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

Servicio web REST para la monitorización mediante streaming de vídeo con Spring Framework y Spring Security

Autor: Pablo Gallardo Hiraldo

Tutor: Mª Teresa Ariza Gómez

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

Sevilla, 2018

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

Servicio web REST para la monitorización mediante streaming de vídeo con Spring Framework y Spring Security

Autor:
Pablo Gallardo Hiraldo

Tutor:
Mª Teresa Ariza Gómez
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Trabajo Fin de Grado: Servicio web REST para la monitorización mediante streaming de vídeo con Spring Framework y Spring Security

Autor: Pablo Gallardo Hiraldo

Tutor: Mª Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi familia

A mis amigos

A mis maestros

Agradecimientos

Quiero dar las gracias a toda aquella persona, que de un modo u otro, haya estado presente en mi vida sirviendo de alguna ayuda. Y concretamente, tengo que agradecer a mis padres el haberme dado tantas oportunidades y haber sido los pilares fundamentales sobre los que me he apoyado para poder llegar hasta aquí, permitiéndome superar grandes dificultades.

"Lo sé. Ha sido un error. No deberíamos ni haber llegado hasta aquí... Pero henos aquí, igual que en las grandes historias, señor Frodo, las que realmente importan, llenas de oscuridad y de constantes peligros. Ésas de las que no quieres saber el final, porque ¿cómo van a acabar bien? ¿Cómo volverá el mundo a ser lo que era después de tanta maldad como ha sufrido? Pero al final, todo es pasajero. Como esta sombra, incluso la oscuridad se acaba, para dar paso a un nuevo día. Y cuando el sol brilla, brilla más radiante aún. Esas son las historias que llenan el corazón, porque tienen mucho sentido, aún cuando eres demasiado pequeño para entenderlas. Pero creo, señor Frodo, que ya lo entiendo. Ahora lo entiendo. Los protagonistas de esas historias se rendirían si quisieran. Pero no lo hacen: siguen adelante, porque todos luchan por algo."

Samsagaz Gamyi – El Señor de los Anillos.

Pablo Gallardo Hiraldo

Sevilla, 2018

Resumen

La tecnología ha avanzado hasta el punto en el que nos permite disponer de dispositivos de todo tipo conectados a Internet. Si además tenemos en cuenta el crecimiento de las redes de telecomunicación, nos encontramos en una situación ideal para el desarrollo de aplicaciones que utilicen transmisión de vídeo para la monitorización. Pero nada de esto sería posible si no nos apoyamos en un servidor eficiente y seguro.

En este proyecto se ha desarrollado la infraestructura necesaria para que, tanto un monitor con su aplicación Java en su ordenador, como un usuario con su aplicación Android en su Smartphone, puedan llevar a cabo conjuntamente una sesión de entrenamiento monitorizada simplemente estando conectados a Internet.

El principal objetivo del proyecto es crear un servicio web RESTful en Java Spring Framework que permita la comunicación de las aplicaciones de monitor (Java) y de cliente (Android) con la base de datos MySQL (en la que está almacenada toda la información sobre los usuarios, sesiones y ejercicios), así como dotarlo de distintas medidas de seguridad tales y como autenticación, autorización y codificación de contraseñas.

Abstract

Technology has evolved up to the point of allowing us to have many different devices with an Internet connection. Furthermore, thanks to the growing use of communication technologies, it is an ideal situation to develop video streaming monitoring applications. But none of this would be possible if we don't base it on a secure and efficient server.

In this project, we have designed the necessary infrastructure to allow an expert with a Java app on his computer and a user with an Android app in his Smartphone, to be able to perform a video-streaming monitored training session simply by having an Internet connection.

The main goal of this project is to create a RESTful web service using Java Spring Framework which allows the communication between the expert/clientes apps and the database (where all the information about users, sessions and exercises is stored), as well as providing security measures to the server, such as authentication, authorization and password codification.

Índice

Agradecimientos **9**

Resumen **11**

Abstract **13**

Índice **15**

Índice de Figuras **17**

1 Introducción **1**
1.1 Motivaciones *1*
1.2 Escenario inicial *1*
1.3 Objetivos *2*
1.4 Agentes del sistema *2*
1.5 Arquitectura del sistema *3*
1.6 Estructura de la memoria *4*

2 Tecnologías y Herramientas **7**
2.1. Tecnologías *7*
2.1.1 Java 7
2.1.2 JSON 8
2.1.3 MySQL 8
2.1.4 Servicio web RESTful 9
2.1.5 Protocolo OAuth2 10
2.1.6 Spring Framework 11
2.1.6.1 Maven 11
2.1.6.2 Spring Boot 12
2.1.6.3 Spring Security 12
2.1.6.4 Patrón DAO 12
2.1.6.5 JDBC 13
2.1.7 YouTube Android API 13
2.1.8 HTTPS 14
2.2 Herramientas *14*
2.2.1 Spring Tool Suite 14
2.2.2 XAMPP 15
2.2.3 Android Studio 15

3 Servicio Web **17**
3.1 Base de datos *17*
3.2 Servicio web desarrollado *19*
3.2.1 Usuarios 20
3.2.2 Sesiones 21
3.2.3 Ejercicios 22
3.3 Seguridad implementada *26*
3.3.1 Autentificación mediante OAuth2 26
3.3.2 Autorización y roles 29
3.3.3 Codificación de contraseñas 29
3.4 Diagramas de Secuencia *30*

4 Aplicación Java **33**
4.1 Gestor de peticiones *33*
4.2 Ventana de registro *35*
4.3 Funcionamiento de la aplicación *37*

5 Aplicación Android **41**
5.1 Gestor de peticiones *41*
5.2 YouTube API *41*
5.3 Funcionamiento de la aplicación *42*

6 Conclusiones y Líneas de mejora **47**
6.1 Objetivos logrados *47*
6.2 Posibles líneas de mejora *48*

Bibliografía **49**

Anexos **51**
1 Manual de instalación *51*
1.1 Manual de instalación de la base de datos 51
1.2 Manual de instalación del servicio web 52
1.3 Manual de instalación de la aplicación Java 56
1.4 Manual de instalación de la aplicación Android 57
2 Código *58*
2.1 Servicio Web 58
2.2 Seguridad Servicio Web 73
2.3 Configuración Servicio Web 78
2.4 Aplicación Java 80
2.5 Aplicación Android 88

Índice de Figuras

Figura 1-1 Arquitectura proyecto Pablo Gil Pereira 2
Figura 1-2 Arquitectura general del sistema 4
Figura 2-1 Logo gráfico de Java 7
Figura 2-2 Logo gráfico de JSON 8
Figura 2-3 Logo gráfico de MySQL 8
Figura 2-4 Logo gráfico de OAuth2 10
Figura 2-5 Logo gráfico de Spring 11
Figura 2-6 Logo gráfico de Maven 12
Figura 2-7 Representación gráfica de JDBC 13
Figura 2-8 Logo gráfico de YouTube 13
Figura 2-9 Logo gráfico de Spring Tool Suite 14
Figura 2-10 Logo gráfico de XAMPP 15
Figura 2-11 Logo gráfico de Android Studio 15
Figura 3-1 Base de datos: modelo entidad-relación. 18
Figura 3-2 Tablas para cumplir con los objetivos 19
Figura 3-3 Agrupación de clases java de nuestro servicio web 20
Figura 3-4 Diagrama de clases de gestión de usuarios 23
Figura 3-5 Diagrama de clases de gestión de sesiones 24
Figura 3-6 Diagrama de clases de gestión de ejercicios 25
Figura 3-7 Diagrama de secuencia del funcionamiento de OAuth2 27
Figura 3-8 Clase *OAuth2SecurityConfiguration* 27
Figura 3-9 Clase *AuthorizationServerConfiguration* 28
Figura 3-10 Clase *ResourceServerConfiguration* 28
Figura 3-11 Clase *CustomUserDetailsService* 29
Figura 3-12 Diagrama de secuencia de autenticación de usuario. 30
Figura 3-13 Diagrama de secuencia de la edición de una sesión. 31
Figura 4-1 Clase *GestorPeticiones* 34
Figura 4-2 Clase *AuthTokenInfo* 34
Figura 4-3 Ventana Login con el nuevo botón REGISTRAR 35
Figura 4-4 Ventana Registro 35
Figura 4-5 Ventana Registro en el caso de usuario repetido 36
Figura 4-6 Ventana Registro en el caso de registro con éxito 36
Figura 4-7 Ventana Login 37
Figura 4-8 Ventana Configuración IP 37
Figura 4-9 Ventana Sesiones 38
Figura 4-10 Ventana Sesiones creando nueva sesión 38

Figura 4-11 Ventana Ejercicios 39
Figura 4-12 Ventana Ejercicios reproduciendo vídeo del cliente 39
Figura 5-1 Reproductor de YouTube dentro de nuestra aplicación 42
Figura 5-2 Inicio de sesión 43
Figura 5-3 Configuración de IP y puerto 43
Figura 5-4 Lista de sesiones activas 44
Figura 5-5 Lista de ejercicios de la sesión 44
Figura 5-6 Reproducción de vídeo del ejercicio 44
Figura 5-7 Información del ejercicio 45
Figura 5-8 Mensaje del monitor 45
Figura 6-1 Interfaz gráfica de XAMPP 51
Figura 6-2 Importar base de datos desde interfaz gráfica PHPMyAdmin 51
Figura 7-1 Interfaz de Spring. Abrir proyecto del sistema de archivos. 52
Figura 7-2 Ventana para importar proyectos desde un directorio 52
Figura 7-3 Interfaz de Spring en la se observa el botón de ejecución (Run Application) 53
Figura 7-4 Resultado del comando 'java InstallCert IPservidor:9999' (parte 1) 54
Figura 7-5 Resultado del comando 'java InstallCert IPservidor:9999' (parte 2) 55
Figura 8-1 Ventana de configuración de la dirección IP del servicio web 56
Figura 9-1 Pantalla de configuración de IP y puerto del servicio web 57

1 INTRODUCCIÓN

"Donde quiera que se encuentre el prejuicio,
siempre nubla la verdad"

- Henry Fonda en 12 hombres sin piedad -

En este primer capítulo se expondrán los diferentes objetivos y motivaciones que han servido de guía para la realización de este proyecto. Además definiremos cuáles han sido los antecedentes a este trabajo. Por último, indicaremos qué estructura va a tener nuestra memoria.

1.1 Motivaciones

En la vida moderna se dan muchas situaciones en las que debido a la saturación del trabajo y al cansancio, se empieza a descuidar la salud. Es demasiado fácil ceder a las tentaciones de la comida basura y el sedentarismo, pero más aún si no se dispone de tiempo necesario para hacer lo que se debe hacer. El objetivo principal de la aplicación es hacer más fácil y accesible todo lo que corresponde al ejercicio físico a los posibles clientes de un gimnasio. Desde la comodidad de tu hogar podrías ser capaz de llevar a cabo una sesión de entrenamiento monitorizada por un entrenador personal. Sería útil tanto para clientes que ocasionalmente no pudieran asistir a una sesión en persona al Gimnasio, como para aquellos que deseen simplemente llevar a cabo todos sus entrenamientos sin salir de casa.

1.2 Escenario inicial

Para la realización de este proyecto partimos de otro realizado con anterioridad por Pablo Gil Pereira [1]. En dicho proyecto, tanto el usuario con dispositivo móvil como el monitor con un ordenador pueden acceder a diferentes sesiones y ejercicios a través de un servidor y una base de datos. Tras intercambiar información a través del servidor, el cliente y el monitor establecen una conexión y se da una transmisión de vídeo desde el cliente hacia el monitor (figura 1-1).

Su objetivo principal fue desarrollar una aplicación Java para el monitor, partiendo a su vez del servidor PHP del proyecto de Antonio José Díaz Lora [2] y de la aplicación Android de Antonio Clavaín Mateo [3]. En nuestro caso partiremos exclusivamente del proyecto de Pablo y de sus modificaciones sobre los proyectos anteriores. Reutilizaremos la base de datos y las aplicaciones Java y Android, realizando sobre todas ellas una serie de modificaciones que se comentarán más adelante.

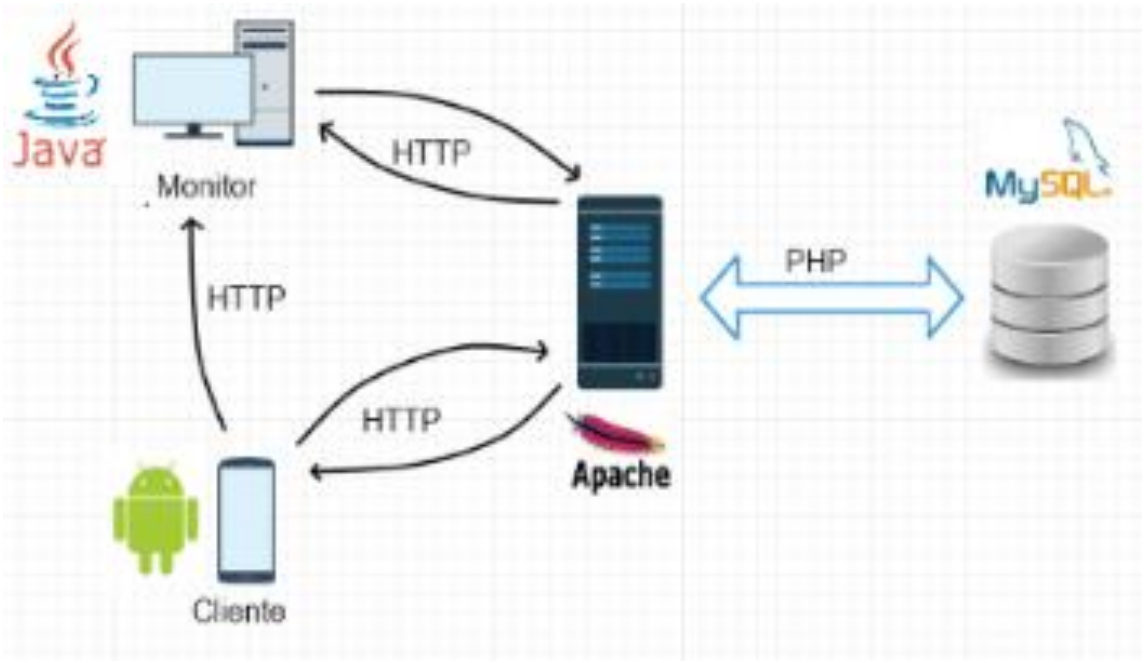

Figura 1-1 Arquitectura proyecto Pablo Gil Pereira

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 un servicio web RESTful en el lenguaje de programación Java, que sirva de puente entre las aplicaciones Android/Java y la base de datos MySQL previamente desarrolladas, para así poder llevar a cabo una correcta monitorización de sesiones de entrenamiento.

A este objetivo principal se le unen varios objetivos secundarios, alguno de gran peso. En primer lugar, se quiere dotar al servicio web de seguridad, puesto que desde el servidor se accede a toda la información disponible en la base de datos y es de vital importancia estar protegidos ante el acceso de usuarios no deseados. Se pretende implementar un sistema de autenticación usando el protocolo Oauth2, un sistema de roles para acceder a los diferentes recursos del servicio web y codificar la información más importante, como las contraseñas, en la base de datos. Además de todo ello, el intercambio de mensajes entre las aplicaciones y el servidor será mediante HTTP sobre SSL.

Por último, además de modificar la aplicación Java, la aplicación Android y la base de datos para su correcta comunicación con el nuevo servicio web y su seguridad, se llevarán a cabo pequeñas mejoras en ambas. Por ejemplo, se desarrollará una ventana de registro para nuevos usuarios en la aplicación del monitor. También se cambiará en la aplicación Android el sistema de reproducción de vídeos explicativos sobre los ejercicios que están alojados en el servidor.

1.4 Agentes del sistema

En esta sección introduciremos los diferentes elementos que componen nuestro sistema, que serán cuatro: servicio web, base de datos, aplicación Java y aplicación Android.

- **Servicio web**: Servicio web con arquitectura RESTful, desarrollado completamente mediante el uso de Java Spring Framework. Su función principal es coordinar la autenticación de usuarios y la

realización de operaciones CRUD (*Create, Read, Update and Delete*, en inglés) relacionadas con los usuarios, las sesiones de entrenamiento y los ejercicios que forman las sesiones. Cabe destacar, que el formato de ficheros de intercambio entre las aplicaciones Java/Android y el servicio web Spring por el que se ha optado ha sido JSON. El servicio web del gimnasio se explicará más profundamente en el capítulo (3).

- **Base de datos**: La base de datos es el lugar donde almacenaremos todos los datos relacionados con información de usuarios, sesiones y ejercicios del sistema.

- **Aplicación Java**: Aplicación desde la cual el monitor podrá crear sesiones para los clientes, así como monitorizarlos mediante streaming de vídeo mientras desarrollan los ejercicios seleccionados por él mismo. El monitor tiene la opción de registrar nuevos usuarios (tanto monitores como clientes) en la aplicación. Para acceder al sistema, el monitor deberá iniciar sesión y, una vez dentro, se le mostrará un menú con las sesiones, tanto las activas como las no activas. Desde dicho menú se pueden crear nuevas sesiones, acceder a ellas e incluso eliminarlas. Una vez accedemos a una sesión, se mostrará una lista con los ejercicios que la forman a la vez que una segunda lista con el resto de ejercicios disponibles para ser añadidos a la sesión si se quiere en cualquier momento. En esta ventana también encontramos los 4 reproductores de vídeo donde, al comenzar una sesión, se podrán reproducir las imágenes que nos lleguen de los respectivos clientes inscritos en la misma. El monitor dispone de la posibilidad de enviar un mensaje de texto instantáneo al usuario al que está monitorizando en caso de que necesite comunicarle algo. Esta aplicación será tratada con más detalle en el capítulo (4).

- **Aplicación Android**: Aplicación que será utilizada por los clientes para llevar a cabo los ejercicios desde cualquier lugar. 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, un cliente puede iniciar sesión en el sistema, lo que le mostrará todas las sesiones activas en el momento. Al seleccionar una, el usuario queda automáticamente inscrito en la misma y le aparecerán la lista de ejercicios asociados a esa sesión. Si la sesión ha comenzado, se podrán seleccionar los ejercicios que la componen para su ejecución. En este momento se empieza a reproducir el vídeo explicativo sobre el ejercicio en la aplicación móvil y a la vez se empieza a transmitir el vídeo que capta la cámara para que el monitor lo vea en su pantalla. Esta aplicación se explica más ampliamente en el capítulo (5).

1.5 Arquitectura del sistema

En la figura (1-2) se pueden observar los agentes del sistema de los que hemos hablado previamente y cómo interactúan entre ellos.

La comunicación entre el servicio web y la base de datos se lleva a cabo a través de JDBC (Java Database Connectivity). La base de datos contiene absolutamente toda la información referente al sistema y el servicio se encargará de hacerle las consultas necesarias para extraer dicha información. Las aplicaciones Java del monitor y Android del usuario usarán exclusivamente peticiones HTTP para comunicarse con el servicio web.

Por último, al reutilizarse ambas aplicaciones del proyecto de Pablo Gil, el envío de streaming de vídeo seguirá siendo de manera directa sin pasar por un servidor intermedio, sino que habrá un servidor en la propia aplicación Android que estará a la espera de conexiones para enviar el vídeo. Una vez el monitor haya enviado una petición para ver al cliente, se transmitirá el vídeo en formato MJPEG sobre HTTP.

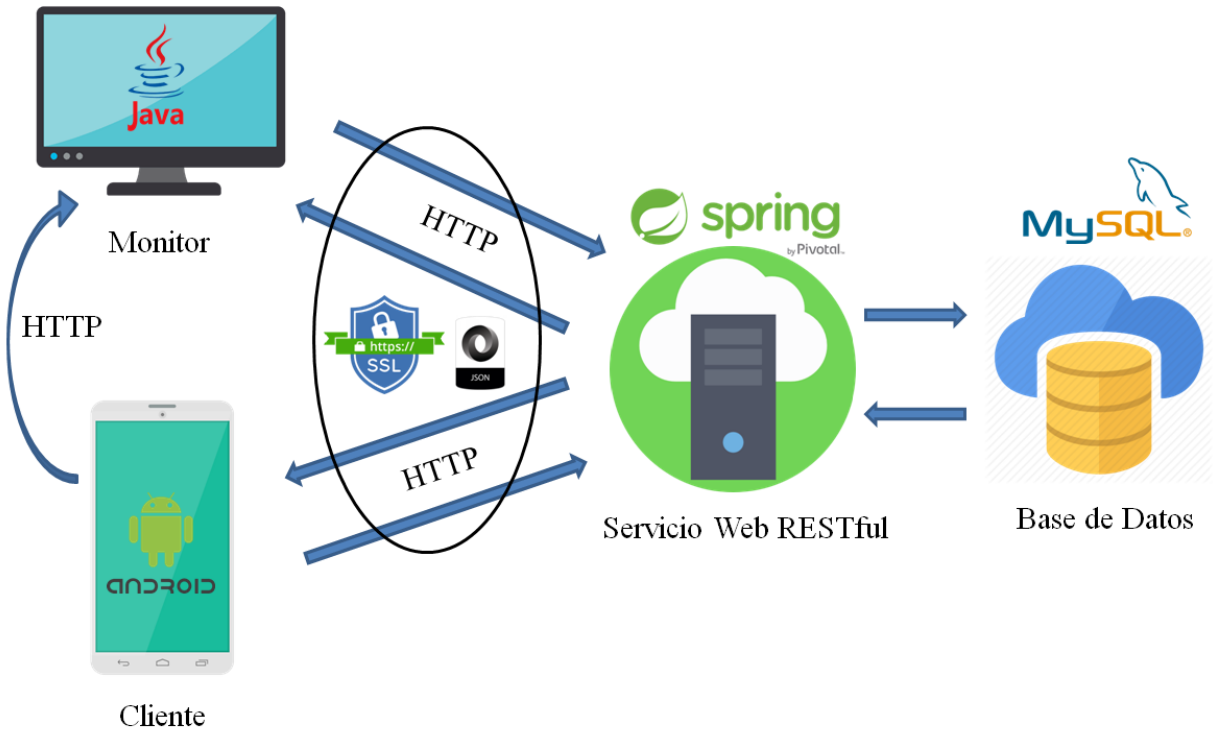

Figura 1-2 Arquitectura general del sistema

1.6 Estructura de la memoria

A continuación, comentaremos brevemente el esquema de capítulos que forman la memoria:

- **Capítulo 1**. Introducción: Este capítulo consta de las motivaciones del proyecto, del escenario inicial del que se ha partido y de los objetivos que se han marcado para el final del proyecto. Sin olvidar la descripción de los agentes del sistema así como de la estructura de éste.
- **Capítulo 2**. Tecnologías y herramientas: Esta sección consta de un pequeño resumen de cada tecnología y herramienta que ha sido usada en el desarrollo de este proyecto en el que se explican el lugar y la razón por la que han sido usadas.
- **Capítulo 3**. Servicio web: Este capítulo consta de una descripción detallada del servicio web del gimnasio y está dividido en varias partes. Se hablará de la base de datos, del servicio web en sí mismo y por último de las medidas de seguridad que implementan. Se usarán diagramas de diversos tipos.
- **Capítulo 4**. Aplicación Java: En esta sección se muestran las funcionalidades de la aplicación y se explican las modificaciones efectuadas para cumplir con los objetivos marcados al principio del proyecto.
- **Capítulo 5**. Aplicación Android: Capítulo en el que se tratarán las funcionalidades de a aplicación y se explicarán las modificaciones llevadas a cabo para completar los objetivos planteados.
- **Capítulo 6**. Conclusiones y líneas de mejora: Sección en la que se lleva a cabo un análisis final de los objetivos para comprobar que se han alcanzado. También se presentan maneras o tecnologías para mejorar la aplicación, así como unas conclusiones personales del proyecto.

- **Bibliografía**. Relación de textos procedentes de diversos soportes utilizados como fuente documental.
- **Anexos**. Se proporcionará un detallado manual de instalación para poner en marcha el sistema.

2 TECNOLOGÍAS Y HERRAMIENTAS

"-Cuenta con mi espada.

-Y cuenta con mi arco.

-¡Y con mi hacha!"

-El señor de los Anillos, por J.R.R. Tolkien-

La tarea de escoger las tecnologías y herramientas para el desarrollo de un proyecto es una fase vital. De ello depende en buena parte el éxito de nuestra empresa y las dificultades que podríamos encontrar durante todo el trayecto hasta llegar a los objetivos propuestos. En este segundo capítulo nos centraremos en explicar en qué consiste cada una de las tecnologías usadas, así como las razones que nos han llevado a tomar la decisión de usarlas en nuestro proyecto. Con las herramientas de desarrollo seguiremos el mismo procedimiento.

2.1. Tecnologías

2.1.1 Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Esto ha hecho que actualmente sea uno de los lenguajes de programación más extendidos, especialmente es las aplicaciones cliente-servidor web.

Figura 2-1 Logo gráfico de Java

Gracias a esta versatilidad, encontramos muy variados entornos de funcionamiento de aplicaciones, como dispositivos móviles, aplicaciones de escritorio y servicios web. De ahí que hayamos decidido usar este entorno para el desarrollo de nuestro servicio web.

2.1.2 JSON

JSON es el acrónimo para JavaScript Object Notation, y aunque su nombre lo diga, no es necesariamente parte de JavaScript, de hecho es un estándar basado en texto plano para el intercambio de información, por lo que se usa en muchos sistemas que requieren mostrar o enviar información para ser interpretada por otros sistemas, la ventaja de JSON al ser un formato que es independiente de cualquier lenguaje de programación, es que los servicios que comparten información por éste método, no necesitan hablar el mismo idioma, es decir, el emisor puede ser Java y el receptor PHP, cada lenguaje tiene su propia librería para codificar y decodificar cadenas de JSON.

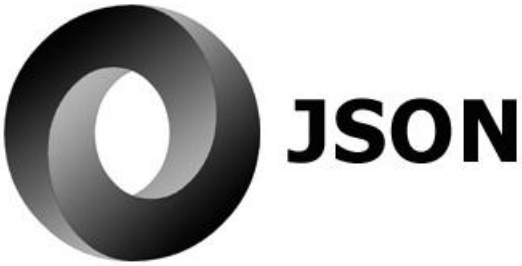

Figura 2-2 Logo gráfico de JSON

Tal y como se puede observar en la Figura 1-2, se ha optado por JSON como estándar de transmisión de datos entre los clientes Java/Android y el servicio web. El motivo principal de la elección de JSON ha sido la sencillez con la que se puede generar y manejar en comparación con otros estándares distintos

2.1.3 MySQL

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation y está considerada como el sistema de base de datos Open Source más popular del mundo. Este gestor de base de datos es multihilo y multiusuario, lo que le permite ser utilizado por varias personas al mismo tiempo, e incluso, realizar varias consultas a la vez, lo que lo hace sumamente versátil.

Figura 2-3 Logo gráfico de MySQL

En nuestro proyecto hemos optado por MySQL como opción de almacenamiento de todos los datos de la aplicación, lo que significa que lo utilizaremos con el fin de guardar la información relacionada con usuarios, sesiones y ejercicios, así como de acceder a la propia información. Los principales motivos son su alta popularidad en el mundo, su facilidad para gestionarla mediante herramientas como XAMPP y por último y más importante, debido a la existencia de una interfaz de conexión a base de datos MySQL compatible con Spring denominada JDBC.

2.1.4 Servicio web RESTful

La Transferencia de Estado Representacional (REpresentation State Transfer - REST) describe un estilo arquitectónico de sistemas en red como, por ejemplo, aplicaciones Web. El término fue utilizado por primera vez en el año 2000 durante una disertación doctoral por Roy Fielding, uno de los principales autores de la especificación HTTP. REST está comprendida por una serie de limitaciones y principios arquitectónicos. Si una aplicación o diseño cumple con esas limitaciones y principios, se considera RESTful.

Hay que mencionar que REST estaba basado en una serie de estándares como pueden ser:

- HTTP
- URL
- Representación de los recursos: JSON, XML, JPEG…
- Tipos MIME: application/json, text/html…

También deben quedar claro los principios sobre los que se apoya REST:

- El uso explícito de los métodos de HTTP de forma que sea consistente con la definición del protocolo. Este principio básico de diseño establece una equivalencia entre las operaciones CRUD (create, read, update and delete) y los métodos HTTP.
    - Para crear un recurso en el servidor, se usa POST
    - Para extraer un recurso, se usa GET
    - Para cambiar el estado de un recurso o para actualizarlo, se usa PUT
    - Para eliminar un recurso, se usa DELETE
- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.
- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso se puede direccionar únicamente a través de su URI (Identificador Uniforme de Recurso).
- El uso de hipermedios como motor para poder indicar las diferentes acciones. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, por el mero hecho de seguir enlaces sin requerir ninguna otra infraestructura adicional.

Tal y como se observa en la figura 1-2, hemos utilizado REST como modelo de diseño para el servicio web. Los motivos han sido su cada vez más extendido uso en el mundo de la computación y por su sencillez a la hora de manipular datos. REST es el estándar más lógico, eficiente y habitual para la creación de APIs para servicios de Internet.

2.1.5 Protocolo OAuth2

OAuth 2.0 es el protocolo estándar de la industria para autorización. OAuth2 reemplaza al protocolo OAuth original creado en 2006. Este protocolo permite a terceros (clientes) acceder a contenidos propiedad de un usuario (alojados en aplicaciones de confianza, un servidor de recursos) sin que éstos tengan que manejar ni conocer las credenciales de usuario. A continuación, explicaremos la información básica sobre el protocolo.

Se distinguen 4 roles distintos en OAuth2:

- **Resource Owner**: Normalmente es el usuario final que es dueño de los recursos.
- **Resource Server**: Es la API o servicio que se protege y en la que se encuentran los recursos.
- **Cliente**: La aplicación (web, móvil o escritorio) que solicita el acceso al Resource Server.
- **Authorization Server**: Servicio que se dedica a emitir tokens de acceso a los clientes para que puedan acceder a los distintos recursos.

Una vez conocemos los roles, podemos hablar de los 4 *modos* de OAuth, llamados *grant types*. Cada uno de ellos tiene un objetivo distinto y se usa de manera diferente:

- **Authorization Code**: Se usa cuando el cliente es un servidor web.
- **Implicit**: Se usa cuando el cliente se ejecuta en un navegador usando un lenguaje basado en scrpits como sería Javascript.
- **Resource Owner Password Credentials**: Con este tipo de autorización, las credenciales del dueño del recurso son enviadas al cliente y luego al Authorization Server.
- **Client Credentials**: Se usa cuando el cliente es a la vez el dueño de los recursos (Resource Owner).

Por último, toca definir los dos tipos de tokens, que son cadenas aleatorias y específicas generadas por el Authorization Server y son mandadas a los clientes que las solicitan:

- **Access Token**: Se envía con cada solicitud, normalmente válidas por un corto periodo de tiempo.
    - **Access Token Scope**: El scope (ámbito en español) es un parámetro usada para limitar los permisos y derechos de los que dispone el token de acceso.
- **Refresh Token**: Principalmente usadas para conseguir un nuevo token de acceso, no se envía en cada solicitud y normalmente viven durante más tiempo.

Figura 2-4 Logo gráfico de OAuth2

En el proyecto se usará el modo *Resource Owner Password Credentials.* La razón es simple, pues tenemos un cliente basado en Java (aplicación de monitor) que dispone de las credenciales del dueño de los recursos (Resource Owner) y se las manda al Authorization Server para luego recibir el token de acceso (y opcionalmente el token de refresco), y así usar el token para acceder a los recursos del servicio web.

2.1.6 Spring Framework

Spring es un framework que se utiliza para el desarrollo de aplicaciones, es de código abierto y utiliza Java como lenguaje de programación. Fue escrito originalmente para la plataforma J2EE de Java, plataforma orientada al desarrollo de aplicaciones web y ha ido evolucionando rápidamente hasta el día de hoy, donde podemos encontrar diferentes ramas de desarrollo de la mano de SpringSource y todo su equipo de desarrolladores.

Spring en la actualidad es la referencia en el mundo de los frameworks de programación para los desarrolladores web de todo el mundo. Su éxito se fundamenta en la constante labor de investigación e innovación que realiza su equipo de desarrollo. Se trata de un framework que impulsa una metodología de trabajo ágil, eficiente y de buena praxis, lo que resulta en la creación de Software de elevada calidad y mantenibilidad.

Algunas características de este framework son:

- **Inversión de Control (IoC)**: Spring implementa un "Contenedor IoC" que se encarga de gestionar las instancias de los objetos del usuario. Por tanto, las aplicaciones como la nuestra que usen Spring usarán Inversión de control.
- **Inyección de Dependencias(DI)**: Patrón de diseño que sirve para "inyectar" componentes a las clases que tenemos implementadas. Para usar la inyección de dependencias hemos definidos unos Beans de Spring y están relacionados con el dataSource de conexión a la base de datos y los DAO de usuarios, sesiones y ejercicios, con el fin de aprovechar al máximo lo que esta capacidad de Spring nos ofrece.
- **Spring Beans**: Son objetos creados y manejados por el contenedor de Spring. El contenedor se encuentra en el núcleo del marco de trabajo de Spring y utiliza inyección de dependencias para gestionar los componentes que forma la aplicación. Se encarga de crea, conectar y alojar los objetos definidos por los Beans. Los hemos usado para definir los elementos DAO relacionados con los usuarios, ejercicios, sesiones y el dataSource.

Figura 2-5 Logo gráfico de Spring

En nuestro proyecto, tal y como se observa en la figura 1-2, hemos utilizado Spring como framework de nuestro servicio web RESTful, ya que aporta una gran cantidad de herramientas que eliminan muchas complejidades relacionadas con el desarrollo de servicios web.

Spring dispone de una gran cantidad de módulos a nuestra disposición, de los cuales hemos utilizado principalmente Spring Boot y Spring Security, además de Spring Relational Data Access.

2.1.6.1 Maven

Maven se utiliza en la gestión y construcción de software. Posee la capacidad de realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. Es decir, hace posible la creación de software con dependencias incluidas dentro de la estructura del JAR. Es necesario definir todas las dependencias del proyecto (librerías externas utilizadas) en un fichero propio de todo proyecto Maven, el POM (Project Object Model). Este es un archivo en formato XML que contiene todo lo necesario para que a la hora de generar el fichero ejecutable de nuestra aplicación este contenga todo lo que necesita para su ejecución en su interior.

Sin embargo, la característica más importante de Maven es su capacidad de trabajar en red. Cuando definimos las dependencias de Maven, este sistema se encargará de ubicar las librerías que deseamos utilizar en Maven Central, el cual es un repositorio que contiene cientos de librerías constantemente actualizadas por sus creadores.

Figura 2-6 Logo gráfico de Maven

Hemos optado por esta herramienta para la construcción de nuestra aplicación porque al definir las dependencias de librerías que necesitamos, el fichero POM.xml de configuración de Maven propio de nuestro proyecto procederá a su descarga y compilación automática. Además está muy bien integrado en el IDE personalizado de Spring

2.1.6.2 Spring Boot

Spring Boot es un módulo de Spring que nos facilita la creación de aplicaciones autónomas de estilo profesional basadas en Spring, basta con ejecutar. Consiste en un mecanismo centrado en una clase principal, que tiene una configuración por defecto y que puede ser personalizado. Es compatible tanto con Gradle como con Maven y a la hora desplegar los formatos que usa son JAR y WAR.

En el presente proyecto se ha usado Spring Boot para el lanzamiento y despliegue de nuestro servicio web, ya que Spring Boot incluye un servidor web Tomcat embebido.

2.1.6.3 Spring Security

Spring Security es un módulo de Spring, que dota al mismo de una serie de servicios de seguridad aplicables para sistemas basados en la arquitectura basados en Java. Los procesos de seguridad están destinados principalmente, a comprobar la identidad del usuario mediante la autenticación y los permisos asociados al mismo mediante la autorización. La autorización es dependiente de la autenticación ya que se produce posteriormente a su proceso.

A la hora de desarrollar en servicio web de este proyecto se han intentado implementar el mayor número posible que elementos de seguridad de Spring Security como han sido la autenticación usando el protocolo OAuth2, la restricción de acceso a diferentes recursos según el nivel de autorización y la codificación de información relevante como las contraseñas.

2.1.6.4 Patrón DAO

El patrón de diseño DAO, Data Access Object (Objeto de Acceso a Datos, en español), pertenece al catálogo de Core J2EE Patterns de Java. Lo que hace este patrón es usar un Objeto de Acceso a Datos para abstraer y encapsular el acceso a dichos datos. El objeto DAO obtiene y guarda los datos en el lugar en el que los almacenemos, manejando la conexión con el mismo. Siempre realizan las operaciones atómicas en la base de datos.

El uso que se le ha dado en el proyecto ha sido el de generar un objeto DAO para cada una de nuestras clases principales (usuarios, sesiones y ejercicios) y así poder realizas operaciones sobre ellos junto a la base de datos.

2.1.6.5 JDBC

Java Database Connectivity (JDBC) es la especificación JavaSoft de una interfaz de programación de aplicaciones (API) estándar que permite que los programas Java accedan a sistemas de gestión de bases de datos. La API JDBC consiste en un conjunto de interfaces y clases escribas en el lenguaje de programación Java.

Puesto que JDBC es una especificación estándar, un programa Java que utilice la API JDBC puede conectar con cualquier sistema de gestión de bases de datos (DBMS), siempre y cuando haya un controlador para dicho DBMS en concreto.

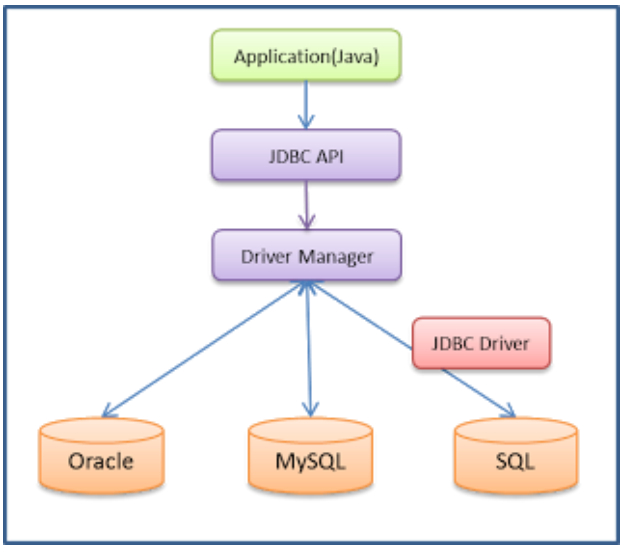

Figura 2-7 Representación gráfica de JDBC

En este trabajo se ha usado concretamente el conector JDBC de MySQL para poder conectarnos a la base de datos desde el servicio web REST y así poder realizar todas las operaciones requerida por los clientes, relacionadas con los datos sobre usuarios, sesiones o ejercicios.

2.1.7 YouTube Android API

YouTube es un sitio web dedicado a compartir vídeos. Presenta una variedad de clips de películas, programas de televisión y vídeos musicales, así como contenidos amateur como videoblogs. Dispone de distintas APIs a disposición de desarrolladores para implementar distintas utilidades de su plataforma.

Figura 2-8 Logo gráfico de YouTube

Hemos usado las herramientas que nos proporciona esta API para implementar el reproductor de YouTube dentro de nuestra aplicación Android de manera más eficiente, al sólo necesitar la URL del vídeo en YouTube y así no tener que disponer de los vídeos guardados en nuestro propio servidor.

2.1.8 HTTPS

El protocolo HTTPS (Hypertext Transfer Protocol Secure) fue creado en 1992 por Netscape Communications para su navegador Netscape Navigator. Solamente era usado originalmente, para SSL, pero esto se volvió obsoleto ante la aparición de TLS. En el año 2000 HTTPS fue adoptado como un estándar web.

La definición de este término es un protocolo de aplicación que se basa en el protocolo HTTP, que está destinado a la transferencia segura de datos de hipertexto. O sea es la versión segura de HTTP. Entre sus características está la utilización de un cifrado establecido en SSL/TLS para crear un canal cifrado más adecuado para la transferencia de información sensible como son usuario y claves de paso, que el HTTP. El nivel de cifrado de este canal va a depender del servidor remoto y del navegador que utilice el cliente. A su vez, de esta manera, se consigue que esa información privada o sensible no pueda ser usada por un atacante que haya podido interceptar la transferencia de datos, porque lo que consigue son datos cifrados que no puede descifrar.

2.2 Herramientas

2.2.1 Spring Tool Suite

Spring Tool Suite (STS) proporciona un entorno ready-to-use para implementar, depurar, ejecutar y desplegar las aplicaciones Spring, incluyendo integraciones para Pivotal tc Server, Pivotal Cloud Foundry, Git, Maven, AspectJ, y viene encima de las últimas versiones de Eclipse.

También soporta el despliegue de aplicaciones tanto en servidores locales, virtuales y en la nube. Es de libre acceso para el desarrollo y uso en operaciones internas sin límite de tiempo, completamente de código abierto y licenciada bajo los términos de la Licencia Pública Eclipse.

Figura 2-9 Logo gráfico de Spring Tool Suite

STS ha sido el IDE principal para el desarrollo del proyecto, tanto para la creación del servicio web REST, como para la modificación de la aplicación Java del monitor Bibliografía. Ha sido muy útil sobre todo para el servicio web REST puesto que cuenta con sugerencias e información sobre que funciones usar o de que atributos disponen.

2.2.2 XAMPP

XAMPP es un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl. Te permite instalar de forma sencilla Apache en tu propio ordenador, sin importar tu sistema operativo (Linux, Windows, MAC o Solaris).

Figura 2-10 Logo gráfico de XAMPP

Se ha usado en el proyecto para administrar y ejecutar la base de datos de la aplicación, mediante la herramienta PHPMyAdmin incluida en dicha aplicación.

2.2.3 Android Studio

IDE desarrollado por Google para la creación de aplicaciones para el sistema operativo Android. Este entorno de desarrollo nos facilita 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.

Figura 2-11 Logo gráfico de Android Studio

En nuestro proyecto se ha empleado para realizar modificaciones sobre la aplicación ya realizada con anterioridad Bibliografía.

3 SERVICIO WEB

"Hazlo o no lo hagas, pero no lo intentes."

- Yoda en Star Wars -

El servicio web RESTful desarrollado para comunicar las aplicaciones Java del monitor y Android del cliente con la base de datos era el objetivo principal que nos marcamos al inicio del proyecto. A este propósito se le añade uno secundario pero de vital importancia, la seguridad.

A lo largo de este capítulo y sus tres secciones analizaremos en detalle la base de datos, el servicio web y para acabar, toda la seguridad que se haya implementado.

3.1 Base de datos

Partiendo de la base de datos del anterior proyecto [1], hemos mantenido la estructura principal y sólo hemos hecho cambios relacionados con la implementación de seguridad en el servicio web. Por lo tanto sólo profundizaremos en dichos cambios y simplemente haremos un resumen del resto de la base de datos.

Podemos estructurar la base de datos en tres grandes pilares que son:

- Usuarios del sistema que se inscriben en sesiones y ahora además cuentan con roles.
- Sesiones de monitor y de usuario, que cuentan con una serie de ejercicios.
- Ejercicios e información detallada sobre ellos.

En la figura 3-1 se puede observar el modelo entidad-relación de nuestra base de datos completa.

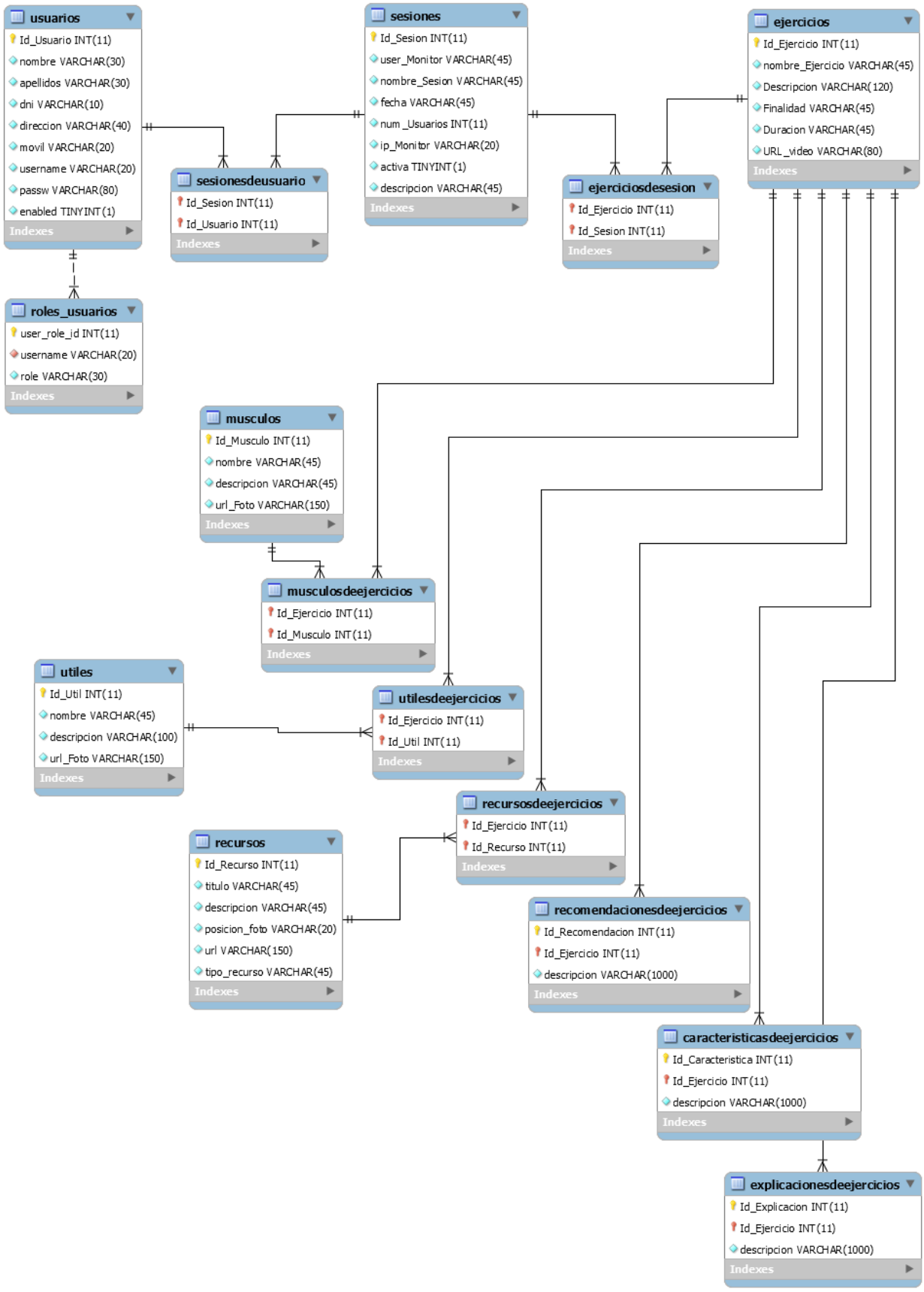

Figura 3-1 Base de datos: modelo entidad-relación.

Por nuestra parte hemos añadido un nuevo campo de carácter Booleano a la tabla *usuarios* llamado *enable*, que indica si la cuenta del usuario está activada y puede acceder a los recursos o si ha sido bloqueada por algún motivo.

El campo *username* de la tabla *usuarios* ha sido convertido en clave única, puesto que el nombre de usuario no debe repetirse. La clave externa de la tabla *roles_usuarios* apunta hacia el campo *username*.

El último cambio hecho en la tabla *usuarios* ha sido el de ampliar la longitud del campo *passw*. Ahora mide más de 60, puesto que las contraseñas codificadas van a necesitar mínimo de esa longitud.

También se ha añadido una tabla nueva llamada *roles_usuarios* que tenemos representada en la figura 3-2 junto a la tabla *usuarios*. En ella se guardan los roles o niveles de autorización de cada usuario. El servicio web consultará está tabla para comprobar qué usuarios pueden acceder a determinados recursos.

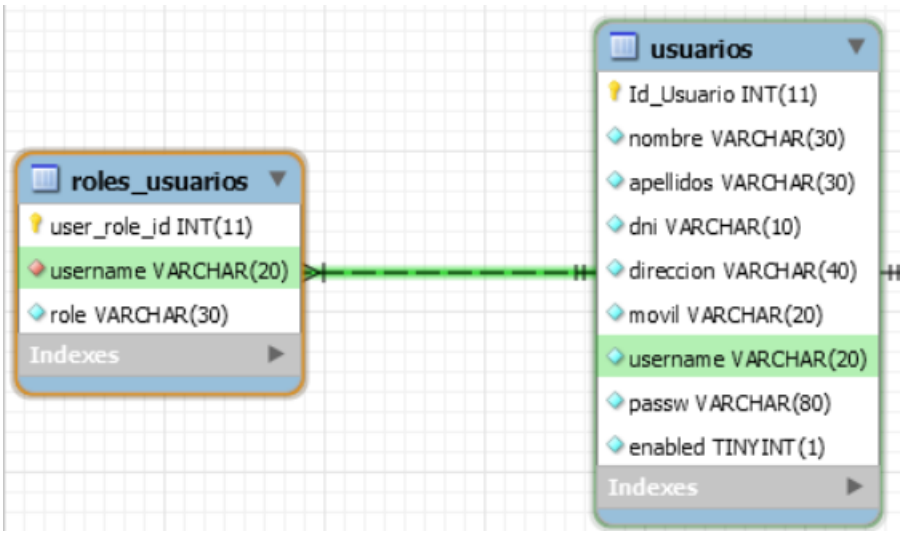

Figura 3-2 Tablas para cumplir con los objetivos

3.2 Servicio web desarrollado

Siguiendo los principios REST explicados en la sección homónima del capítulo 2 de la memoria, se ha construido un servicio web con el objetivo de disponer de una comunicación eficiente y segura entre las aplicaciones Java/Android y la base de datos. En primer lugar vamos a comentar cómo es la comunicación con las aplicaciones de usuario para luego continuar analizando la estructura del código del servicio web apoyándonos en varios diagramas de clases.

El funcionamiento de la comunicación con el servicio web es el siguiente:

- Las aplicaciones de los usuarios envían un mensaje HTTPS hacia nuestro servicio web, indicando la URI donde se encuentra el recurso que se va a consumir. También indicarán el tipo de petición que quieren llevar a cabo (GET, POST o DELETE), puesto que hay URIs idénticas pero según el tipo de petición se realizan operaciones radicalmente distintas. Se pueden enviar parámetros tanto directamente en la URI, como a través de la petición. La información es mandada en un objeto JSON convertido a cadena de texto que el servicio interpretará
- El controlador del servicio web, que está constantemente mapeando, identifica por medio de la URI el recurso que se va a utilizar. Además de los parámetros en caso de que los haya.

- A continuación, el controlador se pone en contacto con las clases DAO encargadas de la comunicación con la base de datos para solicitarles que lleven a cabo una operación determinada. Cuando el controlador recibe la información de la base de datos, que puede ser un objeto, una lista de objetos o una cadena de texto, lo que hace es codificarlo en JSON y seguidamente enviar la respuesta hacia la aplicación que la haya solicitado.
- Por último, la aplicación se encarga de procesar la respuesta JSON adecuadamente.

Ejemplos del funcionamiento de la comunicación de todas las partes del sistema pueden verse en los diagramas de secuencia (figuras 3-12 y 3-13) que se encuentran después de la sección de seguridad.

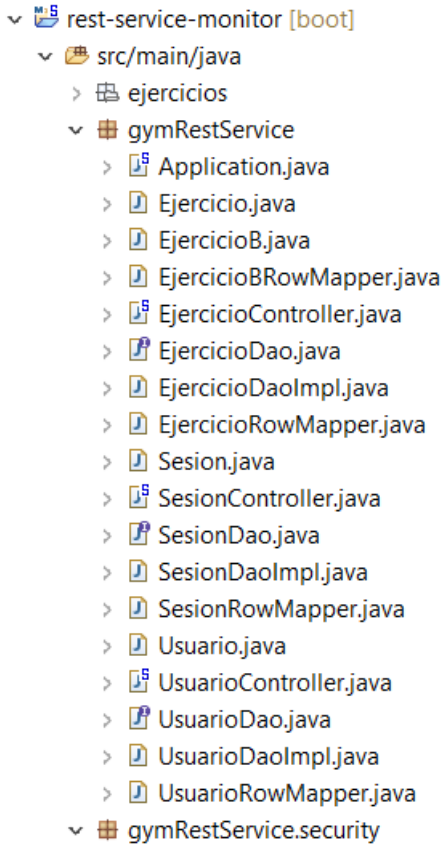

Figura 3-3 Agrupación de clases java de nuestro servicio web

El siguiente paso es adentrarnos en la estructura de nuestro servicio web. Si observamos la figura 3-3 nos damos cuenta de que el servicio se divide en tres partes casi idénticas que corresponden a los tres grandes bloques de información de nuestra base de datos: usuarios, sesiones y ejercicios. Por lo que desarrollaremos en profundidad una de ellas y luego comentaremos las pequeñas diferencias, si es que las hay, con las otras dos partes.

3.2.1 Usuarios

Las clases de las que vamos a hablar a continuación pueden ser observadas en la figura 3-4. Comenzaremos por la clase *Usuario*, que es la que se encarga de representar a un usuario en la aplicación. En ella están todos los campos a los que se puede consultar en la tabla *usuarios* de la base de datos.

Seguimos con la clase *UsuarioController*, que representa un controlador REST de usuarios. Se encarga de mapear las operaciones sobre recursos REST relacionados con usuarios y hace uso del DAO (Data Acces Object) para hacer efectivas dichas operaciones en la base de datos. El DAO se obtiene por inyección de dependencias. Las URIs que mapea este controlador y los métodos a los que llama son los siguientes:

- **/checkUser/**: petición POST. Llama al método *checkUser*, el cual se encarga de comprobar si un usuario ha sido ya dado de alta previamente.
- **/gymRegister/**: petición POST. Llama al método *addUser*, encargado de registrar un nuevo usuario en la base de datos.
- **/gym/login/**: petición POST. Llama al método *login*, que tiene la misión de cotejar que la información de acceso (username y password) es correcta para así entrar en el sistema.
- **/gym/addUserSesion/**: petición POST. Llama al método *addUsuarioSesion*, que simplemente añade un usuario a una determinada sesión.

La clase *UsuarioDao* representa la interfaz DAO de usuarios. Abstrae las operaciones necesarias sobre la base de datos relacionadas con los usuarios a petición del controlador REST de usuarios. Por otro lado, la clase *UsuarioDaoImpl* implementa la interfaz DAO de usuarios. Realiza las operaciones necesarias sobre la base de datos relacionadas con los usuarios a petición del controlador REST de usuarios. Los métodos de esta clase y su labor han sido comentados previamente al tratar las operaciones del controlador con las que se corresponden.

Por último, queda hablar de la clase *UsuarioRowMapper*, que representa un mapeador de usuarios. Obtiene y crea un objeto *Usuario* a partir de los datos obtenidos de la base de datos. O lo que es lo mismo, al hacer una llamada a la base de datos e invocar a la clase RowMapper, se introduce cada campo leído de la base de datos en el campo correspondiente de un objeto de tipo *Usuario*.

3.2.2 Sesiones

Las clases y métodos que vamos a comentar pueden observarse en la figura 3-5. No nos vamos a detener en las clases *Sesion, SesionDao, SesionDaoImpl* y *SesionRowMapper* porque funcionan de igual manera que sus homólogas del apartado Usuarios. Vamos a centrarnos en las URIs que mapea la clase *SesionController* y en sus operaciones:

- **/gym/sesionAdd/**: petición POST. Llama al método *addSesion*, el cual añade una nueva sesión a la base de datos.
- **/gym/sesionesActivas/**: petición GET. Llama al método *getSesionesActivas*, que se encarga de extraer de la base de datos una lista con todas las sesiones activas.
- **/gym/sesion/{id_sesion}/**: petición GET. Llama al método *getSesion*, que es el encargado de obtener toda la información sobre una sesión.
- **/gym/sesion/{id_sesion}/**: petición DELETE. Llama al método *deleteSesion*, que tal y como su nombre indica, elimina la sesión deseada.
- **/gym/sesion/**: petición POST. Llama al método *updateSesion*, el cual modifica determinados campos de información sobre una sesión.
- **/gym/sesionMonitor/{monitor}/**: petición GET. Llama al método *getSesionMonitor* para obtener sólo las sesiones asociadas a un determinado monitor.
- **/gym/sesionMonitor/**: petición POST. Llama al método *updateSesion*, que tiene la misión de actualizar la IP de un monitor en una sesión para que luego la aplicación Android acceda a ella y se pueda establecer la conexión entre ambas aplicaciones.

3.2.3 Ejercicios

Las clases y métodos que vamos a comentar pueden observarse en la figura 3-6. No vamos a detenernos en las clases *Ejercicio, EjercioDao, EjercicioDaoImpl* y *EjercicioRowMapper* porque funcionan de igual manera que sus homólogas del apartado Usuarios. Sin embargo, debemos mencionar que para la gestión de ejercicios se han diseñado dos clases más, *EjercicioB* y *EjercicioBRowMapper*. Dichas clases representan y mapean a un tipo distinto de ejercicio, formado por distintas tablas de la base de datos. Es decir, *EjercicioRowMapper* se encarga de obtener la información la información de la tabla *ejercicios* y de introducirla en un objeto de tipo *Ejercicio*, mientras que *EjercicioBRowMapper* obtiene los datos de las tablas *ejercicios* y *sesiones* y los introduce en un objeto de tipo *EjercicioB*.

Tal y como hicimos en los apartados anteriores, vamos a centrarnos en las URIs que mapea la clase *EjercicioController* y en sus operaciones:

- **/gym/ejercicio/{id_ejercicio}/**: petición GET. Llama al método *getEjercicio*, el cual obtiene la información relativa a un usuario en base de datos.
- **/gym/ejercicio//**: petición GET. Llama al método *getAllEjercicios*, que se encarga de extraer de la base de datos una lista con todos los ejercicios.
- **/gym/ejercicio/{id_sesion}/{id_ejercicio}/**: petición POST. Llama al método *addEjercicioSesion*, el cual añade un ejercicio a una determinada sesión.
- **/gym/ejercicio/{id_sesion}/{id_ejercicio}/**: petición DELETE. Llama al método *deleteEjercicio* para eliminar un ejercicio de una sesión específica.
- **/gym/ejercicioNoSesion/{id_sesion}/**: petición GET. Llama al método *getEjercicioNoSesion*, que es el encargado de obtener todos los ejercicios que no pertenecen a una sesión concreta.
- **/gym/ejercicioSesion/{id_sesion}/** petición GET. Llama al método *getEjercicioSesion*, que tal y como su nombre indica, obtiene todos los ejercicios pertenecientes a una determinada sesión.

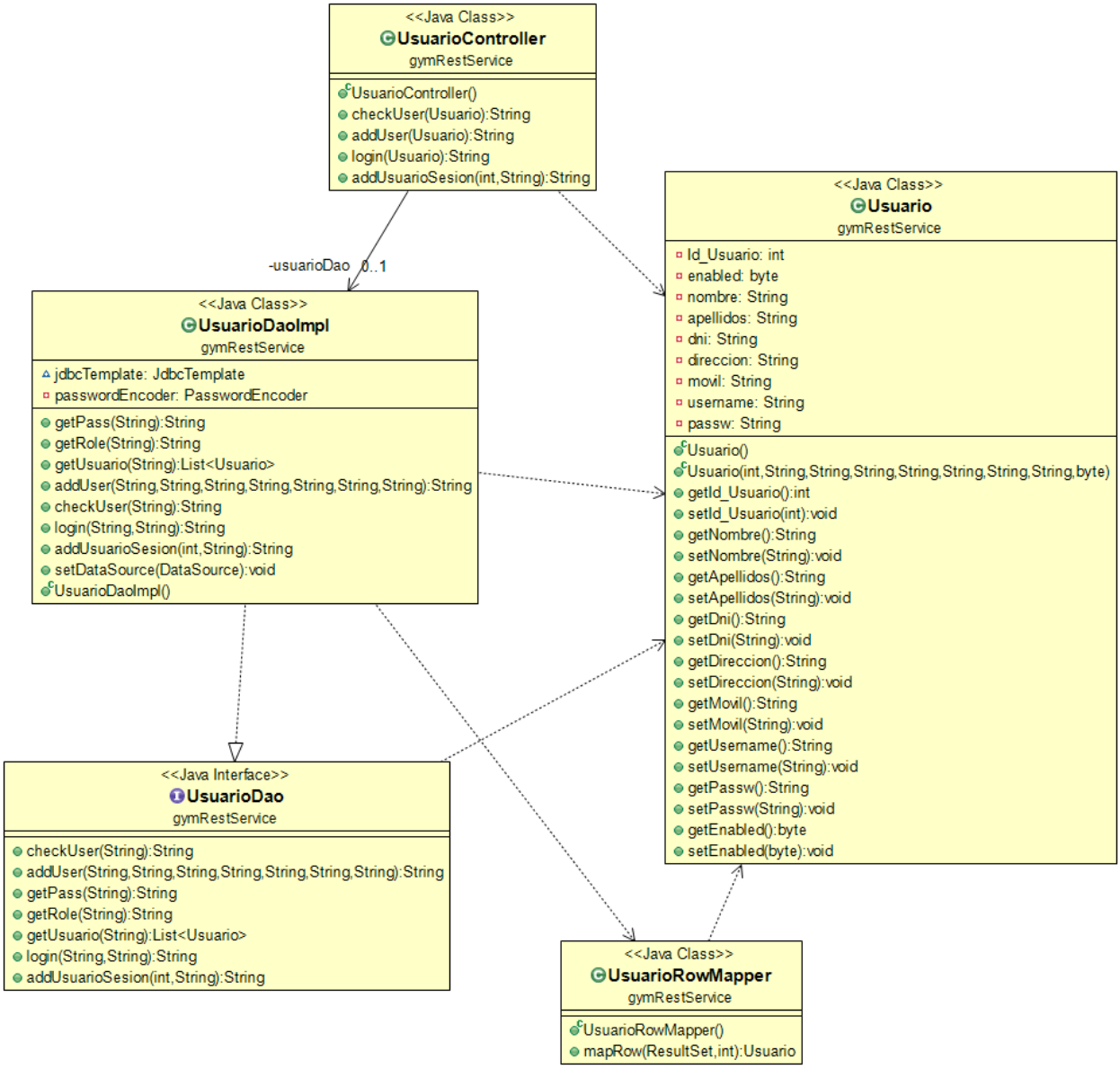

Figura 3-4 Diagrama de clases de gestión de usuarios

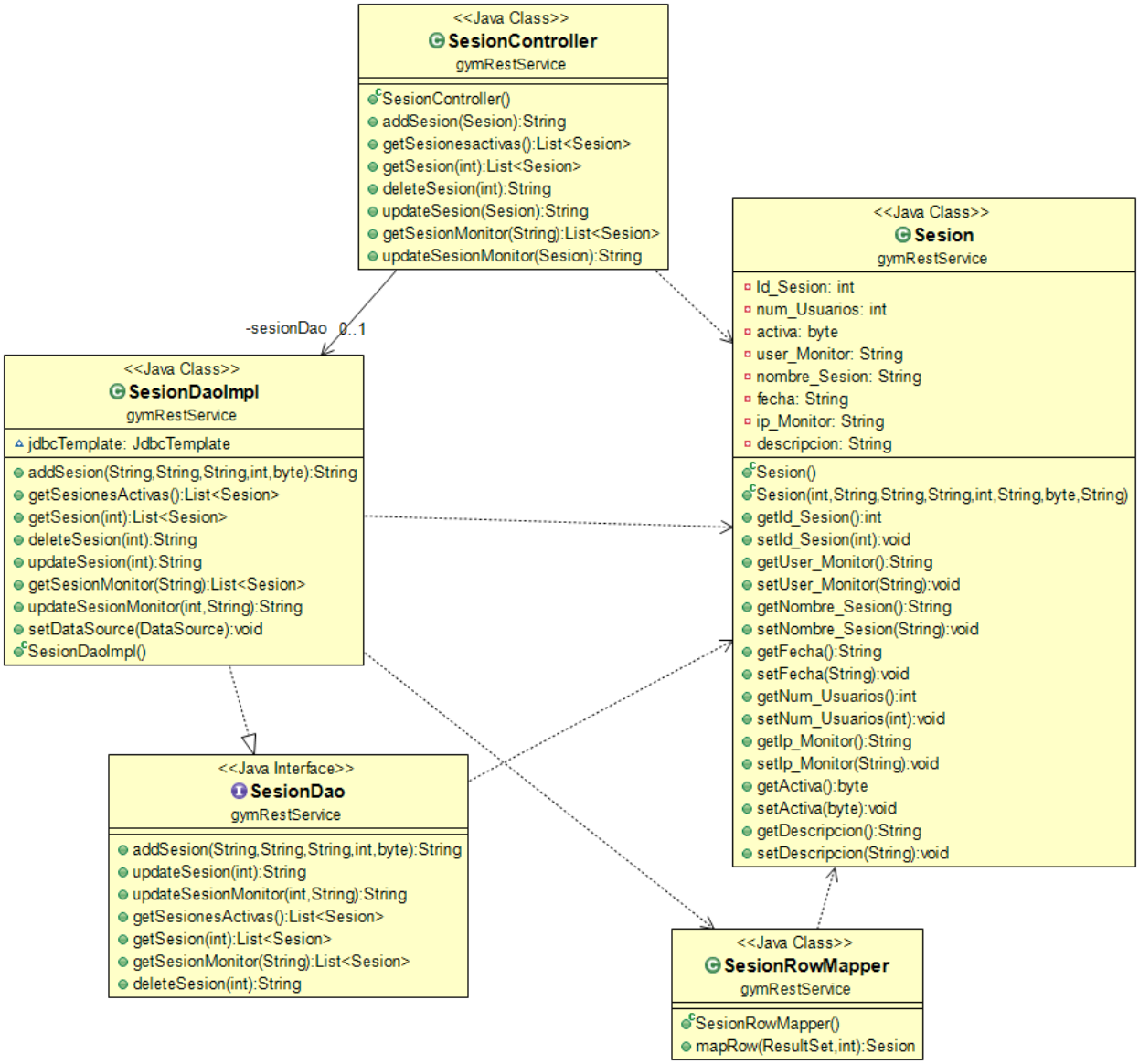

Figura 3-5 Diagrama de clases de gestión de sesiones

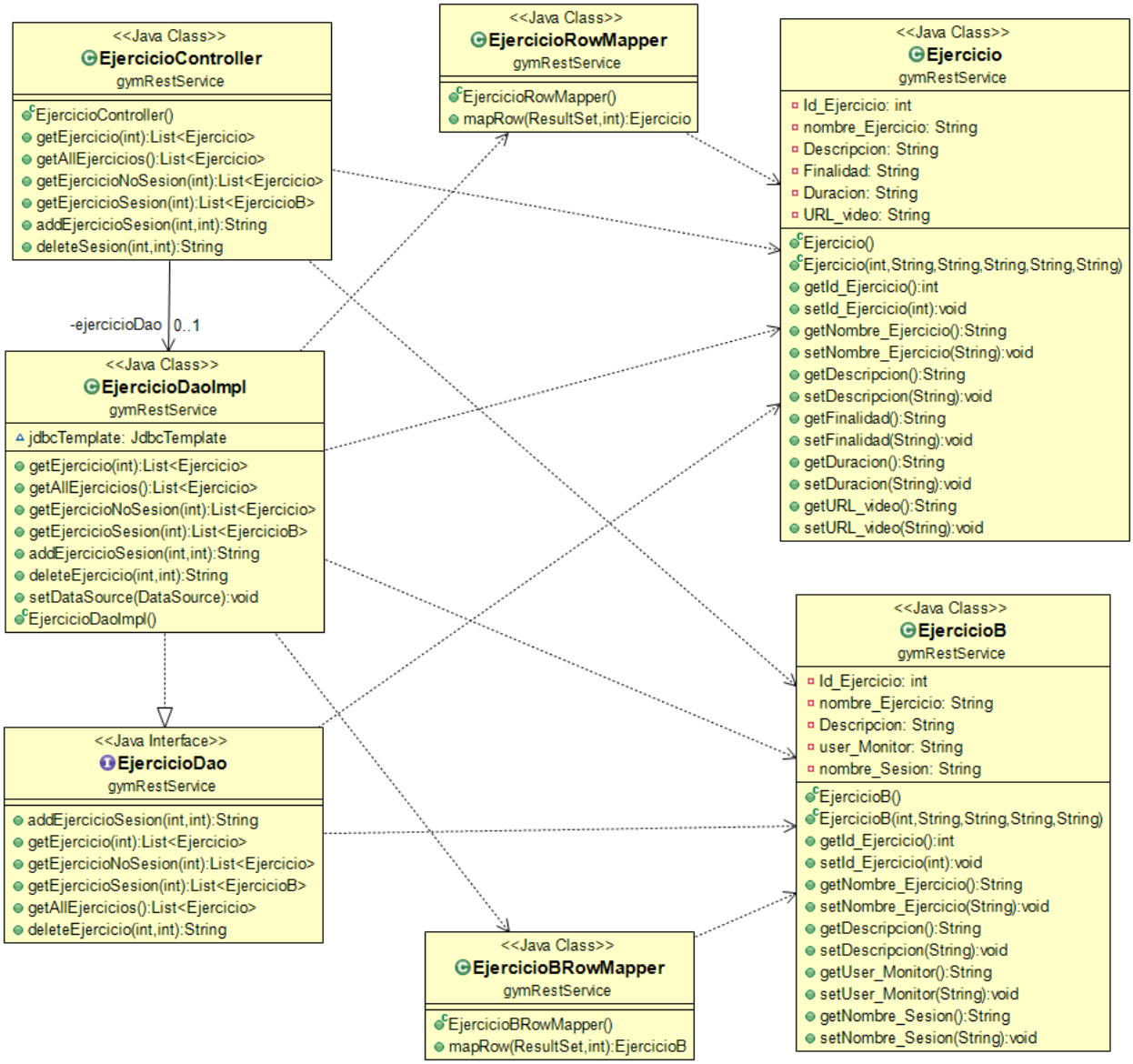

Figura 3-6 Diagrama de clases de gestión de ejercicios

3.3 Seguridad implementada

La seguridad informática es una disciplina que se encarga de proteger la integridad y la privacidad de la información almacenada en un sistema informático. También se ocupa de diseñar los procedimientos, métodos y técnicas, orientados a proveer condiciones seguras y confiables, para el procesamiento de datos en sistemas informáticos. Las cuatro áreas principales que cubre la seguridad informática son:

- Confidencialidad: Sólo los usuarios autorizados pueden acceder a nuestros recursos.
- Integridad: Sólo los usuarios autorizados deben ser capaces de modificar los datos cuando sea necesario.
- Disponibilidad: Los datos deben estar disponibles para los usuarios cuando sea necesario.
- Autenticación: Estás comunicándote con los que piensas que te estás comunicando.

Uno de los objetivos que nos marcamos al principio del proyecto fue, no sólo conseguir desarrollar un servicio web RESTful, sino implementar también distintas medidas de seguridad en él a través de Spring Security. Primero, hemos usado HTTPS para el intercambio de información con las aplicaciones. Segundo, hemos conseguido cubrir la autenticación haciendo uso del protocolo estándar OAuth2 y hemos resuelto también la confidencialidad e integridad al crear un sistema de roles de usuarios mediante el cual, según el nivel de autorización que se tenga se pueden acceder a determinados recursos y a otros no. Por último, se ha añadido codificación a las contraseñas almacenadas en la base de datos. A continuación, procederemos a comentar en detalle cada una de las implementaciones.

3.3.1 Autentificación mediante OAuth2

Tal y como explicamos en el capítulo 2 (Tecnologías y herramientas), OAuth2 es un protocolo de autorización que permite a terceros, clientes o sistemas, acceder a contenidos propiedad de un usuario, alojados en un servidor de recursos, sin tener que compartir las contraseñas. La ventaja de utilizar un mecanismo de seguridad como OAuth2 es que facilita la comunicación entre el usuario y una aplicación tercera y genera más confianza al no tener que ingresar sus credenciales en la misma y es más segura la interacción para el usuario.

Además, expusimos los roles que intervienen en el procedimiento de autorización:

- **Resource Owner**: La entidad que da acceso a recursos protegidos. En nuestro caso, este rol le pertenece a los usuarios de las aplicaciones Java y Android (el monitor y los entrenados).
- **Cliente**: Es la aplicación que hace peticiones a recursos protegidos, con la autorización del usuario que es propietario de ellos. En nuestro sistema, tanto la aplicación Java como la aplicación Android hacen las veces de cliente.
- **Authorization Server**: Es el encargado de generar los Tokens de Acceso para nuestra aplicación. Este servidor de autorización ha sido implementado en nuestro servicio web.
- **Resource Server**: Es el servidor o sistema que nos da acceso a los recursos protegidos. Es capaz de aceptar y responder peticiones usando un Token de Acceso, vía URL. Este servidor es la parte de nuestro servicio web en sí, que realiza las operaciones sobre la base de datos.

Respecto a los tipos de autorización, establecimos que usaremos el llamado *Resource Owner Password Credentials*. En este modo de autorización, el Resource Owner envía las credenciales al Cliente y éste, al Authorization Server. Por lo tanto, sólo se utiliza cuando hay completa confianza entre el Cliente y el Authorization Server, por ejemplo si ambos han sido desarrollados por la misma autoridad (nosotros). En la figura 3-7 podemos observar un diagrama de secuencia en el que se aprecia claramente el funcionamiento. Si lo que se pretende es ver el funcionamiento dentro del sistema completo, basta con dirigirse al diagrama de secuencia de la figura 3-12 en el apartado 3.4 (Diagramas de secuencia).

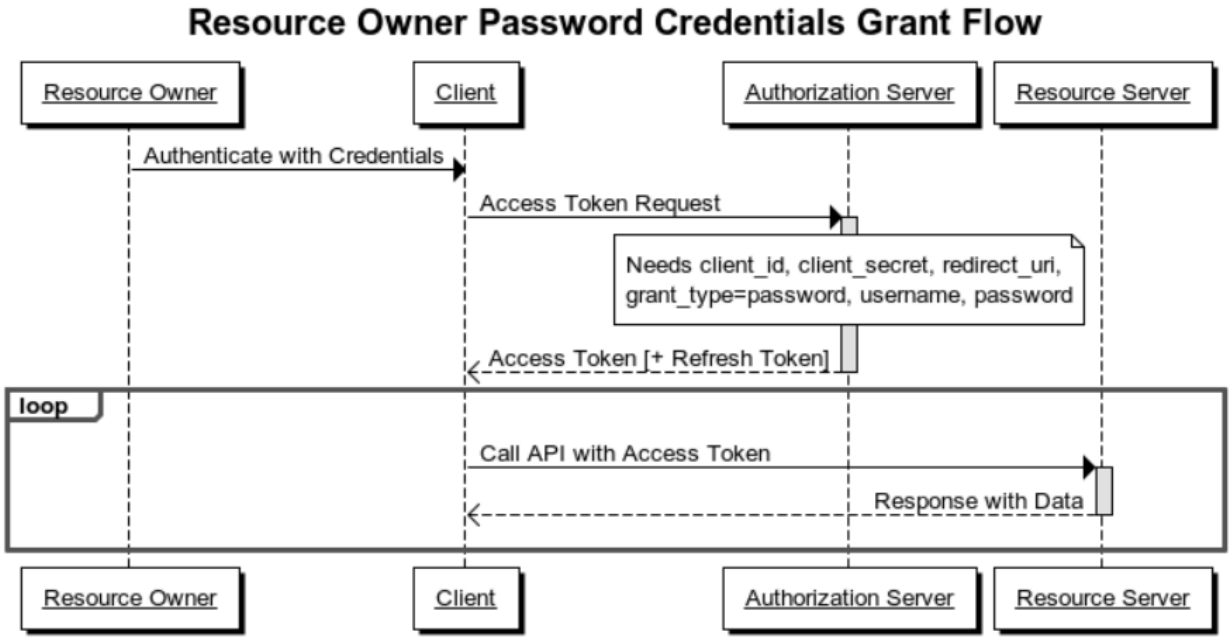

Figura 3-7 Diagrama de secuencia del funcionamiento de OAuth2

A continuación haremos un repaso por las clases de nuestro servicio que tienen alguna parte en la labor de la seguridad. Empezaremos por la clase principal que se encarga de implementar toda la seguridad de la aplicación, *OAuth2SecurityConfiguration* (figura 3-8). En ella hacemos labores de todo tipo:

- Declaramos beans para el almacén en el que guardaremos los Tokens (*tokenStore*), para los gestores de los tokens (*userApprovalHandler y approvalStore*), para el autenticador de usuarios a partir de la información de la base de datos (*DaoAuthenticationProvider*) y para el codificador que usaremos para proteger las contraseñas posteriormente (*passwordEncoder*).
- En *configure* configuramos el endpoint '*/oauth/token'* para poder solicitar un Token de Acceso.
- Dentro de *globalUserDetails* configuramos los propietarios de los recursos. Se accede a la base de datos a través del *DaoAuthenticationProvider* para adquirir la información de los usuarios propietarios de los recursos.

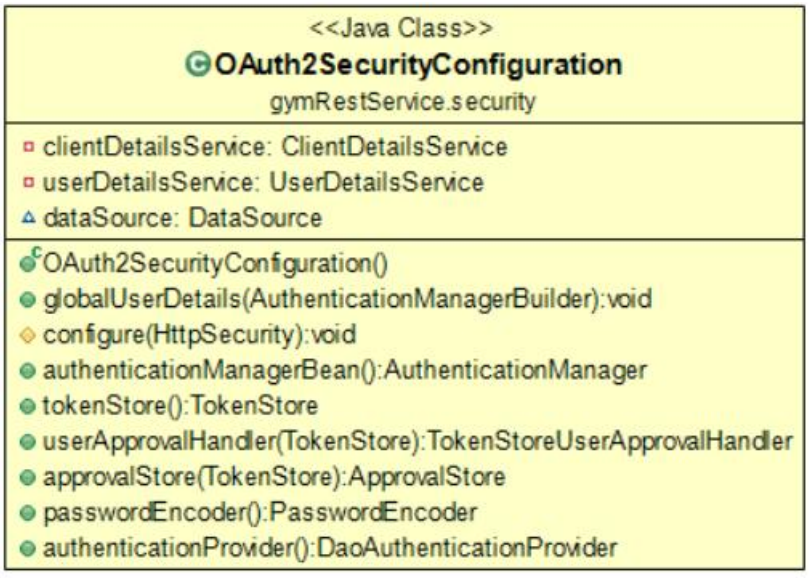

Figura 3-8 Clase *OAuth2SecurityConfiguration*

La siguiente clase es *AuthorizationServerConfiguration* (figura 3-9), responsable de verificar las credenciales y si éstas son correctas, seguidamente provee los Tokens de Acceso. Labores que se realizan:

- Se registran las credenciales con las que el cliente accederá para recibir el Token.
- Se establecen las condiciones del acceso, los permisos de lectura y/o escritura sobre los recursos y el tiempo que durarán activos los Tokens que se manden a los clientes.

Figura 3-9 Clase *AuthorizationServerConfiguration*

Continuamos con la clase *ResourceServerConfiguration* (figura 3-10), que hospeda los recursos (nuestro servicio web REST) en los que el cliente (aplicaciones Java y Android) está interesado. Tareas que se llevan a cabo.

- Activamos un filtro de Spring Security para que autentifique las peticiones a nuestro servicio web usando un Token OAuth2 entrante, el cual ha debido ser solicitado previamente por el cliente al *AuthorizationServer*.
- Dentro de *configure* permitimos el acceso libre a la URI *'/gymRegister'* para que los usuarios puedan registrarse en la aplicación. Al mismo tiempo indicamos que el resto de recursos de nuestro servicio web quedan restringidos a no ser que dispongan de un Token de acceso.

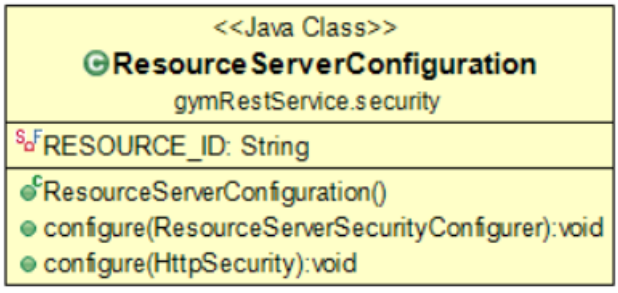

Figura 3-10 Clase *ResourceServerConfiguration*

Por último, comentar la clase *CustomUserDetailsService* (figura 3-11), la cual obtiene información sobre un usuario de la base de datos para luego ser usada para la autenticación. Consta de dos métodos, uno se encarga de cargar la información del usuario y el otro de cargar el rol de autorización del que dispone dicho usuario.

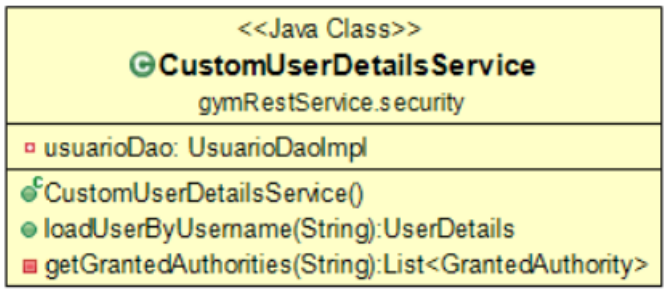

Figura 3-11 Clase *CustomUserDetailsService*

3.3.2 Autorización y roles

Una vez el usuario ya está autenticado en la aplicación y tiene acceso al servicio web, el siguiente paso es lograr que ningún usuario acceda a recursos que no le correspondan o modifique un recurso al cual sólo debería tener acceso para leer. Para ello, Spring Security tiene una lista de roles definidos (user y admin) con los cuales podemos asignar el acceso a recursos concretos según el rol del que disponga el usuario.

En la clase *ResourceServerConfiguration* hemos configurado qué recursos pueden ser accedidos libremente y cuales necesitan un rol determinado. También a la hora de autenticar al cliente, hemos designado que este disponga de los permisos tanto para leer como para escribir.

Además, si necesitáramos más y variados roles para nuestro servicio, Spring nos permite crear nuestros propios roles personalizados con los permisos que más nos convengan.

3.3.3 Codificación de contraseñas

Ningún sistema que piense medianamente en la seguridad puede permitirse el lujo de almacenar la información sensible de sus usuarios (como las contraseñas) en simple formato de texto plano. Tras analizar las distintas opciones que nos ofrecía Spring Security, nos hemos decidido por usar la función Bcrypt.

Bcrypt es una función de hashing de contraseñas diseñado por Niels Provos y David Maxieres, basado en el cifrado de Blowfish. Lleva incorporado un valor llamado *salt*, que es un fragmento aleatorio que se usará para generar el hash asociado a la contraseña, y se guardará junto con ella en la base de datos. Así se evita que dos contraseñas iguales generen el mismo hash y los problemas que ello conlleva, por ejemplo, ataque por fuerza bruta a todas las contraseñas del sistema a la vez.

En nuestro proyecto hemos implementado el *BCryptPasswordEncoder* en dos situaciones:

- A la hora de registrar un nuevo usuario, para generar una contraseña codificada a partir de la original y guardarla en la base de datos.
- Al hacer login en el sistema, se usa el método que compara la contraseña introducida por el usuario con la contraseña codificada de la base de datos para verificar que es la misma.

Cuanto mayor sea el valor del parámetro *salt*, más compleja será la contraseña. Nosotros hemos usado el valor 12 teniendo en cuenta tanto la eficiencia como la seguridad. Por ejemplo, un valor de 10 o menos no se considera demasiado seguro, y por otro lado, un valor de 14 o más requeriría un tiempo de espera excesivo tanto para crear una nueva contraseña como para verificarla.

3.4 Diagramas de Secuencia

Hemos creado varios diagramas de secuencia para mostrar el funcionamiento global del servicio web desarrollado. Pero antes de nada, identificaremos los elementos de los diagramas:

- **Usuario (Resource Owner):** Representa al usuario que utiliza la aplicación Java o Android.
- **Aplicación (Client):** Es la aplicación Java de monitor o la aplicación Android de usuario.
- **Servicio Web (Authorization Server):** Esta parte del servicio es la encargada de la autentificación mediante OAuth2 para poder acceder a los recursos del servicio. Este elemento sólo aparece durante autenticación de usuario, porque a partir de ese momento el usuario ya dispone del Token de Acceso con el que acceder al resto de recursos.
- **Servicio Web (Resource Server):** Es nuestra interfaz REST, que se encarga de realizar las operaciones sobre la base de datos. Se necesita tener un Token de Acceso para comunicarse con ella.
- **Base de datos:** Es el lugar donde se almacenan los datos relativos a información de usuarios, sesiones y ejercicios

Hay que aclarar que sólo elaboramos los diagramas de secuencia de mayor importancia en nuestro sistema, ya que el resto son variaciones, dependiendo de los datos que deseemos obtener o enviar al servicio web.

En el primer diagrama (figura 3-12) podemos observar cómo se realiza la autenticación de usuarios en la aplicación.

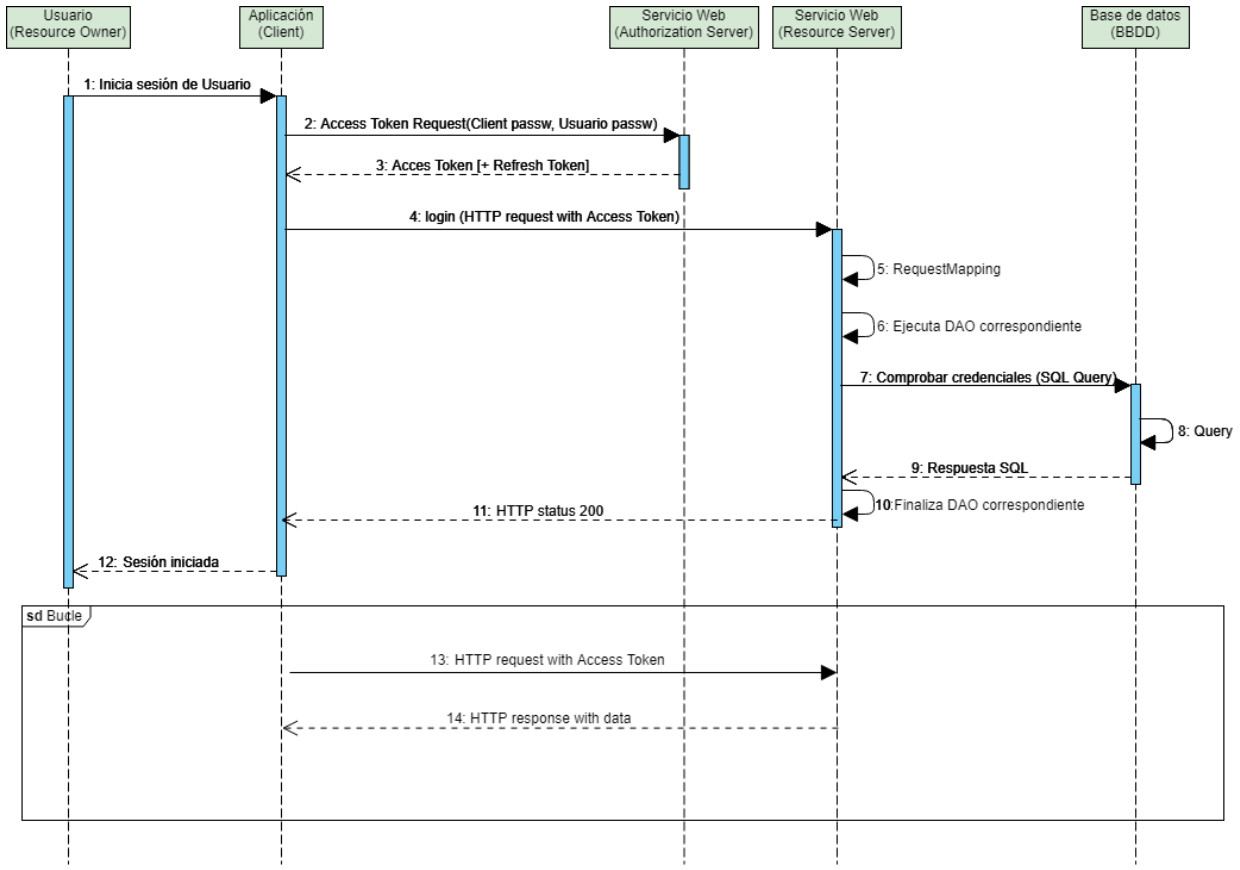

Figura 3-12 Diagrama de secuencia de autenticación de usuario.

En el siguiente diagrama se puede observar cómo obtener, introducir y eliminar datos en la aplicación. Tomaremos como ejemplo la edición de una sesión de ejercicios, en la que primero obtendremos los ejercicios, para posteriormente añadirlos a nuestra sesión o eliminarlos.

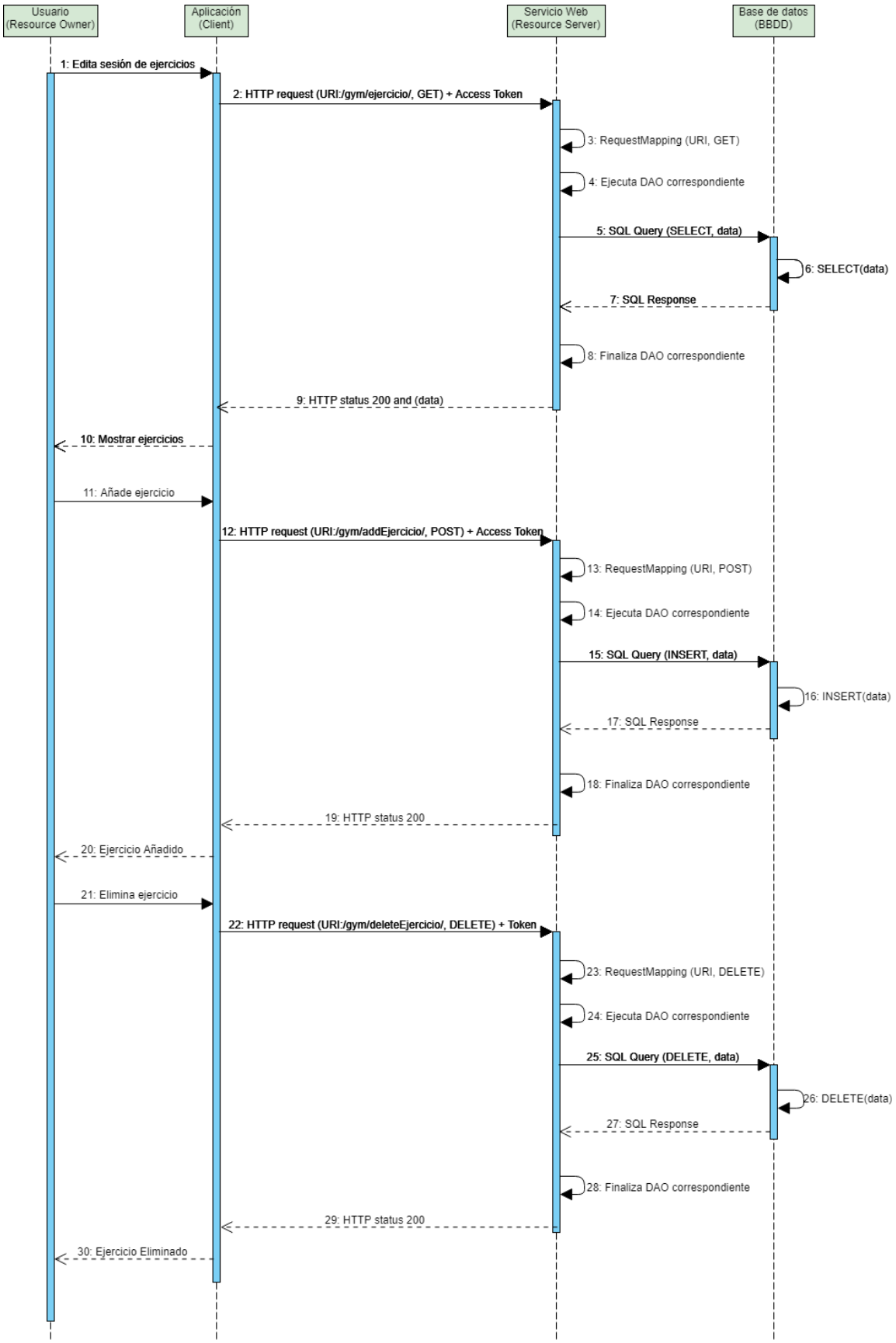

Figura 3-13 Diagrama de secuencia de la edición de una sesión.

4 APLICACIÓN JAVA

"Las máquinas deben funcionar y las personas, pensar."

-Principio de Pollyanna sobre IBM-

En este capítulo se expondrá la aplicación '***monitorgym'*** y profundizaremos en los cambios realizados en la misma respecto al proyecto de Pablo Gil Bibliografía. Con esta aplicación el monitor puede diseñar multitud de sesiones de entrenamiento y, una vez comenzada la sesión, podrá ver mediante streaming de vídeo como los distintos usuarios inscritos en ella realizan los ejercicios, pudiendo comunicarse por mensaje de texto en caso de que fuera necesario. Con esto, se llevaría a cabo una sesión de entrenamiento con el menos número de errores por parte del usuario gracias a las correcciones del monitor.

A continuación, detallaremos las modificaciones realizadas y el funcionamiento de la aplicación completa.

4.1 Gestor de peticiones

La primera gran modificación de la aplicación la encontramos en la clase *GestorPeticiones* (figura 4-1), la cual sirve para gestionar toda la comunicación con el servicio web. Al haber diseñado un nuevo servidor, hemos tenido que rediseñar también la forma en la que se comunicaba la aplicación Java con el servicio web REST.

Uno de los métodos principales de la clase es *sendTokenRequest*, el cual establece una conexión HTTPS con el servicio web en la dirección *'/oauth/token'*. En la misma URL se pasan como parámetros las credenciales del usuario, mientras que las credenciales del cliente van codificadas en base 64 en la cabecera HTTPS *'Authorization'*. Con esta información enviada con el método POST, el servicio web verifica todas las credenciales y nos envía de vuelta el Token de Acceso.

Tras recibir la respuesta del servicio web, se convierte la información recibida a un objeto JSON y a continuación se introduce en una instancia de la clase *AuthTokenInfo*. Esta clase que hemos creado representa a un Token en nuestra aplicación Java. En la figura 4-2 podemos observas sus campos. Para terminar, se devuelve el Token para que la aplicación pueda usarlo posteriormente en cada request que se haga al servicio web.

El segundo método principal del gestor de peticiones es *peticion*, el cual se encarga de enviar las peticiones (POST, GET, DELETE) al servidor, de leer la respuesta e interpretarla. Los parámetros que recibe son el token de acceso (necesario para elaborar cualquier comunicación con el servicio web), parámetros a enviar en la petición (si es que los hubiera) y el tipo de petición (POST, GET, DELETE). El envío del token se hace a través del esquema HTTP "Bearer authentication" en la cabecera "Authorization" de HTTP.

El resto de métodos son llamados por la aplicación Java para realizar una determinada labor relacionada con usuarios, sesiones o ejercicios (por ejemplo obtener sesiones, registrar nuevos usuarios, etc…). Estos métodos reciben los parámetros necesarios de la aplicación y llaman al método *peticion*, que es el que hace el mayor trabajo. Los parámetros son convertidos a cadenas de texto con formato JSON para que el servicio web sea capaz de leerlos.

Por ejemplo, supongamos que en la ventana *Sesiones* de la aplicación de monitor se hace click en una sesión determinada para ver toda la información relativa a ella. El programa llama al método *sesion* y le pasa los parámetros para que éste los transforme en cadenas JSON. Tras esto, el método *sesion* llama al método *peticion* pasándole el token, los parámetros (en este caso el id de la sesión) y el tipo de petición (GET en el

caso actual). Por último, el método *peticion* establece comunicación con el servicio web y se encarga de leer la respuesta JSON recibida.

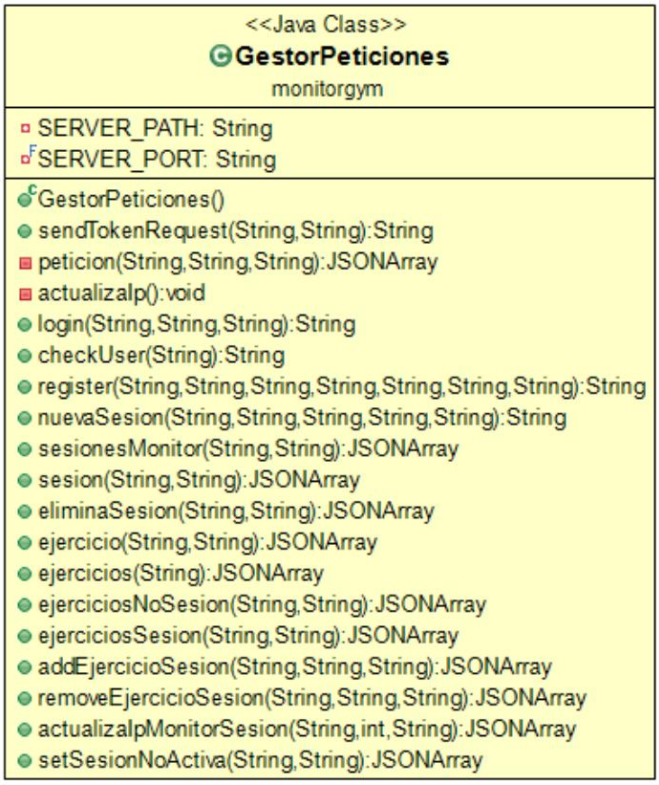

Figura 4-1 Clase *GestorPeticiones*

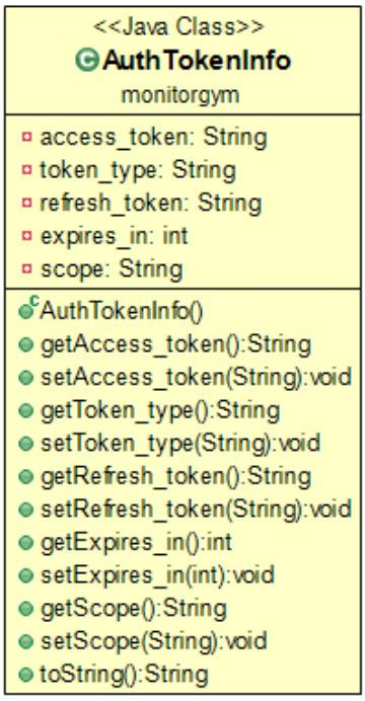

Figura 4-2 Clase *AuthTokenInfo*

4.2 Ventana de registro

La otra gran modificación implementada ha sido la ventana de *Registro*, a través de la cuál se puede inscribir a otros monitores o bien a nuevos usuarios para ser entrenados. Esta nueva función sólo está disponible desde la aplicación de monitor y no requiere estar autenticado para dar de alta a nuevos usuarios. Ahora la ventana *Login* dispone de un segundo botón abajo a la derecha llamado *REGISTRAR* (figura 4-3), que nos llevará a la ventana *Registro* para completar el formulario de registro (figura 4-4). Se ha mantenido el mismo diseño visual del resto de la aplicación.

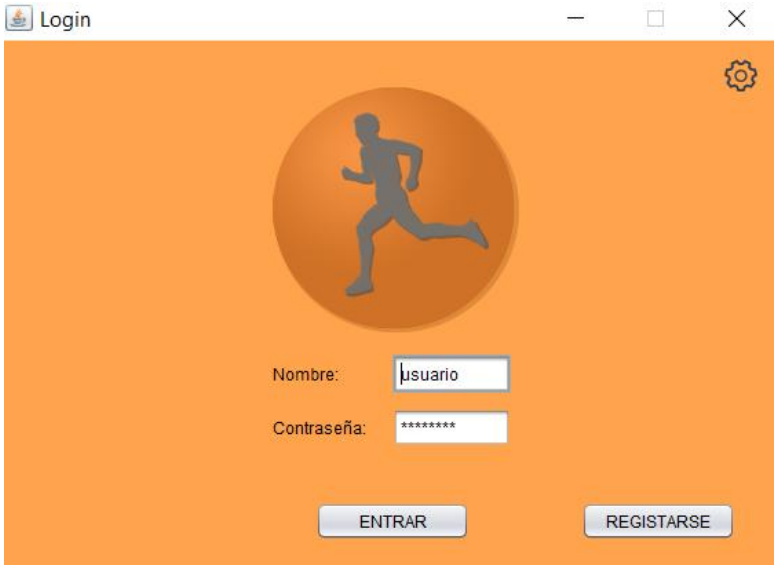

Figura 4-3 Ventana Login con el nuevo botón REGISTRAR

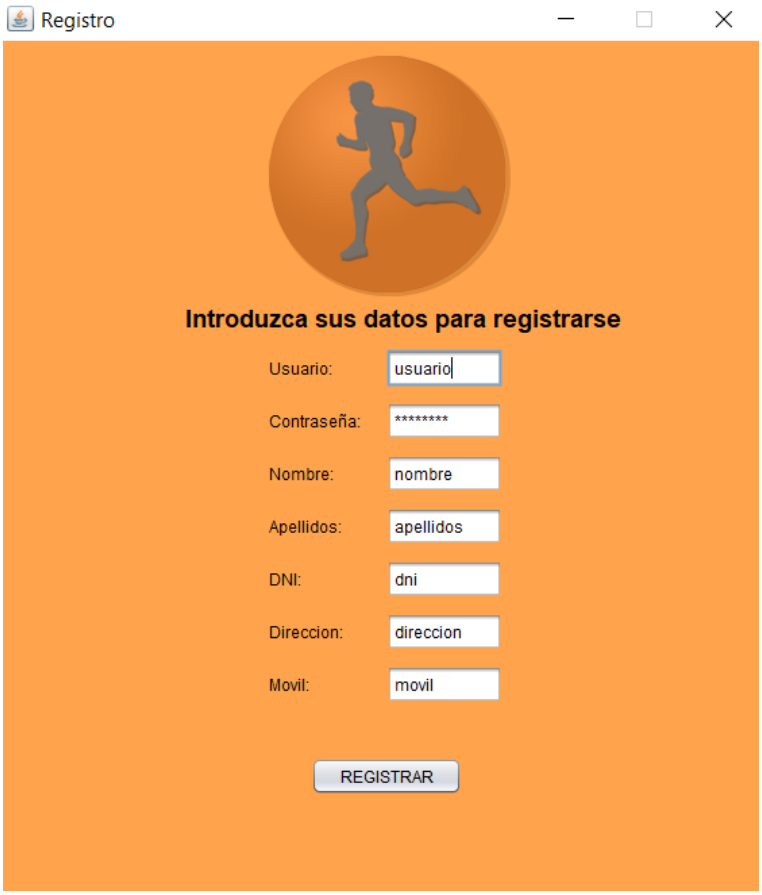

Figura 4-4 Ventana Registro

Al rellenar todos los campos y pulsar el botón *REGISTRAR*, se hace una petición al servidor para comprobar que efectivamente el nombre de usuario no está cogido ya. Si es así, se avisa por pantalla de que debe ingresar un nombre distinto (figura 4-5). En caso de que el registro se desarrolle sin problemas, aparece un nuevo botón con el que continuar a la ventana *Login* de nuevo para poder acceder a la aplicación (figura 4-6).

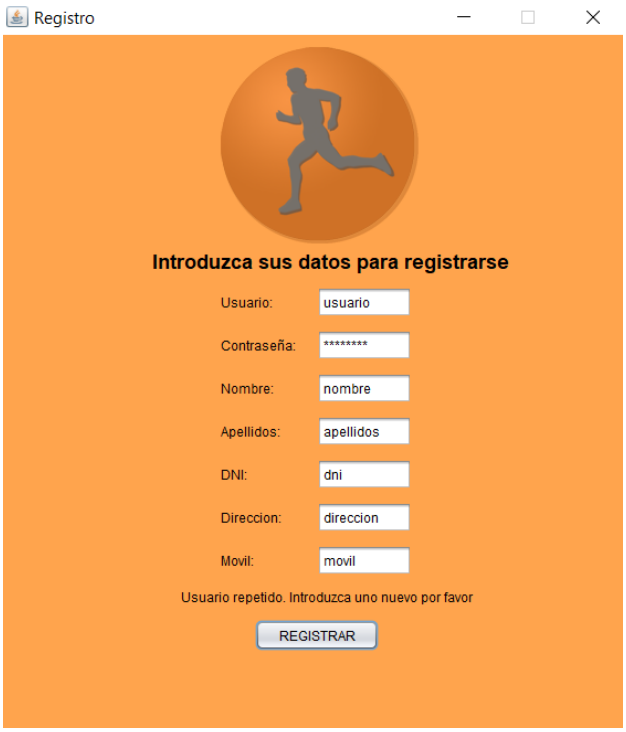

Figura 4-5 Ventana Registro en el caso de usuario repetido

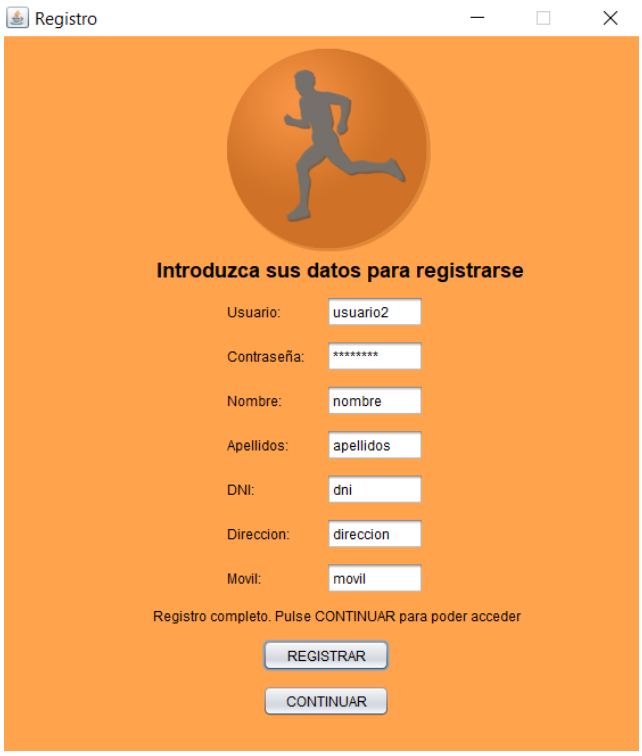

Figura 4-6 Ventana Registro en el caso de registro con éxito

4.3 Funcionamiento de la aplicación

En este apartado se explicará y se mostrará con capturas de pantalla como funciona la aplicación de monitor.

Al ejecutar la aplicación aparece la ventana *Login* (figura 4-7), en la cual se tienen dos opciones: registrarse en el sistema o entrar con las credenciales. La ventana *Registro* puede observarse en la figura 4-4. Antes de hacer login, se debe introducir la IP del servicio web para que la aplicación se comunique con él (figura 4-8).

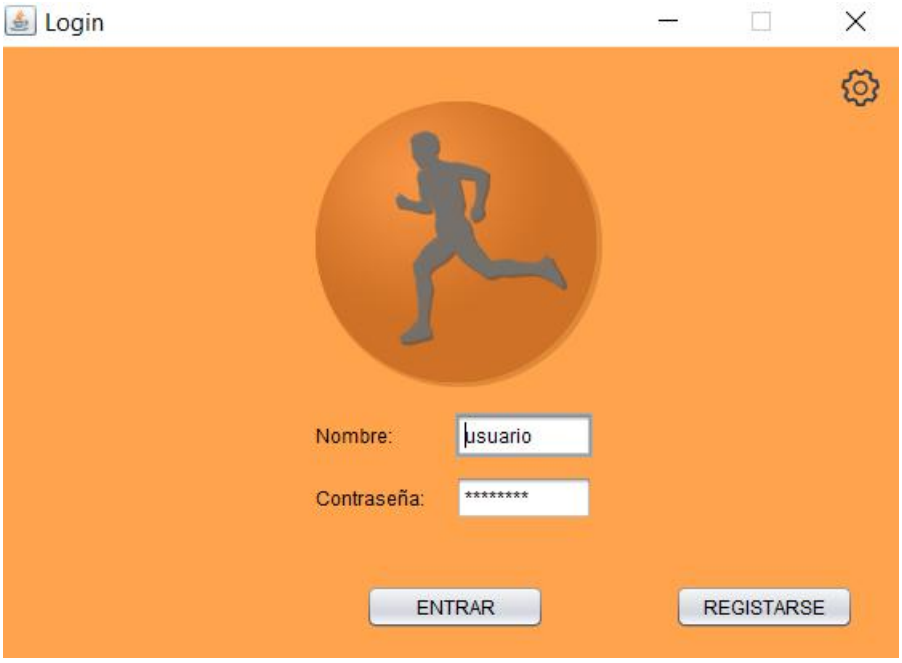

Figura 4-7 Ventana Login

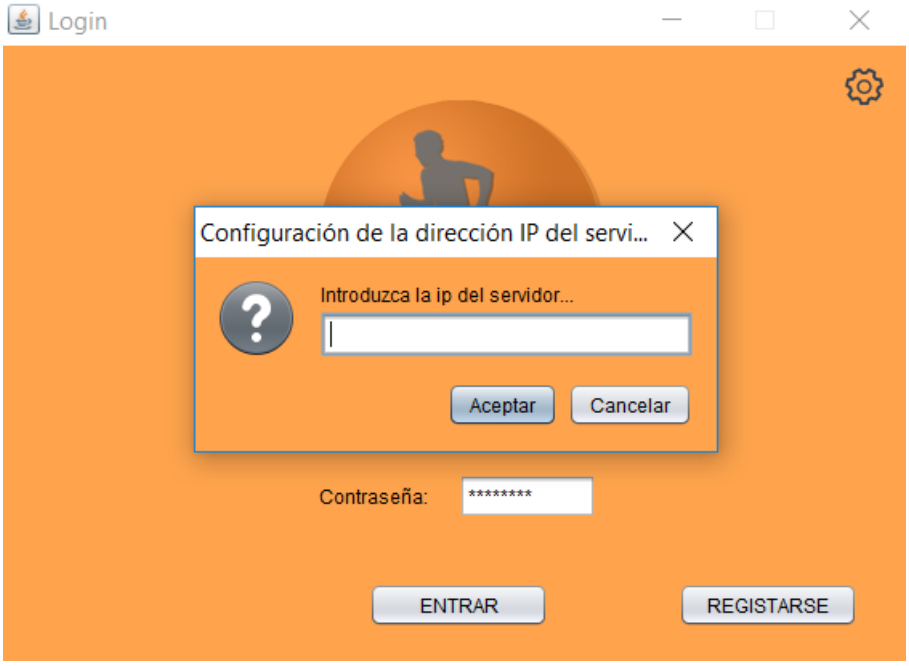

Figura 4-8 Ventana Configuración IP

Una vez estamos identificados en el sistema, nos aparece la ventana *Sesiones* (figura 4-9). En esta ventana podemos ver la lista de sesiones activas, sesiones no activas o todas las sesiones almacenadas en la base de datos. Si pulsamos el botón con la forma de una papelera, se puede eliminar una sesión del sistema. En cambio, si lo que queremos es crear una nueva sesión, hay que pulsar el símbolo +. Al crear una nueva sesión hay que seleccionar nombre, fecha, hora y número máximo de usuarios, y para guardarla en el sistema se debe pulsar el botón con forma de disquete (figura 4-10). Por último, para avanzar a la ventana *Ejercicios* se necesita seleccionar una sesión y a continuación hacer click en el icono del lápiz.

Figura 4-9 Ventana Sesiones

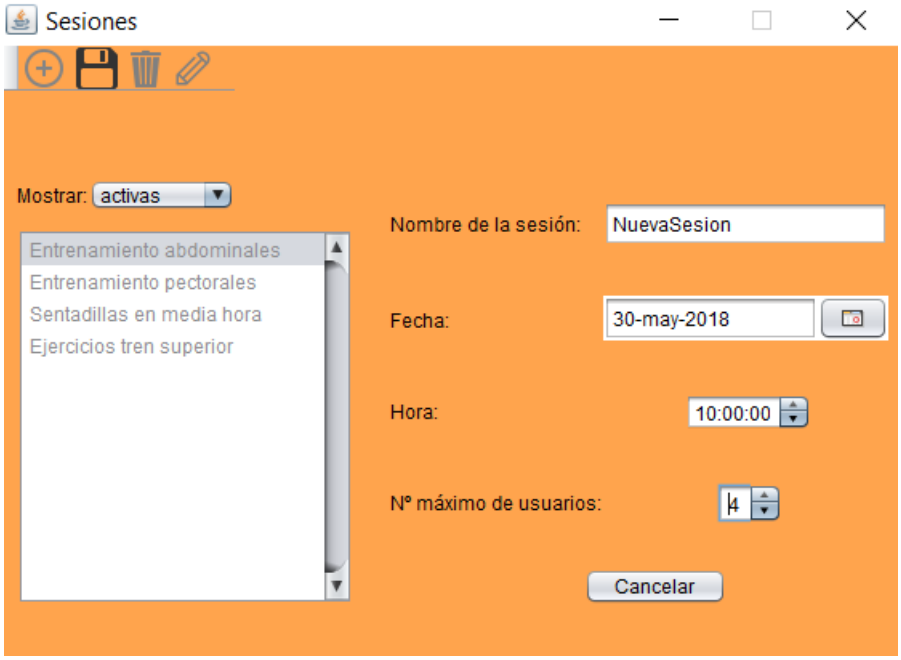

Figura 4-10 Ventana Sesiones creando nueva sesión

En esta última ventana, llamada *Ejercicios*, el monitor puede elegir que ejercicios van a formar parte de la sesión actual. Al hacer click sobre algún ejercicio aparece la información sobre el mismo en la esquina inferior izquierda. Desde el momento que se ha iniciado la sesión, los usuarios de la aplicación Android se pueden suscribir a la sesión y es en ese momento cuando el monitor puede seleecionar a cuál de ellos observar en vídeo (figura 4-11). Una vez que se está monitorizando a un usuario se le pueden envíar mensajes de texto para corregir sus acciones (figura 4-12).

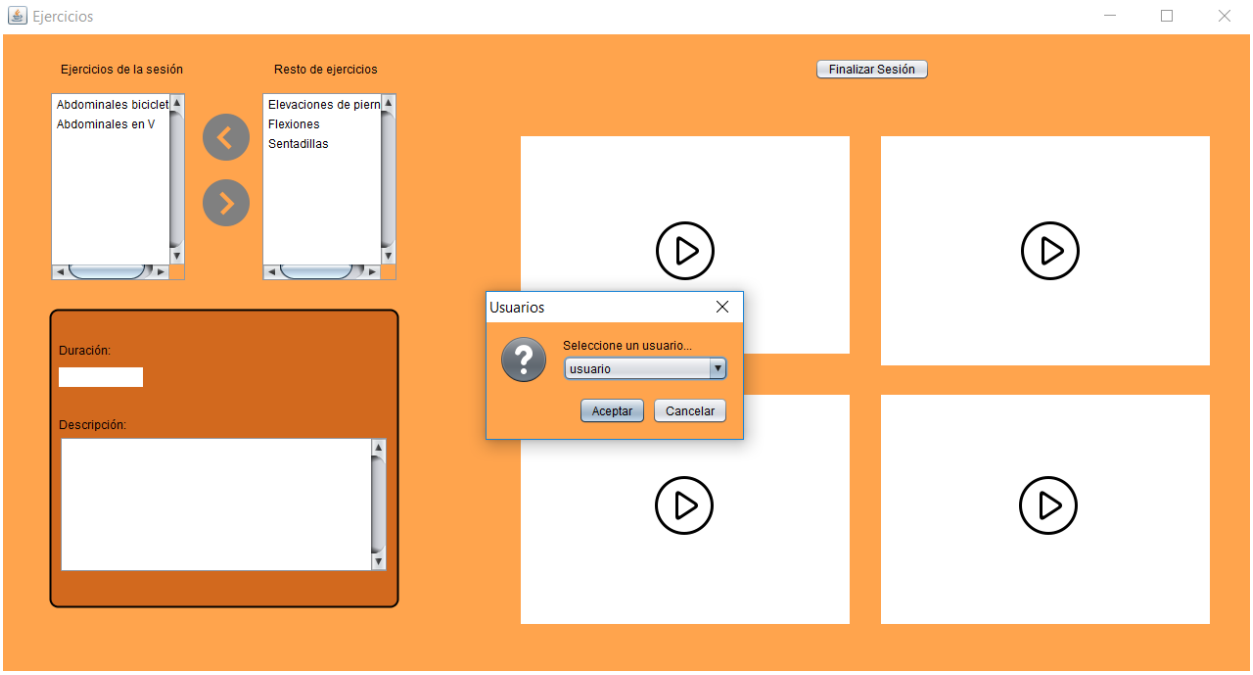

Figura 4-11 Ventana Ejercicios

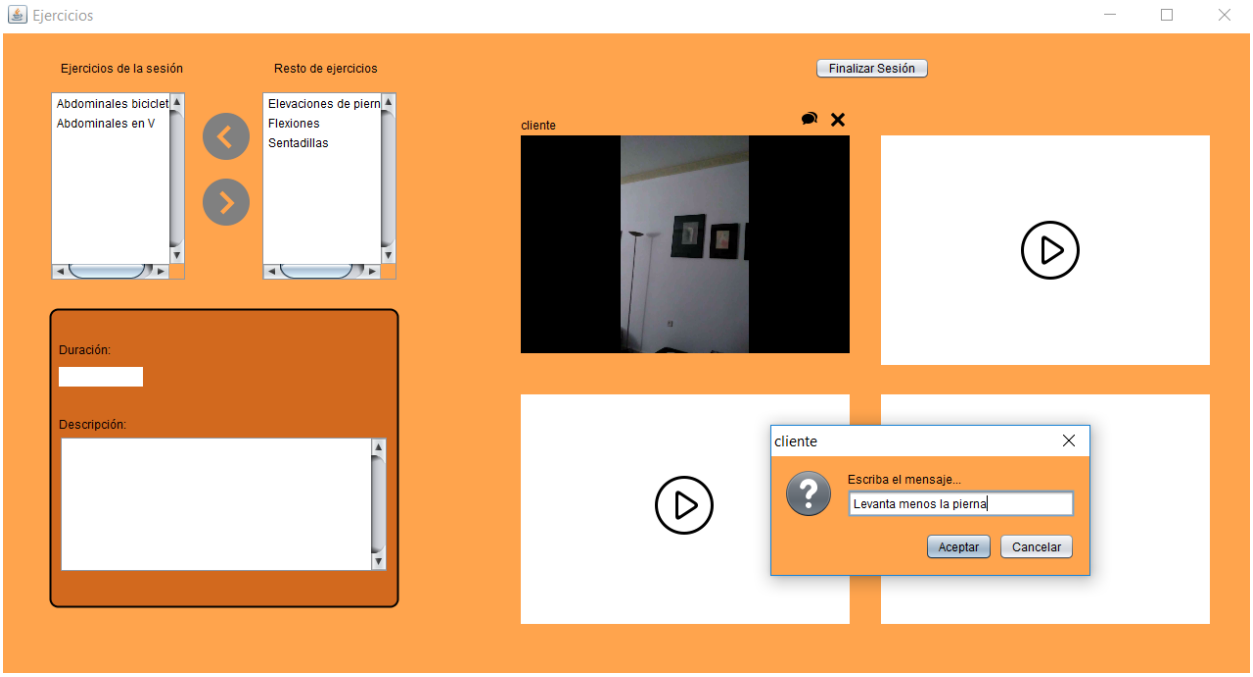

Figura 4-12 Ventana Ejercicios reproduciendo vídeo del cliente

5 APLICACIÓN ANDROID

"Los usuarios no saben realmente lo que quieren,
pero saben con certeza lo que no quieren."
-Ley de Manubay para los programadores-

En este capítulo se comentará la aplicación '***ProjectGym'*** y se profundizará en los cambios realizados en la misma respecto a otros proyectos Bibliografía. Los usuarios del sistema usarán esta aplicación para realizar los ejercicios de las sesiones creadas por el monitor mientras éste los supervisa en tiempo real. La misión más importante que desempeña la aplicación es realizar streaming de vídeo desde la cámara del dispositivo hacia la aplicación del monitor.

A continuación, detallaremos las modificaciones realizadas y el funcionamiento de la aplicación completa.

5.1 Gestor de peticiones

La primera gran modificación de la aplicación la encontramos en la clase *Httppostaux* (figura 4-1), la cual sirve para gestionar toda la comunicación con el servicio web. Al haber diseñado un nuevo servidor, hemos tenido que rediseñar también la forma en la que se comunicaba la aplicación Java con el servicio web REST.

El funcionamiento de esta clase es muy similar al de la clase GestorPeticiones de la aplicación Java. Tenemos dos métodos que cada uno se encargan de una tarea distinta:

- El método *sendTokenRequest* envía una petición POST a la dirección *'/oauth/token'* para conseguir un Token de Acceso. El servidor comprueba todas las credenciales y manda toda la información del Token. Seguidamente se introduce el valor del Token de Acceso en una cadena y se guarda en las preferencias de Android (*SharedPreferences*), con el objetivo de que cualquier método que necesite mandar una petición al servicio web pueda disponer de él.
- El segundo método principal es *httppostaux*, el cual se encarga de enviar las peticiones (POST, GET, DELETE) al servidor, de leer la respuesta e interpretarla. Dentro de la aplicación, cada vez que haga falta comunicarse con el servicio web, se llama a este método y se le pasan los parámetros necesarios.

El resto de métodos son llamados por la aplicación Java para realizar una determinada labor relacionada con usuarios, sesiones o ejercicios (por ejemplo obtener sesiones, registrar nuevos usuarios, etc…). Estos métodos reciben los parámetros necesarios de la aplicación y llaman al método *peticion*, que es el que hace el mayor trabajo. Los parámetros son convertidos a cadenas de texto con formato JSON para que el servicio web sea capaz de leerlos.

5.2 YouTube API

El segundo cambio realizado en la aplicación Android tiene que ver con la manera de reproducir los vídeos explicativos. Antes, los vídeos explicativos de cada uno de los ejercicios estaban almacenados en un directorio de la base de datos y en la tabla *ejercicios* de la base de datos estaban las URIs de cada uno de ellos. Entonces,

cuando seleccionábamos un ejercicio en la aplicación, se hacía una consulta a la base de datos para conseguir la dirección del vídeo. En ese momento, el vídeo se descargaba en el móvil y se reproducía haciendo uso del componente de Webview de Android.

Ahora, en lugar de tener almacenados los vídeos en nuestro servidor, están alojados en el servidor de YouTube y en la bases de datos están sus direcciones. La mejor solución que hemos encontrado es la de crear un fragmento en el layout de la actividad *Ejercicio* el cual llama utiliza la clase *YouTubeFragment* que nos devuelve una instancia del fragmento para reproducir el vídeo. Hemos creado dicha clase para gestionar el reproductor haciendo uso de las herramientas que nos proporciona la API de YouTube. En la figura 5-1 podemos observar el resultado.

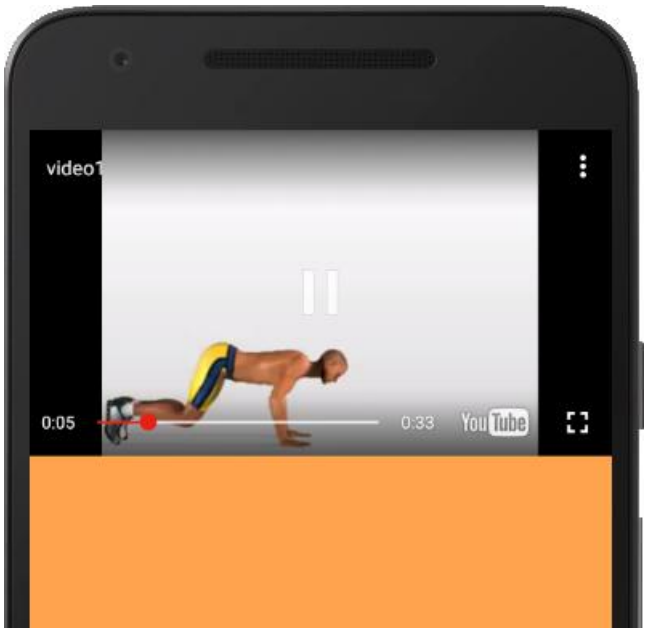

Figura 5-1 Reproductor de YouTube dentro de nuestra aplicación

5.3 Funcionamiento de la aplicación

En este apartado se explicará y se mostrará con capturas de pantalla como funciona la aplicación de entrenado.

Al ejecutar la aplicación aparece un formulario solicitando las credenciales para acceder al sistema (figura 5-2). Antes de hacer login, se debe introducir la IP del servicio web para que la aplicación se comunique con él (figura 5-3).

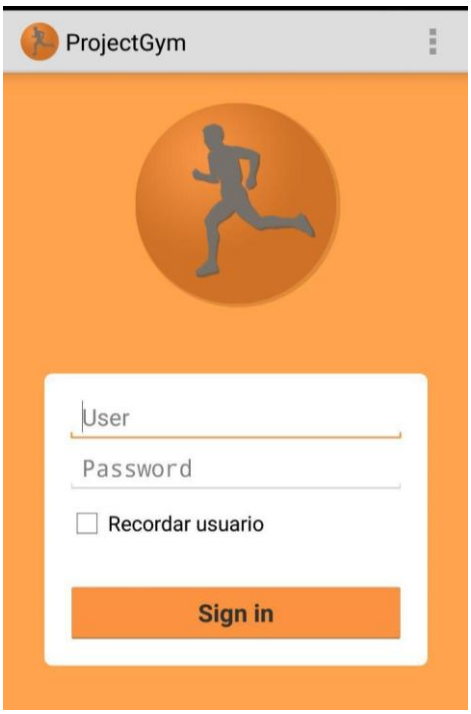

Figura 5-2 Inicio de sesión

Figura 5-3 Configuración de IP y puerto

Una vez estamos identificados en el sistema, nos aparece la lista de la de sesiones activas (figura 5-4) en las cuales podemos suscribirnos para llevar a cabo un entrenamiento. Basta con pulsar sobre la sesión elegida para suscribirse. Una vez lo hemos hecho, podemos ver los distintos ejercicios de la sesión que podemos elegir en el orden que queramos (figura 5-5).

Figura 5-4 Lista de sesiones activas

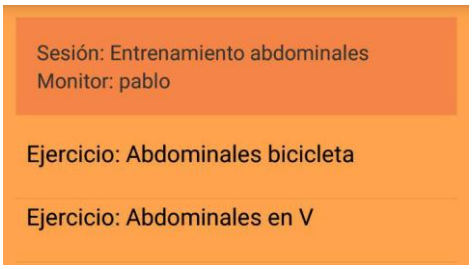

Figura 5-5 Lista de ejercicios de la sesión

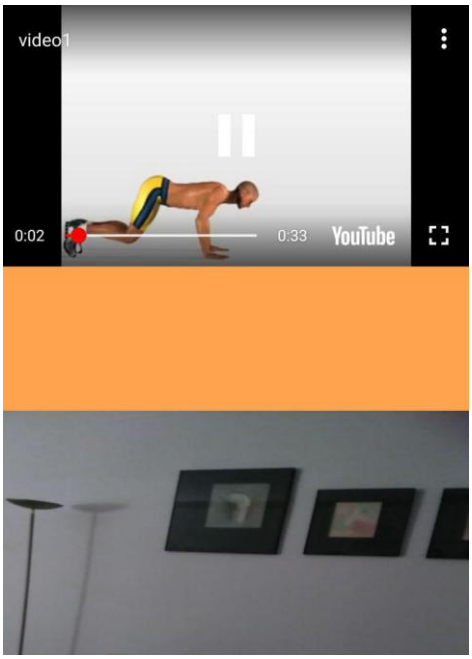

Figura 5-6 Reproducción de vídeo del ejercicio

Cuando pulsemos sobre el ejercicio deseado, entraremos en la última pantalla, en la que se reproduce un vídeo explicativo del ejercicio seleccionado y además se empieza a enviar por streaming el vídeo captado por la cámara del dispositivo móvil (figura 5-6). Si pulsamos en cualquier parte de la pantalla, nos aparece la información relativa al ejercicio, finalidad, duración y descripción (figura 5-7). Por último, podemos ver como se recibe un mensaje en directo del monitor haciendo alguna recomendación sobre el ejercicio que se está realizando (figura 5-8).

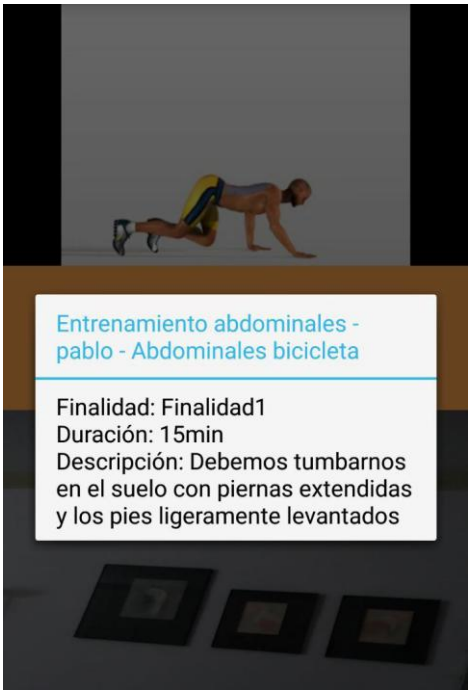

Figura 5-7 Información del ejercicio

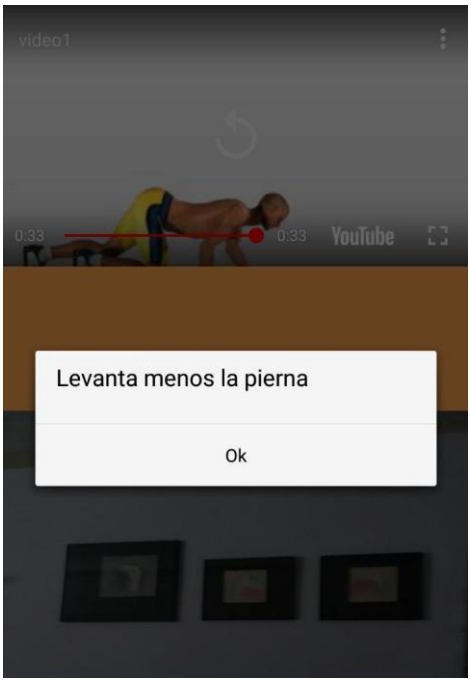

Figura 5-8 Mensaje del monitor

6 CONCLUSIONES Y LÍNEAS DE MEJORA

"No os diré: no lloréis;
pues no todas las lágrimas son amargas."
- Gandalf en el final de El Señor de los Anillos -

El desarrollo de este proyecto ha requerido más esfuerzo y dedicación del que nunca llegué a imaginar que sería capaz de realizar. Ha sido un largo camino de investigación y aprendizaje, sobre todo aprendizaje.

En primer lugar, el partir de otro proyecto ha implicado un profundo estudio del mismo, no sólo para comprender su funcionamiento, sino para poder realizar todas las modificaciones del código que han sido necesarias, tanto en la aplicación Java como en la Aplicación Android. Y en segundo lugar, tanto la elección y uso de las tecnologías y herramientas del proyecto, como la adquisición de conocimientos suficientes para implementarlas, han supuesto un reto aún mayor, puesto que la mayoría de ellas eran completamente nuevas para mí. Cabe destacar la faceta de seguridad del servicio web, que probablemente ha sido la parte más compleja y que proporcionalmente ha requerido más trabajo.

Sin embargo, puedo decir sin miedo que ha merecido la pena. Todo este tiempo trabajando en el proyecto me ha servido para madurar tanto profesional como personalmente, me ha abierto las puertas a la vida real al tenerme que enfrentar a tantísimos problemas y al tener que resolverlos por mí mismo, a base de investigar, equivocarme y mucha constancia.

6.1 Objetivos logrados

Al comenzar el proyecto se establecieron una serie de objetivos principales y secundarios que seguir a lo largo del desarrollo de nuestro sistema. A continuación se expondrán los objetivos que han sido cumplidos:

- Desarrollo de un servicio web RESTful con Spring Framework.
- Implementar seguridad en el servicio web para proteger la información y los recursos:
    - Sistema de autenticación mediante el protocolo OAuth2
    - Sistema de roles y permisos para el acceso a los diferentes recursos
    - Codificación de información sensible como las contraseñas
- Se ha creado una nueva ventana en la aplicación Java para habilitar el registro de nuevos usuarios, ya sean monitores o clientes.

- Se ha cambiado la forma de reproducir los vídeos explicativos de los ejercicios. Ya no están almacenados en nuestro propio servidor, sino que se accede a ellos a través del reproductor de YouTube que se ha implantado en la aplicación Android.

Por lo tanto, se han alcanzado todos y cada uno de los objetivos marcados al principio del proyecto.

6.2 Posibles líneas de mejora

Como todo en esta vida, este sistema es susceptible de mejorar. En esta sección se comentarán líneas futuras de mejora que se pueden seguir a partir de este proyecto:

- Añadir a la aplicación Java un reproductor de vídeo en la ventana de sesión para que el monitor pueda ver también el vídeo explicativo que el entrenado está reproduciendo en su dispositivo.
- Implementar un servidor STUN para la transmisión de vídeo desde la aplicación. Lo que hace este servidor es permitir que los clientes encuentren sus direcciones públicas, el tipo de NAT y el puerto asociado. Esta información se usa para la comunicación entre el cliente y el proveedor.
- En este proyecto se han implementado diversas medidas de seguridad haciendo uso del módulo Spring Security, pero hay muchas más que también podrían ser añadidas. El problema es que la magnitud de la seguridad es lo suficientemente grande como para poder hacer un proyecto completo sobre Spring Security.
- Se podría desarrollar una aplicación de escritorio también para los entrenados que deseen usar un portátil por ejemplo para entrenar en vez del móvil o la tablet. También se podría hacer lo contrario, crear una aplicación Android para el monitor para que éste pueda monitorizar una sesión de entrenamiento incluso desde un móvil o tablet.
- Implementar el envío de mensajes de voz además de los mensajes de texto, para que el entrenado pueda oír las recomendaciones del monitor sin tener que estar atento al dispositivo.
- Añadir comunicación en la dirección del cliente hacia el monitor, para que éste sea capaz de preguntarle cualquier duda fácilmente.
- Usar la especificación Swagger para documentar servicios REST. Swagger proporciona un cliente web para poder acceder de forma muy cómoda y vistosa a la documentación de nuestro servicio.

BIBLIOGRAFÍA

La bibliografía y referencias consultadas para la realización de este trabajo son mayormente recursos electrónicos, que se encuentran en Internet y están en constante actualización. La razón principal es que Spring se encuentra en continua evolución por la gran aceptación que está teniendo entre la comunidad de desarrolladores en general, por lo que cualquier libro que lleve más de seis meses es posible que haga uso de librerías o funciones que se encuentran caducadas a día de hoy (deprecated, en inglés), bien porque se hayan sustituido por una versión mejor o bien se ha dejado de usar por razones de seguridad. Además de que Spring posee una documentación magnífica e incluso ofrece guías sobre sus módulos.

[1] **Pablo Gil Pereira** *Sistema de gestión de sesiones de entrenamiento con monitorización mediante streaming de vídeo*, Sevilla, España, 2015.

[2] **Antonio José Díaz Lora** *Aplicación Android para la rehabilitación física de usuario usando MySQL*, Sevilla, España, 2014.

[3] **Antonio Clavaín Mateo** *Monitorización mediante streaming de vídeo para entrenamiento personal*, Sevilla, España, 2014.

[4] **Leonard Richardson, Sam Ruby** RESTful Web Services, s.1.: O'REILLY, 2007.

[5] **Eugen Paraschiv (Baeldung)** [e-book] Building a REST API with Spring.

[6] **Documentación general de Spring** *https://spring.io/docs*

[7] **Guías de Spring** *https://spring.io/guides*

[8] **Documentación del módulo Spring Boot**
https://docs.spring.io/spring-boot/docs/2.0.2.RELEASE/reference/htmlsingle/

[9] **Documentación del módulo Spring Security**
https://docs.spring.io/spring-security/site/docs/5.0.3.RELEASE/reference/htmlsingle/

[10] **OAuth2** *https://oauth.net/2/*

[11] **WebSystique Spring Security examples** *http://websystique.com/spring-security-tutorial/*

[12] **StackOverflow, Recursos para programadores** *https://stackoverflow.com/*

[13] **YouTube API** *https://developers.google.com/youtube/android/player/?hl=es-419*

1 Manual de instalación

1.1 Manual de instalación de la base de datos

Para poner en funcionamiento la base de datos utilizaremos XAMPP[1]. Por comodidad, recomendamos la instalación de la última versión disponible. Una vez instalado y estemos ante la interfaz gráfica de XAMPP, debemos activar los módulos Apache y MySQL (figura 6-1). A continuación, debemos hacer click en el botón Admin del módulo MySQL para abrir PHPMyAdmin, herramienta que facilitará el despliegue de la base de datos.

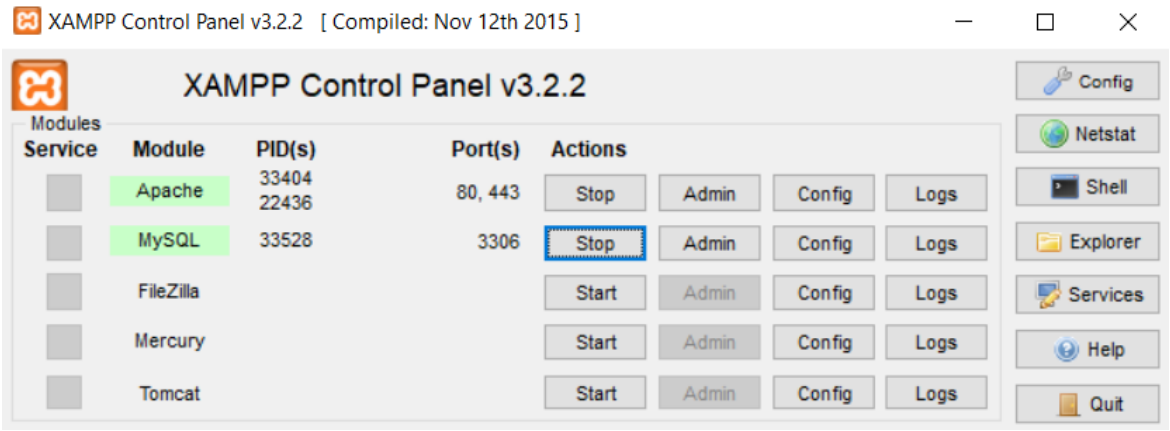

Figura 6-1 Interfaz gráfica de XAMPP

Una vez aquí (figura 6-2), tendremos que importar el fichero sentenciassql.sql para que se ejecute. Después de hacerlo, podemos cerrar la página PHPMyAdmin y pulsar Stop en el módulo Apache de XAMPP`, puesto que sólo necesitamos el módulo MySQL para que nuestra base de datos esté activa. No necesitamos el servidor Apache porque nuestro servicio web se apoya en el servidor Tomcat embebido que nos proporciona Spring.

Tras haber importado la base de datos, para ponerla en funcionamiento en cualquier momento basta con activar el módulo MySQL de XAMPP.

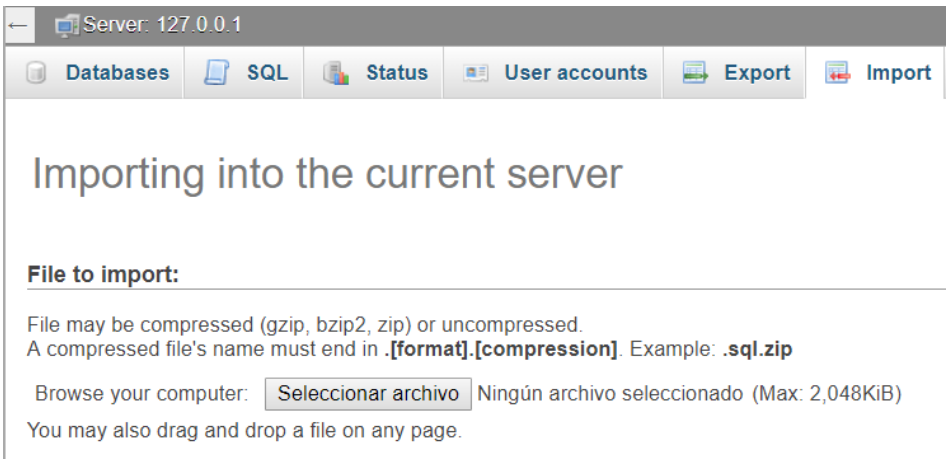

Figura 6-2 Importar base de datos desde interfaz gráfica PHPMyAdmin

[1] https://www.apachefriends.org/es/download.html

1.2 Manual de instalación del servicio web

Para poner a funcionar el servicio web deberemos instalar en nuestro equipo el entorno de desarrollo STS (Spring Tool Suite)[2]. En el enlace a pie de página podemos descargar la última versión del programa.

Una vez en Spring, tan sólo tendremos que abrir el proyecto que se encuentra en la carpeta *gymRestService*. Para ello, debemos acceder a 'File > Open Projects from File System' (figura 7-1).

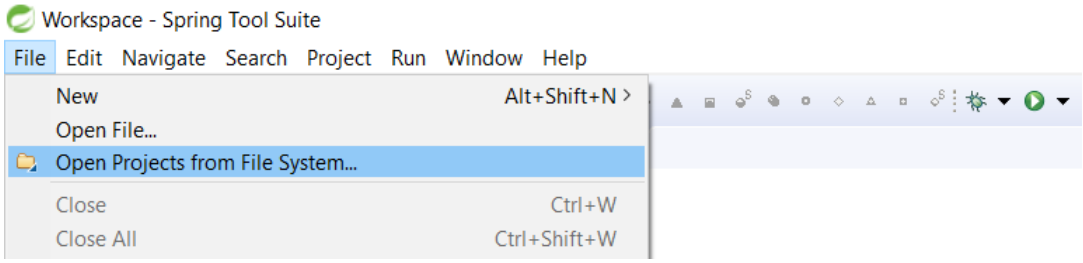

Figura 7-1 Interfaz de Spring. Abrir proyecto del sistema de archivos.

A continuación nos aparecerá una ventana en la que debemos buscar el proyecto y seleccionar el directorio *gymRestService* pulsando en el botón Directory. Antes de pulsar Finish, se debe comprobar que estén activas las casillas 'Search for nested projects' y 'Detect and configure project natures' (figura 7-2).

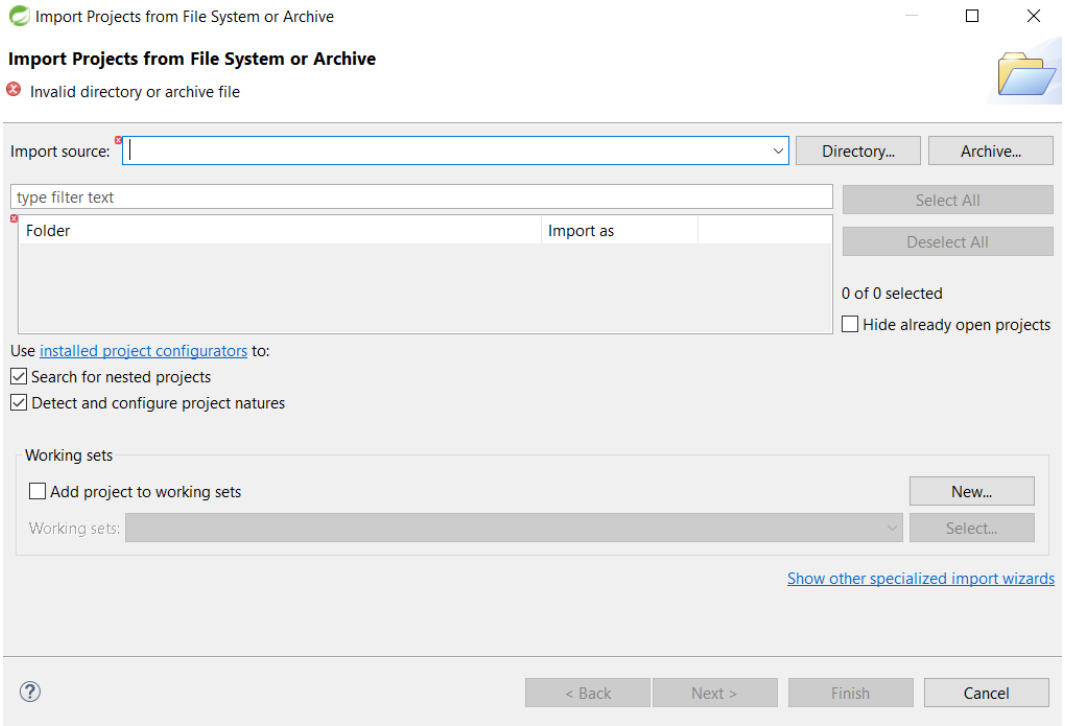

Figura 7-2 Ventana para importar proyectos desde un directorio

Por último, para ejecutar el servicio web basta con seleccionar el proyecto y pulsar el botón verde tal y como vemos en la figura 7-3.

[2] https://spring.io/tools/sts

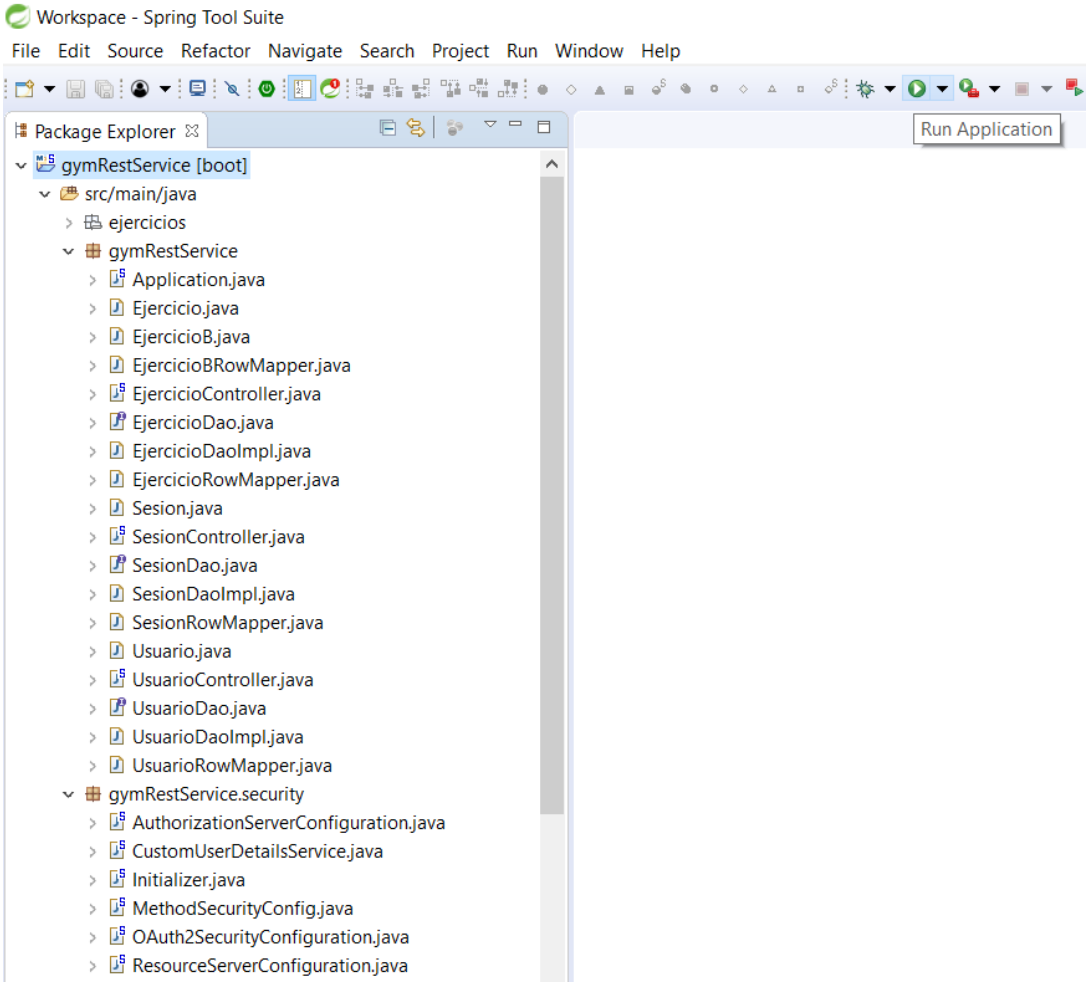

Figura 7-3 Interfaz de Spring en la se observa el botón de ejecución (Run Application)

Por último, para que las aplicaciones se comuniquen correctamente con el servicio web, almacenaremos el certificado SSL del servidor en la lista de Trusted Keystore (almacén de certificados de confianza) de java. Para ello el servicio web debe estar en ejecución y a continuación seguir los siguientes pasos:

- En primer lugar, abriremos la consola de Windows (cmd.exe) y nos situaremos en la carpeta 'Codigo/certificado' usando el comando 'cd'. Seguidamente ejecutaremos esta línea: 'java InstallCert IPservidor:9999'. Sustituir 'IPservidor' por la IP del servidor. Inmediatamente después se nos solicitará que introduzcamos algo, en este momento escribimos '1' y pulsamos 'enter'. El resultado puede verse en las figuras 7-4 y 7-5. Acabamos de generar un archivo llamado 'jssecacerts' en la carpeta 'certificado'.
- En segundo y último lugar, tenemos que copiar el archivo 'jssecacerts' dentro de la carpeta "$JAVA_HOME\jre\lib\security".

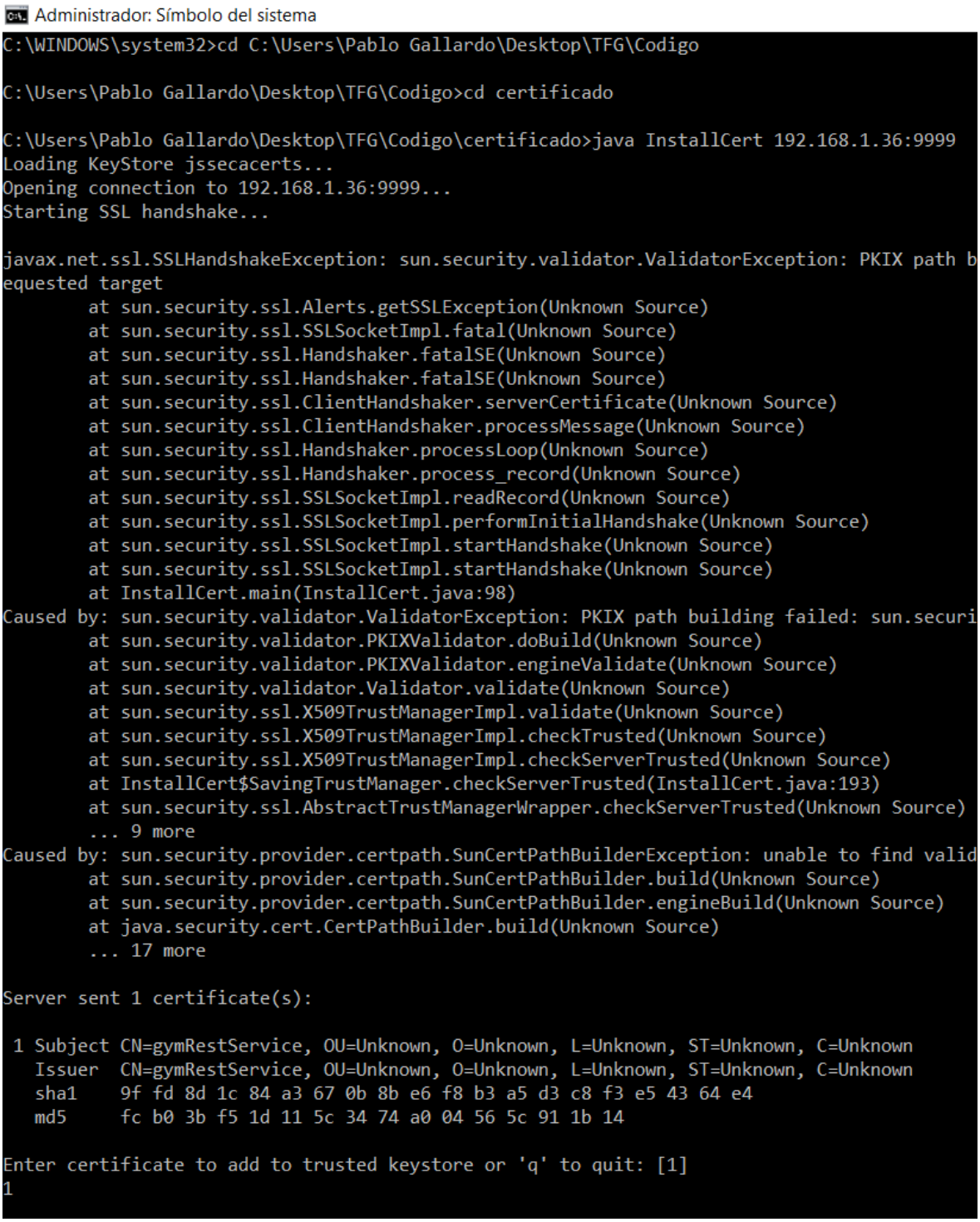

Figura 7-4 Resultado del comando java InstallCert IPservidor:9999 (parte1)

Administrador: Símbolo del sistema

```
1

[
[
  Version: V3
  Subject: CN=gymRestService, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
  Signature Algorithm: SHA256withRSA, OID = 1.2.840.113549.1.1.11

  Key:  Sun RSA public key, 2048 bits
  modulus: 173769434534292899869342664671965421815310268885198625026815510873552296051
491733800874781339817596960813609425316927974144247066429032913601864671846438390515215
239699099088845151054509097361462379056554286583942573467820847476228402660686934358585
  public exponent: 65537
  Validity: [From: Thu May 24 16:52:43 CEST 2018,
              To: Thu Apr 23 16:52:43 CEST 2020]
  Issuer: CN=gymRestService, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
  SerialNumber: [    3a8afbe5]

Certificate Extensions: 1
[1]: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 60 70 5F 53 C0 AE 0E 6F   ED C7 42 DA 2F 5C 5B 90  `p_S...o..B./\[.
0010: E6 4F C7 DF                                        .O..
]
]

]
  Algorithm: [SHA256withRSA]
  Signature:
0000: 2F DD CF F2 44 F7 5F 95   A8 E5 2F FD F8 57 E7 5E  /...D._.../..W.^
0010: B9 28 6C 2A 89 2D C0 34   34 4B 76 2C 90 26 70 87  .(l*.-.44Kv,.&p.
0020: 64 B8 DE DC 11 39 21 72   BE 35 1E 3D BB D1 BB 8A  d....9!r.5.=....
0030: 13 BF 0C 77 C5 CA 89 EF   AD 95 16 6C AF ED 16 AF  ...w.......l....
0040: 58 22 11 EB 22 42 44 A9   76 72 6A 91 9F 86 81 10  X"..."BD.vrj.....
0050: 84 32 18 CE E3 3C FC 78   4E 49 5A C3 46 F4 F3 A4  .2...<.xNIZ.F...
0060: 80 32 36 C2 B0 C1 E1 CD   1C 40 3C 6D A6 8B 5B D3  .26......@<m..[.
0070: 59 C8 0D 4B 0C 3D D8 F0   FC 15 10 9A 5E 96 DF 6F  Y..K.=......^..o
0080: C6 9F AA B4 B1 A9 7E CF   B8 E7 12 64 8E 4D 6B 82  ..........d.Mk.
0090: B5 08 27 FF 37 5A DF 97   C1 5D 5B 80 FE E5 9D 8B  ..'.7Z...][.....
00A0: E5 96 9C 2C EA 2D 5D BB   F8 62 E0 A7 77 E7 DF 4C  ...,.-]..b..w..L
00B0: B8 39 FB D0 4B 20 37 CB   0D C5 8B 42 47 57 5F 92  .9..K 7....BGW_.
00C0: 37 89 5E 6D 21 3C BF 8E   E2 E7 25 B5 D1 FE CA D7  7.^m!<....%.....
00D0: 06 3C D1 F8 AB 41 AF 7C   14 A3 88 B1 76 7B B8 B4  .<...A......v...
00E0: 8E 31 57 D6 36 FB 28 73   29 13 2D DF D8 C7 7A 16  .1W.6.(s).-...z.
00F0: 64 C5 FF FC A8 5E BA 6B   0E DF A3 D7 BE 21 5D 1D  d....^.k.....!].

]

Added certificate to keystore 'jssecacerts' using alias '192.168.1.36-1'
```

Figura 7-5 Resultado del comando java InstallCert IPservidor:9999 (parte 2)

1.3 Manual de instalación de la aplicación Java

Poner en funcionamiento la aplicación Java es muy sencillo. Basta con hacer doble click sobre el JAR ejecutable de la aplicación, que está situado en *monitorgym/monitorgym.jar*. Para que la aplicación funcione correctamente, el servicio debe estar ya en ejecución y el módulo MySQL encendido. En caso contrario, no seremos capaces de pasar más allá de la ventana de inicio de sesión.

Si lo que queremos es importar el proyecto en Spring o Eclipse, tan sólo tendremos que abrir el proyecto que se encuentra en la carpeta *MonitorGym* de la misma forma que hicimos con el servicio web. Para ello, debemos acceder a 'File > Open Projects from File System' (figura 7-1).

A continuación nos aparecerá una ventana en la que debemos buscar el proyecto y seleccionar el directorio *MonitorGym* pulsando en el botón Directory. Antes de pulsar Finish, se debe comprobar que estén activas las casillas 'Search for nested projects' y 'Detect and configure project natures' (figura 7-2)

Por último, para ejecutar la aplicación java basta con seleccionar el proyecto y pulsar el botón verde tal y como vemos en la figura 7-3.

Además, para que la aplicación sea capaz de conectar con el servicio web, tendremos que configurar la dirección IP. Para ello, hacemos click en el icono de configuración que aparece arriba a la derecha en la ventana de inicio de sesión, tras lo que nos aparecerá otra ventana para introducir la dirección del servidor (figura 8-1). En nuestro caso la dirección es la IP privada del equipo que estemos utilizando. Una vez hecho esto, la comunicación entre servicio web y aplicación se dará sin problemas.

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

Una vez dentro de la aplicación, y tras haber configurado la IP del servidor, se nos solicitarán unas credenciales. Debemos introducir tanto en el campo de nombre como de contraseña *'pablo'*, tras lo cual accederemos al sistema pulsando sobre el botón de *ENTRAR* (figura 8-1).

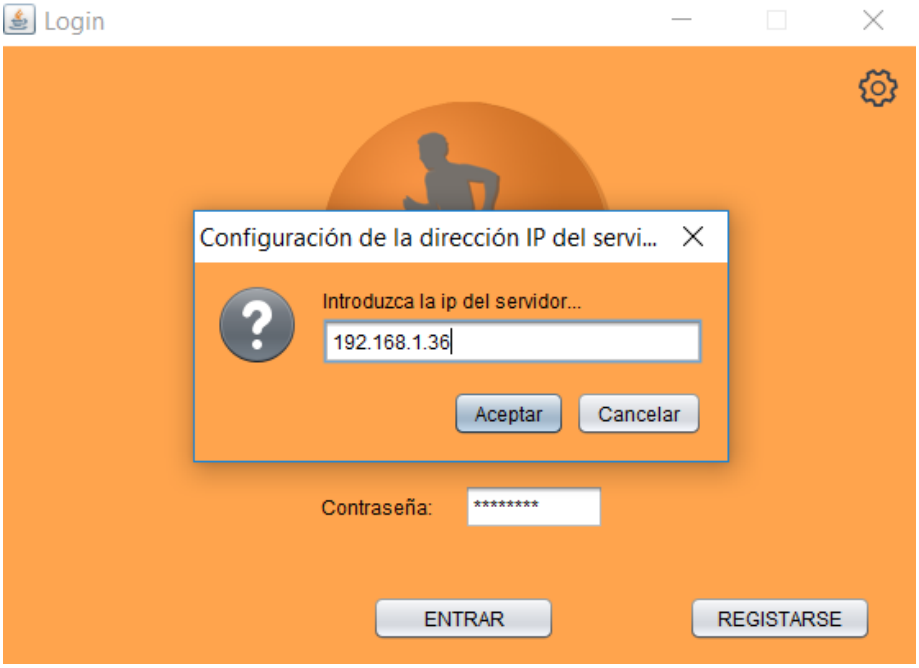

Figura 8-1 Ventana de configuración de la dirección IP del servicio web

[3] https://www.videolan.org/vlc/index.es.html

1.4 Manual de instalación de la aplicación Android

Para instalar esta aplicación, se debe copiar el archivo ProjectGym.apk que encontraremos en el directorio ProjectGym en el dispositivo Android donde queramos instalarla. Con el gestor de archivos, encontramos el archivo y lo ejecutamos. Aunque para ello es probable que antes haya que activar la casilla 'Orígenes desconocidos' que podemos encontrar en 'Ajustes > Seguridad', para así permitir que se instalen aplicaciones que no provengan de la tienda oficial de Google (PlayStore).

Una vez dentro de la aplicación, debemos hacer la misma tarea que con la aplicación Java, configurar la dirección IP del servidor y además, el puerto (figura 9-1). Hemos configurado nuestro servicio web para que mapee el puerto 9999, así que debemos introducir la IP del servidor y 9999 como puerto. A continuación, se nos solicitarán las credenciales para acceder. Debemos introducir tanto en el campo de nombre como de contraseña la palabra 'usuario', tras lo cual accederemos al sistema pulsando el botón de *Sign In*.

Si lo que se pretende es importar el proyecto en Android Studio, tan sólo tendremos que abrir el proyecto que se encuentra en la carpeta *ProjectGym*. Para ello, debemos acceder a 'File > Open Project'. A continuación nos aparecerá una ventana en la que debemos buscar el proyecto y seleccionar el directorio *ProjectGym*. Tras esto, el proyecto queda importado.

Si hemos seguido correctamente todas las indicaciones, tendremos todo el sistema completo funcionado.

Figura 9-1 Pantalla de configuración de IP y puerto del servicio web

2 Código

En este anexo se incluirá el código desarrollado para el servicio web así como todas las modificaciones llevadas a cabo en las aplicaciones Java y Android.

2.1 Servicio Web

Aplicacion.java

```java
package gymRestService;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;

import gymRestService.security.Initializer;

/*
 * Clase principal del servicio web, configurada para la inicialización
 * con Spring Boot
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

@SpringBootApplication
@EnableResourceServer
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(new Class[] { Application.class, Initializer.class }, args);
    }

}
```

Ejercicio.java

```java
package gymRestService;

/*
 * Clase que representa un ejercicio en la aplicación.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

public class Ejercicio {

    private int Id_Ejercicio;
    private String nombre_Ejercicio, Descripcion, Finalidad, Duracion, URL_video;

    public Ejercicio() {
    }

    public Ejercicio(int Id_Ejercicio, String nombre_Ejercicio, String Descripcion, String Finalidad,
        String Duracion, String URL_video) {

    this.setId_Ejercicio(Id_Ejercicio);
    this.setNombre_Ejercicio(nombre_Ejercicio);
    this.setDescripcion(Descripcion);
    this.setFinalidad(Finalidad);
    this.setDuracion(Duracion);
    this.setURL_video(URL_video);
    }

    public int getId_Ejercicio() {
        return Id_Ejercicio;
    }

    public void setId_Ejercicio(int Id_Ejercicio) {
        this.Id_Ejercicio = Id_Ejercicio;
```

```
    }

    public String getNombre_Ejercicio() {
        return nombre_Ejercicio;
    }

    public void setNombre_Ejercicio(String nombre_Ejercicio) {
        this.nombre_Ejercicio = nombre_Ejercicio;
    }

    public String getDescripcion() {
        return Descripcion;
    }

    public void setDescripcion(String Descripcion) {
        this.Descripcion = Descripcion;
    }

    public String getFinalidad() {
        return Finalidad;
    }

    public void setFinalidad(String Finalidad) {
        this.Finalidad = Finalidad;
    }

    public String getDuracion() {
        return Duracion;
    }

    public void setDuracion(String Duracion) {
        this.Duracion = Duracion;
    }

    public String getURL_video() {
        return URL_video;
    }

    public void setURL_video(String URL_video) {
        this.URL_video = URL_video;
    }

}
```

EjercicioB.java

```
package gymRestService;

/*
 * Clase que representa un ejercicio (que también contiene datos de la sesión) en la aplicación.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

public class EjercicioB {

    private int Id_Ejercicio;
    private String nombre_Ejercicio, Descripcion, user_Monitor, nombre_Sesion;

    public EjercicioB() {
    }

    public EjercicioB(int Id_Ejercicio, String nombre_Ejercicio, String Descripcion, String user_Monitor, String
 nombre_Sesion) {
    this.setId_Ejercicio(Id_Ejercicio);
    this.setNombre_Ejercicio(nombre_Ejercicio);
    this.setDescripcion(Descripcion);
    this.setUser_Monitor(user_Monitor);
    this.setNombre_Sesion(nombre_Sesion);
    }

    public int getId_Ejercicio() {
        return Id_Ejercicio;
    }

    public void setId_Ejercicio(int Id_Ejercicio) {
        this.Id_Ejercicio = Id_Ejercicio;
    }

```

```
35.      public String getNombre_Ejercicio() {
36.          return nombre_Ejercicio;
37.      }
38.
39.      public void setNombre_Ejercicio(String nombre_Ejercicio) {
40.          this.nombre_Ejercicio = nombre_Ejercicio;
41.      }
42.
43.      public String getDescripcion() {
44.          return Descripcion;
45.      }
46.
47.      public void setDescripcion(String Descripcion) {
48.          this.Descripcion = Descripcion;
49.      }
50.
51.      public String getUser_Monitor() {
52.          return user_Monitor;
53.      }
54.
55.      public void setUser_Monitor(String user_Monitor) {
56.          this.user_Monitor = user_Monitor;
57.      }
58.
59.      public String getNombre_Sesion() {
60.          return nombre_Sesion;
61.      }
62.
63.      public void setNombre_Sesion(String nombre_Sesion) {
64.          this.nombre_Sesion = nombre_Sesion;
65.      }
66.
67.  }
```

EjercicioBRowMapper.java

```
1.   package gymRestService;
2.
3.   import java.sql.ResultSet;
4.   import java.sql.SQLException;
5.   import org.springframework.jdbc.core.RowMapper;
6.
7.   /*
8.    * Clase que representa un mapeador de ejercicios. Obtiene y crea un
9.    * ejercicio a partir de los datos obtenidos de la base de datos.
10.   * Este RowMapper es para obtener los datos de la tablas 'ejercicios' y 'sesiones'
11.   *
12.   * Diseño por Pablo Gallardo Hiraldo
13.   * Versión: 1.0
14.   *
15.   */
16.
17.  public class EjercicioBRowMapper implements RowMapper<EjercicioB> {
18.
19.      public EjercicioB mapRow(ResultSet resultSet, int arg1) throws SQLException {
20.          // getX(index) para coger los elementos recibidos por la tabla de datos
21.          EjercicioB ejercicioB = new EjercicioB(resultSet.getInt(1), resultSet.getString(2),
22.          resultSet.getString(3), resultSet.getString(4), resultSet.getString(5));
23.
24.      return ejercicioB;
25.      }
26.  }
```

EjercicioController.java

```
1.   package gymRestService;
2.
3.   import java.util.List;
4.   import org.springframework.web.bind.annotation.RequestMapping;
5.   import org.springframework.web.bind.annotation.RestController;
6.   import org.springframework.web.bind.annotation.PathVariable;
7.   import org.springframework.web.bind.annotation.RequestMethod;
8.   import org.springframework.web.bind.annotation.ResponseBody;
9.   import org.springframework.beans.factory.annotation.Autowired;
10.  import org.springframework.context.annotation.ImportResource;
11.
```

```java
/*
 * Clase que representa un controlador REST de ejercicios. Mapea las
 * operaciones sobre recursos REST relacionados con ejercicios y hace uso
 * del DAO para hacerlas efectivas en la base de datos.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

@RestController
@ImportResource("classpath:spring/beanLocations.xml")
public class EjercicioController {

    // Obtenemos el DAO mediante inyección de dependencias
    @Autowired
    private EjercicioDaoImpl ejercicioDao;


    // (J,A) Obtiene un ejercicio
    @RequestMapping(value = "/gym/ejercicio/{id_ejercicio}/", method = RequestMethod.GET, produces = "application/json")
    public @ResponseBody List<Ejercicio> getEjercicio(@PathVariable("id_ejercicio") int id_ejercicio) {

    return ejercicioDao.getEjercicio(id_ejercicio);
    }


    // (J) Obtiene todos los ejercicios
    @RequestMapping(value ="/gym/ejercicio//", method = RequestMethod.GET, produces = "application/json")
    public @ResponseBody List<Ejercicio> getAllEjercicios() {

    return ejercicioDao.getAllEjercicios();
    }


    // (J) Obtiene los ejercicios no incluidos en la sesion
    @RequestMapping(value = "/gym/ejercicioNoSesion/{id_sesion}/", method = RequestMethod.GET, produces = "application/json")
    public @ResponseBody List<Ejercicio> getEjercicioNoSesion(@PathVariable("id_sesion") int id_sesion) {

    return ejercicioDao.getEjercicioNoSesion(id_sesion);
    }


    // (J,A) Obtiene los ejercicios incluidos en sesion
    @RequestMapping(value = "/gym/ejercicioSesion/{id_sesion}/", method = RequestMethod.GET, produces = "application/json")
    public @ResponseBody List<EjercicioB> getEjercicioSesion(@PathVariable("id_sesion") int id_sesion) {

    return ejercicioDao.getEjercicioSesion(id_sesion);
    }


    // (J) Añade un ejercicio a una sesion
    @RequestMapping(value = "/gym/ejercicio/{id_sesion}/{id_ejercicio}/", method = RequestMethod.POST, consumes = "application/json")
    public String addEjercicioSesion(
            @PathVariable("id_sesion") int id_sesion,
            @PathVariable("id_ejercicio") int id_ejercicio) {

    return ejercicioDao.addEjercicioSesion(id_sesion, id_ejercicio);
    }


    // (J) Elimina un ejercicio de sesion
    @RequestMapping(value = "/gym/ejercicio/{id_sesion}/{id_ejercicio}/", method = RequestMethod.DELETE)
    public String deleteSesion(
            @PathVariable("id_sesion") int id_sesion,
            @PathVariable("id_ejercicio") int id_ejercicio) {

    return ejercicioDao.deleteEjercicio(id_sesion, id_ejercicio);
    }

}
```

EjercicioDao.java

```java
package gymRestService;

import java.util.List;

/*
 * Clase que representa la interfaz DAO de ejercicios. Abstrae las
 * operaciones necesarias sobre la base de datos relacionadas con los
 * ejercicios a petición del controlador REST de ejercicios.
 *
 * Diseño por Pablo Gallardo
 * Versión: 1.0
 *
 */

public abstract interface EjercicioDao {

    public abstract String addEjercicioSesion(int id_sesion, int id_ejercicio);
    public abstract List<Ejercicio> getEjercicio (int id_ejercicio);
    public abstract List<Ejercicio> getEjercicioNoSesion (int id_sesion);
    public abstract List<EjercicioB> getEjercicioSesion (int id_sesion);
    public abstract List<Ejercicio> getAllEjercicios();
    public abstract String deleteEjercicio(int id_sesion, int id_ejercicio);

}
```

EjercicioDaoImpl.java

```java
package gymRestService;

import java.util.List;
import javax.sql.DataSource;
import org.springframework.jdbc.core.JdbcTemplate;

/*
 * Clase que implementa la interfaz DAO de ejercicios. Realiza las
 * operaciones necesarias sobre la base de datos relacionadas con los
 * ejercicios a petición del controlador REST de ejercicios. No se
 * comentan los métodos debido a que se corresponden con las operaciones
 * del controlador, con lo que ya están descritos previamente.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

public class EjercicioDaoImpl implements EjercicioDao {

    JdbcTemplate jdbcTemplate;


    public List<Ejercicio> getEjercicio(int id_ejercicio) {
        return jdbcTemplate.query("SELECT * FROM ejercicios WHERE id_ejercicio =?",
            new Object[] { id_ejercicio }, new EjercicioRowMapper());
    }


    public List<Ejercicio> getAllEjercicios() {
        return jdbcTemplate.query("SELECT * FROM ejercicios",
            new EjercicioRowMapper());
    }


    public List<Ejercicio> getEjercicioNoSesion(int id_sesion) {
        return jdbcTemplate.query("SELECT * FROM ejercicios WHERE ejercicios.Id_Ejercicio NOT IN "
                + "(SELECT ejerciciosdesesion.Id_Ejercicio FROM ejerciciosdesesion WHERE ejerciciosdesesion.Id_Sesion=?)",
            new Object[] { id_sesion }, new EjercicioRowMapper());
    }


    public List<EjercicioB> getEjercicioSesion(int id_sesion) {
        return jdbcTemplate.query("SELECT ejercicios.Id_Ejercicio, ejercicios.nombre_Ejercicio, ejercicios.Descripcion, "
                + "sesiones.user_Monitor,sesiones.nombre_Sesion FROM (sesiones INNER JOIN ejerciciosdesesion ON "
                + "sesiones.Id_Sesion=ejerciciosdesesion.Id_Sesion INNER JOIN ejercicios ON "
```

```
                + "ejercicios.Id_Ejercicio=ejerciciosdesesion.Id_Ejercicio) WHERE ejerciciosdesesion.Id_Sesion=?
",
                new Object[] { id_sesion }, new EjercicioBRowMapper());
    }


    public String addEjercicioSesion(int id_sesion, int id_ejercicio) {
        int consulta = jdbcTemplate.update("INSERT INTO ejerciciosdesesion (Id_Ejercicio, Id_Sesion) VALUES(?,?)
",
                new Object[] { id_ejercicio, id_sesion });

        // Devolvemos una cadena comunicando si la operación ha tenido éxito
        String resul ="";
        if (consulta==0){
            resul = "[{\"addEjercicioSesion\":\"0\"}]";
        }else{
            resul = "[{\"addEjercicioSesion\":\"1\"}]";
        }
        return resul;
    }


    public String deleteEjercicio(int id_sesion, int id_ejercicio) {
        int consulta= jdbcTemplate.update("DELETE FROM ejerciciosdesesion WHERE Id_Sesion=? AND Id_Ejercicio=?",

            new Object[] { id_sesion, id_ejercicio });

        // Devolvemos una cadena comunicando si la operación ha tenido éxito
        String resul ="";
        if (consulta==0){
            resul = "[{\"eliminaEjercicioSesion\":\"0\"}]";
        }else{
            resul = "[{\"eliminaEjercicioSesion\":\"1\"}]";
        }
        return resul;
    }


    // Inyección del dataSource mediante el constructor
    public void setDataSource(DataSource dataSource) {
    this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public EjercicioDaoImpl() {
    }
}
```

EjercicioRowMapper.java

```
package gymRestService;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

/*
 * Clase que representa un mapeador de ejercicios. Obtiene y crea un
 * ejercicio a partir de los datos obtenidos de la base de datos.
 * Este RowMapper es para obtener los datos de la tabla 'ejercicios'
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

public class EjercicioRowMapper implements RowMapper<Ejercicio> {

    public Ejercicio mapRow(ResultSet resultSet, int arg1) throws SQLException {
        // getX(index) para coger los elementos recibidos por la tabla de datos
        Ejercicio ejercicio = new Ejercicio(resultSet.getInt(1), resultSet.getString(2),
        resultSet.getString(3), resultSet.getString(4), resultSet.getString(5),
        resultSet.getString(6));

    return ejercicio;
    }
}
```

Sesion.java

```java
package gymRestService;

/*
 * Clase que representa una sesión en la aplicación.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

public class Sesion {

    private int Id_Sesion, num_Usuarios;
    private byte activa;
    private String user_Monitor, nombre_Sesion, fecha, ip_Monitor, descripcion;

    public Sesion() {
    }

    public Sesion(int Id_Sesion, String user_Monitor, String nombre_Sesion, String fecha,
        int num_Usuarios, String ip_Monitor, byte activa, String descripcion) {

    this.setId_Sesion(Id_Sesion);
    this.setUser_Monitor(user_Monitor);
    this.setNombre_Sesion(nombre_Sesion);
    this.setFecha(fecha);
    this.setNum_Usuarios(num_Usuarios);
    this.setIp_Monitor(ip_Monitor);
    this.setActiva(activa);
    this.setDescripcion(descripcion);
    }



    public int getId_Sesion() {
        return Id_Sesion;
    }

    public void setId_Sesion(int id_Sesion) {
        Id_Sesion = id_Sesion;
    }

    public String getUser_Monitor() {
        return user_Monitor;
    }

    public void setUser_Monitor(String user_Monitor) {
        this.user_Monitor = user_Monitor;
    }

    public String getNombre_Sesion() {
        return nombre_Sesion;
    }

    public void setNombre_Sesion(String nombre_Sesion) {
        this.nombre_Sesion = nombre_Sesion;
    }

    public String getFecha() {
        return fecha;
    }

    public void setFecha(String fecha) {
        this.fecha = fecha;
    }

    public int getNum_Usuarios() {
        return num_Usuarios;
    }

    public void setNum_Usuarios(int num_Usuarios) {
        this.num_Usuarios = num_Usuarios;
    }

    public String getIp_Monitor() {
        return ip_Monitor;
    }

    public void setIp_Monitor(String ip_Monitor) {
        this.ip_Monitor = ip_Monitor;
    }
```

```
82.
83.     public byte getActiva() {
84.         return activa;
85.     }
86.
87.     public void setActiva(byte activa) {
88.         this.activa = activa;
89.     }
90.
91.     public String getDescripcion() {
92.         return descripcion;
93.     }
94.
95.     public void setDescripcion(String descripcion) {
96.         this.descripcion = descripcion;
97.     }
98.
99. }
```

SesionController.java

```
1.  package gymRestService;
2.
3.  import java.util.List;
4.  import org.springframework.web.bind.annotation.RequestMapping;
5.  import org.springframework.web.bind.annotation.RestController;
6.  import org.springframework.web.bind.annotation.PathVariable;
7.  import org.springframework.web.bind.annotation.RequestBody;
8.  import org.springframework.web.bind.annotation.RequestMethod;
9.  import org.springframework.web.bind.annotation.ResponseBody;
10. import org.springframework.beans.factory.annotation.Autowired;
11. import org.springframework.context.annotation.ImportResource;
12.
13. /*
14.  * Clase que representa un controlador REST de sesiones. Mapea las
15.  * operaciones sobre recursos REST relacionados con sesiones y hace uso
16.  * del DAO para hacerlas efectivas en la base de datos.
17.  *
18.  * Diseño por Pablo Gallardo Hiraldo
19.  * Versión: 1.0
20.  *
21.  */
22.
23. @RestController
24. @ImportResource("classpath:spring/beanLocations.xml")
25. public class SesionController {
26.
27.     // Obtenemos el DAO mediante inyección de dependencias
28.     @Autowired
29.     private SesionDaoImpl sesionDao;
30.
31.
32.     // (J) Añade una sesión
33.     @RequestMapping(value = "/gym/sesionAdd/", method = RequestMethod.POST, consumes = "application/json")
34.     public @ResponseBody String addSesion(@RequestBody Sesion sesion) {
35.
36.     return sesionDao.addSesion(sesion.getUser_Monitor(), sesion.getNombre_Sesion(),
37.         sesion.getFecha(), sesion.getNum_Usuarios(), sesion.getActiva());
38.     }
39.
40.
41.     // (A) Obtiene las sesiones activas
42.     @RequestMapping(value = "/gym/sesionesActivas/", method = RequestMethod.GET, produces = "application/json")

43.     public @ResponseBody List<Sesion> getSesionesactivas() {
44.
45.     return sesionDao.getSesionesActivas();
46.     }
47.
48.
49.     // (J,A) Obtiene una sesión
50.     @RequestMapping(value = "/gym/sesion/{id_sesion}/", method = RequestMethod.GET, produces = "application/json
    ")
51.     public @ResponseBody List<Sesion> getSesion(@PathVariable("id_sesion") int id_sesion) {
52.
53.     return sesionDao.getSesion(id_sesion);
54.     }
55.
56.
57.     // (J) Elimina una sesión
58.     @RequestMapping(value = "/gym/sesion/{id_sesion}/", method = RequestMethod.DELETE)
```

```
59.      public String deleteSesion(@PathVariable("id_sesion") int id_sesion) {
60.
61.      return sesionDao.deleteSesion(id_sesion);
62.      }
63.
64.
65.      // (J) Modifica una sesión
66.      @RequestMapping(value = "/gym/sesion/", method = RequestMethod.POST, consumes = "application/json")
67.      public String updateSesion(@RequestBody Sesion sesion) {
68.
69.      return sesionDao.updateSesion(sesion.getId_Sesion());
70.      }
71.
72.
73.      // (J) Obtiene las sesiones de un determinado monitor
74.      @RequestMapping(value = "/gym/sesionMonitor/{monitor}/", method = RequestMethod.GET, produces = "application
    /json")
75.      public @ResponseBody List<Sesion> getSesionMonitor(@PathVariable("monitor") String monitor) {
76.
77.      return sesionDao.getSesionMonitor(monitor);
78.      }
79.
80.
81.      // (J) Añade la IP del monitor para que el cliente Android pueda conectarse más tarde
82.      @RequestMapping(value = "/gym/sesionMonitor/", method = RequestMethod.POST, consumes = "application/json")
83.      public String updateSesionMonitor(@RequestBody Sesion sesion) {
84.
85.      return sesionDao.updateSesionMonitor(sesion.getId_Sesion(), sesion.getIp_Monitor());
86.      }
87.
88.  }
```

SesionDao.java

```
1.   package gymRestService;
2.
3.   import java.util.List;
4.
5.   /*
6.    * Clase que representa la interfaz DAO de sesiones. Abstrae las
7.    * operaciones necesarias sobre la base de datos relacionadas con los
8.    * sesiones a petición del controlador REST de sesiones.
9.    *
10.   * Diseño por Pablo Gallardo
11.   * Versión: 1.0
12.   *
13.   */
14.
15.  public abstract interface SesionDao {
16.
17.      public abstract String addSesion( String user_monitor, String nombre_sesion, String fecha,
18.              int num_usuarios, byte activa);
19.      public abstract String updateSesion(int id_sesion);
20.      public abstract String updateSesionMonitor(int id_sesion, String ip_monitor);
21.      public abstract List<Sesion> getSesionesActivas();
22.      public abstract List<Sesion> getSesion(int id_sesion);
23.      public abstract List<Sesion> getSesionMonitor(String monitor);
24.      public abstract String deleteSesion(int id_sesion);
25.
26.  }
```

SesionDaoImpl.java

```
1.   package gymRestService;
2.
3.   import java.util.List;
4.   import javax.sql.DataSource;
5.   import org.springframework.jdbc.core.JdbcTemplate;
6.
7.   /*
8.    * Clase que implementa la interfaz DAO de sesiones. Realiza las
9.    * operaciones necesarias sobre la base de datos relacionadas con las
10.   * sesiones a petición del controlador REST de sesiones. No se
11.   * comentan los métodos debido a que se corresponden con las operaciones
12.   * del controlador, con lo que ya están descritos previamente.
13.   *
```

```
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

public class SesionDaoImpl implements SesionDao {

    JdbcTemplate jdbcTemplate;


    public String addSesion(String user_monitor, String nombre_sesion, String fecha,
            int num_usuarios, byte activa) {

        int consulta = jdbcTemplate.update("INSERT INTO sesiones (nombre_Sesion, user_Monitor, fecha, num_Usuari
    os, "
                + "activa, ip_Monitor, descripcion) VALUES (?,?,?,?,?, '', '')",
                new Object[] { nombre_sesion, user_monitor, fecha, num_usuarios, activa });

        // Devolvemos una cadena comunicando si la operación ha tenido éxito
        String resul ="";
        if (consulta==0){
            resul = "[{\"nuevaSesion\":\"0\"}]";
        }else{
            resul = "[{\"nuevaSesion\":\"1\"}]";
        }
        return resul;
    }


    public List<Sesion> getSesionesActivas() {
        return jdbcTemplate.query("SELECT * FROM sesiones WHERE activa=1",
                new SesionRowMapper());
    }


    public List<Sesion> getSesion(int id_sesion) {
        return jdbcTemplate.query("SELECT * FROM sesiones WHERE Id_Sesion=?",
                new Object[] { id_sesion }, new SesionRowMapper());
    }


    public String deleteSesion(int id_sesion) {
        int consulta= jdbcTemplate.update("DELETE FROM sesiones WHERE Id_Sesion=?",
                new Object[] { id_sesion });

        // Devolvemos una cadena comunicando si la operación ha tenido éxito
        String resul ="";
        if (consulta==0){
            resul = "[{\"eliminaSesion\":\"0\"}]";
        }else{
            resul = "[{\"eliminaSesion\":\"1\"}]";
        }
        return resul;
    }


    public String updateSesion (int id_sesion) {
        int consulta = jdbcTemplate.update("UPDATE sesiones SET activa=0 WHERE Id_Sesion=?",
                new Object[] { id_sesion });

        // Devolvemos una cadena comunicando si la operación ha tenido éxito
        String resul ="";
        if (consulta==0){
            resul = "[{\"setNoActiva\":\"0\"}]";
        }else{
            resul = "[{\"setNoActiva\":\"1\"}]";
        }
        return resul;
    }


    public List<Sesion> getSesionMonitor(String monitor) {
        return jdbcTemplate.query("SELECT * FROM sesiones WHERE user_Monitor=?",
            new Object[] { monitor }, new SesionRowMapper());
    }


    public String updateSesionMonitor(int id_sesion, String ip_monitor) {
        String ipypuerto = "192.168.1.36:"+ip_monitor;
        int consulta = jdbcTemplate.update("UPDATE sesiones SET ip_Monitor=? WHERE Id_Sesion=?",
                new Object[] { ipypuerto, id_sesion });

        // Devolvemos una cadena comunicando si la operación ha tenido éxito
        String resul ="";
        if (consulta==0){
```

```
98.             resul = "[{\"setIpMonitor\":\"0\"}]";
99.         }else{
100.            resul = "[{\"setIpMonitor\":\"1\"}]";
101.        }
102.        return resul;
103.    }
104.
105.
106.    // Inyección del dataSource mediante el constructor
107.    public void setDataSource(DataSource dataSource) {
108.    this.jdbcTemplate = new JdbcTemplate(dataSource);
109.    }
110.
111.    public SesionDaoImpl() {
112.    }
113. }
```

SesionRowMapper.java

```
1.  package gymRestService;
2.
3.  import java.sql.ResultSet;
4.  import java.sql.SQLException;
5.  import org.springframework.jdbc.core.RowMapper;
6.
7.  /*
8.   * Clase que representa un mapeador de sesiones. Obtiene y crea una
9.   * sesión a partir de los datos obtenidos de la base de datos.
10.  * Este RowMapper es para obtener los datos de la tabla 'sesiones'
11.  *
12.  * Diseño por Pablo Gallardo Hiraldo
13.  * Versión: 1.0
14.  *
15.  */
16.
17. public class SesionRowMapper implements RowMapper<Sesion> {
18.
19.     public Sesion mapRow(ResultSet resultSet, int arg1) throws SQLException {
20.         // getX(index) para coger los elementos recibidos por la tabla de datos
21.         Sesion sesion = new Sesion(resultSet.getInt(1), resultSet.getString(2),
22.         resultSet.getString(3), resultSet.getString(4), resultSet.getInt(5),
23.         resultSet.getString(6), resultSet.getByte(7), resultSet.getString(8));
24.
25.     return sesion;
26.     }
27. }
```

Usuario.java

```
1.  package gymRestService;
2.
3.  /*
4.   * Clase que representa un usuario en la aplicación.
5.   *
6.   * Diseño por Pablo Gallardo Hiraldo
7.   * Versión: 1.0
8.   *
9.   */
10.
11. public class Usuario {
12.
13.     private int Id_Usuario;
14.     private byte enabled;
15.     private String nombre, apellidos, dni, direccion, movil, username, passw;
16.
17.     public Usuario() {
18.     }
19.
20.     public Usuario(int Id_Usuario, String nombre, String apellidos, String dni,
21.         String direccion, String movil, String username, String passw, byte enabled) {
22.
23.     this.setId_Usuario(Id_Usuario);
24.     this.setNombre(nombre);
25.     this.setApellidos(apellidos);
26.     this.setDni(dni);
27.     this.setDireccion(direccion);
```

```
28.       this.setMovil(movil);
29.       this.setUsername(username);
30.       this.setPassw(passw);
31.       this.setEnabled(enabled);
32.       }
33.
34.       public int getId_Usuario() {
35.           return Id_Usuario;
36.       }
37.
38.       public void setId_Usuario(int Id_Usuario) {
39.           this.Id_Usuario = Id_Usuario;
40.       }
41.
42.       public String getNombre() {
43.           return nombre;
44.       }
45.
46.       public void setNombre(String nombre) {
47.           this.nombre = nombre;
48.       }
49.
50.       public String getApellidos() {
51.           return apellidos;
52.       }
53.
54.       public void setApellidos(String apellidos) {
55.           this.apellidos = apellidos;
56.       }
57.
58.       public String getDni() {
59.           return dni;
60.       }
61.
62.       public void setDni(String dni) {
63.           this.dni = dni;
64.       }
65.
66.       public String getDireccion() {
67.           return direccion;
68.       }
69.
70.       public void setDireccion(String direccion) {
71.           this.direccion = direccion;
72.       }
73.
74.       public String getMovil() {
75.           return movil;
76.       }
77.
78.       public void setMovil(String movil) {
79.           this.movil = movil;
80.       }
81.
82.       public String getUsername() {
83.           return username;
84.       }
85.
86.       public void setUsername(String username) {
87.           this.username = username;
88.       }
89.
90.       public String getPassw() {
91.           return passw;
92.       }
93.
94.       public void setPassw(String passw) {
95.           this.passw = passw;
96.       }
97.
98.       public byte getEnabled() {
99.           return enabled;
100.      }
101.
102.      public void setEnabled(byte enabled) {
103.          this.enabled = enabled;
104.      }
105.
106. }
```

UsuarioController.java

```java
package gymRestService;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.ImportResource;

/*
 * Clase que representa un controlador REST de usuarios. Mapea las
 * operaciones sobre recursos REST relacionados con usuarios y hace uso
 * del DAO para hacerlas efectivas en la base de datos.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

@RestController
@ImportResource("classpath:spring/beanLocations.xml")
public class UsuarioController {

    // Obtenemos el DAO mediante inyección de dependencias
    @Autowired
    private UsuarioDaoImpl usuarioDao;


    // (J) Comprueba si un usuario no está dado de alta aún
    @RequestMapping(value = "/checkUser/", method = RequestMethod.POST, consumes = "application/json")
    public @ResponseBody String checkUser (@RequestBody Usuario usuario) {

    return usuarioDao.checkUser(usuario.getUsername());
    }


    // (J) Añade un usuario
    @RequestMapping(value = "/gymRegister/", method = RequestMethod.POST, consumes = "application/json")
    public @ResponseBody String addUser (@RequestBody Usuario usuario) {

    return usuarioDao.addUser(usuario.getNombre(), usuario.getApellidos(), usuario.getDni(),
            usuario.getDireccion(), usuario.getMovil(),usuario.getUsername(), usuario.getPassw());
    }


    // (J,A) Comprueba que la información de acceso (user y pass) es correcta
    @RequestMapping(value = "/gym/login/", method = RequestMethod.POST, consumes = "application/json")
    public @ResponseBody String login(@RequestBody Usuario usuario) {

    return usuarioDao.login(usuario.getUsername(), usuario.getPassw());
    }


    // (A) Añade un usuario a una sesión
    @RequestMapping(value = "/gym/addUserSesion/{id_sesion}/{username}", method = RequestMethod.POST, consumes =
 "application/json")
    public @ResponseBody String addUsuarioSesion(
            @PathVariable("id_sesion") int id_sesion,
            @PathVariable("username") String username) {

    return usuarioDao.addUsuarioSesion(id_sesion, username);
    }

}
```

UsuarioDao.java

```java
package gymRestService;

import java.util.List;

/*
 * Clase que representa la interfaz DAO de usuarios. Abstrae las
 * operaciones necesarias sobre la base de datos relacionadas con los
 * usuarios a petición del controlador REST de usuarios.
```

```
 *
 * Diseño por Pablo Gallardo
 * Versión: 1.0
 *
 */

public abstract interface UsuarioDao {

    public abstract String checkUser(String user);
    public abstract String addUser (String nombre, String apellidos, String dni,
            String direccion, String movil, String username, String password);

    public abstract String getPass(String username); // Obtiene contraseña de un usuario
    public abstract String getRole(String username); // Obtiene rol de un usuario
    public abstract List<Usuario> getUsuario(String username);

    public abstract String login(String user, String passw);
    public abstract String addUsuarioSesion(int id_sesion, String username);

}
```

UsuarioDaoImpl.java

```
package gymRestService;

import java.util.List;

import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.security.crypto.password.PasswordEncoder;

/*
 * Clase que implementa la interfaz DAO de usuarios. Realiza las
 * operaciones necesarias sobre la base de datos relacionadas con los
 * uarios a petición del controlador REST de usuarios. No se
 * comentan los métodos debido a que se corresponden con las operaciones
 * del controlador, con lo que ya están descritos previamente.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

public class UsuarioDaoImpl implements UsuarioDao {

    JdbcTemplate jdbcTemplate;

    // Obtenemos el codificador mediante inyección de dependencias
    @Autowired
    private PasswordEncoder passwordEncoder;

    // Obtiene contraseña de un usuario
    public String getPass(String username) {
        String pass = jdbcTemplate.queryForObject("SELECT passw FROM usuarios WHERE username =?",
                new Object[] { username }, String.class);
        return pass;
    }

    // Obtiene rol de un usuario
    public String getRole(String username) {
        String role = jdbcTemplate.queryForObject("SELECT role FROM roles_usuarios WHERE username =?",
                new Object[] { username }, String.class);
        return role;
    }

    // Obtiene un usuario
    public List<Usuario> getUsuario(String username){
        return jdbcTemplate.query("SELECT * FROM usuarios WHERE username =?",
                new Object[] { username }, new UsuarioRowMapper());
    }


    public String addUser (String nombre, String apellidos, String dni,
            String direccion, String movil, String username, String password) {

        String encodedPassword = passwordEncoder.encode(password); // Codificamos la contraseña antes de guardar
la

        int consulta1 = jdbcTemplate.update("INSERT INTO usuarios "
                + "(nombre, apellidos, dni, direccion, movil, username, passw) "
```

```
58.                 + "VALUES(?,?,?,?,?,?,?)",
59.                 new Object[] { nombre, apellidos, dni, direccion, movil, username, encodedPassword });
60.
61.         int consulta2 = jdbcTemplate.update("INSERT INTO roles_usuarios (username, role) "
62.                 + "VALUES(?,'ROLE_USER')",
63.                 new Object[] { username });
64.
65.         // Devolvemos una cadena comunicando si la operación ha tenido éxito
66.         String resul ="";
67.         if (consulta1==0 || consulta2==0){
68.              resul = "[{\"nuevoRegistro\":\"0\"}]";
69.         }else{
70.             resul = "[{\"nuevoRegistro\":\"1\"}]";
71.         }
72.         return resul;
73.     }
74.
75.
76.     public String checkUser(String user) {
77.
78.         int count = jdbcTemplate.queryForObject("SELECT COUNT(*) FROM usuarios WHERE username=?",
79.                 new Object[] { user }, Integer.class);
80.
81.         // Devolvemos una cadena comunicando si la operación ha tenido éxito
82.         String resul="";
83.         if (count==0){
84.              resul = "[{\"uniqueUser\":\"1\"}]";
85.         }else{
86.             resul = "[{\"uniqueUser\":\"0\"}]";
87.         }
88.         return resul;
89.     }
90.
91.
92.     public String login(String user, String passw) {
93.
94.         String passBCrypt= jdbcTemplate.queryForObject("SELECT passw FROM usuarios WHERE username=?",
95.                 new Object[] { user }, String.class);
96.
97.         boolean count = passwordEncoder.matches(passw, passBCrypt); // Comprobamos la contraseña
98.
99.         // Devolvemos una cadena comunicando si la operación ha tenido éxito
100.        String resul="";
101.        if (count==false){
102.             resul = "[{\"logstatus\":\"0\"}]";
103.        }else{
104.            resul = "[{\"logstatus\":\"1\"}]";
105.        }
106.        return resul;
107.    }
108.
109.
110.    public String addUsuarioSesion(int id_sesion, String username) {
111.
112.        String resul="";
113.
114.        int id_usuario = jdbcTemplate.queryForObject("SELECT Id_Usuario FROM usuarios WHERE username = ?",
115.                new Object[] { username}, Integer.class);
116.
117.        int count = jdbcTemplate.queryForObject("SELECT COUNT(*) FROM sesionesdeusuario WHERE Id_Sesion=? AND Id
    _Usuario=?",
118.                new Object[] { id_sesion, id_usuario }, Integer.class);
119.
120.        // Comprobamos que el usuario no está subscrito a la sesión antes de añadirlo
121.        // Y devolvemos una cadena comunicando si toda la operación ha tenido éxito
122.        if (count==1){
123.            resul = "[{\"addUserSesion\":\"1\"}]";
124.        }else{
125.            int consulta = jdbcTemplate.update("INSERT INTO sesionesdeusuario (Id_Sesion, Id_Usuario) VALUES(?,
    ?)",
126.                    new Object[] { id_sesion, id_usuario });
127.
128.            if(consulta==1){
129.                resul = "[{\"addUserSesion\":\"1\"}]";
130.            } else{
131.                resul = "[{\"addUserSesion\":\"0\"}]";
132.            }
133.        }
134.        return resul;
135.    }
136.
137.
138.    // Inyección del dataSource mediante el constructor
139.    public void setDataSource(DataSource dataSource) {
```

```
140.     this.jdbcTemplate = new JdbcTemplate(dataSource);
141.     }
142.
143.     public UsuarioDaoImpl() {
144.     }
145.
146. }
```

UsuarioRowMapper.java

```
1.  package gymRestService;
2.
3.  import java.sql.ResultSet;
4.  import java.sql.SQLException;
5.  import org.springframework.jdbc.core.RowMapper;
6.
7.  /*
8.   * Clase que representa un mapeador de usuarios. Obtiene y crea un
9.   * usuario a partir de los datos obtenidos de la base de datos.
10.  * Este RowMapper es para obtener los datos de la tabla 'usuarios'
11.  *
12.  * Diseño por Pablo Gallardo Hiraldo
13.  * Versión: 1.0
14.  *
15.  */
16.
17. public class UsuarioRowMapper implements RowMapper<Usuario> {
18.
19.     public Usuario mapRow(ResultSet resultSet, int arg1) throws SQLException {
20.         // getX(index) para coger los elementos recibidos por la tabla de datos
21.         Usuario usuario = new Usuario(resultSet.getInt(1), resultSet.getString(2),
22.         resultSet.getString(3), resultSet.getString(4), resultSet.getString(5),
23.         resultSet.getString(6), resultSet.getString(7), resultSet.getString(8), resultSet.getByte(9));
24.
25.     return usuario;
26.     }
27. }
```

2.2 Seguridad Servicio Web

OAuth2SecurityConfiguration.java

```
1.  package gymRestService.security;
2.
3.  import javax.sql.DataSource;
4.  import org.springframework.beans.factory.annotation.Autowired;
5.  import org.springframework.beans.factory.annotation.Qualifier;
6.  import org.springframework.context.annotation.Bean;
7.  import org.springframework.context.annotation.Configuration;
8.  import org.springframework.security.authentication.AuthenticationManager;
9.  import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
10. import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
11. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
12. import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
13. import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
14. import org.springframework.security.core.userdetails.UserDetailsService;
15. import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
16. import org.springframework.security.crypto.password.PasswordEncoder;
17. import org.springframework.security.oauth2.provider.ClientDetailsService;
18. import org.springframework.security.oauth2.provider.approval.ApprovalStore;
19. import org.springframework.security.oauth2.provider.approval.TokenApprovalStore;
20. import org.springframework.security.oauth2.provider.approval.TokenStoreUserApprovalHandler;
21. import org.springframework.security.oauth2.provider.request.DefaultOAuth2RequestFactory;
22. import org.springframework.security.oauth2.provider.token.TokenStore;
23. import org.springframework.security.oauth2.provider.token.store.InMemoryTokenStore;
24.
25. /*
26.  * Clase que implementa toda la seguridad de la aplicación.
27.  *
28.  * Diseño por Pablo Gallardo Hiraldo
29.  * Versión: 1.0
```

```
30.  *
31.  */
32.
33. @Configuration
34. @EnableWebSecurity
35. public class OAuth2SecurityConfiguration extends WebSecurityConfigurerAdapter {
36.
37.     @Autowired
38.     private ClientDetailsService clientDetailsService;
39.
40.     @Autowired
41.     @Qualifier("customUserDetailsService")
42.     private UserDetailsService userDetailsService;
43.
44.     @Autowired
45.     DataSource dataSource;
46.
47.     @Autowired
48.     public void globalUserDetails(AuthenticationManagerBuilder auth) throws Exception {
49.         // Aquí configuramos los propietarios de los recursos
50.         // Se accede a la BBDD parar comprobar si el solicitante tiene autorización
51.         auth.userDetailsService(userDetailsService);
52.         auth.authenticationProvider(authenticationProvider());
53.     }
54.
55.     @Override
56.     protected void configure(HttpSecurity http) throws Exception {
57.         http
58.         .csrf().disable()
59.         .anonymous().disable()
60.         .authorizeRequests()   //El endpoint '/oauth/token' se usa para solicitar un token de acceso
61.         .antMatchers("/oauth/token").permitAll();
62.     }
63.
64.     @Override
65.     @Bean
66.     public AuthenticationManager authenticationManagerBean() throws Exception {
67.         return super.authenticationManagerBean();
68.     }
69.
70.     @Bean
71.     public TokenStore tokenStore() {
72.         return new InMemoryTokenStore();
73.     }
74.
75.     @Bean
76.     @Autowired
77.     public TokenStoreUserApprovalHandler userApprovalHandler(TokenStore tokenStore){
78.         TokenStoreUserApprovalHandler handler = new TokenStoreUserApprovalHandler();
79.         handler.setTokenStore(tokenStore);
80.         handler.setRequestFactory(new DefaultOAuth2RequestFactory(clientDetailsService));
81.         handler.setClientDetailsService(clientDetailsService);
82.         return handler;
83.     }
84.
85.     @Bean
86.     @Autowired
87.     public ApprovalStore approvalStore(TokenStore tokenStore) throws Exception {
88.         TokenApprovalStore store = new TokenApprovalStore();
89.         store.setTokenStore(tokenStore);
90.         return store;
91.     }
92.
93.     // Nuestro encoder para codificar las contraseñas y para verificar las que nos lleguen
94.     @Bean
95.     public PasswordEncoder passwordEncoder() {
96.          return new BCryptPasswordEncoder(12); // Cuanto mayor sea el número más segura es la codificación
97.     }
98.
99.     // Nos permite comunicarnos con la base de datos para conseguir la info de un usuario y comprobar con ellos
    las credenciales
100.     @Bean
101.     public DaoAuthenticationProvider authenticationProvider() {
102.         DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
103.         authenticationProvider.setUserDetailsService(userDetailsService);
104.         authenticationProvider.setPasswordEncoder(passwordEncoder());
105.         return authenticationProvider;
106.     }
107.
108. }
```

AuthorizationServerConfiguration.java

```java
package gymRestService.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;
import org.springframework.security.oauth2.provider.approval.UserApprovalHandler;
import org.springframework.security.oauth2.provider.token.TokenStore;

/*
 * Clase responsable de verificar las credenciales
 * y si éstas son correctas, provee los tokens de acceso.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

@SuppressWarnings("deprecation")
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfiguration extends AuthorizationServerConfigurerAdapter {

    private static String REALM="MY_OAUTH_REALM";

    @Autowired
    private TokenStore tokenStore; //Para almacenar los tokens

    @Autowired
    private UserApprovalHandler userApprovalHandler;

    @Autowired
    @Qualifier("authenticationManagerBean")
    private AuthenticationManager authenticationManager;

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

        //Registramos las credenciales con las que el cliente accederá para recibir el token
        clients.inMemory()
            .withClient("trusted-monitor")
            .authorizedGrantTypes("password", "authorization_code", "refresh_token", "implicit")
            .authorities("ROLE_CLIENT", "ROLE_TRUSTED_CLIENT")
            .scopes("read", "write", "trust")
            .secret("topsecret")
            .accessTokenValiditySeconds(900); //Access token solo es valido por 15 minutos
            //.refreshTokenValiditySeconds(1800); //Refresh token solo es valido por 30 minutos
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        endpoints.tokenStore(tokenStore).userApprovalHandler(userApprovalHandler)
                .authenticationManager(authenticationManager);
    }


    @Override
    public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {
        oauthServer.realm(REALM+"/client");
        oauthServer.passwordEncoder(NoOpPasswordEncoder.getInstance()); // Para que no use BCrypt aquí
    }

}
```

ResourceServerConfiguration.java

```java
package gymRestService.security;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;
import org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;

import org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecurityConfigurer;
import org.springframework.security.oauth2.provider.error.OAuth2AccessDeniedHandler;

/*
 * Clase que hospeda los recursos (nuestra api rest) en los
 * que el cliente está interesado
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

@Configuration
@EnableResourceServer //Activa un filtro de Spring Security que autentifica peticiones usando un token Oauth2 en trante
public class ResourceServerConfiguration extends ResourceServerConfigurerAdapter {

    private static final String RESOURCE_ID = "rest-service-monitor";

    @Override
    public void configure(ResourceServerSecurityConfigurer resources) {
        resources.resourceId(RESOURCE_ID).stateless(false);
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
        .antMatchers("/gymRegister/**").permitAll(); //Permitimos el acceso para poder registrarse
        http.
        anonymous().disable()
        .requestMatchers().antMatchers("/gym/**") //Resto de recursos del servidor necesitan autorización para acceder
        .and().authorizeRequests()
        .antMatchers("/gym/**").access("hasRole('USER')")
        .and().exceptionHandling().accessDeniedHandler(new OAuth2AccessDeniedHandler());
    }

}
```

CustomUserDetailsService.java

```java
package gymRestService.security;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import gymRestService.UsuarioDaoImpl;

/*
 * Clase que obtiene información sobre un usuario de la base de datos
 * para luego ser usada por ejemplo para la autenticación
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

@Service("customUserDetailsService")
public class CustomUserDetailsService implements UserDetailsService{

    // Obtenemos el DAO mediante inyección de dependencias
    @Autowired
```

```
30.     private UsuarioDaoImpl usuarioDao;
31.
32.     // Carga la información de un usuario a través de su username
33.     @Transactional(readOnly=true)
34.     public UserDetails loadUserByUsername(String user) throws UsernameNotFoundException {
35.
36.         String pass = usuarioDao.getPass(user);
37.         System.out.println("User : "+user + pass);
38.         if(user==null){
39.             System.out.println("User not found");
40.             throw new UsernameNotFoundException("Username not found");
41.         }
42.             return new org.springframework.security.core.userdetails.User(user, pass, true, true, true, true, ge
    tGrantedAuthorities(user));
43.     }
44.
45.     // Carga el rol del usuario
46.     private List<GrantedAuthority> getGrantedAuthorities(String user){
47.         List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
48.
49.             String role = usuarioDao.getRole(user);
50.             authorities.add(new SimpleGrantedAuthority(role));
51.
52.         System.out.print("authorities :"+authorities);
53.         return authorities;
54.     }
55.
56. }
```

MethodSecurityConfig.java

```
1.  package gymRestService.security;
2.
3.  import org.springframework.context.annotation.Configuration;
4.  import org.springframework.security.access.expression.method.MethodSecurityExpressionHandler;
5.  import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
6.  import org.springframework.security.config.annotation.method.configuration.GlobalMethodSecurityConfiguration;
7.  import org.springframework.security.oauth2.provider.expression.OAuth2MethodSecurityExpressionHandler;
8.
9.  /*
10.  * Activa GlobalMethodSecurity, lo que activa las anotaciones @PreFilter,
11.  * @PostFilter, @PreAuthorize @PostAuthorize por si queremos usarlas.
12.  *
13.  * Diseño por Pablo Gallardo Hiraldo
14.  * Versión: 1.0
15.  *
16.  */
17.
18. @Configuration
19. @EnableGlobalMethodSecurity(prePostEnabled = true, proxyTargetClass = true)
20. public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {
21.
22.     @Override
23.     protected MethodSecurityExpressionHandler createExpressionHandler() {
24.         return new OAuth2MethodSecurityExpressionHandler();
25.     }
26.
27. }
```

Initializer.java

```
1.  package gymRestService.security;
2.
3.  import javax.servlet.ServletContext;
4.  import javax.servlet.ServletException;
5.  import org.springframework.boot.web.servlet.ServletContextInitializer;
6.  import org.springframework.context.annotation.Configuration;
7.
8.  /*
9.   * Clase que inicializa el ServletContext nada más ejecutarse el programa
10.  * para que la configuración de seguridad pueda disponer de el inmediatamente.
11.  *
12.  * Diseño por Pablo Gallardo Hiraldo
13.  * Versión: 1.0
14.  *
15.  */
16.
```

```
@Configuration
public class Initializer implements ServletContextInitializer {

    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        System.err.println("Iniciando...");
    }

}
```

2.3 Configuración Servicio Web

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springframework</groupId>
    <artifactId>rest-service-monitor</artifactId>
    <version>0.1.0</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.1.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

         <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </dependency>

        <dependency>
            <groupId>com.jayway.jsonpath</groupId>
            <artifactId>json-path</artifactId>
            <scope>test</scope>
        </dependency>

       <dependency>
          <groupId>mysql</groupId>
          <artifactId>mysql-connector-java</artifactId>
          <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
        </dependency>

        <dependency>
```

```
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.security.oauth</groupId>
            <artifactId>spring-security-oauth2</artifactId>
            <version>2.0.10.RELEASE</version>
        </dependency>

    </dependencies>


    <properties>
        <java.version>1.8</java.version>
    </properties>


    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>


</project>
```

application.properties

```
server.port=9999
server.ssl.key-alias=selfsigned_localhost_sslserver
server.ssl.key-password=changeit
server.ssl.key-store=classpath:ssl-server.jks
server.ssl.key-store-provider=SUN
server.ssl.key-store-type=JKS
```

database.properties

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/datosgym
jdbc.username=root
jdbc.password=
```

beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

    <bean id="usuarioDao" class="gymRestService.UsuarioDaoImpl" >
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <bean id="sesionDao" class="gymRestService.SesionDaoImpl" >
        <property name="dataSource" ref="dataSource"></property>
    </bean>
    <bean id="ejercicioDao" class="gymRestService.EjercicioDaoImpl" >
        <property name="dataSource" ref="dataSource"></property>
    </bean>

</beans>
```

beanLocations.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <import resource="beans.xml" />
    <import resource="datasource.xml" />
</beans>
```

dataSource.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">

    <bean
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location">
            <value>properties/database.properties</value>
        </property>
    </bean>
    <bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>
    <bean id="jdbcTemplate"
    class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>
</beans>
```

2.4 Aplicación Java

GestorPeticiones.java

```java
package monitorgym;

import java.io.BufferedReader;
import java.io.OutputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Base64;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLSession;

import org.json.JSONArray;
import monitorgym.AuthTokenInfo;

/*
 * Clase que gestionará todas las conexiones con el servidor.
 * Cada vez que se realice una petición al servidor, deberá
 * utilizar un método de esta clase.
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

```

```
27. public class GestorPeticiones {
28. 
29.     private String SERVER_PATH;
30.     private final String SERVER_PORT="9999";
31. 
32.     /*
33.      * Envía un POST request a '/oauth/token' para conseguir un token de acceso,
34.      * el cual será mandado en cada request posterior
35.      */
36.     public String sendTokenRequest (String username, String password) throws Exception{
37. 
38.         StringBuffer response = null;
39.         JSONArray object = null;
40. 
41.         try{
42.             // Actualizamos la IP del servidor por si ha habido algun cambio
43.             actualizaIp();
44.             // Establecemos conexión con el servidor
45.             String urlString = SERVER_PATH+":"+SERVER_PORT+"/oauth/token";//?grant_type=password&username="+user
    name+"&password="+password;
46.             URL urlObj = new URL(urlString);
47.             HttpsURLConnection con = (HttpsURLConnection) urlObj.openConnection();
48. 
49.             //Esto solo lo hacemos para que Java no de problemas con el certificado self-signed.
50.             con.setHostnameVerifier(new HostnameVerifier(){
51.                 public boolean verify(String hostname, SSLSession session){
52.                     return true;
53.                 }
54.             });
55. 
56.             // Credenciales con las que solicitamos el token de acceso
57.             String userCredentials = "trusted-monitor:topsecret";
58.             String basicAuth = "Basic " + new String(Base64.getEncoder().encode(userCredentials.getBytes()));
59.             // Mandamos las credenciales en el header http
60.             con.setRequestProperty ("Authorization", basicAuth);
61.             con.setRequestMethod("POST");
62.             con.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
63.             con.setUseCaches(false);
64.             con.setDoInput(true);
65.             con.setDoOutput(true);
66. 
67.             // Enviamos la petición por POST con las credenciales de usuario
68.             OutputStream os = con.getOutputStream();
69.             String input = "grant_type=password&username="+username+"&password="+password;
70.             os.write(input.getBytes());
71.             os.flush();
72. 
73.             int responseCode = con.getResponseCode();
74.             System.out.println("\n Sending request to URL : " + urlString);
75.             System.out.println("Response Code : " + responseCode);
76. 
77.             // Leemos la respuesta
78.             BufferedReader in = new BufferedReader(
79.                     new InputStreamReader(con.getInputStream(), "UTF-8"));
80.             String inputLine;
81.             response = new StringBuffer();
82. 
83.             while ((inputLine = in.readLine()) != null) {
84.                 response.append(inputLine);
85.             }
86.             // Convertimos a formato JSON
87.             String responseSTR = response.toString();
88.             String responseJson = "["+responseSTR+"]";
89.             object = new JSONArray(responseJson);
90.             AuthTokenInfo tokenInfo = null;
91. 
92.             // Guardamos toda la información relativa al token en un objeto de tipo AuthTokenInfo
93.             if(object!=null){
94.                 tokenInfo = new AuthTokenInfo();
95.                 tokenInfo.setAccess_token(object.getJSONObject(0).getString("access_token"));
96.                 tokenInfo.setToken_type(object.getJSONObject(0).getString("token_type"));
97.                 tokenInfo.setRefresh_token(object.getJSONObject(0).getString("refresh_token"));
98.                 tokenInfo.setExpires_in(object.getJSONObject(0).getInt("expires_in"));
99.                 tokenInfo.setScope(object.getJSONObject(0).getString("scope"));
100.                System.out.println(tokenInfo);
101.            }else{
102.                System.out.println("No user exist----------");
103.            }
104.          //Cerramos la conexión
105.            in.close();
106. 
107.        } catch (Exception e){
108.            throw e;
109.        }
110. 
```

```
111.         //Devolvemos el token de acceso para para usarlo en cada request que hagamos
112.         return object.getJSONObject(0).getString("access_token");
113.     }
114.
115.     /*
116.      * Envía las peticiones al servidor
117.      */
118.     private JSONArray peticion(String url,String params, String access_token) throws Exception{
119.
120.         StringBuffer response = null;
121.         JSONArray object = null;
122.
123.         try{
124.             // Actualizamos la IP del servidor por si ha habido algun cambio
125.             actualizaIp();
126.
127.             String urlString ="";
128.             String bearerAuth="";
129.             // Añadimos el token de acceso a todas las peticiones excepto a las que vayan dirigidas
130.             // al registro de un nuevo usuario en la aplicación, ya que son de libre acceso
131.             if(url=="gymRegister/") {
132.                 urlString = SERVER_PATH +":"+SERVER_PORT+"/"+url;
133.             }else if (url=="checkUser/") {
134.                 urlString = SERVER_PATH +":"+SERVER_PORT+"/"+url;
135.             }else {
136.                 urlString = SERVER_PATH +":"+SERVER_PORT+"/"+url;//+"?access_token="+ access_token;
137.                 bearerAuth="Bearer "+access_token;
138.
139.             }
140.             // Establecemos conexión con el servidor
141.             URL urlObj = new URL(urlString);
142.             HttpsURLConnection con = (HttpsURLConnection) urlObj.openConnection();
143.
144.             //Esto solo lo hacemos para que Java no de problemas con el certificado self-signed.
145.             con.setHostnameVerifier(new HostnameVerifier(){
146.                 public boolean verify(String hostname, SSLSession session){
147.                     return true;
148.                 }
149.             });
150.
151.             if(bearerAuth!="") {
152.             con.setRequestProperty ("Authorization", bearerAuth);
153.             }
154.
155.             // Según el tipo de petición, definimos distintas propiedades en la cabecera
156.             if(params=="GET") {
157.                 con.setRequestProperty("Accept", "application/json");
158.                 con.setRequestMethod("GET");
159.             }else if(params=="DELETE") {
160.                 con.setRequestMethod("DELETE");
161.             }else {
162.                 con.setDoOutput(true);
163.                 con.setRequestMethod("POST");
164.                 con.setRequestProperty("Content-Type", "application/json");
165.
166.                 // Enviamos la petición por POST
167.                 String input = params;
168.                 OutputStream os = con.getOutputStream();
169.                 os.write(input.getBytes());
170.                 os.flush();
171.             }
172.
173.             int responseCode = con.getResponseCode();
174.             System.out.println("\nSending request to URL : " + url);
175.             System.out.println("Response Code : " + responseCode);
176.
177.             // Leemos la respuesta
178.             BufferedReader in = new BufferedReader(
179.                     new InputStreamReader(con.getInputStream(), "UTF-8"));
180.             String inputLine;
181.             response = new StringBuffer();
182.
183.             while ((inputLine = in.readLine()) != null) {
184.                 response.append(inputLine);
185.             }
186.             // Convertimos a formato JSON
187.             String responseSTR = response.toString();
188.             object = new JSONArray(responseSTR);
189.
190.             //Mostramos la respuesta del servidor por consola
191.             if(object.length()>0){
192.                 System.out.println("Respuesta del servidor: "+ object.getJSONObject(0).toString());
193.                 System.out.println();
194.             }
```

```
195.            //Cerramos la conexión
196.            in.close();
197.
198.        } catch (Exception e){
199.            throw e;
200.        }
201.        //Devolvemos objeto JSON con la respuesta completa del servidor
202.        return object;
203.    }
204.
205.
206.    /*
207.     * Método para actualizar la IP del servidor según la configurada por el usuario
208.     */
209.    private void actualizaIp(){
210.        ConfiguracionIpServidor config = new ConfiguracionIpServidor();
211.        SERVER_PATH = "https://"+config.getIpServer();
212.    }
213.
214.    /*
215.     * Método para enviar las credenciales de usuario al servidor. Éste nos
216.     * indicará en la respuestas si son válidas o no.
217.     */
218.    public String login(String usuario, String passw, String access_token) throws Exception{
219.        String url = "gym/login/";
220.        String params = "{\"username\":\""+usuario+"\", \"passw\":\""+passw+"\" }";
221.        String codigo = "logstatus";
222.        return this.peticion(url, params, access_token).getJSONObject(0).getString(codigo);
223.    }
224.
225.    /*
226.     * Método para comprobar si un usuario está disponible. Éste nos
227.     * indicará en la respuestaa si es válido o no.
228.     */
229.    public String checkUser(String usuario) throws Exception{
230.        String url = "checkUser/";
231.        String params = "{\"username\":\""+usuario+"\"}";
232.        String codigo = "uniqueUser";
233.        return this.peticion(url, params, "").getJSONObject(0).getString(codigo);
234.    }
235.
236.    /*
237.     * Método para enviar al servidor la información relativa a un nuevo usuario del sistema.
238.     */
239.    public String register(String username, String password, String nombre, String apellidos, String dni,
240.            String direccion, String movil) throws Exception{
241.        String url = "gymRegister/";
242.        String params = "{\"username\":\""+username+"\", \"passw\":\""+password+"\" "
243.                        +",\"nombre\":\""+nombre+"\", \"apellidos\":\""+apellidos+"\" "
244.                        +",\"dni\":\""+dni+"\", \"direccion\":\""+direccion+"\" "
245.                        +",\"movil\":\""+movil+"\" }";
246.        String codigo = "nuevoRegistro";
247.        return this.peticion(url, params, "").getJSONObject(0).getString(codigo);
248.    }
249.
250.    /*
251.     * Método para enviar al servidor la información relativa a una sesión.
252.     */
253.    public String nuevaSesion(String nomSesion, String monitor, String fecha, String numUsuarios, String access_
    token) throws Exception{
254.        String url = "gym/sesionAdd/";
255.
256.        String params = "{\"user_Monitor\":\""+monitor+"\", \"nombre_Sesion\":\""+nomSesion+"\" "
257.                        +",\"fecha\":\""+fecha+"\", \"num_Usuarios\":\""+numUsuarios+"\" "
258.                        +",\"activa\":\""+"1"+"\" }";
259.        String codigo = "nuevaSesion";
260.        return this.peticion(url, params, access_token).getJSONObject(0).getString(codigo);
261.    }
262.
263.    /*
264.     * Método para solicitar al servidor todas las sesiones creadas por
265.     * cierto monitor.
266.     */
267.    public JSONArray sesionesMonitor(String monitor, String access_token) throws Exception{
268.        String url = "gym/sesionMonitor/"+monitor+"/";
269.        String params = "GET";
270.        return this.peticion(url, params, access_token);
271.    }
272.
273.    /*
274.     * Método que solicita al servidor toda la información de una sesión
275.     */
276.    public JSONArray sesion(String id, String access_token) throws Exception{
277.        String url = "gym/sesion/"+id+"/";
278.        String params = "GET";
```

```
279.         return this.peticion(url, params, access_token);
280.     }
281.
282.     /*
283.      * Método para indicarle al servidor que debe eliminar cierta sesión.
284.      */
285.     public JSONArray eliminaSesion(String id, String access_token) throws Exception{
286.         String url = "gym/sesion/"+id+"/";
287.         String params = "DELETE";
288.         return this.peticion(url, params, access_token);
289.     }
290.
291.     /*
292.      * Método para obtener del servidor toda la información de cierto ejercicio
293.      */
294.     public JSONArray ejercicio(String idEjercicio, String access_token) throws Exception{
295.         String url = "gym/ejercicio/"+idEjercicio+"/";
296.         String params = "GET";
297.         return this.peticion(url, params, access_token);
298.     }
299.
300.     /*
301.      * Método para obtenet todos los ejercicios almacenados en la base de datos.
302.      */
303.     public JSONArray ejercicios(String access_token) throws Exception{
304.         String url = "gym/ejercicio/";
305.         String params = "GET";
306.         return this.peticion(url, params, access_token);
307.     }
308.
309.     /*
310.      * Método para solicitar al servidor todos los ejercicios que no han sido
311.      * incluidos en una sesión.
312.      */
313.     public JSONArray ejerciciosNoSesion(String idSesion, String access_token) throws Exception{
314.         String url = "gym/ejercicioNoSesion/"+idSesion+"/";
315.         String params = "GET";
316.         return this.peticion(url, params, access_token);
317.     }
318.
319.     /*
320.      * Método para solicitar al servidor todos los ejercicios que forman
321.      * parte de una sesión.
322.      */
323.     public JSONArray ejerciciosSesion(String idSesion, String access_token) throws Exception{
324.         String url = "gym/ejercicioSesion/"+idSesion+"/";
325.         String params = "GET";
326.         return this.peticion(url, params, access_token);
327.     }
328.
329.     /*
330.      * Método para añadir un ejercicio a una sesión.
331.      */
332.     public JSONArray addEjercicioSesion(String idSesion, String idEjercicio, String access_token) throws Excepti
    on{
333.         String url = "gym/ejercicio/"+idSesion+"/"+idEjercicio+"/";
334.         String params = "{\"id_Ejercicio\":\""+idEjercicio+"\"}";
335.         return this.peticion(url, params, access_token);
336.     }
337.
338.     /*
339.      * Método para eliminar un ejercicio de una sesión.
340.      */
341.     public JSONArray removeEjercicioSesion(String idSesion, String idEjercicio, String access_token) throws Exce
    ption{
342.         String url = "gym/ejercicio/"+idSesion+"/"+idEjercicio+"/";
343.         String params = "DELETE";
344.         return this.peticion(url, params, access_token);
345.     }
346.
347.     /*
348.      * Método utilizado para actualizar la IP del monitor asociada a una sesión.
349.      * Con este método evitamos problemas si el monitor quiere  monitorizar a los
350.      * usuarios desde un PC distinto desde el que creó la sesión.
351.      */
352.     public JSONArray actualizaIpMonitorSesion(String idSesion, int puerto, String access_token) throws Exception
    {
353.         String url = "gym/sesionMonitor/";
354.         String params = "{\"id_Sesion\":\""+idSesion+"\", \"ip_Monitor\":\""+puerto+"\" }";
355.         return this.peticion(url, params, access_token);
356.     }
357.
358.     /*
359.      * Método para indicar al servidor que una sesión ha finalizado,
```

```
     * pasando a estar en el estado de NO activa
     */
    public JSONArray setSesionNoActiva(String idSesion, String access_token) throws Exception{
        String url = "gym/sesion/";
        String params = "{\"id_Sesion\":\""+idSesion+"\"}";
        return this.peticion(url, params, access_token);
    }
}
```

AuthTokenInfo.java

```
package monitorgym;

/*
 * Clase que representa un token
 *
 * Diseño por Pablo Gallardo Hiraldo
 * Versión: 1.0
 *
 */

public class AuthTokenInfo {

    private String access_token;
    private String token_type;
    private String refresh_token;
    private int expires_in;
    private String scope;
    public String getAccess_token() {
        return access_token;
    }
    public void setAccess_token(String access_token) {
        this.access_token = access_token;
    }
    public String getToken_type() {
        return token_type;
    }
    public void setToken_type(String token_type) {
        this.token_type = token_type;
    }
    public String getRefresh_token() {
        return refresh_token;
    }
    public void setRefresh_token(String refresh_token) {
        this.refresh_token = refresh_token;
    }
    public int getExpires_in() {
        return expires_in;
    }
    public void setExpires_in(int expires_in) {
        this.expires_in = expires_in;
    }
    public String getScope() {
        return scope;
    }
    public void setScope(String scope) {
        this.scope = scope;
    }
    @Override
    public String toString() {
        return "AuthTokenInfo [access_token=" + access_token + ", token_type=" + token_type + ", refresh_token="
                + refresh_token + ", expires_in=" + expires_in + ", scope=" + scope + "]";
    }

}
```

VentanaRegistro.java

```
package monitorgym;

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

```
10. import javax.swing.JLabel;
11. import javax.swing.ImageIcon;
12. 
13. import java.awt.Font;
14. import java.awt.event.ActionListener;
15. import java.awt.event.ActionEvent;
16. 
17. /*
18.  * Ventana de usuario en la que se mostrará un formulario
19.  * de registro en el sistema.
20.  */
21. 
22. @SuppressWarnings("serial")
23. public class VentanaRegistro extends JFrame {
24. 
25.     private JPanel contentPane;
26.     private JTextField txtUsuario, txtNombre, txtApellidos, txtDni, txtDireccion, txtMovil;
27.     private JPasswordField pwdPassword;
28.     private JLabel label;
29.     private JButton btnRegistrar, btnContinuar;
30.     private GestorPeticiones gestor = new GestorPeticiones();
31.     private String usuario, passwd, nombre, apellidos, dni, direccion, movil;
32. 
33.     /**
34.      * Create the frame.
35.      */
36.     public VentanaRegistro() {
37.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
38.         setBounds(100, 100, 570, 666);
39.         setLocationRelativeTo(null); // Ventana centrada
40.         setTitle("Registro");
41.         setResizable(false);
42. 
43.         contentPane = new JPanel();
44.         contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
45.         setContentPane(contentPane);
46.         contentPane.setLayout(null);
47. 
48.         // Campo para introducir el nombre de usuario
49.         txtUsuario = new JTextField();
50.         txtUsuario.setText("usuario");
51.         txtUsuario.setBounds(284, 228, 86, 28);
52.         contentPane.add(txtUsuario);
53.         txtUsuario.setColumns(10);
54. 
55.         // Campo para introducir la contraseña del usuario
56.         pwdPassword = new JPasswordField();
57.         pwdPassword.setText("password");
58.         pwdPassword.setBounds(284, 267, 86, 28);
59.         contentPane.add(pwdPassword);
60. 
61.         // Campo para introducir nombre del usuario
62.         txtNombre = new JTextField();
63.         txtNombre.setText("nombre");
64.         txtNombre.setBounds(284, 306, 86, 28);
65.         contentPane.add(txtNombre);
66.         txtNombre.setColumns(10);
67. 
68.         // Campo para introducir apellidos del usuario
69.         txtApellidos = new JTextField();
70.         txtApellidos.setText("apellidos");
71.         txtApellidos.setBounds(284, 345, 86, 28);
72.         contentPane.add(txtApellidos);
73.         txtApellidos.setColumns(10);
74. 
75.         // Campo para introducir dni del usuario
76.         txtDni = new JTextField();
77.         txtDni.setText("dni");
78.         txtDni.setBounds(284, 384, 86, 28);
79.         contentPane.add(txtDni);
80.         txtDni.setColumns(10);
81. 
82.         // Campo para introducir la direccion del usuario
83.         txtDireccion = new JTextField();
84.         txtDireccion.setText("direccion");
85.         txtDireccion.setBounds(284, 423, 86, 28);
86.         contentPane.add(txtDireccion);
87.         txtDireccion.setColumns(10);
88. 
89. 
90.         // Campo para introducir el movil del usuario
91.         txtMovil = new JTextField();
92.         txtMovil.setText("movil");
93.         txtMovil.setBounds(284, 462, 86, 28);
```

```
94.         contentPane.add(txtMovil);
95.         txtMovil.setColumns(10);
96.
97.
98.
99.         label = new JLabel(" ");
100.        label.setBounds(81, 499, 424, 20);
101.        label.setHorizontalAlignment(SwingConstants.CENTER);
102.        contentPane.add(label);
103.
104.
105.        // Botón para enviar los datos de registro al sistema
106.        btnRegistrar = new JButton("REGISTRAR");
107.        btnRegistrar.addActionListener(new ActionListener() {
108.            public void actionPerformed(ActionEvent e) {
109.                try{
110.                    // Obtenemos la información introducida por el usuario
111.                    usuario = txtUsuario.getText();
112.                    passwd = new String(pwdPassword.getPassword());
113.                    nombre = txtNombre.getText();
114.                    apellidos = txtApellidos.getText();
115.                    dni = txtDni.getText();
116.                    direccion = txtDireccion.getText();
117.                    movil = txtMovil.getText();
118.
119.                    // Consulta al servidor para comprobar si ya hay un usuario con el mismo nombre
120.                    if (gestor.checkUser(usuario).equals("1")){
121.                        // Enviamos los datos para registrarlos en la base de datos
122.                        if (gestor.register(usuario, passwd, nombre, apellidos, dni, direccion, movil).equals("1
    ")){
123.
124.                        label.setText("Registro completo. Pulse CONTINUAR para poder acceder");
125.
126.                        btnContinuar = new JButton("CONTINUAR");
127.                        btnContinuar.addActionListener(new ActionListener() {
128.                            public void actionPerformed(ActionEvent e) {
129.                                try{
130.                                    // Lanzamos de nuevo la ventana login para acceder
131.                                    VentanaLogin ventana = new VentanaLogin();
132.                                    ventana.setVisible(true);
133.                                    dispose();
134.
135.                                } catch(Exception ex){
136.                                    // Si no es posible conectar con el servidor se lo indicamos al usuario
137.                                    System.err.println("Obtenido el siguiente error: " + ex.getMessage());
138.                                    label.setText("Problemas con el servidor. Vuelva a intentarlo en unos minuto
    s");
139.                                }
140.                            }
141.                        });
142.                        btnContinuar.setBounds(228, 570, 112, 28);
143.                        contentPane.add(btnContinuar);
144.
145.                        } else{
146.                            // Le mostramos al usuario que ha habido algún problema al efectur el registro
147.                            label.setText("No ha sido posible completar el registro");
148.                        }
149.
150.                    } else{
151.                        // Le mostramos al usuario el usuario introducido no está disponible
152.                        label.setText("Usuario repetido. Introduzca uno nuevo por favor");
153.                    }
154.                } catch(Exception ex){
155.                    // Si no es posible conectar con el servidor se lo indicamos al usuario
156.                    System.err.println("Obtenido el siguiente error: " + ex.getMessage());
157.                    label.setText("Problemas con el servidor. Vuelva a intentarlo en unos minutos");
158.                }
159.            }
160.        });
161.        btnRegistrar.setBounds(228, 530, 112, 28);
162.        contentPane.add(btnRegistrar);
163.
164.        JLabel lblUsuario = new JLabel("Usuario:");
165.        lblUsuario.setBounds(197, 235, 60, 14);
166.        contentPane.add(lblUsuario);
167.
168.        JLabel lblContrasea = new JLabel("Contrase\u00F1a:");
169.        lblContrasea.setBounds(197, 274, 76, 14);
170.        contentPane.add(lblContrasea);
171.
172.        JLabel lblNombre = new JLabel("Nombre:");
173.        lblNombre.setBounds(197, 313, 76, 14);
174.        contentPane.add(lblNombre);
175.
176.        JLabel lblApellidos = new JLabel("Apellidos:");
```

```
177.         lblApellidos.setBounds(197, 352, 76, 14);
178.         contentPane.add(lblApellidos);
179.
180.         JLabel lblDni = new JLabel("DNI:");
181.         lblDni.setBounds(197, 391, 76, 14);
182.         contentPane.add(lblDni);
183.
184.         JLabel lblDireccion = new JLabel("Direccion:");
185.         lblDireccion.setBounds(197, 430, 76, 14);
186.         contentPane.add(lblDireccion);
187.
188.         JLabel lblMovil = new JLabel("Movil:");
189.         lblMovil.setBounds(197, 469, 76, 14);
190.         contentPane.add(lblMovil);
191.
192.
193.
194.         JLabel label_1 = new JLabel("");
195.         label_1.setIcon(new ImageIcon(VentanaLogin.class.getResource("/icons/logo.png")));
196.         label_1.setBounds(189, 5, 192, 189);
197.         contentPane.add(label_1);
198.
199.         JLabel label_2 = new JLabel("Introduzca sus datos para registrarse");
200.         label_2.setBounds(135, 111, 369, 189);
201.         Font font = label_2.getFont();
202.         Font boldFont = new Font(font.getFontName(), Font.BOLD, 18);
203.         label_2.setFont(boldFont);
204.         contentPane.add(label_2);
205.
206.     }
207. }
```

2.5 Aplicación Android

Httppostaux.java

```
1.  package com.proyecto.antonio.projectdesign.library;
2.
3.  import java.io.BufferedReader;
4.  import java.io.InputStream;
5.  import java.io.InputStreamReader;
6.  import java.io.OutputStream;
7.  import java.net.HttpURLConnection;
8.  import java.net.URL;
9.
10. import org.apache.http.conn.ssl.AllowAllHostnameVerifier;
11. import org.json.JSONArray;
12. import org.json.JSONException;
13.
14. import android.net.SSLCertificateSocketFactory;
15. import android.util.Base64;
16. import android.util.Log;
17.
18. import javax.net.ssl.HostnameVerifier;
19. import javax.net.ssl.HttpsURLConnection;
20. import javax.net.ssl.SSLContext;
21. import javax.net.ssl.SSLSession;
22.
23. /*
24.  * Diseño por Pablo Gallardo Hiraldo
25.  * Versión: 1.0
26.  *
27.  */
28.
29. /*CLASE AUXILIAR PARA EL ENVIO DE PETICIONES A NUESTRO SISTEMA
30.  * Y MANEJO DE RESPUESTA.*/
31.
32. public class Httppostaux{
33.
34.     InputStream is = null;
35.     String result = "";
36.
37.     /*
38.      * Envía un POST request a '/oauth/token' para conseguir un token de acceso,
39.      * el cual será mandado en cada request posterior
40.      */
```

```
41.     public String sendTokenRequest (String username, String password, String URL_connect){
42.
43.         StringBuffer response = null;
44.         JSONArray object = null;
45.
46.         try{
47.             // Establecemos conexión con el servidor
48.             String urlString = "https://"+URL_connect+"/oauth/token?grant_type=password&username="+username+"&pa
    ssword="+password;
49.             URL urlObj = new URL(urlString);
50.             HttpsURLConnection con = (HttpsURLConnection) urlObj.openConnection();
51.
52.             if (con instanceof HttpsURLConnection) {
53.                 HttpsURLConnection httpsConn = (HttpsURLConnection) con;
54.                 httpsConn.setSSLSocketFactory(SSLCertificateSocketFactory.getInsecure(0, null));
55.                 httpsConn.setHostnameVerifier(new AllowAllHostnameVerifier());
56.             }
57.
58.
59.             // Credenciales con las que solicitamos el token de acceso
60.             String userCredentials = "trusted-monitor:topsecret";
61.             String basicAuth = "Basic " + new String(Base64.encode(userCredentials.getBytes(), Base64.DEFAULT));

62.
63.             // Mandamos las credenciales en el header http
64.             con.setRequestProperty ("Authorization", basicAuth);
65.             con.setRequestMethod("POST");
66.             con.setRequestProperty("Content-Type", "application/json");
67.             con.setUseCaches(false);
68.             con.setDoInput(true);
69.             con.setDoOutput(true);
70.
71.             // Enviamos la petición por POST
72.             OutputStream os = con.getOutputStream();
73.             os.flush();
74.
75.             // Leemos la respuesta
76.             BufferedReader in = new BufferedReader(
77.                     new InputStreamReader(con.getInputStream(), "UTF-8"));
78.             String inputLine;
79.             response = new StringBuffer();
80.
81.             while ((inputLine = in.readLine()) != null) {
82.                 response.append(inputLine);
83.             }
84.             // Convertimos a formato JSON
85.             String responseSTR = response.toString();
86.             String responseJson = "["+responseSTR+"]";
87.             object = new JSONArray(responseJson);
88.             String token = object.getJSONObject(0).getString("access_token");
89.
90.             in.close();
91.
92.             //Devolvemos el token de acceso para para usarlo en cada request que hagamos
93.             return token;
94.
95.         } catch (Exception e){
96.             Log.e("log_tag", "Error CON EL TOKEN "+e.toString());
97.             return null;
98.         }
99.
100.    }
101.
102.
103.
104.    public JSONArray getserverdata(String parameters, String accessToken, String urlwebserver ){
105.
106.        // Conecta via http y envia un post.
107.        httppostconnect(parameters,urlwebserver, accessToken);
108.        // Convierte el resultado a JSON
109.        JSONArray resul = getjsonarray();
110.            return resul;
111.    }
112.
113.
114.    /*
115.     * Envía las peticiones al servidor
116.     */
117.    private void httppostconnect(String parametros, String urlwebserver, String accessToken){
118.
119.        try{
120.            // Añadimos el token de acceso a todas las peticiones  y extablecemos conexión
121.            String urlstring = urlwebserver+"?access_token="+accessToken;
122.            URL urlObj = new URL(urlstring);
123.            HttpsURLConnection con = (HttpsURLConnection) urlObj.openConnection();
```

```
124.
125.             if (con instanceof HttpsURLConnection) {
126.                 HttpsURLConnection httpsConn = (HttpsURLConnection) con;
127.                 httpsConn.setSSLSocketFactory(SSLCertificateSocketFactory.getInsecure(0, null));
128.                 httpsConn.setHostnameVerifier(new AllowAllHostnameVerifier());
129.             }
130.
131.             // Según el tipo de petición, definimos distintas propiedades en la cabecera
132.             if(parametros=="GET") {
133.                 con.setRequestProperty("Accept", "application/json");
134.                 con.setRequestMethod("GET");
135.             }else {
136.                 con.setDoOutput(true);
137.                 con.setRequestMethod("POST");
138.                 con.setRequestProperty("Content-Type", "application/json");
139.
140.                 // Enviamos la petición por POST
141.                 OutputStream os = con.getOutputStream();
142.                 os.write(parametros.getBytes());
143.                 os.flush();
144.
145.             }
146.             // Leemos la respuesta
147.             BufferedReader in = new BufferedReader(
148.                     new InputStreamReader(con.getInputStream(), "UTF-8"));
149.             String inputline;
150.             StringBuffer response= new StringBuffer();
151.             while((inputline = in.readLine()) != null){
152.                 response.append(inputline);
153.             }
154.             result=response.toString();
155.             in.close();
156.
157.         }catch(Exception e){
158.             Log.e("log_tag", "Error in http connection "+e.toString());
159.         }
160.
161.     }
162.
163.
164.     public JSONArray getjsonarray(){
165.         //Convierte a JSON
166.         try{
167.             JSONArray jArray = new JSONArray(result);
168.             return jArray;
169.         }
170.         catch(JSONException e){
171.             Log.e("log_tag", "Error parsing data "+e.toString());
172.             return null;
173.         }
174.     }
175. }
```

YoutubeFragment.java

```
1.  package com.proyecto.antonio.projectdesign.library;
2.
3.  import android.os.Bundle;
4.  import android.widget.Toast;
5.  import com.google.android.youtube.player.YouTubeInitializationResult;
6.  import com.google.android.youtube.player.YouTubePlayer;
7.  import com.google.android.youtube.player.YouTubePlayerSupportFragment;
8.
9.
10. public class YouTubeFragment extends YouTubePlayerSupportFragment implements YouTubePlayer.OnInitializedListener
     {
11.
12.     private static final int RECOVERY_DIALOG_REQUEST = 1;
13.
14.     private static final String KEY_VIDEO_ID = "KEY_VIDEO_ID";
15.
16.     private String mVideoId;
17.
18.     //Empty constructor
19.     public YouTubeFragment() {
20.     }
21.
22.     /**
23.      * Devuelve una nueva instancia de este fragmento
24.      * El parámetro videoId es la ID del video a reproducir, no la URL completa
```

```
25.       */
26.      public static YouTubeFragment newInstance(final String videoId) {
27.          final YouTubeFragment youTubeFragment = new YouTubeFragment();
28.          final Bundle bundle = new Bundle();
29.          bundle.putString(KEY_VIDEO_ID, videoId);
30.          youTubeFragment.setArguments(bundle);
31.          return youTubeFragment;
32.      }
33.
34.      @Override
35.      public void onCreate(Bundle bundle) {
36.          super.onCreate(bundle);
37.
38.          final Bundle arguments = getArguments();
39.
40.          if (bundle != null && bundle.containsKey(KEY_VIDEO_ID)) {
41.              mVideoId = bundle.getString(KEY_VIDEO_ID);
42.          } else if (arguments != null && arguments.containsKey(KEY_VIDEO_ID)) {
43.              mVideoId = arguments.getString(KEY_VIDEO_ID);
44.          }
45.
46.          initialize("AIzaSyBk9X7iUxKbrvUNgts6rn5reE-ky5P0XoM", this);
47.      }
48.
49.      /**
50.       * Asigna el id del video e inicializa el reproductor
51.       * El parámetro videoId es la ID del video a reproducir, no la URL completa
52.       */
53.      public void setVideoId(final String videoId) {
54.          mVideoId = videoId;
55.          initialize("AIzaSyBk9X7iUxKbrvUNgts6rn5reE-ky5P0XoM", this);
56.      }
57.
58.      @Override
59.      public void onInitializationSuccess(YouTubePlayer.Provider provider, YouTubePlayer youTubePlayer, boolean re
     stored) {
60.
61.          if (mVideoId != null) {
62.              if (restored) {
63.                  youTubePlayer.play();
64.              } else {
65.                  youTubePlayer.loadVideo(mVideoId);
66.              }
67.          }
68.      }
69.
70.      @Override
71.      public void onInitializationFailure(YouTubePlayer.Provider provider, YouTubeInitializationResult youTubeInit
     ializationResult) {
72.          if (youTubeInitializationResult.isUserRecoverableError()) {
73.              youTubeInitializationResult.getErrorDialog(getActivity(), RECOVERY_DIALOG_REQUEST).show();
74.          } else {
75.              //Handle the failure
76.              Toast.makeText(getActivity(), "Fallose al iniciar", Toast.LENGTH_LONG).show();
77.          }
78.      }
79.
80.      @Override
81.      public void onSaveInstanceState(Bundle bundle) {
82.          super.onSaveInstanceState(bundle);
83.          bundle.putString(KEY_VIDEO_ID, mVideoId);
84.      }
85.  }
```