT COUNT(*) FROM usuarios WHERE username='$user' ");
54                  $count = mysql_fetch_row($result);
55                  //usuario unico
56                  if ($count[0]==0){
57                          $resul = "0"; // Esto estaba inicialmente al reves
58                  }else{
59                          $resul = "1";
60                  }
61                  return $resul;
62          }
63
64
65          /*
66           * Para modificar los datos de un usuario en la base de datos
67           */
68          public function modifica($Nomb, $Apell, $DNI, $Direc, $Movil, $usuario, $passw) {
69                  $sql = "UPDATE 'usuarios' SET 'nombre'='$Nomb','apellidos'='$Apell',
70                                  'dni'='$DNI','direccion'='$Direc','movil'='$Movil',
71                                  'passw'='$passw' WHERE username='$usuario'";
72                  $consulta = mysql_query($sql);
73                  //Si se modifica el usuario de manera correcta
74                  if ($consulta) {
75                          //Devuelve true
76                          $resul = "1";
77                          //si no fue de manera correcta
78                  } else {
79                          //Devuelve false
80                          $resul = "0";
81                  }
82                  return $resul;
83          }
84
85
86          /*
87           * Metodo para modificar la IP del monitor de una sesion
88           */
89          public function setIpMonitorSesion($Id_Sesion, $ip_Monitor){
90                  $sql = "UPDATE sesiones SET ip_Monitor='$ip_Monitor'
91                                  WHERE Id_Sesion='$Id_Sesion'";
92                  $consulta = mysql_query($sql);
93                  if($consulta){
94                          $resul = "1";
95                  } else{
96                          $resul = "0";
97                  }
98                  return $resul;
99          }
100
101         /*
102          * Metodo para que una sesion pase a ser no activa
103          */
104         public function setSesionNoActiva($Id_Sesion){
105                 $sql = "UPDATE sesiones SET activa=0 WHERE Id_Sesion='$Id_Sesion'";
106                 $consulta = mysql_query($sql);
107                 if($consulta){
108                         $resul = "1";
109                 } else{
110                         $resul = "0";
111                 }
112                 return $resul;
113         }
114
115
116         /*
117          * Para agregar a un nuevo nuevo usuario a la base de datos
118          */
119         public function addUsuario($Nomb, $Apell, $DNI, $Direc, $Movil, $usuario, $passw) {
120                 $sql = "INSERT INTO usuarios (nombre, apellidos, dni, direccion,
121                                 movil, username, passw) VALUES('$Nomb', '$Apell', '$DNI',
122                                 '$Direc', '$Movil', '$usuario', '$passw')";
123                 $consulta = mysql_query($sql);
124                 //Si se guardo el usuario de manera correcta
125                 if ($consulta) {
126                         //Devuelve true
127                         $resul = "1";
128                 //si no fue de manera correcta
129                 } else {
130                         //Devuelve false
131                         $resul = "0";
```

```
132                 }
133                 return $resul;
134         }
135
136
137         /*
138          * Metodo para agregar una nueva sesion a la base de datos
139          */
140         public function nuevaSesion($nomb, $monitor, $fecha, $numUsuarios, $ip, $activa){
141                 $sql = "INSERT INTO sesiones (nombre_Sesion, user_Monitor,
142                                 fecha, num_Usuarios, ip_Monitor, activa) VALUES('$nomb',
143                                 '$monitor', '$fecha', '$numUsuarios', '$ip', '$activa')";
144                 $consulta = mysql_query($sql);
145                 if($consulta) {
146                         $resul = "1";
147                 } else {
148                         $resul = "0";
149                 }
150                 return $resul;
151         }
152
153
154         /*
155          * Metodo para eliminar una sesion
156          */
157         public function deleteSesion($Id_Sesion){
158                 $sql = "DELETE FROM sesiones WHERE Id_Sesion='$Id_Sesion'";
159                 $consulta = mysql_query($sql);
160                 if($consulta) {
161                         $resul = "1";
162                 } else {
163                         $resul = "0";
164                 }
165                 return $resul;
166         }
167
168
169         /*
170          * Metodo para introducir un ejercicio en una sesion
171          */
172         public function addEjercicioSesion($Id_Sesion, $Id_Ejercicio){
173                 $sql = "INSERT INTO ejerciciosdesesion (Id_Ejercicio, Id_Sesion)
174                                 VALUES('$Id_Ejercicio','$Id_Sesion')";
175                 $consulta = mysql_query($sql);
176                 if($consulta){
177                         $resul = "1";
178                 } else{
179                         $resul = "0";
180                 }
181                 return $resul;
182         }
183
184
185         /*
186          * Metodo para eliminar un ejercicio de una sesion
187          */
188         public function deleteEjercicioSesion($Id_Sesion, $Id_Ejercicio){
189                 $sql = "DELETE FROM ejerciciosdesesion WHERE Id_Sesion='$Id_Sesion'
190                                 AND Id_Ejercicio='$Id_Ejercicio'";
191                 $consulta = mysql_query($sql);
192                 if($consulta){
193                         $resul = "1";
194                 } else{
195                         $resul = "0";
196                 }
197                 return $resul;
198         }
199
200
201         /*
202          * Metodo para la subscripcion de un usuario a una sesion
203          */
204         public function addUsuarioSesion($Id_Sesion, $Id_Usuario){
205                 //Comprobamos que el usuario no esta subscrito a la sesion
206                 $sql1 = "SELECT * FROM sesionesdeusuario WHERE Id_Sesion='$Id_Sesion'
207                                 AND Id_Usuario='$Id_Usuario'";
208                 $consulta1 = mysql_query($sql1);
209                 if(mysql_num_rows($consulta1)){
210                         $resul = "1";
211                 } else{
212                         $sql2 = "INSERT INTO sesionesdeusuario (Id_Sesion, Id_Usuario)
213                                         VALUES('$Id_Sesion', '$Id_Usuario')";
214                         $consulta2 = mysql_query($sql2);
215                         if($consulta2){
216                                 $resul = "1";
```

```
217                     } else{
218                             $resul = "0";
219                     }
220             }
221             return $resul;
222     }
223
224
225     /*
226      * Metodo para obtener la direccion IP del monitor de una sesion
227      */
228     public function getIpMonitorSesion($Id_Sesion){
229             $sql = "SELECT ip_Monitor FROM sesiones WHERE Id_Sesion='$Id_Sesion'";
230             return $this->consultaSql($sql);
231     }
232
233
234     /*
235      * Metodo para enviar un email al usuario
236      */
237     public function enviomail($user, $direccion){
238             $squl = "SELECT 'passw' FROM 'usuarios' WHERE username = '$user'
239                             and direccion = '$direccion'";
240             $consulta = mysql_query($sql);
241             $count = mysql_fetch_row($consulta);
242             //Comprueba que la dupla usuario-mail es correcta
243             if ($count[0]==0){
244                     //Si no existe dicha dupla, devuelve error
245                     $resul = 0;
246                     //Si existe, se envia mail.
247             }else{
248
249                     while ($fila = mysql_fetch_array($result, MYSQL_NUM)) {
250
251                             $myvar = implode($fila);
252                             //$myvar = print_r($fila[0], true);
253                             //$myvar = printf(": %d", $fila[0]);
254                     }
255
256                     //envio de mail
257                     $header = 'From: ' . $direccion . " \r\n";
258                     $header .= "X-Mailer: PHP/" . phpversion() . " \r\n";
259                     $header .= "Mime-Version: 1.0 \r\n";
260                     $header .= "Content-Type: text/plain";
261
262                     $mensaje = "Este mensaje es enviado para el usuario: " . $user . " \r\n";
263                     $mensaje .= "Con el siguiente e-mail: " . $direccion . " \r\n";
264                     $mensaje .= "La contraseña del usuario es: " . $myvar . " \r\n";
265                     $mensaje .= "Enviado el " . date('d/m/Y', time());
266
267                     $para = $direccion;
268                     $asunto = 'Recordatorio password';
269                     //funcion que envia mail
270                     mail($para, $asunto, utf8_decode($mensaje), $header);
271                     $resul = "1";
272             }
273             return $resul;
274     }
275
276
277     /*
278      * Metodo para obtener todas las sesiones (activas y no activas) asociadas a un monitor
279      */
280     public function obtenerSesionesMonitor($monitor){
281             $sql = "SELECT * FROM sesiones WHERE user_Monitor='$monitor'";
282             return $this->consultaSql($sql);
283     }
284
285
286     /*
287      * Metodo para obtener todas las sesiones activas en la base de datos
288      */
289     public function obtenerSesionesActivas(){
290             $sql = "SELECT * FROM sesiones WHERE activa=1";
291             return $this->consultaSql($sql);
292     }
293
294
295     /*
296      * Metodo que devuelve la sesion cuyo id coincide con el pasado como parametro
297      */
298     public function devuelveSesion($Id_Sesion){
299             $sql = "SELECT * FROM sesiones WHERE Id_Sesion='$Id_Sesion'";
300             return $this->consultaSql($sql);
301     }
```

```
302
303
304         /*
305          * Metodo para obtener los datos del usuario
306          */
307         public function datosUsuario($usuario){
308                 $sql = "SELECT * FROM usuarios WHERE username = '$usuario'";
309                 return $this->consultaSql($sql);
310         }
311
312
313         /*
314          * Metodo para obtener todos los ejercicios en la base de datos
315          */
316         public function obtenerEjercicios(){
317                 $sql = "SELECT * FROM ejercicios";
318                 return $this->consultaSql($sql);
319         }
320
321
322         /*
323          * Metodo para obtener los ejercicios asignados a una rutina
324          */
325         public function ejerciciosSesion($Id_Sesion){
326                 $sql = "SELECT ejercicios.Id_Ejercicio, ejercicios.nombre_Ejercicio,
327                                 ejercicios.Descripcion, sesiones.user_Monitor,sesiones.nombre_Sesion FROM
328                                 (sesiones INNER JOIN ejerciciosdesesion ON
329                                 sesiones.Id_Sesion=ejerciciosdesesion.Id_Sesion
330                                 INNER JOIN ejercicios ON ejercicios.Id_Ejercicio=ejerciciosdesesion.Id_Ejercicio)
331                                 WHERE ejerciciosdesesion.Id_Sesion='$Id_Sesion'";
332                 return $this->consultaSql($sql);
333         }
334
335
336         /*
337          * Metodo para obtener todos los ejercicios que no estan asignados a una cierta sesion
338          */
339         public function ejerciciosNoSesion($Id_Sesion){
340                 $sql = "SELECT * FROM ejercicios WHERE ejercicios.Id_Ejercicio NOT IN
341                                 (SELECT ejerciciosdesesion.Id_Ejercicio FROM ejerciciosdesesion
342                                 WHERE ejerciciosdesesion.Id_Sesion=13)";
343                 return $this->consultaSql($sql);
344         }
345
346
347         /*
348          * Metodo para obtener la URL del video asignado a un ejercicio
349          */
350         public function getVideoEjercicio($Id_Ejercicio){
351                 $sql = "SELECT URL_video FROM ejercicios WHERE Id_Ejercicio='$Id_Ejercicio'";
352                 return $this->consultaSql($sql);
353         }
354
355         public function videoEjercio($idejercicio){
356                 $sql = "SELECT Recursos.url FROM ( ejercicios
357                                 INNER JOIN recursosdeejercicios ON ejercicios.Id_Ejercicio =
358                                 recursosdeejercicios.Id_Ejercicio INNER JOIN recursos ON
359                                 recursos.Id_recurso = recursosdeejercicios.Id_recurso )
360                                 WHERE recursosdeejercicios.Id_Ejercicio =  '$idejercicio'
361                                 and recursos.tipo_recurso = 'video'";
362                 return $this->consultaSql($sql);
363         }
364
365
366         /*
367          * Metodo para obtener la informacion de un ejercicio
368          */
369         public function infoEjercicio($idejercicio){
370                 $sql = "SELECT * FROM ejercicios WHERE id_ejercicio = '$idejercicio'";
371                 return $this->consultaSql($sql);
372         }
373
374
375         /*
376          * Metodo para obtener la descripcion de un ejercicio
377          */
378         public function descripcionEjercicio($idejercicio){
379                 $sql = "SELECT descripcion FROM 'explicacionesdeejercicios'
380                                 WHERE id_ejercicio = '$idejercicio'";
381                 return $this->consultaSql($sql);
382         }
383
384
385         /*
386          * Metodo para obtener las caracteristicas de un ejercicio
```

```
387          */
388         public function caracteristicasEjercicio($idejercicio){
389                 $sql = "SELECT descripcion FROM 'carasteristicasdeejercicios'
390                         WHERE id_ejercicio = '$idejercicio'";
391                 return $this->consultaSql($sql);
392         }
393
394
395         /*
396          * Metodo para obtener las recomendaciones de un ejercicio
397          */
398         public function recomendacionesEjercicio($idejercicio){
399                 $sql = "SELECT descripcion FROM 'recomendacionesdeejercicios'
400                         WHERE id_ejercicio = '$idejercicio'";
401                 return $this->consultaSql($sql);
402         }
403
404
405         /*
406          * Metodo para obtener la foto de un ejercicio
407          */
408         public function fotoEjercicio($idejercicio){
409                 $sql = "SELECT Recursos.posicion_foto, Recursos.url FROM ( ejercicios
410                         INNER JOIN recursosdeejercicios ON ejercicios.Id_Ejercicio =
411                         recursosdeejercicios.Id_Ejercicio INNER JOIN recursos ON
412                         recursos.Id_recurso = recursosdeejercicios.Id_recurso ) WHERE
413                         recursosdeejercicios.Id_Ejercicio =  '$idejercicio' and
414                         recursos.tipo_recurso = 'foto'";
415                 return $this->consultaSql($sql);
416         }
417
418
419         /*
420          * Metodo para obtener los utiles de un ejercicio
421          */
422         public function utilesEjercicio($idejercicio){
423                 $sql = "SELECT utiles.Id_Util, utiles.descripcion FROM ( ejercicios
424                         INNER JOIN utilesdeejercicios ON ejercicios.Id_Ejercicio =
425                         utilesdeejercicios.Id_Ejercicio INNER JOIN utiles ON
426                         utiles.Id_Util = utilesdeejercicios.Id_Util )
427                         WHERE utilesdeejercicios.Id_Ejercicio =  '$idejercicio'";
428                 return $this->consultaSql($sql);
429         }
430
431
432         /*
433          * Metodo para obtener la foto de un util
434          */
435         public function fotoUtil($idutil){
436                 $sql = "SELECT urlfoto FROM 'utiles' WHERE Id_util = '$idutil'";
437                 return $this->consultaSql($sql);
438         }
439
440
441         /*
442          * Metodo para obtener los musculos de un ejercicio
443          */
444         public function musculosEjercicio($idejercicio){
445                 $sql = "SELECT musculos.Id_musculo, musculos.descripcion FROM ( ejercicios
446                         INNER JOIN musculosdeejercicios ON ejercicios.Id_Ejercicio =
447                         musculosdeejercicios.Id_Ejercicio INNER JOIN musculos ON
448                         musculos.Id_musculo = musculosdeejercicios.Id_musculo )
449                         WHERE musculosdeejercicios.Id_Ejercicio = '$idejercicio'";
450                 return $this->consultaSql($sql);
451         }
452
453
454         /*
455          * Metodo para obtener la foto de un musculo
456          */
457         public function fotoMusculo($idmusculo){
458                 $sql = "SELECT urlfoto FROM 'musculos' WHERE Id_musculo = '$idmusculo'";
459                 return $this->consultaSql($sql);
460         }
461
462
463         /*
464          * Metodo para obtener el numero de usuarios maximos de una sesion
465          */
466         public function numeroUsuariosRutina($monitor){
467                 $sql = "SELECT 'num_usuarios' FROM sesiones WHERE 'user_monitor'='$monitor'";
468                 return $this->consultaSql($sql);
469         }
470 }
471 ?>
```

connectbd.php

"connectbd.php"

```php
<?php
 /**
* Script Php con las funciones de conexion con la base de datos.
* @Autor Antonio Jose Diaz Lora (antdialor84@gmail.com)
* @fecha Mayo 2014
*/
class DB_Connect {
    // constructor
    function __construct() {
    }
    // destructor
    function __destruct() {
    }
    //Conexion a la base de datos
    public function connect() {
                //Archivo con los datos de configuracion
        require_once 'config.php';
        //Conexion
        mysql_connect(DB_HOST, DB_USER, DB_PASSWORD);
        //Seleccion base de datos
        mysql_select_db(DB_DATABASE);
    }
    //Cierre de la conexion
    public function close() {
        mysql_close();
    }
}
?>
```

addEjercicioSesion.php

"addEjercicioSesion.php"

```php
<?php

iconv_set_encoding("output_encoding","UTF-8");
//Toma los valores enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];
$Id_Ejercicio = $_POST['Id_Ejercicio'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
//Realizamos la consulta
$consulta = $gestor->addEjercicioSesion($Id_Sesion, $Id_Ejercicio);
$json[] = array("addEjercicioSesion"=>$consulta);
// Codificamos el resultado en JSON
echo json_encode($json);

?>
```

addSesion.php

"addSesion.php"

```php
<?php

iconv_set_encoding("output_encoding","UTF-8");
//Toma los valores enviados por la aplicacion
$nomb = $_POST['nombre_Sesion'];
$monitor = $_POST['user_Monitor'];
$fecha = $_POST['fecha'];
$numUsuarios = $_POST['num_Usuarios'];
$activa = $_POST['activa'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$consulta = $gestor->nuevaSesion($nomb, $monitor, $fecha, $numUsuarios, $ip, $activa);
$json[] = array("nuevaSesion"=>$consulta);
// Codificamos el resultado en JSON
echo json_encode($json);
?>
```

addUserSesion.php

"addUserSesion.php"

```php
<?php

//Toma los valores enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];
$nom_Usuario = $_POST['nom_Usuario'];
```

```

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
//Obtenemos el ID del usuario
$consultaUsuario = $gestor->datosUsuario($nom_Usuario);
//Realizamos la consulta a la base de datos
$consulta = $gestor->addUsuarioSesion($Id_Sesion, $consultaUsuario[0]['Id_Usuario']);
$json[] = array("addUserSesion"=>$consulta);
//Codifica en JSON
echo json_encode($json);

?>
```

addUsuario.php

"addUsuario.php"

```
<?php
/**
* Script Php que se ejecuta en el servidor Apache,
* con la finalidad de insertar a un usuario nuevo.
* Despues se envia respuesta codificada en JSON.
* @Autor Antonio Jose Diaz Lora (antdialor84@gmail.com)
* @fecha Mayo 2014
*/
//Toma los valores enviados por la aplicacion
$Nomb = $_POST['nombre'];
$Apell = $_POST['apellidos'];
$DNI = $_POST['dni'];
$Direc = $_POST['direccion'];
$Movil = $_POST['movil'];
$usuario = $_POST['usuario'];
$passw = $_POST['password'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$consulta = $gestor->addUsuario($Nomb, $Apell, $DNI, $Direc, $Movil, $usuario, $passw);
$json[] = array("regstatus"=>$consulta);
//Codifica resultado en JSON
echo json_encode($json);
?>
```

eliminaEjercicioSesion.php

"eliminaEjercicioSesion.php"

```
<?php

//Toma los valores enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];
$Id_Ejercicio = $_POST['Id_Ejercicio'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$consulta = $gestor->deleteEjercicioSesion($Id_Sesion, $Id_Ejercicio);
$json[] = array("eliminaEjercicioSesion"=>$consulta);
echo json_encode($json);
?>
```

eliminaSesion.php

"eliminaSesion.php"

```
<?php

iconv_set_encoding("output_encoding","UTF-8");
//Toma los valores enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$consulta = $gestor->deleteSesion($Id_Sesion);
$json[] = array("eliminaSesion"=>$consulta);
// Codificamos el resultado en JSON
echo json_encode($json);

?>
```

getEjercicio.php

"getEjercicio.php"

```
<?php
// Obtenemos los parametros enviados por la aplicacion android
$id_ejercicio = $_POST['id_ejercicio'];

```

```php
require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$json = $gestor->infoEjercicio($id_ejercicio);
//Codifica en JSON
echo json_encode($json);
?>
```

getSesion.php

"getSesion.php"

```php
<?php

// Obtenemos los parametros enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$json = $gestor->devuelveSesion($Id_Sesion);
// Codificamos en JSON
echo json_encode($json);
?>
```

getVideoEjercicio.php

"getVideoEjercicio.php"

```php
<?php

// Toma los valores enviados por la aplicacion
$Id_Ejercicio = $_POST['id_ejercicio'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$json = $gestor->getVideoEjercicio($Id_Ejercicio);
//Codifica en JSON
echo json_encode($json);
?>
```

listaEjercicios.php

"listaEjercicios.php"

```php
<?php
require_once 'gestor_bd.php';
$gestor = new gestor_BD();
//Realizamos la consulta a la base de datos
$json = $gestor->obtenerEjercicios();
//Codifica en JSON
echo json_encode($json);
?>
```

listaEjerciciosNoSesion.php

"listaEjerciciosNoSesion.php"

```php
<?php

//Toma los valores enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
//Realizamos la consulta a la base de datos
$json = $gestor->ejerciciosNoSesion($Id_Sesion);
//Codifica en JSON
echo json_encode($json);
?>
```

listaEjerciciosSesion.php

"listaEjerciciosSesion.php"

```php
<?php

//Toma los valores enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
//Realizamos la consulta a la base de datos
$json = $gestor->ejerciciosSesion($Id_Sesion);
//Codifica en JSON
echo json_encode($json);
?>
```

listaSesiones.php

"listaSesiones.php"

```php
<?php

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
//Codifica en JSON
echo json_encode($gestor->obtenerSesionesActivas());

?>
```

modificaUsuario.php

"modificaUsuario.php"

```php
<?php
/**
* Script Php que se ejecuta en el servidor Apache,
* con la finalidad de modificar la tupla informacion
* del usuario existente. Para ello usa la funcion "modifica" que
* que devuelve true (si modificacion correcta) o false (si  no ok).
* Despues se envia respuesta codificada en JSON.
* @Autor Antonio Jose Diaz Lora (antdialor84@gmail.com)
* @fecha Mayo 2014
*/
//Toma los valores enviados por la aplicacion
$Nomb = $_POST['nombre'];
$Apell = $_POST['apellidos'];
$DNI = $_POST['dni'];
$Direc = $_POST['direccion'];
$Movil = $_POST['movil'];
$usuario = $_POST['usuario'];
$passw = $_POST['password'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$consulta = $gestor->modifica($Nomb, $Apell, $DNI, $Direc, $Movil, $usuario, $passw);
$json[] = array("modificaok"=>$consulta);
//Codifica resultado en JSON
echo json_encode($json);
?>
```

setIpMonitorSesion.php

"setIpMonitorSesion.php"

```php
<?php

// Toma los valores enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];
$ip_Monitor = getenv('REMOTE_ADDR').":".$_POST['Puerto'];


require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$consulta = $gestor->setIpMonitorSesion($Id_Sesion, $ip_Monitor);
$json[] = array("setIpMonitor"=>$consulta);
// Codifica el resultado en JSON
echo json_encode($json);
?>
```

setSesionNoActiva.php

"setSesionNoActiva.php"

```php
<?php

// Tomamos los valores enviados por la aplicacion
$Id_Sesion = $_POST['Id_Sesion'];

require_once 'gestor_bd.php';
$gestor = new gestor_BD();
$consulta = $gestor->setSesionNoActiva($Id_Sesion);
$json[] = array("setNoActiva"=>$consulta);
// Codifica el resultado en JSON
echo json_encode($json);
?>
```

2 Manual de instalación

2.1 Manual de instalación del servidor

Para poner en funcionamiento el servidor precisaremos de un servidor Apache con el módulo PHP instalado. Por comodidad, recomendamos la instalación de la última versión del paquete XAMPP[1], que incluye todo el software necesario. Una vez instalado, solo tendremos que copiar la carpeta *gymserver* dentro de la carpeta pública del servidor HTTP.

Por otro lado, para desplegar la base de datos tendremos que ejecutar las sentencias SQL contenidas en el fichero *sentenciassql.sql*. Si se ha instalado XAMPP, al incorporar PHPMyAdmin podremos ejecutarlas desde su interfaz gráfica simplemente importando el archivo SQL, como se muestra en la figura (1), lo cual sin duda resulta bastante más cómodo.

Figura 1 Importar base de datos desde interfaz gráfica PHPMyAdmin.

Cada vez que queramos poner el funcionamiento el servidor del gimnasio, podremos hacerlo de una manera sencilla desde la interfaz gráfica de XAMPP (figura 2).

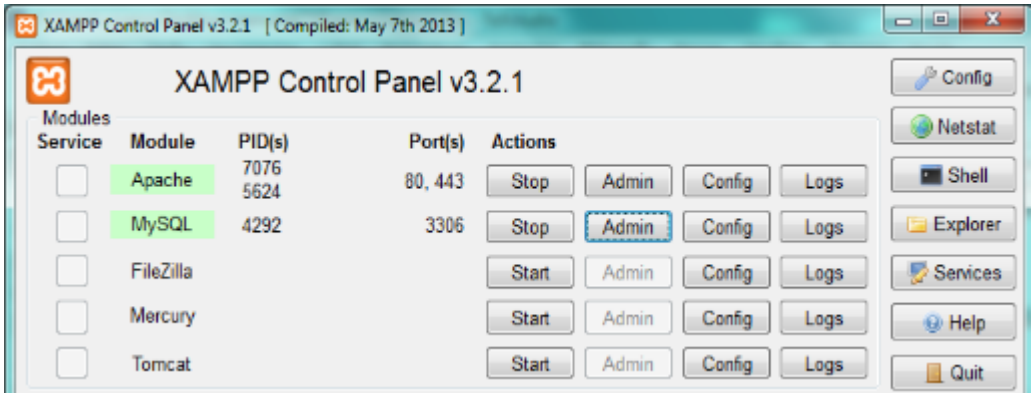

Figura 2 Interfaz gráfica de XAMPP con el servidor en funcionamiento.

2.2 Manual de instalación de la aplicación Java

Para poner en funcionamiento la aplicación Java deberemos instalar en nuestro equipo el IDE Eclipse con las herramientas para Java. Podemos encontras este programa en el enlace a pie de página[2], que es la versión de

[1] https://www.apachefriends.org/es/download.html
[2] https://eclipse.org/downloads/packages/eclipse-ide-java-developers/lunar

Eclipse para desarrolladores de Java, aunque funcionaría con cualquier otra versión de Eclipse si descargamos e instalamos todas las herramientas para la compilación y ejecución de aplicaciones Java.

Una vez en Eclipse, tan solo tendremos que abrir el proyecto Java que se encuentra en la carpeta *monitorgym*. Para que la aplicación funcione correctamente el servidor del gimnasio debe estar encendido. En caso contrario, no seremos capaces de pasar más allá de la ventana de inicio de sesión. Por otro lado, para que nuestra aplicación sea capaz de conectar con el servidor, tendremos que configurar la su dirección IP. Para ello hacemos click en el icono de configuración que aparece en la ventana de inicio de sesión, tras lo que nos aparecerá una ventana para introducir la dirección del servidor (figura 3). Una vez hecho esto la aplicación podrá comunicarse con el servidor del gimnasio sin problemas.

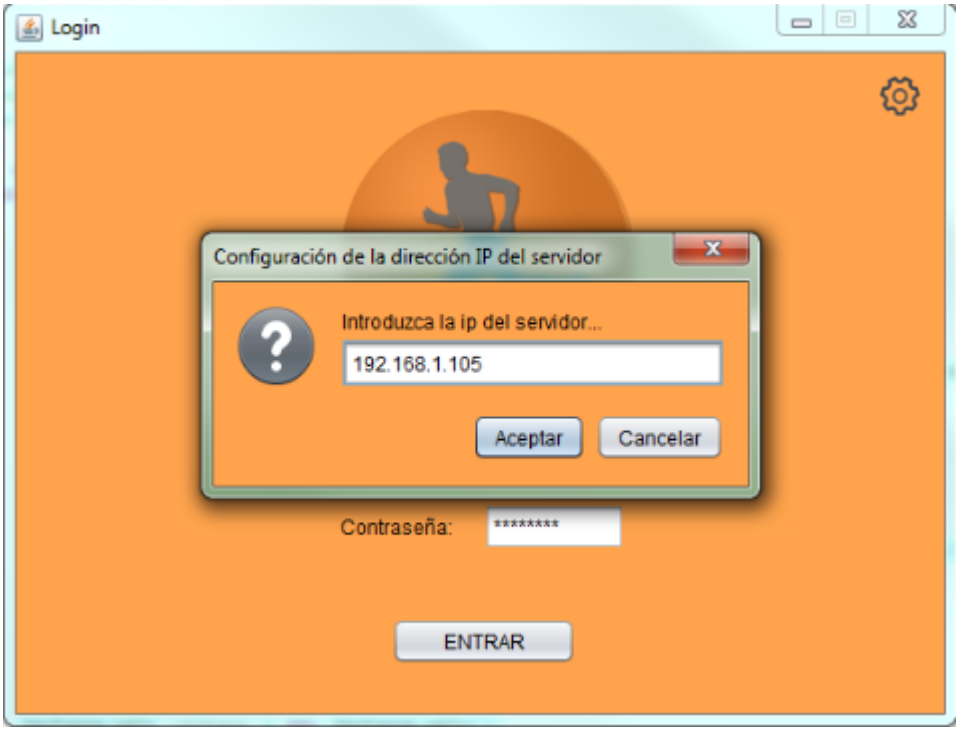

Figura 3 Ventana de configuración de la dirección IP del servidor de gimnasio.

Por último, para poder reproducir el vídeo de los usuarios será necesario tener instalada en nuestro equipo la última versión de VLC[3], ya que algunas librerías de nuestra aplicación utilizan la versión de VLC instalada en nuestro equipo.

Una vez dentro de la aplicación se nos solicitarán una credenciales. Deberemos introducir tanto en el campo de nombre como de contraseña la palabra *'pablo'*, tras lo cual accederemos al sistema pulsando sobre el botón de *Entrar*.

2.3 Manual de instalación de la aplicación Android

Para instalar esta aplicación deberemos copiar el archivo *app-debug-unaligned.apk*, que encontraremos en el directorio *ProjectGym/app/build/outputs/apk* en el dispositivo Android donde queramos instalar la aplicación. Con un gestor de ficheros en el dispositivo, buscamos el archivo y lo ejecutamos.

Posiblemente sea necesario realizar cambios en los ajustes del dispositivo para permitir la ejecución de aplicaciones de Órígnes desconocidos', para así permitir que se instalen aplicaciones que no provengan del *PlayStore*, tienda de aplicaciones de Android.

Una vez dentro de la aplicación se nos solicitarán una credenciales. Deberemos introducir tanto en el campo de nombre como de contraseña la palabra *'cliente'*, tras lo cual accederemos al sistema pulsando sobre el botón de *Entrar*.

[3] http://www.videolan.org/vlc/