Universidad de Almería

ESCUELA POLITÉCNICA SUPERIOR Y FACULTAD DE CIENCIAS EXPERIMENTALES

INGENIERÍA EN INFORMÁTICA

Estudio y auditoría de LOPD en el diseño e implementación de un gestor de servicios peer to peer

Marzo de 2014

Realizado por:
Jorge Manzano Rodríguez
Director:
José Antonio Torres Arriaza

Contenidos

1. Introducción

La aparición de servicios que permiten el intercambio de productos y servicios entre usuarios en la red se ha incrementado en los últimos tiempos con la aparición de los dispositivos inteligentes o smartphones. Esta situación, y el hecho derivado de las normativas que impone la administración para tratar los datos personales relativos a los clientes de estos servicios hace interesante plantear, desde las fases iniciales de diseño, la implementación de las normativas en materia de protección de datos que dicta la LOPD.

Este proyecto hace uso de una aplicación de intercambio de servicios para implantar, de manera estructurada, los condicionantes legales de la ley de protección de datos.

1.1. Objetivos

El proyecto persigue diversos objetivos entre los que se encuentran, de forma muy resumida, los siguientes:

1. Crear una plataforma móvil segura con la cual se puedan crear y consumir cualquier tipo de servicio.
2. Crear un servidor seguro que se encargue de gestionar las peticiones de los clientes móviles.
3. Estudiar la Ley Orgánica de Protección de datos para poder aplicarla a cada una de las etapas del desarrollo del proyecto y así, tanto servidor como dispositivo móvil se ajusten a los requisitos marcados por dicha ley.

1.2. Recursos

Para la realización de este proyecto se ha usado un portátil Packard Bell EasyNote Ts44 hr, con un procesador Intel i5, 4 gb de memoria principal y tarjeta gráfica Nvidia Geforce GT540M. En cuanto al dispositivo móvil se ha utilizado un Sony Xperia S con 1 gb de memoria ram, un procesador Qualcomm MSM8260 dual-core 1.5GHz y GPU Adreno 220.

En el apartado software, la aplicación se ha desarrollado sobre el sistema operativo Windows 7 (portátil) y para dispositivos cuyo sistema operativo sea Android (móvil), ambos con acceso a Internet.

Para la implementación del servidor se ha usado Spring Tools Suite, un entorno basado en Eclipse pero personalizado para el desarrollo y ejecución de aplicaciones que usen Spring Framework.

Para la implementación de la aplicación móvil se ha utilizado el llamado ADT (Android Developers Tools) para Windows. Éste incluye el entorno de programación

Eclipse con el plugin ADT instalado, el SDK de Android, una imagen de un dispositivo para la emulación de un terminal (se desaconseja su uso puesto que su ejecución es muy lenta, siempre que sea posible se debería utilizar un dispositivo físico), diversas herramientas para la plataforma y la última versión de dicho sistema operativo.

En el apartado del sistema de gestión de base de datos relacional, se ha optado por MySQL, por ser una base de datos muy rápida y por haber trabajado previamente con ella.

Se ha empleado un control de versiones, GIT, y un servidor con un repositorio privado, BitBucket. Con ello se ha conseguido tener el código bien organizado, estructurado y actualizado para tener una copia de seguridad del desarrollo.

La implementación ha estado basada en metodologías ágiles, Scrum. El problema radica en que esta metodología está muy focalizada en grupos de trabajo y el proyecto ha sido desarrollado de forma individual. Por este motivo no se ha podido aplicar estrictamente dicho método.

Para aplicar la metodología comentada en el párrafo anterior se ha hecho uso de la plataforma ScrumDo. Desde esta herramienta se ha gestionado las distintas historias de usuario y sprints creados a lo largo del desarrollo. Al finalizar cada uno de los sprints se ha intentado presentar una parte del software potencialmente entregable a un supuesto cliente. A continuación se puede ver una captura de pantalla de la aplicación en la que muestra tres historias de usuario finalizadas en uno de los sprints programados:

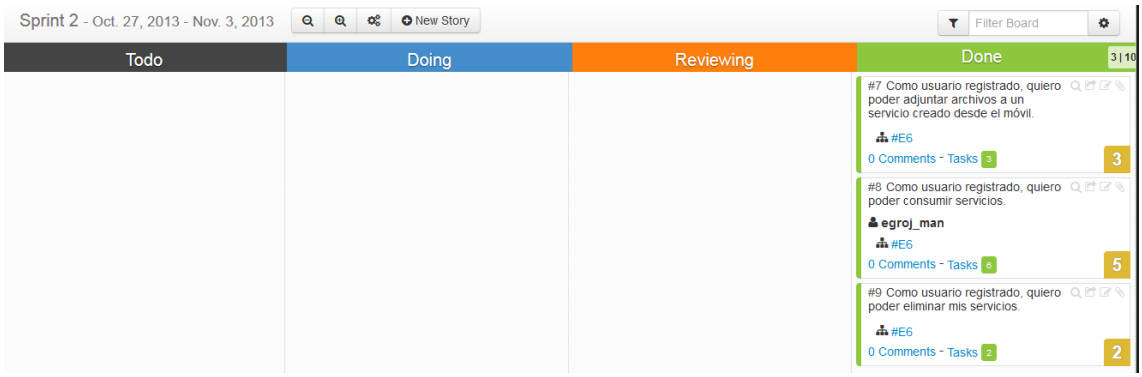


Imagen 1

Se han realizado capturas de paquetes de las conexiones entre cliente y servidor para verificar su correcto funcionamiento. Se capturaron los datos antes y después de aplicar el cifrado en las conexiones usando *RawCap* y *Wireshark*. *RawCap* se utiliza para capturar paquetes desde la línea de comandos y con *Wireshark,* aparte de poder realizar estas capturas, se usó para analizar los paquetes que circulaban por la red y comprobar si viajaban en texto plano o se cifraban tal como se pretendía.

1.3. Plan de trabajo

A continuación se verá una aproximación de las distintas fases de trabajo llevadas a cabo durante la ejecución del proyecto:

- Primer estudio de la Ley Orgánica de Protección de Datos. Se llevará a cabo una primera toma de contacto con la LOPD realizando una lectura comprensiva de la misma y tomando las referencias que se crean oportunas para el futuro desarrollo.
- Implementación básica del servidor. Se comenzará con el análisis de requisitos para después diseñar la base de datos e implementar la funcionalidad requerida.
- Implementación básica de la aplicación móvil. Se diseñará la interfaz de usuario básica y se implementará la comunicación con el servidor.
- Implementación de la seguridad en el servidor, en el móvil y en la comunicación entre ambos. Tanto para el servidor como para la aplicación móvil es necesario añadir cierta configuración adicional para preservar la seguridad en el sistema. Esta parte se tendrá en cuenta una vez implementada la funcionalidad completa de la aplicación móvil y del servidor.
- Repaso de la Ley Orgánica de Protección de Datos. Una vez el sistema se dé por finalizado se comprobará que efectivamente nuestro sistema no incumple ninguna de las normas establecidas por la LOPD.
- En cada paso hemos se han realizado las pruebas oportunas, se han depurado ambas aplicaciones y se han ido corrigiendo los errores que iban apareciendo.

2. Estado del arte y conceptos teóricos

Antes de comenzar con la implementación se ha de contextualizar previamente el proyecto. Por ello se analizará la actualidad móvil y en concreto el sistema operativo Android, se verán algunas de las herramientas usadas en la implementación y se explicarán algunos de los conceptos adquiridos durante la realización de este proyecto. Estos conceptos serán necesarios para una correcta comprensión del desarrollo de la idea.

También se resumirán los puntos más importantes marcados por la Ley Orgánica de Protección de Datos para tener una amplia visión de la misma e intentar cuidar la implementación del sistema bajo su normativa.

2.1. Servicios móviles

Hace 40 años Martin Cooper[1], directivo de Motorola realizó la primera llamada desde el primer teléfono móvil personal. En aquellos años disponer de uno de estos estaba al alcance de muy pocos debido, sobre todo, a su alto precio. A parte, se contaba con una infraestructura de comunicaciones de muy baja calidad, los terminales eran de un tamaño considerable y la cobertura era muy limitada. Fue en los '90 cuando empezaron a aparecer dispositivos móviles similares a los que hoy en día podemos encontrarnos en el mercado. Mejoraron considerablemente las telecomunicaciones consiguiendo una transmisión de voz de gran calidad y, aumentando y mejorando el nivel de cobertura.

Sobre el año 2000 aparecieron los primeros dispositivos, conocidos como smartphones, con capacidad de computación y conectividad a Internet, que, aparte de permitir realizar llamadas, eran capaces de hacer muchas otras cosas. Gracias a la inclusión de Internet en el móvil fueron apareciendo un gran número de servicios para estos dispositivos, desde conocer el tráfico de una determinada vía hasta, por ejemplo, poder consultar la meteorología de un determinado lugar, entre otros muchos. Este hecho despertó el interés de los usuarios en dichos terminales provocando la necesidad de realizar grandes inversiones por parte de las operadoras para mejorar sus infraestructuras de telecomunicaciones. Por otra parte las empresas tecnológicas vieron la oportunidad de poder crear servicios más complejos sobre la red y con una mayor exigencia de conectividad, por lo que se consiguieron grandes avances, tanto en servicios como en tecnologías de red.

En la actualidad, la velocidad de las conexiones y la potencia de cómputo de los dispositivos móviles continúan creciendo de forma exponencial. Esto es debido a que las grandes marcas en el mundo de los smartphones han entrado en una lucha constante por mantener el primer puesto en cuanto a tecnología y rendimiento en los terminales, para así, mantener un nivel de popularidad alto. En algunos casos, los dispositivos móviles han conseguido igualar o incluso a batir en capacidad de

[1] http://es.wikipedia.org/wiki/Martin_Cooper

procesamiento a los ordenadores convencionales y, por otra parte, gracias al ancho de banda que ofrecen las distintas tecnologías de red, como por ejemplo la cuarta generación o 4G, podemos consumir cualquier tipo de contenido multimedia en nuestro móvil. Lo anterior motiva a que gran cantidad de empresas tecnológicas lancen un gran número de sus servicios para smartphones o, incluso, que se hayan creado entidades exclusivamente para ofrecer, exclusivamente a móviles, servicios de lo más variado. Desde el 2010 una gran parte de estos servicios se ofertan en forma de aplicaciones o apps. [1]

El Ministerio de Industria, Energía y Turismo ha presentado recientemente un informe[2] sobre el uso de Internet en los hogares. En él podemos ver cómo por ejemplo el 94% de los hogares españoles dispone de telefonía móvil o que el 85,8% de los españoles de 15 y más años usan el teléfono móvil de forma habitual. Esto pone de manifiesto la penetración existente en la actualidad de terminales móviles en España.

2.2. Android

Es un sistema operativo basado en Linux diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas. Inicialmente fue desarrollado por Android, Inc. Google lo respaldó económicamente y más tarde compró esta empresa en 2005. Android fue presentado en 2007 junto la fundación del Open Handset Alliance, un consorcio de compañías de hardware, software y telecomunicaciones, para avanzar en los estándares abiertos de los dispositivos móviles. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008. [2]

España se ha convertido en el país europeo con mayor penetración de smartphones, según la agencia de noticias Europapress[3], actualmente cuenta con un 66 por ciento de penetración. Además, Android, es el sistema líder en nuestro país llegando al 64,6 por ciento de cuota de mercado. Por otro lado, según Kantar Worldpanel ComTech[4], vemos que Android tendría en el mercado un 92 por ciento de la cuota, 9 de cada 10 smartphones llevarían el sistema opetativo de Google. Además, según el último estudio de la misma consultora sobre cuotas de mercado de los sistemas operativos, Android ha alcanzado el 64,6 por ciento de la cuota de smartphones a nivel mundial, convirtiéndose así en la primera plataforma de venta de teléfonos inteligentes.

[2] http://www.minetur.gob.es/en-us/gabineteprensa/notasprensa/2013/Paginas/npinformepenetracioninter141113.aspx
[3] http://www.europapress.es/portaltic/sector/noticia-espana-lidera-europa-uso-smartphones-66-penetracion-20130820134026.html

[4] http://www.kantarworldpanel.com/es/Noticias/Android-ya-est-en-9-de-cada-10-nuevos-smartphones

%Cuota Sistemas Operativos sobre nuevos *Smartphones*	Dic - Feb 2012	Dic - Feb 2013
Android	70,9%	92,1%
iOS	5,9%	4,4%
Symbian	10,5%	1,1%
RIM	10,2%	1,0%
Windows	2,5%	0,9%
Otros	0,0%	0,5%
Total *Smartphones*	100%	100%

Fuente: Worldpanel ComTech

Tabla 1

En cuanto a Europa, Android ha acaparado el 70,4 por ciento de la cuota de mercado, iOS un 17,8 por ciento y Windows Phone un 6,8 por ciento. En Estados Unidos, aunque Android lidera el mercado con un 52 por ciento de la cuota, su ventaja con respecto a iOS, con un 41,9 por ciento, no es tanta como en el viejo continente; y Windows se mantiene en tercera posición con un 4,6 por ciento. Japón sería la excepción, ya que iOS es líder de mercado con un 49,2 por ciento frente a un 45,8 por ciento de Android.

Analizando todos estos datos se puede apreciar que, comenzar desarrollando la aplicación para Android es claramente la mejor opción. Debido a su gran cuota de mercado se podría llegar a tener una gran cantidad de usuarios usando potencialmente nuestra aplicación.

A parte, otro dato a tener en cuenta es el único pago de 25 dólares requerido hasta la fecha para darse de alta como desarrollador en Google Play, la tienda de aplicaciones de Android. Sin embargo, para el sistema operativo IOS, segundo en cuota de mercado a nivel europeo, darse de alta en su tienda de aplicaciones para publicar apps tiene un coste de 99 dólares anuales.

Actualmente el sistema operativo Android se encuentra en su versión 4.4 KitKat cuyo nivel de api es el 19. Pero esta versión no se encuentra disponible en el terminal usado para ejecutar la aplicación, por lo que todas las pruebas han sido realizadas sobre la versión 4.1.2 JellyBean y su nivel de api 16.

2.3. Patrón MVC

MVC es un patrón o modelo de abstracción de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos. Por un lado define componentes para la representación de la información y por otro lado para la interacción del usuario. [3]

El principio fundamental del patrón MVC es definir una arquitectura separada con claras responsabilidades para diferenciar componentes. En MVC hay tres participantes:

- **Modelo:** Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).

- **Vista:** Presenta el los datos del modelo en un formato adecuado para interactuar con ellos. Solicita al modelo la información que debe representar como salida.

- **Controlador:** Hace de intermediario entre la vista y el modelo. Responde a eventos (generados usualmente por el usuario en la vista) y realiza peticiones al modelo si se hace alguna solicitud sobre la información. Incluso puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se representa el modelo.

2.4. Spring Framework

Spring es un framework de código abierto y un contenedor ligero para el desarrollo de aplicaciones para la plataforma Java.

Proporciona una infraestructura que actúa como soporte para que sólo te tengas que centrar en tu aplicación. Intenta hacer el desarrollo de aplicaciones JEE más fácil, por ello los componentes u objetos, la interacción entre ellos y el ciclo de vida de la aplicación lo maneja internamente Spring por nosotros.

La primera versión fue escrita por Rod Johnson, quien lo lanzó junto a la publicación de su libro *Expert One-on-One J2EE Design and Development* (Wrox Press, octubre 2002). Hoy en día es uno de los frameworks más utilizados gracias, en gran parte, a la comunidad que ha ido creciendo poco a poco y apoyando este proyecto desde su lanzamiento.

Como hemos señalado anteriormente, Spring es un contenedor ligero (“lightweight container”). Por esto podemos afirmar que está orientado principalmente a aplicaciones compactas. En desarrollos J2EE generalmente se usa un contenedor como Tomcat, JBoss, Websphere, Weblogic, Glassfish, etc. La ventaja de estos contenedores es que ofrecen ya varios servicios que puede utilizar la aplicación que desarrollemos. Pero si nuestra aplicación no llega a ser tan compleja, o no va a usar casi ninguno de estos servicios y el tiempo de arranque de la misma va a ser importante para nosotros, necesitamos un contenedor como Spring Framework. Su uso generalmente es en aplicaciones web aunque también es cierto que Spring puede usarse en el desarrollo de cualquier tipo de software.

Spring Framework está basado en la técnica Inversión de Control. Esta técnica es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales. Se establecen unas respuestas

adecuadas a eventos concretos delegando el control del programa a algún tipo de entidad o arquitectura externa.

Spring fomenta el acoplamiento débil entre clases usando la técnica llamada inyección de dependencia (DI). El acoplamiento entre clases o subsistemas es una medida de interconexión entre esas clases o subsistemas. El acoplamiento fuerte significa que las clases relacionadas necesitan saber detalles internos unas de otras, los cambios se propagan por el sistema y el sistema es posiblemente más difícil de entender. En resumen, los objetivos detrás de la obtención de acoplamiento débil entre clases y módulos son:

1. Hacer que el código sea más fácil de leer.

2. Hacer que nuestras clases sean más fáciles de consumir por otros desarrolladores, ocultando la parte del funcionamiento interno de nuestras clases en APIs bien diseñadas.

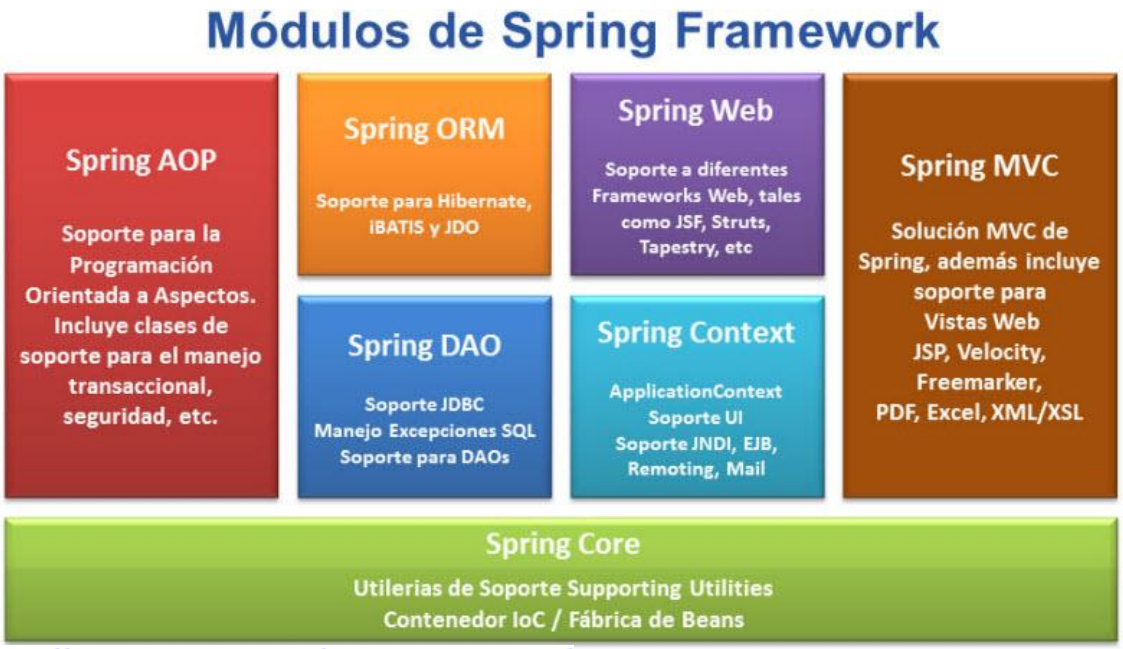
3. Aislar posibles cambios en un área pequeña del código.

4. Reutilizar clases en contextos completamente nuevos.

Cuando aplicamos la inyección de dependencia, se otorga a los objetos de forma pasiva sus dependencias, en lugar de crear o buscar objetos dependientes de sí mismo, o en otras palabras, suministramos los objetos a una clase en lugar de ser la propia clase quien cree dichos objetos.

Por otro lado, Spring tiene un amplio soporte para la programación orientada a aspectos (AOP) cuyo objetivo es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de los diferentes aspectos del sistema. Cada módulo se ocupa de lo que se supone que debe hacer, para nada conoce ni son responsables del resto del comportamiento del sistema. [4]

Spring MVC (Model – View – Controller) es uno de los módulos del framework Spring. Este provee un exhaustivo soporte para el patrón MVC, comentado con anterioridad, así como también provee soporte de otras características, una de ellas es facilitar la implementación de la capa de presentación.



http://globalmentoring.com.mx/curso-spring-framework/

Imagen 2

Actualmente el proyecto Spring Framework se encuentra en su versión estable 4.0.0 pero, cuando se inició la implementación de nuestro proyecto, la versión liberada estable era la 3.2.3, por lo que en la documentación nos remitiremos exclusivamente a ésta última.

2.5. Apache Tomcat

Tomcat es un servidor web con soporte de servlets y JSPs (Java Server Pages). Tomcat fue desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages de Sun Microsystems.

Tomcat es mantenido y desarrollado por miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la *Apache Software License*. [5]

2.6. Maven

Es una herramienta de software para la gestión y construcción de proyectos Java similar en funcionalidad a Apache Ant, pero con un modelo de configuración de construcción más simple, basado en un formato XML.

Maven utiliza un Project Object Model (POM), un archivo XML que describe el proyecto de software a construir, las dependencias de otros módulos y los componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas, como la compilación del código y su empaquetado. [6]

Maven se suele utilizar habitualmente bajo línea de comandos aunque, para cada IDE, suele haber algún plugin con interfaz gráfica que te facilita la realización de ciertas tareas de esta herramienta.

Para crear un proyecto de ejemplo a través de la línea de comandos, previamente hemos tenidos que instalar Maven, deberíamos escribir lo siguiente:

```
mvn archetype:create -DgroupId=com.pfc -DartifactId=EjemploMaven
```

Explicamos los parámetros de la línea anterior:

- **`archetype:create`**: Le indica a Maven que quieres crear un proyecto nuevo.

- **`DgroupId=`***`com.pfc`*: Indicamos a Maven cuál será el grupo de proyectos al que pertenecerá el que estamos creando. Todos han de pertenecer a un grupo aunque este grupo sólo lo ocupe un proyecto.

- **`DartifactId=`***`EjemploMaven`*: Con este comando especificamos el nombre del proyecto. Todos los proyectos Maven se identifican a través de su *artifactId*.

2.7. Hibernate

Para facilitar la relación entre las clases del modelo de datos de la aplicación a desarrollar y la base de datos relacional y después de ver la buena integración que mantiene con Spring Framework, se ha decidido usar Hibernate en el proyecto como herramienta ORM.

Hibernate es una herramienta de Mapeo Objeto-Relacional (ORM). Dicho de otro modo, Hibernate es un framework que nos simplifica la creación del código necesario para almacenar el valor de nuestras clases en la base de datos relacional. También nos ayuda a diferenciar los dos modelos de datos que coexisten en nuestra aplicación: el modelo orientado a objetos, que se usará en la memoria de la computadora y el modelo relacional, que será usado en la base de datos. [7]

Para lograr esto nos permite detallar, mediante archivos XML (desde la aparición del framework) o anotaciones en el propio código (en sus versiones más recientes) cómo es el modelo de datos y las relaciones que tienen las clases del mismo.

Una vez se han definido todas las clases del modelo de datos y se han añadido las anotaciones o creado los archivos XMLs, Hibernate generará las sentencias

correspondientes en SQL para la creación y manipulación de la base de datos librando al desarrollador de esa tediosa tarea.

2.8. Servidor REST

De las distintas técnicas o estilos de arquitectura del software para la creación de servidores en la web, la más acertada para la implementación del sistema era crear un servidor que siga los principios marcados por REST, es decir, un servidor RESTful.

REST (Representational State Transfer) surge sobre el año 2000 sin tener mucho éxito en su presentación. Sin embargo, en estos últimos años ha emergido como el modelo predominante para el diseño de los servicios web, siendo una alternativa a los servicios SOAP y a las interfaces basadas en WSDL por tener un estilo más fácil de usar e implementar. [8]

REST se centra en los recursos del sistema, incluye la forma de acceso a los mismos y cómo se transfieren por HTTP hacia diferentes clientes.

Una implementación cualquiera de un servicio web REST se apoya en cuatro principios básicos:

- **Usa los métodos HTTP de manera explícita.** Estos métodos son *GET*, para obtener un recurso, *POST*, para crear un recurso en el servidor, *PUT*, para actualizar un recurso y *DELETE*, para eliminar un recurso del servidor.

- **El protocolo cliente/servidor no mantiene estado.** Cada mensaje HTTP contiene toda la información necesaria para comprender una petición. Así, ni cliente ni servidor tienen que almacenar dato alguno en relación a las peticiones haciendo que la implementación y el escalado del sistema sea mucho más sencillo.

- **Usa URIs con forma de directorios.** Las URIs de un servidor REST han de ser sencillas e intuitivas, por eso se presentan en forma de directorios. Sólo con ver una URI de un servidor RESTful deberíamos ser capaz de entender a qué recurso estamos accediendo. Cada recurso es accesible únicamente a través de su URI.

- **Transfiere JSON, XML, XHTML o todos.** Es recomendable que los servicios usen el atributo HTTP Accept del encabezado. Así diferentes clientes podrán usar distintos tipos MIME con cualquiera de los valores que hayamos incluido en el servidor. Los tipos más usados en los servicios REST son JSON, XML y XHTML por su simplicidad y por la legibilidad por humanos.

2.9. JSON

¿Qué es JSON?

JSON (JavaScript Object Notation) es un formato ligero para el intercambio de datos [9] que ha alcanzado gran popularidad en estos años. Una de sus mayores ventajas es que puede ser leído por cualquier lenguaje de programación. Para muchos, es el compañero perfecto de AJAX y se ha convertido en el más usado en los desarrollos de servicios REST gracias a su sencillez.

Estructura

JSON ha sido constituido con dos estructuras:

1. Colección de pares de nombre/valor. Dependiendo del lenguaje de programación un objeto, un registro, una estructura, etc.

2. Lista ordenada de valores. Que puede ser un array, un vector, etc.

Ejemplo

Para verlo con más claridad a continuación se puede ver un ejemplo muy sencillo de un array de personas en formato JSON:

```
{“Personas”:[
     {“nombre”: “Pepe”, “edad”: 45},
     {“nombre”: “Luisa”, “edad”: 20}
]}
```

En el ejemplo anterior se puede ver un array llamado *Personas* el cual contiene dos valores, *Pepe* y *Luisa*.

2.10. Ley Orgánica de Protección de Datos

Los distintos acontecimientos recientes, cada vez más habituales, relacionados con el robo o venta de datos personales están creando cierto recelo a ceder dichos datos a cualquier aplicación o portal que los solicite. Los usuarios se comienzan a hacer preguntas como ¿por qué tengo que dar mis datos?, ¿dónde se almacenan? o ¿qué hacen con ellos? La preocupación de cómo se usan nuestros datos aumenta, aunque todavía hay un gran número de personas que, por un "suculento regalo" o por algún tipo de descuento, no tienen ningún pudor a regalar su perfil completo.

2.10.1. ¿Qué hacen las empresas con nuestros datos?

Tener un gran número de perfiles de usuario en la actualidad es muy valioso. Es muy común que empresas vendan y/o compren dichos perfiles para, gracias a la información que aportan, poder lanzar campañas de marketing dirigidas, por ejemplo.

Los perfiles de usuario también se usan para lo que se conoce como publicidad por comportamiento o behavioral targeting (Usando las cookies de los navegadores). Con ello se consigue una publicidad mucho más efectiva debido a que personas interesadas en un determinado producto les llegará "exclusivamente" publicidad de dicho producto o, al menos, relacionada con el mismo.

Aunque el principal uso de dichos perfiles sea para el puro marketing, estos datos se suelen utilizar en otros tipos de estudios como por ejemplo el estudio de tendencias. En general, el Big Data, de forma muy resumida, la gestión de grandes cantidades de datos, está registrando un notable impulso.

Son muchas entidades las que se han beneficiado, o actualmente se están beneficiando con el negocio de la información de carácter personal.

Algunas empresas consiguen y usan los datos personales de los usuarios de forma totalmente legítima y transparente pero, en otras entidades, no existe una política tan clara de la forma en la que se obtienen o del uso que se le da a dicha información. Por este motivo era necesario una ley que velara por la protección de estos datos de carácter personal, la llamada: "Ley Orgánica de Protección de datos" (LOPD).

2.10.2. LOPD

Para profundizar un poco en la ley es conveniente citar el artículo 1 de la LOPD. Este artículo expone lo siguiente:

> "*Artículo 1 Objeto*
>
> La presente Ley Orgánica tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor e intimidad personal y familiar."

2.10.2.1. Principios de la protección de datos

El primer principio de esta ley es recoger datos personales estrictamente necesarios para el desarrollo de la funcionalidad de la aplicación y que, una vez hayan

dejado de ser necesarios, éstos han de ser cancelados. Por ello se ha de analizar exhaustivamente qué tipo de datos serán necesarios para el desarrollo de las funciones del sistema a desarrollar para evitar obtener datos excesivos en relación a el ámbito de la aplicación.

El usuario deberá ser informado de los datos que se obtienen de él, de cuál será la finalidad de la obtención de dicha información y del responsable de fichero asignado por la empresa. Queda fuera de este ámbito el tratamiento posterior de los datos para aquellos fines que sean históricos, estadísticos o científicos.

La ley deja bien claro que la información privada de los usuarios que se tenga que almacenar ha de ser exacta y actualizada, los datos que sean inexactos u obsoletos han de ser rectificados, completados o actualizados.

El usuario podrá oponerse en todo momento a dar sus datos personales y tendrá la capacidad de acceder, rectificar, cancelar dichos datos, salvo que éstos estén previamente cancelados.

Tanto el responsable de fichero como cualquiera que maneje los datos de carácter personal están obligados al secreto personal de los mismos y al deber de guardarlos, permaneciendo esta obligación una vez finalizada cualquier relación con el titular de los datos.

2.10.2.2. Resumen

La LOPD aplica a cualquier tratamiento de datos personales con un fin profesional o empresarial, o por una Administración Pública, realizado por una entidad española y/o en el territorio español**.**

Esta ley vela por la protección de datos privados de los usuarios, para ello y de forma muy resumida, siempre que se obtenga algún tipo de información personal estás obligado a:

- Informar a los usuarios sobre los datos personales que se recogen, qué es lo que se hará con ellos y se ha de obtener su consentimiento previo.

- Crear las medidas de seguridad necesarias para la protección de dichos datos. La ley distingue tres niveles de seguridad para los datos, el que afecta directamente al sistema que se desarrollará es el nivel básico y obliga a:

    - Evitar el acceso a los datos sensibles a personas no autorizadas por medio de una clave.

    - Realizar una copia de seguridad de los datos semanalmente.

- Cambiar la contraseña de acceso a los datos una vez al año, como mínimo.

- Crear un documento explicativo que recoja la "política de privacidad" de la empresa.

- Registrar el fichero de datos en la Agencia Española de Protección de Datos (AEPD). Este "fichero" deberá ser el conjunto de tablas de la base de datos, ficheros de texto, excel, etc, en los que se almacenan los datos privados de los usuarios. En nuestro caso debemos incluir las tablas de la base de datos y el archivo almacenado en el terminal móvil.

2.10.2.3. Régimen sancionador

¿Quiénes pueden ser multados?

- **Los Responsables de los ficheros**; es decir, aquellas entidades que tratan datos personales de forma directa, por ejemplo, de sus empleados o de sus clientes.

- **Los Encargados del tratamiento**; es decir, aquellas entidades que tratan datos personales por encargo del Responsable, por ejemplo, la gestoría que hace las nóminas de los empleados del Responsable, o el *callcenter* que atiende las llamadas de los clientes del Responsable. [10]

¿Qué sanciones pueden ser aplicadas?

En cuanto a las sanciones por incumplimiento de esta norma, a continuación se muestra tabla con los tipos de infracciones, las principales causas, y las sanciones aplicables en cada caso**.**

Infracción	Causa	Sanción
Leves	No atender por motivos formales solicitudes de rectificación o cancelación No proporcionar información requerida por la AEPD No inscribir ficheros en el Registro General de Protección de Datos No cumplir el deber de información Vulnerar el deber de secreto	Entre 600 y 60.000 euros
Graves	No recabar el consentimiento de los titulares Mantener datos inexactos No aplicar las medidas de seguridad correspondientes Obstrucción de la labor inspectora	Entre 60.000 y 300.000 euros
Muy graves	Recogida de datos de forma engañosas Comunicación de datos sin requisitos legales Tratar datos de nivel alto sin consentimiento expreso/expreso y escrito	Entre 300.000 y 600.000 euros

Tabla 2

2.11. Cifrado

Un requisito indispensable para este proyecto es almacenar información personal de los usuarios que hagan uso de la aplicación. Esta información se guardará en una base de datos e incluso en ocasiones, los datos deberán viajar a través de la red. Por ejemplo, si un cliente se da de alta en nuestro sistema a través de la aplicación móvil, los datos tendrán que ser enviados al servidor para que los almacene correctamente.

2.11.1. Cifrar

Cifrar es transformar información, mediante una clave, con el fin de protegerla de miradas ajenas. Al aplicar el cifrado a un determinado mensaje, éste mensaje queda alterado llegando a parecer irreconocible o incomprensible, pero la información no se pierde, es posible recuperarla si conocemos la clave de cifrado. [11]

2.11.2. Tipos de cifrado

Clásico: Se ha usado mucho históricamente pero ha caído en desuso. En él, las letras del mensaje son sustituidas o traspuestas por otras. Es un tipo de cifrado muy básico, lo que lo hace poco fiable. Los métodos más modernos operan con bits o bytes.

Simétrico (Clave secreta): En este tipo de cifrado se usa una única clave para descifrar el mensaje. Ambas partes, emisor y receptor, tienen que poseer dicha clave para poder comunicarse entre sí (comparten la misma clave).

Asimétrico (Clave pública): En este caso se usan dos claves para el envío de un mensaje cifrado. Esta pareja de claves son propiedad del emisor, una de ellas es pública y la otra privada. La pareja de claves sólo puede ser generada una vez, por eso este método es seguro. Para ver el funcionamiento de este cifrado se mostrarán a continuación dos ejemplos gráficos:

En la siguiente imagen vemos cómo Roberto y Ana poseen sus dos parejas de claves, una pública y otra privada, y un mensaje que se quieren transmitir:

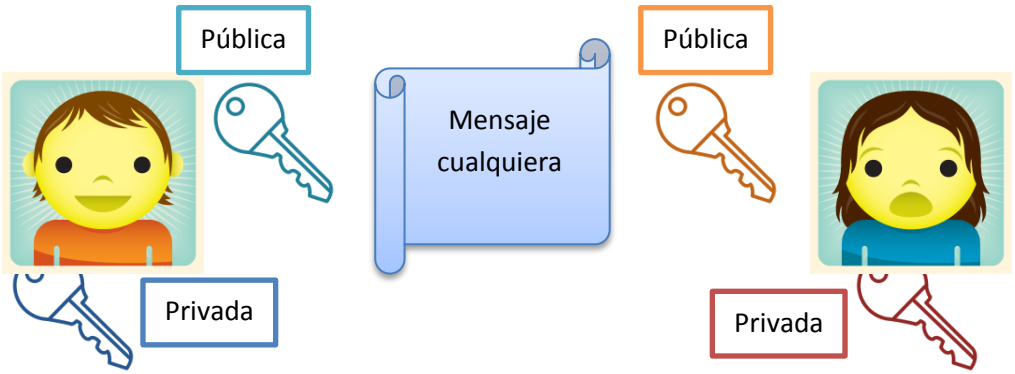


Imagen 3

Roberto envía un mensaje cifrado a Ana:

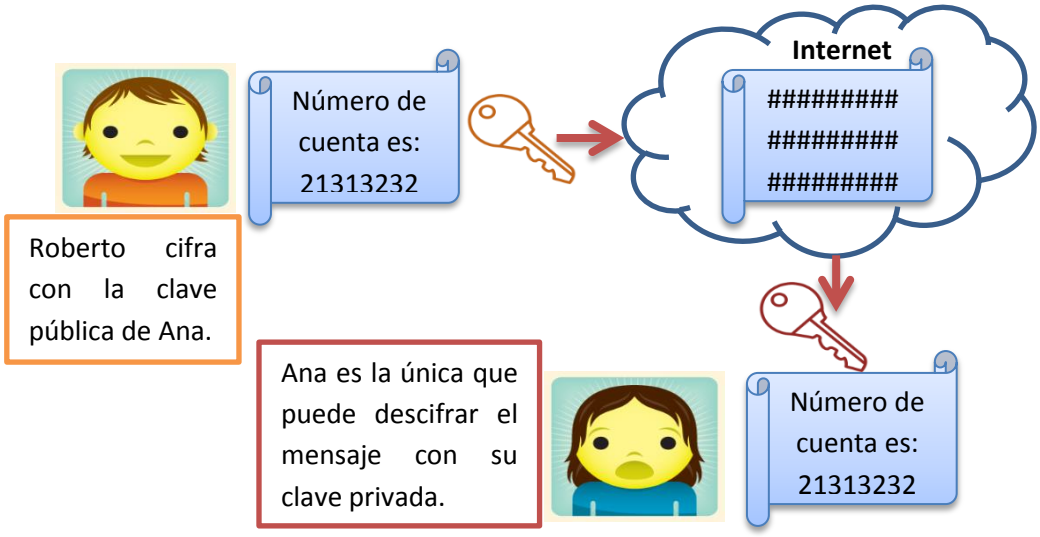


Imagen 4

Roberto firma digitalmente un mensaje para Ana:

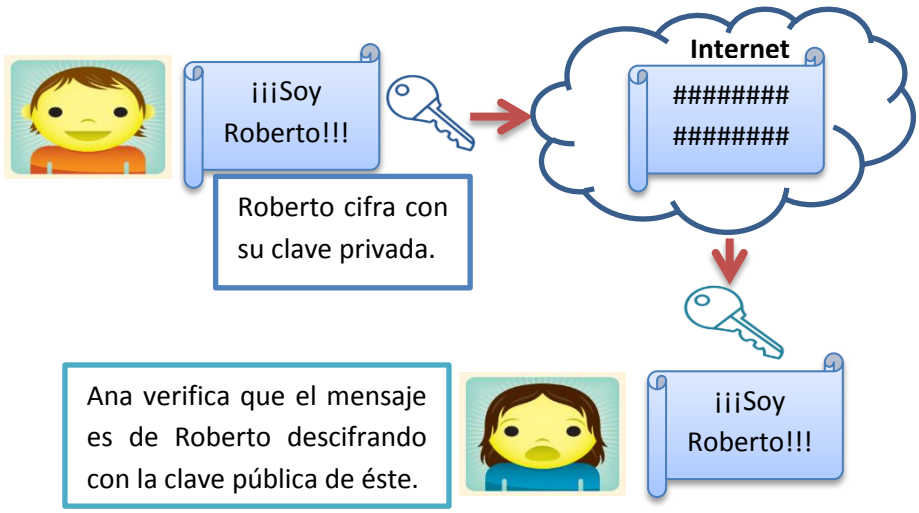


Imagen 5

2.11.2.1. Relación con la LOPD

El primer punto del **Artículo 9 de la LOPD** apunta lo siguiente:

> "*Artículo 9 Seguridad de los datos*
> **1.** El responsable del fichero, y, en su caso, el encargado del tratamiento deberán adoptar las medidas de índole técnica y organizativas necesarias que garanticen la seguridad de los datos de carácter personal y eviten su alteración, pérdida, tratamiento o acceso no autorizado, habida cuenta del estado de la tecnología, la naturaleza de los datos almacenados y los riesgos a que están expuestos, ya provengan de la acción humana o del medio físico o natural**."**

Para evitar "su alteración" y el "tratamiento o acceso no autorizado" a los datos de los usuarios de la aplicación a desarrollar, se ha usado la técnica de cifrado en distintas partes de la misma (base de datos, aplicación móvil y comunicaciones). Este artículo en concreto se ha querido destacar por ser uno de los más importantes y al que más cuidado se ha tenido que prestar por su repercusión en la implementación del sistema.

2.11.3. Jasypt

Según la página oficial, Jasypt es una librería java que permite al desarrollador añadir capacidades básicas de cifrado a los proyectos con un esfuerzo mínimo y sin la necesidad de tener profundos conocimientos sobre el funcionamiento de la criptografía. [12]

Entre sus características más interesantes se encuentran:

- Sigue los estándares y normas marcados por RSA tanto para message digesting como para cifrado basado en clave, que el API de Java de por sí no sigue.
- Es thread-safe por lo que evita todos los problemas relacionados con la concurrencia.
- Se puede usar de forma sencilla en un contenedor IoC como Spring.
- Se integra con Hibernate 3 de forma transparente.
- Permite que los usuarios avanzados que lo necesiten lo configuren según sus necesidades, mientras que otros usuarios pueden usarlo directamente sin configuración adicional.

Resumiendo lo anterior, esta librería consigue ahorrarnos el trabajo necesario para realizar las tareas de cifrado frente al uso de la API de Java. Veremos más detenidamente el uso de esta librería en el desarrollo del servidor.

2.12. Certificados de seguridad

Es un archivo usado por el protocolo SSL firmado electrónicamente por un prestador de servicios de certificación que vincula unos datos de verificación de firma a un firmante y confirma su identidad. Un certificado de seguridad sirve para:

- Autentificar la identidad del usuario, de forma electrónica, ante terceros.
- Firmar electrónicamente de forma que se garantice la integridad de los datos trasmitidos y su procedencia. Un documento firmado no puede ser manipulado, ya que la firma está asociada matemáticamente tanto al documento como al firmante.
- Cifrar datos para que sólo el destinatario del documento pueda acceder a su contenido.

Gracias a los certificados de seguridad podemos mostrar de manera clara tu identidad y cifrar los datos entre el dispositivo cliente y el servidor que ofrezca una página web o algún servicio seguro, de esta forma los datos de carácter personal no pueden ser interceptados por un tercero. [13] [14]

Un certificado de seguridad contiene los siguientes campos: Versión, Número de serie, ID del algoritmo, Emisor, Validez, Sujeto, Información de clave pública del sujeto (Algoritmo de clave pública y Clave pública del sujeto), Identificador único de emisor (opcional), Identificador único de sujeto (opcional), Extensiones (opcional), Algoritmo usado para firmar el certificado y la Firma digital del certificado.

2.12.1. PKI (Public Key Infrastructure)

Una infraestructura de clave pública es una combinación de harware, software, políticas y procedimientos de seguridad que permiten a los usuarios autenticarse frente a otros usuarios y usar la información de los certificados de identidad (por ejemplo, las claves públicas de otros usuarios) para cifrar y descifrar mensajes, firmar digitalmente información, garantizar el no repudio de un envío, entre otros. Esta infraestructura usa el tipo de cifrado asimétrico o de clave pública explicado anteriormente.

2.13. Secure Sockets Layer (SSL)

Secure Sockets Layer y Transport Layer Security (TLS), su sucesor, son protocolos criptográficos que proporcionan comunicaciones seguras entre un cliente y un servidor cifrando los datos que viajan a través de una red. Si disponemos de un certificado de seguridad, visto con anterioridad, la comunicación se cifrará usando dicho certificado, por lo que podemos asegurar que ninguna tercera persona será capaz de leer el contenido de los mensajes que sean enviados por la red. [15]

De forma muy resumida vemos cómo funciona una conexión bajo este protocolo:

- El cliente y el servidor entran en un proceso de negociación, conocido como *handshake* (apretón de manos). Este proceso sirve para que se establezca algunos parámetros para realizar la conexión de forma segura (Intercambio de claves públicas, negociar algoritmos de cifrado, etc).

- Una vez terminada la negociación, la conexión segura es establecida.

- Usando claves preestablecidas, se codifica y descodifica todo lo que sea enviado hasta que la conexión se cierre.

2.14. HTTPS

Hypertext Transfer Protocol Secure es un protocolo basado en HTTP pero que incluye un cifrado basado en SSL/TLS. Con HTTPS se crea un canal cifrado apropiado para la transferencia de información sensible, función que no debería desempeñarse sobre HTTP ya que, en éste último, los datos circulan en texto plano y cualquiera podría acceder a ellos. [16]

La principal diferencia con respecto a HTTP es que HTTPS está libre de ataques *man-in-the-middle* y eavesdropping. También se aprecian diferencias en la forma de las URLs y el puerto en el que opera HTTPS, las URLs comienzan por *https://* en vez de *http://* y utiliza el puerto 443 y no el 80.

Para que un servidor acepte peticiones HTTPS, el administrador, entre otras cosas, ha de crear un certificado de clave pública para dicho servidor. También deberá distribuir la clave pública de ese certificado entre sus clientes para que se puedan conectar de forma segura.

3. Desarrollo del sistema

En este proyecto existen dos desarrollos claramente diferenciados, por un lado se ha de implementar un servidor, encargado de tener la base de datos actualizada y de atender a las llamadas realizadas por los distintos clientes móviles; y por otro lado, se tiene que implementar una aplicación móvil en Android, que será la encargada de posibilitar la creación, el listado o la obtención de los diferentes servicios.

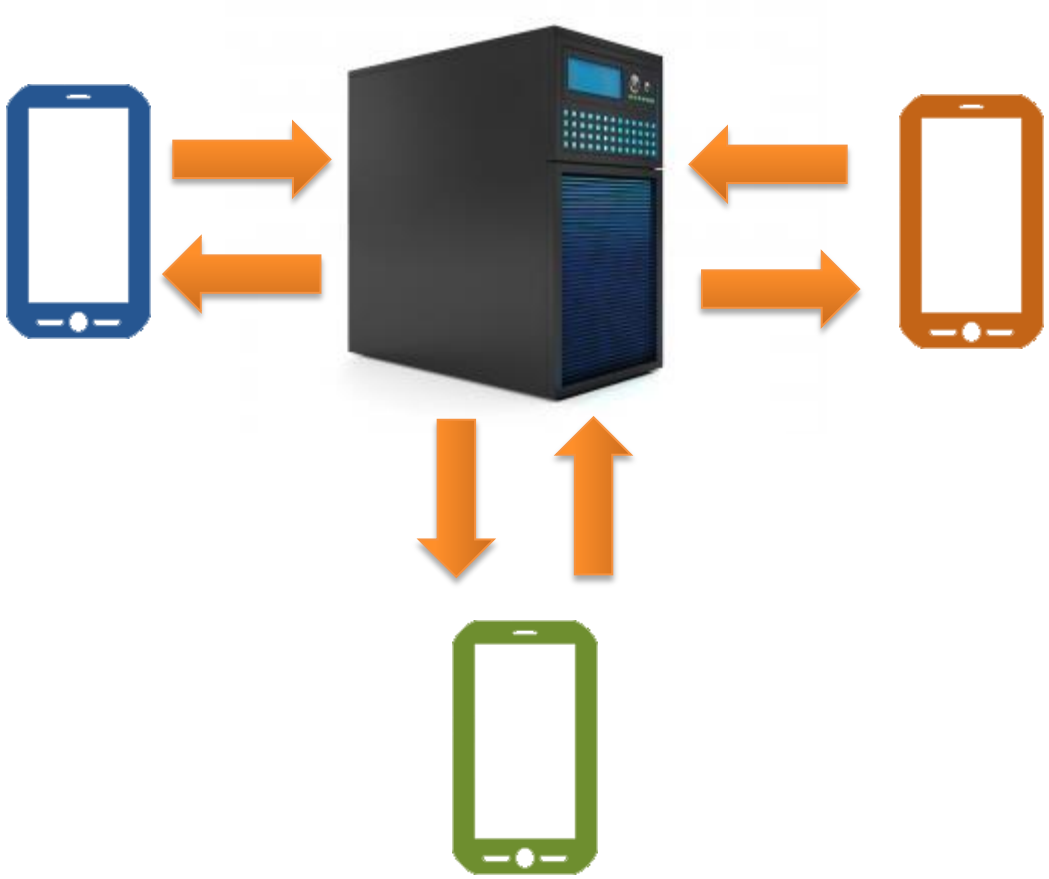


Imagen 6

Se aprecia en la Imagen 6 que el esquema es bastante sencillo. Inicialmente se comenzará describiendo de forma detallada la creación del servidor para después continuar con un profundo análisis de la implementación de la aplicación móvil.

3.1. Implementación del servidor

A continuación se verá paso por paso la implementación del servidor destacando los aspectos más relevantes del sistema. Primero se realizará un estudio sobre el problema para obtener los requisitos y especificaciones del servidor y así dar una solución lo más acertada posible.

3.1.1. Planteamiento del problema

Para que los usuarios se puedan conectar entre sí, puedan crear servicios, consultar los servicios que se ofrecen en un determinado instante de tiempo y consumir alguno de estos, se debe implementar un servidor, situado en la nube y que sea capaz de manejar todas esas peticiones.

El principal inconveniente que surge en la mayoría de desarrollos enfocados a la obtención de servicios a través de internet es la escalabilidad del servidor. En cuanto se ofrece un servicio por la red se ha de tener en mente que el número de usuarios conectados a la aplicación puede crecer de forma exponencial y en cualquier momento, por lo que el servidor ha de estar preparado para poder atender todas las peticiones y dar un buen servicio a los usuarios. De lo contrario la compañía se enfrentará a clientes insatisfechos con el producto en concreto y a la consecuente pérdida de cuota de mercado.

Para que el sistema cumpla su función de forma correcta es necesario recabar cierta información personal de los usuarios de la aplicación. Es primordial mantener la seguridad de estos datos sensibles, impidiendo el acceso de terceros en todo momento. Esta información, en ocasiones, ha de viajar a través de internet, por lo que se deben proporcionar conexiones seguras en todo momento. Esto se puede conseguir mediante técnicas de cifrado.

Se dispone de un periodo de tiempo limitado para la finalización del proyecto, por lo que se debe realizar un desarrollo del software lo más ágil posible. Se deben evitar el uso de tecnologías desconocidas, novedosas o con poca documentación, así se conseguirá implementar la aplicación de una forma rápida.

Hay que prestar especial cuidado con la robustez del software. Éste debe estar libre de fallos conseguir así fidelizar a los usuarios de la aplicación. Se deberá preparar una extensa batería de pruebas durante el desarrollo y al finalizar el mismo.

Resumiendo todo lo anterior, se tiene que crear un servidor ligero, seguro, robusto, escalable y en un periodo de tiempo lo más breve posible.

3.1.2. Soluciones propuestas

Para encuadrar, en la medida de lo posible, todo lo comentado anteriormente y tras haber barajado múltiples opciones, se ha concluido que la solución más acertada para desarrollar el servidor de datos es usar un framework como Spring.

Spring Framework, como se ha contado en la introducción de este documento, ayuda a concentrar el esfuerzo del programador en la funcionalidad de la aplicación, y no en la infraestructura del sistema. Promueve un mejor análisis y diseño del software y permite establecer un desarrollo ágil y escalable para las aplicaciones.

Su gran soporte para el patrón MVC facilita la creación de una aplicación modulable con una buena separación de responsabilidades, permitiendo obtener un código final con una mayor legibilidad y, por tanto, una mayor facilidad de modificación ante cambios en la especificación de la aplicación.

El apartado de la seguridad también está considerado en Spring Framework. Su módulo Spring Security añade una capa superficial a la aplicación con un acoplamiento escaso, por lo que su inclusión en el proyecto se realizó en una fase posterior, una vez finalizado todo el núcleo de la aplicación servidora.

Además de eso, Spring Framework tiene dos ventajas importantes: una de ellas es que se programa en Java. Evitamos con esto la pérdida de tiempo que conlleva el aprendizaje de un nuevo lenguaje de programación, puesto que hemos realizado previamente varios proyectos usando dicho lenguaje. Y la otra es que tiene su propio entorno de desarrollo basado en Eclipse, software con el cual estamos muy familiarizados.

3.1.3. Configuración de Spring Framework

Para comenzar con Spring Framework se comenzará explicando los archivos de configuración que hacen posible su correcto funcionamiento. Todos estos ficheros de configuración están en lenguaje XML por lo que su comprensión es relativamente fácil.

3.1.3.1. Esquema del proyecto Spring

En la Imagen 7 se puede ver el esquema del proyecto creado en Spring Tool Suite. Contiene gran número de carpetas y de archivos de configuración por lo que puede parecer algo confuso. Se intentará explicar de forma detallada cada uno de los estos archivos para esclarecer su funcionamiento.

Es un proyecto Maven-web con el complemento de las dependencias del framework de Spring:

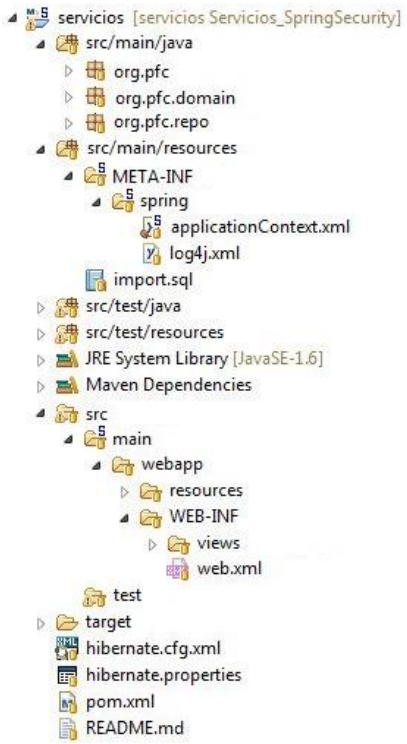


Imagen 7

Los directorios fundamentales del proyecto Spring son los siguientes:

- *src/main/java*: Contiene el código Java del proyecto. En él se han creado varios paquetes para mantener una estructura ordenada de las clases. Los paquetes son:
    - *org.pfc*: Contiene todos los controladores implementados para el proyecto.
    - *org.pfc.domain*: En su interior se encuentran las clases del modelo de datos.
    - *org.pfc.repo*: Incluye las clases necesarias para la conexión con la base de datos (DAO).
- *src/main/resources*: Este directorio contiene archivos XML de configuración que serán empaquetados e incluidos en el CLASSPATH. En este proyecto en concreto tenemos el fichero *log4j.xml*, para la configuración de los registros de la aplicación y *applicationContext.xml*, que se detallará posteriormente.
- *src/test/java* y *src/test/resources*: Son directorios relacionados con las pruebas de la aplicación. El primero contendrá los fuentes de los test y el segundo aquellos recursos necesarios, que no sean código java, necesarios para la ejecución correcta de las pruebas.

- *src/main/webapp/WEB-INF*: Contiene a la carpeta *views*, en la que se encuentran las vistas de la aplicación, y el archivo *web.xml*, que se analizará con detenimiento más adelante.

- *Hibernate.cfg.xml*: Es el archivo propio de la configuración de Hibernate. Se verá con más detalle posteriormente.

- *pom.xml*: *Project Object Model*. Es el archivo usado por Maven para describir el proyecto, resolver sus las dependencias, etc. A continuación se detalla la función y el contenido de este archivo:

3.1.3.2. El archivo pom.xml

El contenido del archivo *pom.xml* de este proyecto es demasiado extenso como para analizarlo íntegramente. No es relevante puesto que la mayoría de las etiquetas añadidas en este fichero cumplen la misma función. Se analizan las más importantes:

Todo el contenido del archivo va entre las etiquetas `<project>` y `</project>`. Es donde se incluirá el nombre del proyecto, la versión, sus dependencias, etc. A continuación vemos algunas de las etiquetas añadidas:

- `<repositories>`: Por defecto Maven descarga todas las dependencias de su repositorio central[5], pero algunas librerías pueden no estar en dicho repositorio. Así que, para casos como éste, se debe de indicar manualmente el repositorio que se quiere usar para que Maven pueda obtener dicha librería. En este proyecto se ha indicado el repositorio de Spring externo al propio de Maven de la siguiente forma:

```
<repositories>
    <repository>
      <id>spring-libs-milestone</id>
      <name>Spring Milestone Repository</name>
      <url>http://repo.springsource.org/milestone</url>
    </repository>
</repositories>
```

- `<dependencies>`: Esta es una de las etiquetas más destacadas del archivo. En ella se definen todas las dependencias necesarias para el proyecto. A continuación veremos unos ejemplos de ellas:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>3.2.3.RELEASE</version>
</dependency>
```

[5] http://central.maven.org/maven2/

El código anterior incluye al proyecto la dependencia del núcleo de Spring Framework. Un proyecto Spring en concreto requiere gran número de dependencias. No se incluirán más puesto que no añaden novedad alguna con respecto a la etiqueta que se está describiendo.

```
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.3</version>
</dependency>

<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-core-asl</artifactId>
    <version>1.9.3</version>
</dependency>
```

Las dos dependencias anteriores son imprescindibles para la conversión de los objetos al formato JSON.

```
<dependency>
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>1.3</version>
</dependency>
```

Commons-fileupload es necesaria para el envío y recepción de archivos a través de la red.

3.1.3.3. El archivo web.xml

Como en todos los proyectos web desarrollados con la especificación J2EE, es necesario un archivo llamado *web.xml* cuyo elemento raíz es `<web-app></ web-app>`. Este archivo se conoce cómo descriptor de despliegue y contiene la configuración del comportamiento general de la aplicación web. Por ejemplo los servlets que contendrá la aplicación web, la asignación de rutas por cada servlet, filtros, políticas de seguridad, etc. Se va a describir la forma en la que se ha ido construyendo este archivo de configuración.

- `<context-param>`: Este elemento permite definir parámetros que serán accesibles desde cualquier servlet de la aplicación web. Los dos elementos definidos en su interior especifican cuál será el archivo de configuración del contexto de la aplicación web:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
      classpath:/META-INF/spring/applicationContext.xml
  </param-value>
</context-param>
```

- `<listener>`: Esta etiqueta permite crear listeners. Estos listeners permiten realizar ciertas operaciones en el momento en el que sucede algún evento. Solamente se han tenido que definir dos de ellos para este proyecto:

```
<listener>
      <listener-class>
            org.springframework.web.context.ContextLoaderListener
      </listener-class>
</listener>

<listener>
      <listener-class>
            org.springframework.web.util.Log4jConfigListener
      </listener-class>
</listener>
```

  El primero de los listeners está relacionado con el arranque y parada del contexto raíz de Spring, WebApplicationContext y el segundo se crea para obtener registros o logs de la aplicación, para así poder detectar y solventar posibles errores.

- `<servlet>`: Es uno de los elementos más importantes del archivo *web.xml*. Mediante el uso de estas etiquetas se definen cada uno de los servlets que tendrá la aplicación web. Para este proyecto, solamente se ha necesitado un servlet y su definición es la siguiente:

```
<servlet>
    <servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            classpath:/META-INF/spring/applicationContext.xml
       </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

  Con este código se consigue crear un servlet cuyo nombre es *`Spring MVC Dispatcher Servlet`* y cuyo archivo de configuración es `applicationContext.xml`.

- `<servlet-mapping>`: Mediante este elemento se permite especificar un patrón de URL para asignarla a un servlet. Este servlet será el definido en el elemento hijo `<servlet-name>`:

```
<servlet-mapping>
    <servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

  Así se ha especificado que cualquier URLs será asignada a un único servlet. En este caso al servlet llamado `Spring MVC Dispatcher Servlet`.

Por defecto las llamadas HTTP que un cliente envía es pasada directamente a un servlet y la respuesta de este servlet es pasada directamente al cliente. En algunos casos, como por ejemplo en el ámbito de la seguridad, es conveniente preprocesar estas llamadas y las respuestas correspondientes. En esta aplicación servidora se capturan todas las peticiones y respuestas aplicándole un filtro de seguridad. Este filtro permite cumplir con el Artículo 9, Punto 1 de la LOPD visto anteriormente. Para realizar esto en el archivo *web.xml* se han definido las siguientes etiquetas:

- `<filter>`: Con este elemento se define el filtro comentado de la aplicación.

```
<filter>
          <filter-name>springSecurityFilterChain</filter-name>
          <filter-class>
             org.springframework.web.filter.DelegatingFilterProxy
          </filter-class>
</filter>
```

- `<filter-mapping>`: Este elemento define el patrón URL que especifica a qué destino se debe aplicar el filtro. El destino se especifica mediante la etiqueta `<filter-name>`.

```
<filter-mapping>
      <filter-name>springSecurityFilterChain</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
```

  El patrón definido en `<url-pattern>/*</url-pattern>` indica que el filtro se aplicará a todas las URLs de la aplicación.

3.1.3.4. El archivo applicationContext.xml

Spring Framework permite construir aplicaciones con una jerarquía entre contextos de servlets. Así, si un bean no se encuentra presente en el contexto de un servlet se buscará en el contexto del padre.

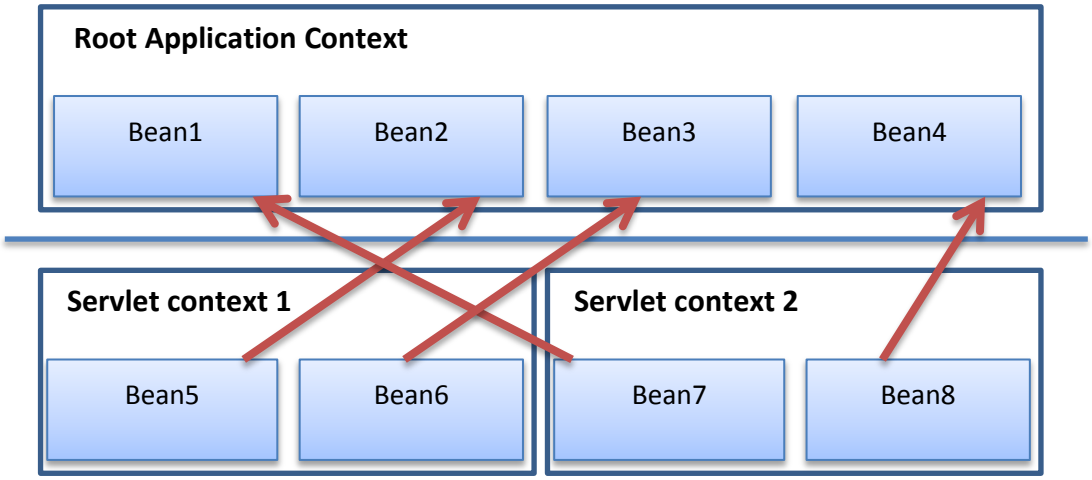


Imagen 8

Si se desea que un bean sea accesible desde cualquier servlet de la aplicación deberá ser añadido al archivo *applicationContext.xml*. Por tanto es el que define el contexto raíz de la aplicación web.

Debido a que en el proyecto sólo existe un servlet encargado de atender a las distintas peticiones de los clientes, solamente se usará este archivo de configuración, no es necesario definir ningún contexto secundario.

Ahora se verán los puntos clave del archivo *applicationContext.xml*:

- `<context:component-scan base-package="org.pfc" />`: Spring brinda da la posibilidad de automatizar el registro de clases estereotipadas como beans. Con esta etiqueta se le indica a Spring que detecte estas clases desde el paquete base `"org.pfc"` para que registre los bean en el contexto de la aplicación.

- `<mvc:annotation-driven />`: Con esta etiqueta se registrarán dos beans, DefaultAnnotationHandlerMapping y AnnotationMethodHandlerAdapter con unos valores por defecto. Estos beans son los encargados de enviar las peticiones HTTP a los aquellos otros beans registrados de las clases anotadas con `@Controller`.

- `<tx:annotation-driven />`: Esta etiqueta permite que los bean registrados de aquellas clases que esten anotadas con `@Transactional` sean envueltos en proxies. Estos proxies serán los encargados de ejecutar los métodos transaccionales. Si por algún motivo se lanzara una RuntimeException, entonces en la base de datos se haría rollback, volviendo a un estado posterior y manteniendo la consistencia de los datos. Sin embargo, si se lanza otro tipo de excepción (que no sea RuntimeException), no se haría rollback (a menos que configures en la propia anotación que para esa excepción concreta sí tenga ese comportamiento).

- `<bean />`: Un bean es un componente software reutilizable con unas particularidades determinadas. En Spring Framework son creados y manejados por el contenedor Spring. Gracias a la etiqueta `<bean />` se pueden definir los beans necesarios en el archivo *applicationContext.xml*. Se han incluido en el archivo 7 beans:

    - `viewResolver`: Encargado de obtener el nombre de una vista del controlador, añadir un prefijo (`"/WEB-INF/views/"`) y un sufijo (`".jsp"`), enviar el directorio resultado y delegar el trabajo de mostrar la vista.

    - `sessionFactory`: Su función es la de cargar toda la configuración de Hibernate desde el archivo `"hibernate.cfg.xml"`.

    - `transactionManager`: Este bean pertenece a Hibernate y se usa para las transacciones que se realizarán en la base de datos.

    - `log4jInitialization`: Es el encargado de los registros o logs de la aplicación usados para la detección de errores.

  - *encoder*: Gracias a este bean la clave secreta de acceso a la aplicación de la tabla *usuarios* permanece cifrada. Para ello se hace uso de la clase *"BCryptPasswordEncoder"*.

  - *dataSource*: Este es el bean que contiene la información de conexión a la base de datos.

  - *multiPartResolver*: La obtención o envío de un archivo por partes se consigue gracias a este bean.

- `<security:http />`: Este elemento es el encargado de crear el *FilterChainProxy* llamado "*springSecurityFilterChain*" que mantiene la pila de filtros de seguridad y el que crea la configuración de la seguridad. Se ha indicado una configuración básica para la aplicación con el valor `auto-config="true"` y se han definido los puertos a asegurar con `<security:port-mapping http="8080" https="8443"/>`.

- `<security:global-method-security secured-annotations="enabled" />`: También se permite la configuración de la seguridad de la aplicación mediante anotaciones propias de Spring Framework. Con esta etiqueta se habilita el uso de estas anotaciones en el código Java para permitir que sólo aquellos usuarios indicados en la una anotación concreta sean lo que puedan llamar a los métodos anotados.

- `<security:authentication-manager />`: Para realizar un proceso de autenticación en la aplicación, es necesario añadir esta etiqueta. En la configuración se ha añadido un `<authentication-provider />` que es el que devolverá si un usuario está o no en la base de datos. También hace uso del bean comentado anteriormente, *encoder*, para el cifrado de la clave privada de los usuarios a través de Spring.

- `<security:jdbc-user-service />`: Esta etiqueta define el `<authentication-provider />` para la aplicación puesto que se está utilizando una base de datos MySQL. Aquí, se ha de incluir las consultas SQL que deberá hacer Spring para la obtención de los usuarios y de los roles de estos usuarios en el proceso de autenticación:

  - Para obtener los usuarios de la base de datos: `users-by-username-query="select email as username, claveAcceso as password, true as enabled from usuarios join usuarios_privado on usuarios.idUsuario = usuarios_privado.idUsuario and email=?"`

  - Para obtener los roles de estos usuarios: `authorities-by-username-query="select email as username, rol as authorities from usuarios where email=?"`

3.1.4. Implementación del modelo de datos

El modelo de la aplicación del servidor se resume en tres clases principales: la clase *Usuario*, la clase *UsuarioPrivado* y la clase *Servicio*.

El diseño del modelo de datos de la aplicación ha tenido un matiz a resaltar, la separación de los datos de un usuario en dos clases distintas, por un lado tenemos la clase *UsuarioPrivado* y por otro la clase *Usuario*. Esta división ha sido realizada para mantener dos tablas aisladas en la base de datos, una con los datos sensibles de un usuario y la otra con los datos menos comprometidos del mismo, proporcionando con esta decisión una mayor seguridad. Se puede apreciar en el diagrama de clases siguiente lo comentado en este párrafo junto con el detalle de la clase *Servicio*:

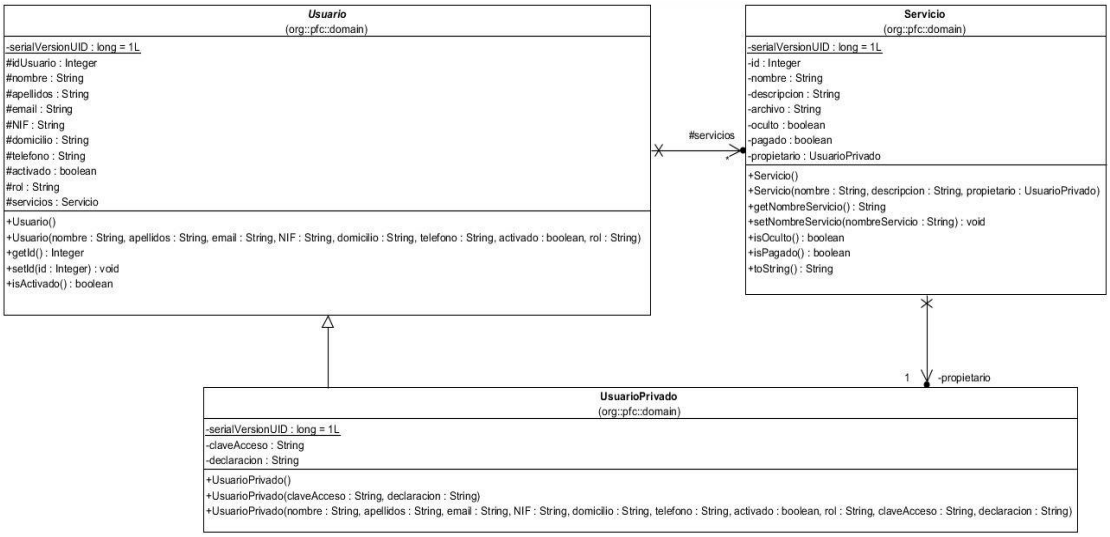


Diagrama de clase 1

Por otro lado, para almacenar la compra de un servicio por un usuario en la aplicación, se ha creado una segunda entidad llamada *ServicioUsuario*. Esta clase también contiene un atributo que almacenará el valor de la fecha en la cual el usuario obtuvo el servicio:

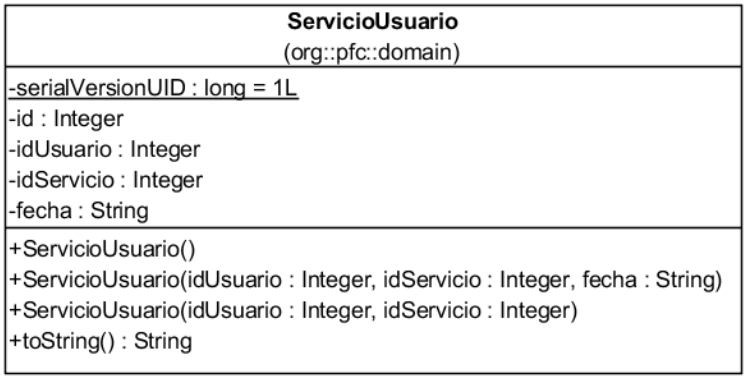


Diagrama de clase 2

3.1.4.1. Configuración de Hibernate

El archivo principal de Hibernate en un proyecto es el llamado *hibernate.cfg.xml*, se encuentra situado en la raíz del proyecto y en él se establecerán los parámetros de conexión a la base de datos MySQL y las clases que queramos que Hibernate relacione con tablas en la base de datos. A continuación se muestra el contenido de dicho archivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory name="configLocation">
  <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="connection.url">jdbc:mysql://localhost:3306/pfc</property>
  <property name="connection.username">root</property>
  <property name="connection.password"/>

  <mapping class="org.pfc.domain.Usuario"/>
  <mapping class="org.pfc.domain.UsuarioPrivado"/>
  <mapping class="org.pfc.domain.Servicio"/>
  <mapping class="org.pfc.domain.ServicioUsuario"/>

 </session-factory>
</hibernate-configuration>
```

Como se puede ver, la primera parte en la cual se definen los parámetros de conexión a la base de datos se incluyen con el tag `<property>`. Si se desea que una clase tenga una tabla asociada directamente en la base de datos se añadirá usando la etiqueta `<mapping>`, tal y como se puede ver en el código XML anterior.

Existen dos formas de comunicarle a Hibernate la información necesaria para que pueda crear las tablas en la base de datos de forma correcta. Se puede indicar esta información a través de archivos XML, creando un archivo por cada clase que queramos persistir; o directamente desde el código Java de la clase, usando anotaciones.

Hibernate es una implementación de la API de persistencia de Java (JPA), por lo que puedes usar indistintamente las anotaciones de Hibernate o las propias de JPA. Pero, al ser JPA el estándar, se aconseja usar las anotaciones de ésta API a menos que se desee usar alguna funcionalidad específica implementada en Hibernate y no en JPA.

Los proyectos web, y en concreto los proyectos realizados con Spring Framework, generan muchos archivos XML de configuración, así que se ha optado por el uso de las anotaciones de la JPA que están presentes desde el desarrollo de la rama 3.x de Hibernate.

Se muestra a continuación cómo queda la clase Java junto con las anotaciones de JPA para la persistencia en la base de datos:

```
@Entity
@Table(name="servicios")
@JsonIgnoreProperties(ignoreUnknown=true)
public class Servicio implements Serializable{

	private static final long serialVersionUID = 1L;
	@Id
	@GeneratedValue
	@Column
	private Integer id;
	@Column
	private String nombre;
	@Column
	private String descripcion;
	@Column
	private String archivo;
	@Column
	private boolean oculto;
	@Column
	private boolean pagado;
	@ManyToOne
	@JoinColumn(name="id_usuario")
	private UsuarioPrivado propietario;

	public Servicio() {
	}

	public Servicio(String nombre, String descripcion,
UsuarioPrivado propietario){
		this.nombre = nombre;
		this.descripcion = descripcion;
		this.propietario = propietario;
	}

	/*Getters and setters*/
}
```

Se han ocultado los getter y setter de la clase ya que no son relevantes. Se analizarán a continuación los aspectos más importantes del código anterior relacionados con Hibernate:

- `@Entity`: Indica a Hibernate que la clase será una entidad.

- `@Table(name="servicios")`: Se especifica a Hibernate que esa clase se verá reflejada en una tabla cuyo nombre será `"servicios"`.

- `@Id`: Esta anotación indica que el atributo sobre el que está situado será clave primaria de una entidad, en este caso concreto de la entidad *servicios.*

- `@GeneratedValue`: Indica que el atributo será autogenerado. Se usa para las claves primarias.

- `@Column`: Crea una fila en la tabla. Esta fila tendrá el mismo tipo y nombre del atributo por defecto.

- `@ManyToOne`: Define una asociación con otra entidad o clase. Esta asociación tiene una multiplicidad muchos a uno, un usuario puede tener muchos servicios, pero los servicios son propiedad exclusivamente de un solo usuario.

- `@JoinColumn(name="id_usuario")`: Especifica una columna en la tabla para unir una asociación. Es atributo `name="id_usuario"` indica el nombre de la columna de la tabla con la que se está realizando la unión. Esta columna es la clave externa.

Nota:

Para indicar a Hibernate que las clases UsuarioPrivado y Usuario debían relacionarse en la base de datos como tablas separadas, conformaban ambas una misma entidad y que las tablas compartirían identificador único, se han tenido que seguir 4 pasos:

1. ***La clase Usuario se ha definido como abstracta.***

2. ***La clase UsuarioPrivado hereda de Usuario.***

3. ***A la clase abstracta Usuario se le ha añadido la anotación*** `@Inheritance(strategy=InheritanceType.JOINED)`.

4. ***A la clase UsuarioPrivado se le ha añadido la anotación*** `@PrimaryKeyJoinColumn(name="idUsuario")`***, en la que*** `name="idUsuario"` ***hace referencia a la clave primaria de la clase abstracta Usuario.***

3.1.4.2. Implementación de las interfaces DAO

El Data Access Object es un componente software que suministra interfaces entre la aplicación y los dispositivos de almacenamiento, en este proyecto el almacenamiento sería la base de datos SQL.

Como la implementación de las interfaces de acceso a la base de datos es muy similar entre ellas, se explicará solamente la interfaz correspondiente a los servicios y la clase que la implementa. A continuación se muestra el diagrama de *IServicioDAO* y de la clase *ServicioDAO*.

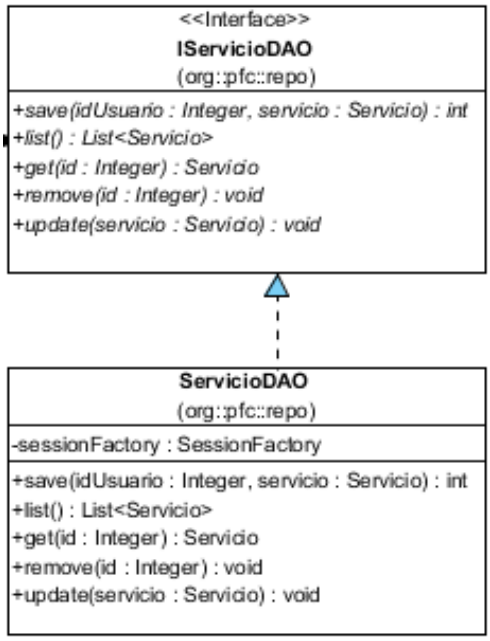


Diagrama de clase 3

Los métodos de la interfaz *IServicioDAO* son los tradicionales para la gestión de entidades en una base de datos. Con estos métodos se permite almacenar un servicio, listar el total de los servicios almacenados, obtener un único servicio a través de su identificador, eliminar y actualizar un servicio determinado.

El código de esta interfaz se puede ver a continuación:

```
@Repository
public interface IServicioDAO {
	@Transactional
	public abstract int save(Integer idUsuario, Servicio servicio);
	@Transactional
	public abstract List<Servicio> list();
	@Transactional
	public abstract Servicio get(Integer id);
	@Transactional
	public abstract void remove(Integer id);
	@Transactional
	public abstract void update(Servicio servicio);
}
```

En el código se puede ver cómo se han anotado tanto la definición de la interfaz como la definición de los métodos que contiene. Por un lado, con la anotación `@Repository` se le indica a Spring que esta interfaz está relacionada de alguna forma con una capa de persistencia de datos. Por otra parte, con la anotación `@Transactional`, se permite que los métodos sean transaccionales, es decir, que si ocurre algún error en uno de los métodos anotados, en la base de datos se vuelva a un punto en el que permanezca la integridad y consistencia de los datos.

Una vez vista la interfaz se analizará la clase que implementa esa interfaz, *ServicioDAO*. No se verán todos los métodos de esta clase puesto que son similares, solo se comentará el método *save()*:

```
@Repository
public class ServicioDAO implements IServicioDAO {

	@Autowired
	private SessionFactory sessionFactory;

	@Secured("ROLE_USUARIO")
	@Override
	public int save(Integer idUsuario, Servicio servicio){
		Session session = sessionFactory.getCurrentSession();
		UsuarioPrivado usuario = (UsuarioPrivado) session.get(
			UsuarioPrivado.class, idUsuario);
		servicio.setPropietario(usuario);
		session.save(servicio);
		session.flush();
		session.update(servicio);
		return servicio.getId();
	}
}
```

Puede resultar peculiar ver la anotación `@Repository` en la definición de la clase al igual que se encontraba en la definición de la interfaz que implementa, esto es necesario puesto que las anotaciones no se heredan por lo que hemos de comunicar a Spring que esta clase también está relacionada con la capa de persistencia de datos.

La siguiente anotación que se puede ver es `@Autowired`. Aquí se presenta una de las características principales de Spring, la inyección de dependencias. Con `@Autowired`, se le está indicando a Spring que cuando vaya a crear la instancia de *ServicioDAO* debe pasarle una referencia a la clase *SessionFactory*. Esta referencia será un bean incluido, mediante la etiqueta `<bean />`, en el archivo *applicationContext.xml* como ya se ha visto previamente.

La siguiente anotación importante en esta clase es `@Secured("ROLE_USUARIO")`. Justo aquí entra en juego la capa de seguridad incluida por Spring Security. Como se puede observar, esta anotación acepta un parámetro (String o un array de Strings), en este caso `"ROLE_USUARIO"`, que indica qué rol de usuario permite realizar una llamada al método que contiene dicha anotación. La tabla de la base de datos que contiene a los usuarios de la aplicación ha de tener un campo que indique el rol de ese usuario en nuestro sistema.

Es posible incluir un conjunto de roles en esta anotación para permitir que distintos tipos de usuarios puedan realizar la llamada a un mismo método. Por ejemplo el método *remove()* de la clase *ServicioDAO* permite que tanto un usuario con rol `"ROLE_USUARIO"`, como otro usuario con rol `"ROLE_ADMIN"`, puedan llamar ambos a dicho método. La definición este método es la siguiente:

```
@Secured({"ROLE_USUARIO","ROLE_ADMIN"})
@Override
public void remove(Integer id) {}
```

Más adelante se podrá ver cómo han de activarse las anotaciones `@Secured()` desde el archivo *applicationContext.xml*.

Aunque no se explicarán el resto de interfaces creadas para esta la aplicación del servidor, añadiremos simplemente su diagrama de clases para ver la similitud entre éstas.

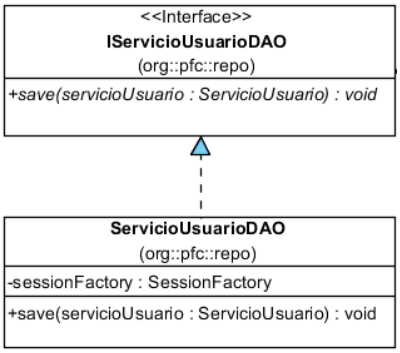


Diagrama de clase 4

La clase *ServicioUsuarioDAO* sólo implementa el método *save()* porque simplemente se desea tener un registro de los usuarios que consumen un servicio determinado. Por ahora es suficiente implementar únicamente esa funcionalidad, en un futuro si se estimara oportuno se podrán añadir demás métodos.

La clase *UsuarioDAO* es muy similar a la clase *ServicioDAO* explicada con detalle anteriormente. Para la clase *UsuarioDAO* sí que dispone de todos los métodos para la manipulación de la base de datos puesto que eran necesarios para el correcto funcionamiento del sistema:

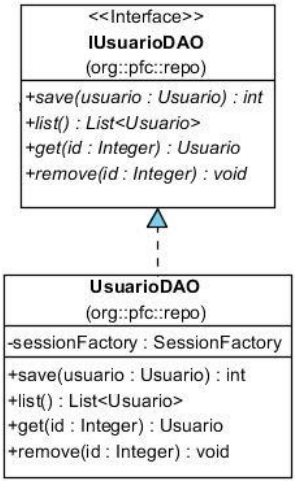


Diagrama de clase 5

3.1.5. Implementación de los controladores

Una vez se ha finalizado la implementación del modelo de la aplicación se verá cómo se han creado los controladores que atenderán a las distintas peticiones de los usuarios.

En este sistema sólo ha bastado con el desarrollo de dos controladores, uno que se encargará de la gestión de los servicios y otro encargado de la gestión de los usuarios. Debido a una mayor complejidad en la implementación, se profundizará en el controlador *ServicioController*.

Así se queda el diagrama de clases de ambos controladores:

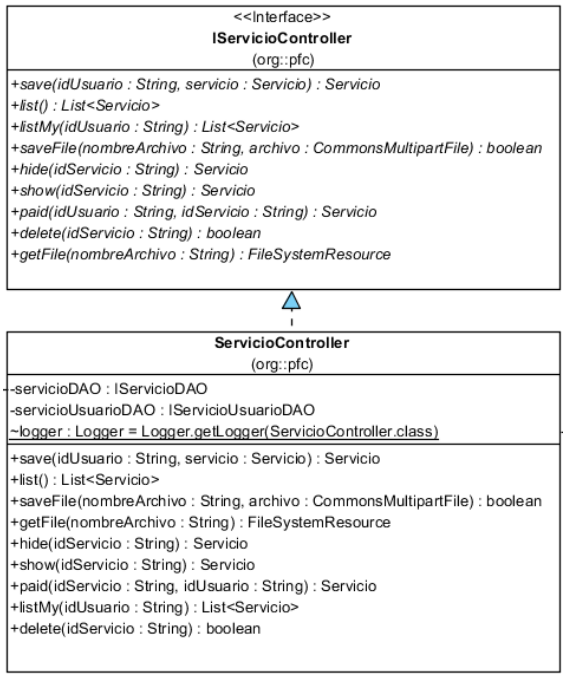


Diagrama de clase 6

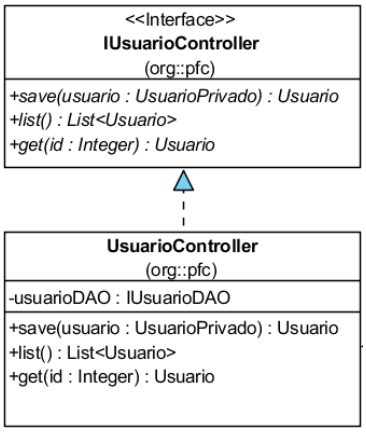


Diagrama de clase 7

Como se puede ver en el Diagrama de clase 7, con sólo tres métodos en el controlador de los usuarios ha bastado hasta ahora para el correcto funcionamiento de la aplicación. Sin embargo, para gestionar los servicios se ha necesitado algo más de lógica de negocio.

3.1.5.1. Interfaz IServicioController

Se analizará por partes la construcción del controlador, comenzando por la definición de la interfaz *IServicioController*:

```
@Controller
@RequestMapping(value="/servicio")
public interface IServicioController {

}
```

Podemos ver que tiene dos anotaciones encima de la definición de la clase. La primera, `@Controller`, indica a Spring que esta interfaz va a actuar con el rol de controlador y, como se ha permitido previamente la auto-detección con `<mvc:annotation-driven />` en el archivo *applicationContext.xml*, se evita tener que definir el bean por nosotros mismos.

La siguiente anotación que podemos ver es la de `@RequestMapping( value="/servicio")`. Esta es la forma de indicar a Spring que las peticiones que se realicen a ***https://miservidor:mipuerto/servicio*** las manejará este controlador.

Una vez vista la definición de la interfaz se analizarán cada uno de sus métodos:

Método save()

La definición del primer método de la interfaz *IServicioController* es la siguiente:

```
@RequestMapping(value = "/save", consumes="application/json", method =
        RequestMethod.POST)
public abstract @ResponseBody Servicio save(String idUsuario, Servicio
        servicio);
```

Nuevamente aparece la anotación `@RequestMapping`, en este caso algo más completa. Como se puede apreciar tiene tres atributos:

- `value = "/save"`: Este método se ejecutará cuando desde un cliente se realice una petición a la dirección ***https://miservidor:mipuerto/servicio/save***.

- `consumes="application/json"`: Espera que la petición que llama a este método contenga datos en el formato JSON para poder consumirlos.

- `RequestMethod.POST`: Por último la petición tiene que haber sido realizada a través del método POST del protocolo HTTP.

La anotación `@ResponseBody` configura la respuesta a la petición realizada por el cliente. Con ella se consigue que envíe como respuesta el objeto que ha sido devuelto por el método que haya manejado la petición en vez de devolver una página JSP. Dicha respuesta puede ser devuelta en distintos formatos como XML o JSON.

Si la petición del cliente tiene en la cabecera *accept=application/json* devolverá los datos en JSON, siempre que en el classpath del proyecto se encuentre añadida la librería *Jackson-Mapper*; si por otro lado, el cliente tuviera en la cabecera de la petición HTTP *accept=application/xml*, datos serían devueltos en formato XML.

Este comportamiento puede ser cambiado si queremos que exclusivamente devuelva el objeto en un formato determinado. Si por ejemplo se quisiera obligar a que los datos sólo retornaran en formato JSON, se debería añadir a la anotación `@RequestMapping` el atributo `produces="application/json"`.

Método list()

```
@RequestMapping(value = "/list", method = RequestMethod.GET, produces
        = "application/json")
public abstract @ResponseBody List<Servicio> list();
```

Este método la única novedad no vista hasta ahora es el uso de `method = RequestMethod.GET`, indicando que la petición ha de ser a través del método GET.

Método listMy()

```
@RequestMapping(value = "/{idUsuario}/listMy", method =
        RequestMethod.GET, produces = "application/json")
public abstract @ResponseBody List<Servicio> listMy(String idUsuario);
```

Se puede ver que el atributo de la anotación `@RequestMapping`, `value = "/{idUsuario}/listMy"`, el *idUsuario* va entre corchetes. Esto se utiliza, junto con la anotación `@PathVariable` para realizar peticiones al servidor con parámetros. Se verá con más detalle cuando se analice la implementación de este método.

Método saveFile()

```
@RequestMapping(value = "/saveFile", method = RequestMethod.POST)
public abstract @ResponseBody boolean saveFile(String nombreArchivo,
        CommonsMultipartFile archivo);
```

Este método se implementará con la función de almacenar un archivo, pasado desde un cliente, en el servidor. Para ello se hace uso de la clase `CommonsMultipartFile` y la anotación `@RequestParam`, que se colocará en la implementación del método.

Método getFile()

```
@RequestMapping(value = "/getFile", method = RequestMethod.GET,
       produces = MediaType.APPLICATION_OCTET_STREAM_VALUE)
public abstract @ResponseBody FileSystemResource getFile(String
       nombreArchivo);
```

Fijándonos en el valor del atributo `produces` de la anotación `@RequestMapping` vemos que éste es el único método que no devuelve un JSON como respuesta a la petición, usamos `produces = MediaType.APPLICATION_OCTET_STREAM_VALUE` para devolver un fichero.

3.1.5.2. Clase ServicioController

La definición de la clase *ServicioController* que implementa la interfaz *IServicioController* no tiene nada relevante a destacar salvo que ha de anotarse también con `@Controller` y que posee tres atributos:

```
@Autowired
private IServicioDAO servicioDAO;
@Autowired
private IServicioUsuarioDAO servicioUsuarioDAO;
static Logger logger = Logger.getLogger(ServicioController.class);
```

Los dos atributos primeros son referencias a las distintas clases relacionadas con los servicios que se comunican con la base de datos, están anotados con `@Autowired`, anotación comentada con anterioridad. El tercer atributo lo se usa para llevar un registro de sucesos en la aplicación. Ahora se repasarán los métodos relevantes de la implementación de *IServicioController*.

Método saveFile()

```
@Override
public boolean saveFile(@RequestParam String nombreArchivo,
             @RequestParam CommonsMultipartFile archivo) {
       String ruta = "C:"+File.separator
              +"archivosSpring" + File.separator + nombreArchivo;
       if (!archivo.isEmpty()) {
      try {
          byte[] bytes = archivo.getBytes();
          BufferedOutputStream stream =
            new BufferedOutputStream(new FileOutputStream(new
            File(ruta)));
          stream.write(bytes);
          stream.close();
          return true;
      } catch (Exception e) {
          return false;
      }
  }
       return false;
}
```

Como se ha comentado anteriormente, el cliente realiza una petición para almacenar el archivo en el servidor. Esta petición contiene dos parámetros, el nombre del archivo y el archivo en sí. Estos dos parámetros se indican por medio de la anotación `@RequestParam`.

El resto del método es una simple escritura en disco en Java, lo más relevante es que los datos los obtiene de los bytes del atributo `archivo`, recibido a través de internet.

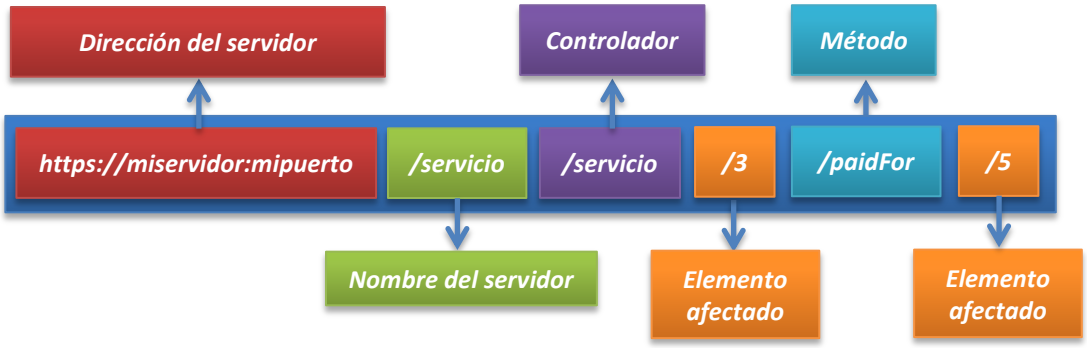
Método paid()

```
@Override
public Servicio paid(@PathVariable String idServicio, @PathVariable
        String idUsuario) {

        Servicio servicio = servicioDAO.get(
                Integer.valueOf(idServicio));

        if(servicio.isPagado()) return null;

        servicio.setPagado(true);
        servicioDAO.update(servicio);
        ServicioUsuario servicioUsuario = new ServicioUsuario(
                Integer.valueOf(idUsuario), Integer.valueOf(idServicio));
        servicioUsuarioDAO.save(servicioUsuario);
        return servicio;
}
```

Este método emula la realización de una compra de un servicio por un usuario, su implementación es bastante sencilla pero deja muy clara la responsabilidad de cada uno de los módulos de la aplicación y el uso de la anotación `@PathVariable`.

Se puede observar que los dos parámetros del método *idServicio* e i*dUsuario* están acompañados de `@PathVariable`. Con esta anotación se pueden pasar parámetros a través de la URL directamente, no como se hacía con la anotación `@RequestParam`.

Un ejemplo de paso de parámetros a través de una URL podría ser: ***https://miservidor:mipuerto/servicio/3/paidFor/5***, donde ***3*** es el identificador de un servicio y ***5*** es el identificador del usuario que ha consumido el servicio ***3***. En la imagen siguiente se explica la URL de forma detallada:



Siguiendo con el ejemplo y para terminar de explicar el método, la primera función es obtener el servicio de la base de datos cuyo identificador es 3 y comprobar que no ha sido pagado previamente. Si no ha sido pagado se "efectúa el pago" modificando el valor del atributo *pagado* del objeto *Servicio* y se actualiza la base de datos. Seguidamente se crea y se almacena en la base de datos un objeto *ServicioUsuario* para llevar un registro acerca de los servicios que han comprado los usuarios de nuestro sistema.

El resto de métodos de la clase no se analizarán puesto que cumplen funciones similares a las ya comentadas, acceso a la base de datos, actualización de su contenido, etc.

3.1.6. Cifrado del contenido de la base de datos

Para evitar que personas no autorizadas puedan obtener los datos de carácter personal de los usuarios de nuestro sistema se ha optado por cifrar el contenido de la base de datos. Para ello se ha usado la librería Jasypt y el cifrado de claves de Spring.

3.1.6.1. Jasypt

Como ya se ha visto en la introducción, Jasypt es una librería que permite añadir capacidades básicas de cifrado. Esta librería es bastante dinámica y muy configurable, se puede añadir desde un cifrado simple hasta tener una configuración bastante completa. Además, se integra perfectamente con Spring e Hibernate.

Jasypt puede ser configurado tanto por archivos XML como por anotaciones. Se ha decidido usar anotaciones siempre que se pueda, así el proyecto no se verá sobrecargado de archivos XML de configuración y estos serán a su vez más pequeños, facilitando así su comprensión.

La web de Jasypt expone que la definición de los tipos de cifrado con `@TypeDef`, puede realizarse dentro de la propia clase o también definiendo `@TypeDefs` en un archivo *package-info.java*. Como únicamente se necesitará un tipo de cifrado ha bastado con añadir un solo `@TypeDef` en la clase *UsuarioPrivado.*

La configuración queda de la siguiente manera:

```
    @TypeDef(
   name="encryptedString",
         typeClass=EncryptedStringType.class,
         parameters= {
             @Parameter(name="registeredName",
                  value="strongHibernateStringEncryptor"),
             @Parameter(name="password", value="jasypt"),
             @Parameter(name="algorithm",
                  value="PBEWithMD5AndTripleDES")
         }
)
```

En el tipo de cifrado se puede ver que es de la clase `EncryptedStringType`, puesto que se cifrarán cadenas de caracteres, y los parámetros siguientes son pares clave-valor. El parámetro más importante de la lista es `@Parameter (name="algorithm", value = "PBEWithMD5AndTripleDES")`, que indica el tipo de algoritmo a utilizar en el cifrado de los datos, en este caso se ha usado MD5 para la generación de la clave y triple DES para el cifrado.

Para indicar qué campos se desean cifrar en la base de datos solamente se ha de indicar con la anotación `@Type(type = "encryptedString")` directamente sobre los atributos de la clase Java, tal como se muestra a continuación:

```
@Type(type="encryptedString")
private String declaracion;
```

La anotación anterior también se encuentra en todos los atributos de la clase *Usuario* (salvo el atributo *claveAcceso* por lo que se explicará en el siguiente apartado), con ello esto se dificultará el acceso a los datos por terceras personas ajenas a la organización cumpliendo de nuevo con el Artículo 9, Punto 1 de la LOPD.

3.1.6.2. Cifrado de password Spring

Nota:

Al intentar integrar Jasypt en el proyecto para cifrar la información de la base de datos e intentar acceder desde la aplicación móvil al servidor, se obtuvo un error 401 Unauthorized. Esto era debido a que Spring estaba comparando la clave no cifrada que tenía almacenada la aplicación móvil con la clave cifrada que se encontraba almacenada en la base de datos. Cuando se descubrió este error para solventarlo se cifró todos los datos con Jasypt menos el campo de la contraseña del usuario, éste último se dejó para que fuera Spring el que lo cifrara.

Como se ha comentado con previamente, para ceder el cifrado de este campo a Spring se debe añadir a la etiqueta `<security:authentication-provider>` situada en el archivo *applicationContext.xml* la siguiente línea:

```
<security:password-encoder ref="encoder"></security:password-encoder>
```

Y añadir en el método *save()* de la clase UsuarioController las siguientes sentencias:

```
PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
usuario.setClaveAcceso(passwordEncoder.encode(
      usuario.getClaveAcceso()));
usuarioDAO.save(usuario);
```

Analizando el código se puede apreciar cómo se cifra la clave de acceso del usuario antes de almacenar dicho usuario en la base de datos. Este método aplica el algoritmo SHA-1 combinado con 8 bytes o más de un salt generado de forma aleatoria.

3.1.7. Conexiones seguras mediante HTTPS

Para asegurar la conexión entre el servidor y los clientes móviles se ha de crear un certificado de seguridad que contenga un par de claves, una pública y otra privada. Con las claves del certificado se cifrará la conexión de la misma forma que se ha mostrado en la introducción del cifrado asimétrico.

3.1.7.1. Creación del certificado con Keytool

Keytool es una herramienta del JRE de gestión de claves y certificados. Permite gestionar un almacén de claves criptográficas y crear certificados de seguridad a través de comandos.

Para generar un certificado basta con introducir en la consola de comandos la siguiente línea:

```
keytool -genkey -alias tomcat -keyalg RSA
```

A continuación el programa irá solicitando la siguiente información:

1. **Contraseña para el almacén de claves:** almacentomcatservicios.
2. **Nombre y apellido:** Jorge Manzano.
3. **Unidad de organización:** Servicios.
4. **Nombre de organización:** Servicios.
5. **Ciudad:** Guadix.
6. **Provincia:** Granada.
7. **Código País:** 34.
8. **Contraseña clave tomcat:** almacentomcatservicios.

El programa al finalizar la solicitud de datos queda como muestra la siguiente imagen:

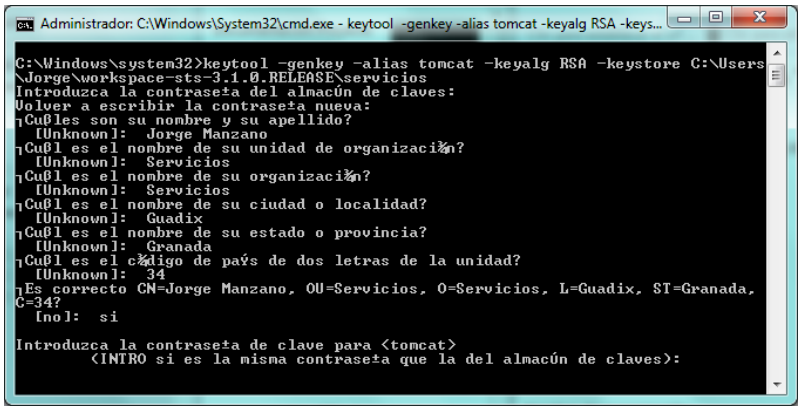


Imagen 9

Una vez finalizado el proceso se puede verificar que no ha ocurrido ningún fallo visualizando el almacén de claves con el siguiente comando e introduciendo la contraseña del almacén de claves:

```
keytool -list
```

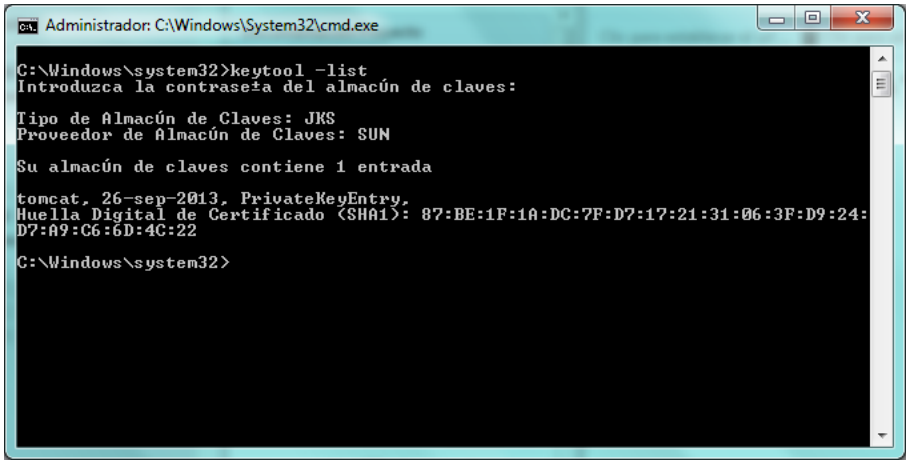


Imagen 10

Se puede ver en la imagen que existe una entrada en el almacén de claves por lo que el certificado se ha creado correctamente.

Si analizamos el almacén de claves se puede ver que su tipo es *JKS*, esto resultó ser un problema debido a que Android no soporta este formato. Un almacén de claves compatible para Android ha de ser del tipo *BKS*.

Para conseguir un almacén de este tipo se debe hacer uso de la librería *BouncyCastle,* más concretamente, el archivo que se descargó en su momento fue *bcprov-jdk16-145.jar.*

Una vez almacenado este fichero en el ordenador se han de seguir los siguientes pasos de configuración:

1. Se copia el archivo *bcprov-jdk16-145.jar* a todas las siguientes carpetas para evitar problemas:

    a. C:\Program Files (x86)\Java\jre7\lib\ext
    b. C:\Program Files\Java\jdk1.7.0_17\jre\lib\ext
    c. C:\Program Files\Java\jre7\lib\ext

2. Se añadirá la siguiente línea: *security.provider.7=org.bouncycastle.jce.provider.BouncyCastleProvider* a los archivos *java.security* de las siguientes carpetas:

    a. C:\Program Files (x86)\Java\jre7\lib\security
    b. C:\Program Files\Java\jdk1.7.0_17\jre\lib\security
    c. C:\Program Files\Java\jre7\lib\security

3. Se debe agregar la siguiente variable de entorno en "Variables de usuario": *CLASSPATH=%CLASSPATH%;c:\bcprov-ext-jdk15on-1.46.jar*

Ahora se puede usar correctamente *BouncyCastle*. Para crear el nuevo almacén de claves se ejecutará el siguiente comando y se insertarán los datos de la misma manera que para el primer certificado:

```
keytool -genkey -alias certservicios -keystore C:\servicios.keystore –storepass almacentomcatservicios -storetype BKS -provider org.bouncycastle.jce.provider.BouncyCastleProvider
```

Una vez finalizado el proceso se vuelve a comprobar que todo haya salido correctamente ejecutando:

```
keytool -list -keystore C:\servicios.keystore –storetype BKS
```

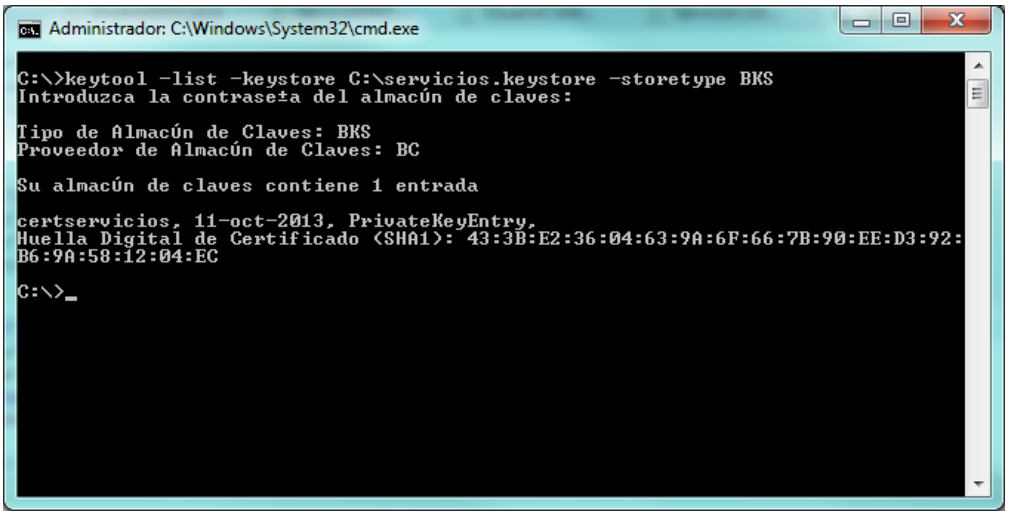


Imagen 11

Ahora se puede observar cómo el almacén de claves sí es de tipo *BKS* por lo que no habrá ningún problema al enviar el certificado a la aplicación Android.

Para revisar los certificados y los almacenes de claves de una forma más gráfica y más sencilla, se hizo uso de otra herramienta llamada *Portecle*.

3.1.7.2. Configuración de Tomcat para HTTPS

Una vez se ha creado correctamente el almacén de claves y el certificado de seguridad, el siguiente paso es configurar tomcat para permitir el tráfico HTTPS usando el almacén de claves creado.

Para ello se debe modificar el archivo *server.xml* del servidor. Usando la vista *Package Explorer* de Spring Tool Suite se puede visualizar el fichero de la siguiente forma:

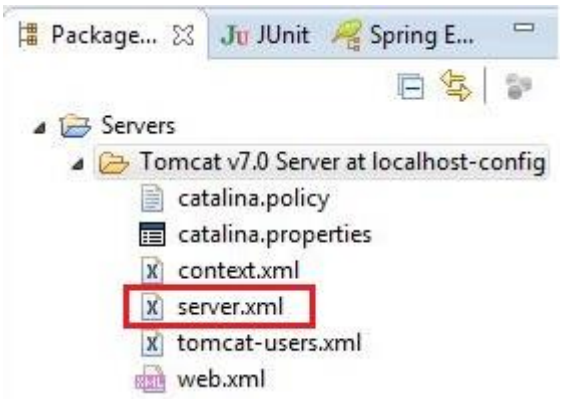


Imagen 12

Una vez abierto, hemos de se debe agregar el siguiente trozo de código:

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" keystore=
"C:\Users\Jorge\.keystore" maxThreads="150" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS" keystorePass="almacentomcatservicios" />
```

Con el código anterior se configura Tomcat para permitir el tráfico SSL a través del puerto 8443 usando el almacén de claves *"C:\Users\Jorge\.keystore"* que ha sido creado en los pasos anteriores.

3.2. Implementación de la aplicación móvil

Hasta aquí se ha implementado un servidor RESTful creado con el framework de Spring que atiende a peticiones HTTP y que devuelve objetos en formato JSON. Una vez finalizada esta parte se debe crear una vista con la que se pueda interactuar con dicho servidor para que un usuario pueda registrarse como tal en nuestra aplicación, crear servicios, listar servicios o consumir servicios.

3.2.1. Planteamiento del problema

La plataforma sobre la que los servicios se gestionan ya está creada, ahora necesitamos un medio mediante el cual un usuario pueda recibir información sobre los servicios que se están ofertando en cada momento e incluso poder consumir algún servicio si éste estuviera interesado. Por la versatilidad y la movilidad que aporta y la gran extensión de usuarios que abarca, parece lo más sensato optar por un desarrollo para alguna plataforma móvil.

Para conseguir que el sistema se haga popular necesitamos llegar a un gran volumen de usuarios. Un grave error sería realizar un desarrollo para una plataforma móvil concreta sin un estudio previo de la cuota de mercado que posee cada uno de los sistemas operativos móviles en la actualidad. El proyecto podría implementarse para una plataforma móvil minoritaria y fracasar por esa escasez de usuarios, aun habiendo hecho un buen trabajo en el desarrollo de la aplicación.

Un inconveniente insalvable es la necesidad de disponer de una conexión a internet en el dispositivo móvil. Los usuarios pueden crear, eliminar y consumir en cualquier momento los servicios por lo que es indispensable tener los datos actualizados en todo momento.

3.2.2. Soluciones propuestas

En la introducción se mostraban dos estudios realizados, uno por la agencia de noticias EuropaPress y otro por la consultora Kantar Worldpanel Comtech. En ellos se explicaba la penetración de smartphones en España, a nivel Europeo y a nivel mundial. Analizando estos estudios se puede corroborar que el sistema operativo de Google tiene una cuota de mercado superior, en general, a su directo competidor IOS, quedándose muy por debajo otras plataformas móviles como son Windows Phone o BlackBerry.

Se puede afirmar entonces que Android ofrece mayor visibilidad basándose en el volumen de usuarios, pero, de acuerdo al último estudio de Distimo[6], el 73 por ciento de los ingresos de apps es generado por la App Store de Apple, comparado con el 23 por ciento de Google Play.

[6] http://www.distimo.com/blog/2013_05_a-granular-app-level-look-at-revenues-google-play-vs-apple-app-store/

Las aplicaciones para Android también se programan bajo el lenguaje Java, mientras que el lenguaje de las aplicaciones para IOS es Objetive-C. Esto es una clara ventaja ya que no es necesario aprender otro lenguaje de programación para el desarrollo de la aplicación móvil al coincidir con el lenguaje usado en la implementación del servidor.

Tras haber analizado todos los datos comentados anteriormente, se ha decidido realizar la implementación de la aplicación para el sistema operativo de Google. Ésta hará de cliente para el servidor de datos en Spring.

Gráfico 1

En el Gráfico 1 tenemos resumida la funcionalidad de la aplicación a desarrollar. Cada función que puede realizar el usuario tiene asociada a una petición al servidor, por ejemplo, si desde la aplicación se crea un servicio, éste ha de ser enviado al servidor para que lo almacene correctamente en su base de datos y el resto de usuarios tenga acceso al mismo.

3.2.3. Esquema del proyecto en Android

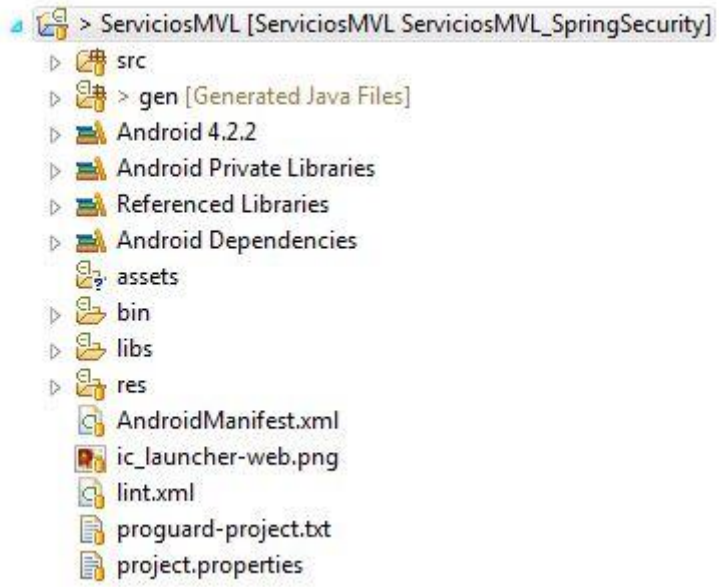


Imagen 13

La imagen anterior muestra la estructura del proyecto creado en el ADT para el desarrollo de la aplicación en Android. Se irá detallando cada uno de los directorios y archivos que componen este proyecto, comenzando desde la parte superior de la imagen:

- *src*: Es la carpeta que contiene el código fuente de la aplicación. En ella se crearán los paquetes necesarios y se incluirán los archivos *.java* que se estimen oportunos. La aplicación se ha distribuido en 5 paquetes:

    - ***org.pfc.serviciosMVL***, que contiene todas las actividades de la aplicación y la clase *ServiciosAdapter*.
    - ***org.pfc.serviciosMVL.core***, que contiene las clases análogas al modelo de datos del servidor.
    - ***org.pfc.serviciosMVL.core.connections***, cuyo contenido son las clases que hacen posible la conexión con el servidor.
    - ***org.pfc.serviciosMVL.pagos***, que contiene la interfaz *IServicioPago* y su implementación *ServicioPagoFake* para emular un pago de un servicio determinado.
    - ***org.pfc.serviciosMVL.utils***, que tiene las clases relacionadas con el cifrado de archivos y con el cliente HTTPS.

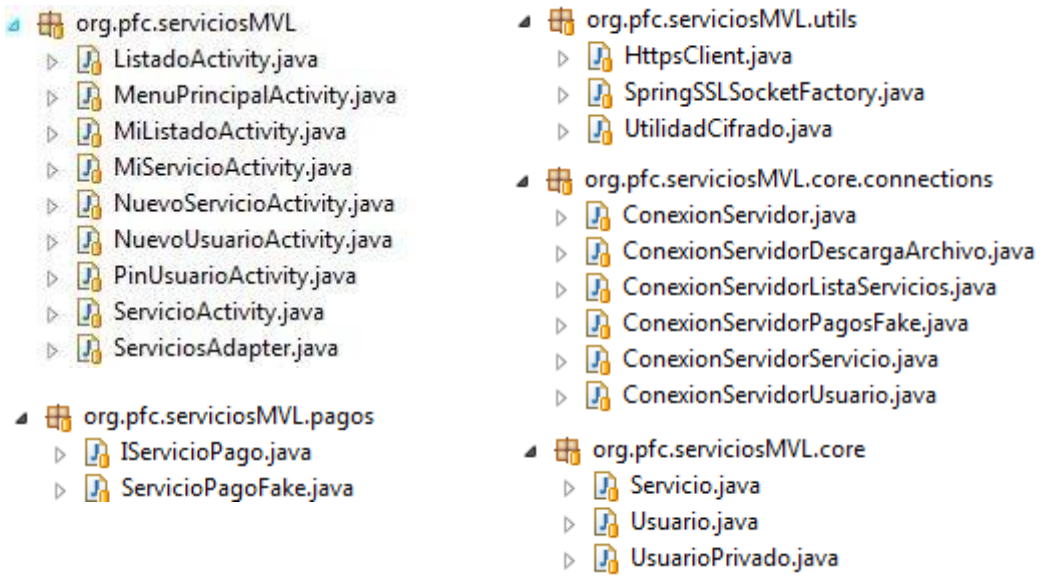


Imagen 14

- *gen*: El contenido de este directorio es autogenerado. No se recomienda la modificación de ninguno de los archivos que contiene.

- Diferentes tipos de librerías: Como se puede apreciar en Android hay definidas diferentes tipos de librerías. Entre ellas se observa la dependencia de la versión del sistema operativo para el cual está compilada la aplicación, librerías privadas del proyecto, referenciadas y las dependencias propias de proyecto que se está desarrollando.

- *assets:* Este directorio se usa en Android para almacenar archivos de tipos solo lectura (RAW). Para acceder a estos recursos se debe hacer uso de la clase *AssetManager*.

- *bin*: Aquí se encuentra el código de la aplicación compilado y en el que se genera el fichero *.apk* o instalador.

- *libs*: En esta carpeta se encuentran los archivos *.jar* de las librerías enlazadas.

- *res*: Todos los recursos usados por la aplicación que crean una entrada en el fichero *R.java* se encuentran en este directorio. *R.java* es un archivo que almacena identificadores en formato hexadecimal de los recursos incluidos en esta carpeta. Este fichero se encuentra en el directorio *gen* y facilita enormemente el acceso a los distintos recursos almacenados en *res*. En *res* se pueden encontrar los siguientes directorios:

    - *drawable*: Destinado para las imágenes. Suelen ser varias carpetas con un sufijo. Este sufijo diferencia los distintos tipos de densidades de pantalla de los diferentes dispositivos Android.

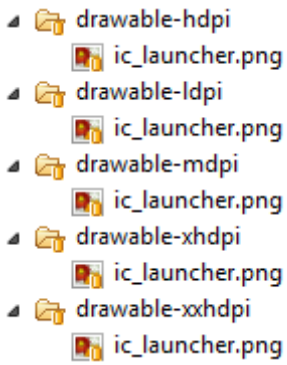


Imagen 15

En esta aplicación no hace uso de ninguna imagen aparte de *ic_laucher.png,* por eso éste es el único archivo que se observa en el interior de las carpetas *drawable*.

- *layout*: En él se encuentran los ficheros XML con las vistas de la aplicación.

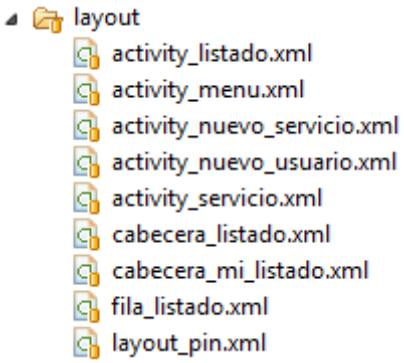


Imagen 16

Por cada actividad se ha tenido que crear un archivo XML asociado en esta carpeta. A parte de los layout de las actividades también se debe incluir en esta carpeta los archivos XML asociados a las filas de un *listView*, en esta aplicación dicho archivo es *fila_listado.xml*.

También se han definido unas cabeceras para dar algo de estilo a algunas actividades, estos archivos son *cabecera_listado.xml* y *cabecera_mi_listado.xml*.

- *menú*: También son ficheros XML que contienen los distintos menús de las actividades.

Aunque existe un menú por cada actividad en esta aplicación no se ha hecho uso de ellos aunque en otras versiones del software se podrá implementar su funcionalidad.

Imagen 17

- *values*: Archivos XML que almacenan strings, colores, o estilos. De este tipo también existen diferentes carpetas en relación al tamaño del dispositivo y versión del sistema Android. En la aplicación el único archivo modificado ha sido *strings.xml,* situado en la carpeta *values* (sin sufijo). En él se encuentran todos los strings relacionados con las vistas que se usan en la aplicación.

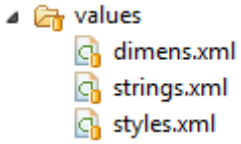


Imagen 18

Un proyecto Android puede contener otros directorios como:

- *xml*: Otros archivos XML requeridos por la aplicación.

- *raw*: Otros ficheros que no se encuentren en formato XML. En esta carpeta es donde se ha descargado el *keystoreandroid.bks*, usado para establecer las comunicaciones seguras a través HTTPS.

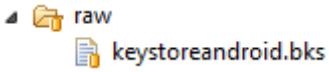


Imagen 19

- *anim*: Contiene las animaciones en XML.

- *AndroidManifest.xml*: Este fichero es el descriptor de la aplicación Android. En su interior se debe indicar a qué versión del sistema operativo está dirigida la aplicación y hasta cual soporta, las actividades que tiene, la versión de la propia aplicación, etc.

  Debido a la gran fragmentación que actualmente tiene el sistema operativo Android se ha decidido que la aplicación pueda soportarse desde la API nivel 8 que corresponde al nombre comercial *Froyo*, aunque todas las pruebas están hechas sobre la última versión llamada *Kit-Kat,* cuyo nivel de API es el nivel 19. Para especificar esta configuración en el archivo *AndroidManifest.xml* se debe añadir el siguiente código:

```
<uses-sdk android:minSdkVersion="8"
      android:targetSdkVersion="19" />
```

También se debe indicar en este archivo qué permisos necesita esta aplicación. Es muy importante especificar aquellos permisos que sean estrictamente necesarios para el correcto funcionamiento del sistema. Si un usuario al instalar la aplicación observa que los permisos a priori exceden el ámbito de la misma, puede desconfiar de los fines de este software.

En esta primera versión únicamente se han requerido dos de los permisos: el relacionado con la salida a internet (para poder establecer una comunicación con el servidor) y el que permite la escritura en la memoria interna del dispositivo (para almacenar los datos cifrados del usuario). Estos dos permisos los se especifican de la siguiente manera en el *AndroidManifest.xml*:

```
<uses-permission
      android:name="android.permission.INTERNET"/>
<uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Otra de las etiquetas de *AndroidManifest* es `<application>`. Con ella se define la aplicación. En su interior se han de indicar cada uno de sus componentes y los atributos que afecten a todos estos componentes.

```
<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
```

Con el atributo *allowBackup* se está indicando si la aplicación permite realizar copia de seguridad y restauración de los datos de la misma.

Para el icono de la aplicación se ha de indicar al atributo *icon* y proporcionarle una imagen válida en la carpeta *drawable*.

El nombre de la aplicación se define con el atributo *label* y tiene que tener un valor válido en el archivo *strings.xml*.

Y por último, el atributo *theme* especifica el tema que tendrá la aplicación. El tema se tendrá que definir en el archivo *style.xml*.

En el interior de `<application>` es donde se añaden las distintas actividades que componen la aplicación. Por cada actividad ha de haber una entrada `<activity>` en el interior de `<application>`. A continuación se pueden dos ejemplos:

```
<activity
    android:name="org.pfc.serviciosMVL.MenuPrincipalActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

Esta es la actividad principal de la aplicación. El atributo *name* se ha especificado el nombre de la clase que implementa esa actividad y el atributo *label* el nombre de la actividad en sí.

Después se puede ver la etiqueta `<intent-filter>`, con dos hijos `<action>` y `<category>`. En este caso es necesario incluirlos ya que, como se ha comentado previamente, esta es la definición de la clase principal. En el archivo *AndroidManifest.xml* sólo puede haber una actividad principal y ésta actividad debe tener las etiquetas `<action>` y `<category>` bien definidas.

```
<activity
     android:name="org.pfc.serviciosMVL.ListadoActivity"
     android:label="@string/title_activity_listado"
     android:parentActivityName=
           "org.pfc.serviciosMVL.MenuPrincipalActivity" >
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="org.pfc.serviciosMVL.MenuPrincipal" />
</activity>
```

En la definición de esta segunda actividad se puede ver que también posee los atributos *name* y *label*, pero no tiene añadidas las etiquetas `<intent-filter>`, `<action>` y `<category>`. Como ya se ha comentado sólo puede haber una actividad principal en la aplicación.

Sin embargo, al ver el código de la definición de la actividad anterior se puede apreciar que posee un atributo *parentActivityName* y una etiqueta `<meta-data>` con otros dos atributos, *name* y *value*, que no tenía la actividad anterior. Las actividades se crean a partir de otras, excepto la actividad principal, por este motivo se ha de indicar el padre de cada una de las actividades, permitiendo así la navegación jerárquica entre las mismas. Por lo tanto se añade el atributo *parentActivityName* y la etiqueta `<meta-data>` con los dos atributos, *name* y *value* para indicar qué clase será el padre que creará esta actividad.

- *ic_launcher_web.png*: Este archivo es un icono de gran tamaño para ser usado en páginas web.

- *lint.xml*: Este archivo es usado por Android Lint, un analizador de código estático para la detección de fallos, mejorar el rendimiento de la aplicación, etc.

- *proguard_project*: Fichero de configuración de la herramienta *ProGuard*, que permite optimizar y ofuscar el código generado. Con él se obtiene un *.apk* más pequeño y donde resulta más difícil hacer ingeniería inversa.

- *project.properties*: Este archivo es generado por el SDK. No es conveniente modificarlo.

3.2.4. Clases principales

Para no sobrecargar el documento con fragmentos de código y clases irrelevantes se seleccionarán y se analizarán con más detenimiento aquellas que consideren más ilustrativas para la comprensión del proyecto.

3.2.4.1. Clases del modelo

Las tres clases, cuyo diagrama se puede ver a continuación, son las equivalentes al modelo de datos implementado en la parte del servidor.

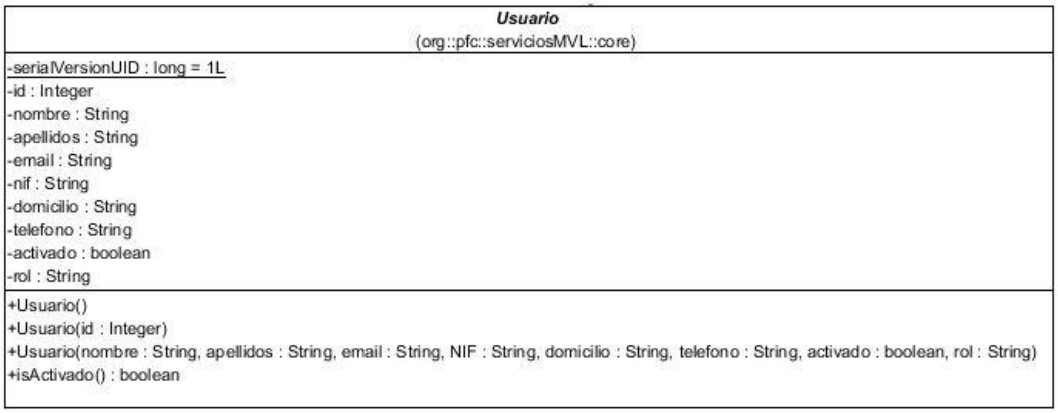


Diagrama de clase 8

Diagrama de clase 9

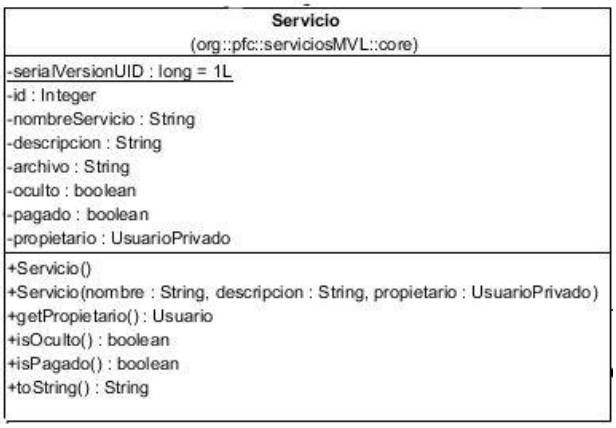


Diagrama de clase 10

Estas clases permiten trabajar con los datos que se encuentran almacenados en el servidor en la misma memoria del dispositivo, habiendo realizado la petición correspondiente a dicho servidor a través de internet. Revisando los diagramas de clase anteriores se puede ver que las clases no necesitan ser comentadas con más detalle puesto que sólo contienen atributos y los métodos necesarios obtener y actualizar los valores de estos atributos (getters y setters).

3.2.4.2. NuevoUsuarioActivity

La primera actividad que aparece al iniciar por primera vez la aplicación es la de *NuevoUsuarioActivity*. Esta clase es la encargada de obtener los datos de los usuarios y enviarlos al servidor para que se almacenen en la base de datos. Como son datos sensibles hay que analizar exhaustivamente los puntos de la LOPD referentes a la recogida de información personal.

Los datos que se recogen de los usuarios para el registro son los siguientes:

- Nombre.
- Apellidos.
- NIF.
- Domicilio.
- Email.
- Teléfono.
- Declaración jurada.
- Clave de desbloqueo.

Los datos de carácter personal que el usuario aporta a la aplicación no son excesivos para el ámbito de la misma. Es necesario poder identificar a una persona física para intentar evitar fraudes y usuarios falsos en la medida de lo posible. Con esto estamos respetando el **Artículo 4 de la LOPD**.

En esta misma actividad, al mismo nivel del botón “Registrarse” se ha colocado otro llamado “Aviso legal”. Al pulsar sobre éste último, al usuario se le despliega un diálogo para notificarle la existencia de un fichero al que se añadirán sus datos personales, del fin de la recogida de los mismos y del destinatario de la información. Si el usuario tras leer esto finalizase su registro se entenderá su consentimiento para almacenar sus datos de carácter personal. También se le comunica la vía para ejercer su derecho de acceso, modificación o baja de estos datos. Cumpliendo así los **Artículos 5, 6 y 15 de la LOPD**. A continuación se pueden ver las dos pantallas de la aplicación en las que se puede observar lo comentado anteriormente:

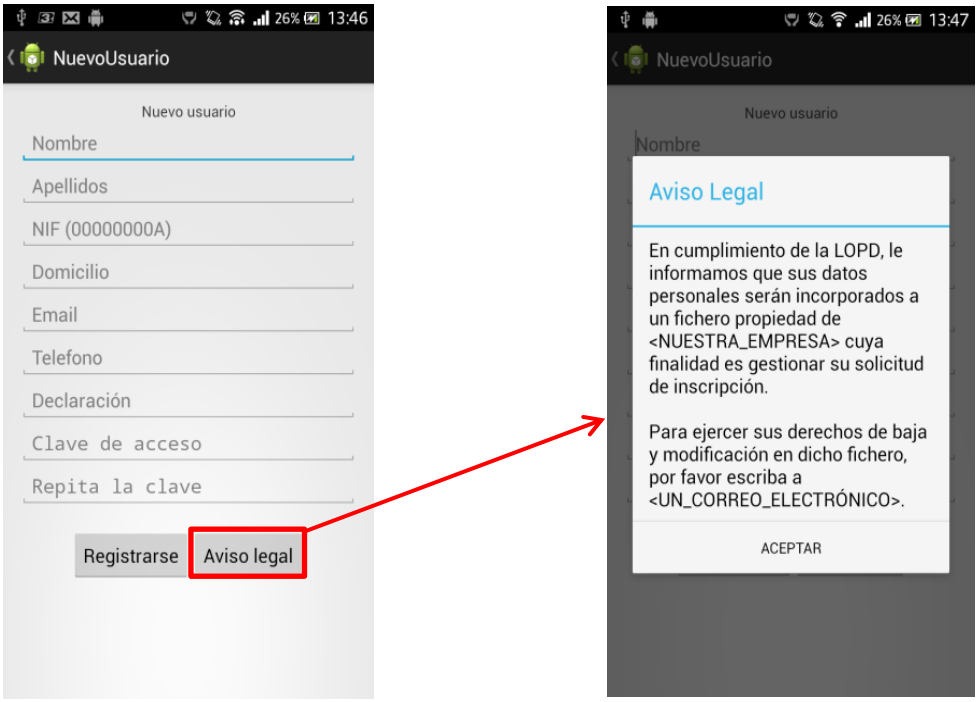


Imagen 20

Puesto que en la aplicación no se recogen datos especialmente protegidos, datos relacionados con la salud de los usuarios y no se transmitirán ni tendrán acceso terceras partes, **no se aplicarán los Artículos 7, 8, 11 y 12** respectivamente.

3.2.4.3. MenuPrincipalActivity (ActividadPrincipal)

Esta es la actividad desde la cual se lanzan el resto de actividades de la aplicación desarrollada por lo que es conveniente analizarla detenidamente. A continuación se puede ver su diagrama de clase en el que se aprecian los distintos atributos y métodos de esta actividad:

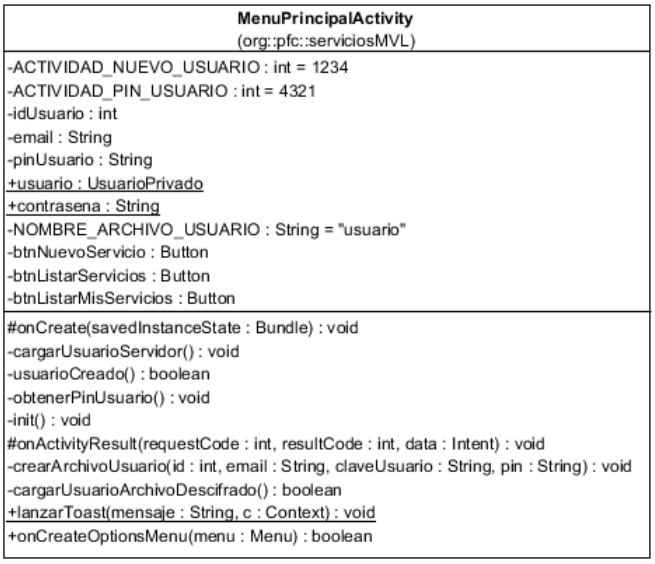


Diagrama de clase 11

Al iniciarse la aplicación, esta es la primera actividad que se crea. Vemos el método *onCreate()* de la misma:

```
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	if (!usuarioCreado()) {
		Intent i = new Intent(this, NuevoUsuarioActivity.class);
		startActivityForResult(i, ACTIVIDAD_NUEVO_USUARIO);
	} else {
		obtenerPinUsuario();
	}
	setContentView(R.layout.activity_menu);
	init();
}
```

El método anterior es el encargado de crear la actividad *MenuPrincipalActivity* pero, antes de iniciarse, verifica si existe un usuario creado en el móvil llamando al método *usuarioCreado()*. Esta verificación se lleva a cabo comprobando si existe un fichero que contiene los datos cifrados de un usuario que se haya conectado previamente a la aplicación. Si existe este fichero se llama al método *obtenerPinUsuario()* que lanza la actividad *PinUsuarioActivity*.

PinUsuarioActivity permitirá al usuario introducir un pin que descifrará el archivo, obtendrá los datos del usuario y los verificará enviándolos al servidor. Justo después la actividad principal entonces será creada correctamente. Pero, si por el contrario el archivo no existe, se lanzará *NuevoUsuarioActivity*.

NuevoUsuarioActivity solicita los datos del usuario para darse de alta en el sistema y envía esos datos al servidor, éste los almacenará en la base de datos. Si todo ha funcionado correctamente la actividad principal se creará.

MenuPrincipalActivity consta de tres botones “Nuevo Servicio”, “Listar Servicios” y “ListarMisServicios”. Estos tres botones lanzan cada uno una actividad distinta pero de forma diferente.

- “Nuevo Servicio”: Lanza la actividad *NuevoServicioActivity* para crear un servicio nuevo. El usuario introduce los datos del servicio en esta actividad que los recoge y los envía al servidor.

- “Listar Servicios”: Este botón lo primero que hace es lanzar una petición al servidor de todos los servicios. Si no ocurre ningún fallo en la red se creará la actividad *ListadoActivity* con todos los servicios listados.

- “Listar Mis Servicios”: Su comportamiento es muy similar al botón anterior. Realiza una petición al servidor para obtener todos los servicios propios del usuario que está registrado en la aplicación. Después creará la actividad *MiListadoActivity* que mostrará los servicios creados por el usuario en una lista.

MenuPrincipalActivity posee dos métodos importantes, uno se encarga de crear el archivo cifrado con los datos del usuario en el terminal (*crearArchivoUsuario()*) y el otro se usa para descifrar este archivo y cargar los datos del mismo cuando el usuario introduce su pin al arrancar la aplicación (*cargarUsuarioArchivoDescifrado()*). A continuación se muestra el primero de ellos:

crearArchivoUsuario()

```
private void crearArchivoUsuario(int id, String email, String
      claveUsuario, String pin) {
      try {
            FileOutputStream fos = this.openFileOutput(
                  NOMBRE_ARCHIVO_USUARIO, MODE_PRIVATE);
            byte[] claveGenerada = UtilidadCifrado.generarClave(pin);
            byte[] contenidoArchivoCifrado =
            UtilidadCifrado.cifrarArchivo(claveGenerada, (id + "/" +
            email + "/" + claveUsuario).getBytes("UTF-8"));

            fos.write(contenidoArchivoCifrado);
            fos.close();
      } catch (Exception e) {
            e.printStackTrace();
      }
}
```

Lo primero se crea en este método es un `FileOutputStream` en el que se escribirán los datos cifrados del usuario. Se genera una clave partiendo del pin del usuario y haciendo uso de la clase *UtilidadCifrado,* clase estática que facilita las labores relacionadas con el cifrado y el descifrado de archivos. A continuación se concatenan los datos del usuario, se cifra todo con la clave generada y se almacena toda esta información en el archivo que se abrió al inicio del método.

cargarUsuarioArchivoDescifrado()

```
private boolean cargarUsuarioArchivoDescifrado() {
	try {
		FileInputStream fis = this.openFileInput(
			NOMBRE_ARCHIVO_USUARIO);
		byte[] contenidoCifrado = new byte[fis.available()];
		fis.read(contenidoCifrado);
		byte[] claveGenerada = UtilidadCifrado.generarClave(
			pinUsuario);
		byte[] contenidoDescifrado = UtilidadCifrado.
			descrifrarArchivo(claveGenerada, contenidoCifrado);
		String contenido = new String(
			contenidoDescifrado, "UTF-8");
		String[] idEmailContrasena = contenido.split("/");
		idUsuario = Integer.valueOf(idEmailContrasena[0]);
		email = idEmailContrasena[1];
		contrasena = idEmailContrasena[2];
		usuario = new UsuarioPrivado(idUsuario,
			email, contrasena);
		fis.close();
		return true;
	} catch (Exception e) {
		MenuPrincipalActivity.lanzarToast("Puede que el pin sea
			incorrecto, introduzcalo de nuevo",  this);
		return false;
	}
}
```

Este método es llamado cuando un usuario inicia la aplicación e introduce su pin privado. En ese momento se abre el fichero y se lee su contenido. A continuación, se genera la clave con el pin introducido por el usuario y se intenta descifrar el contenido del archivo con dicha clave. Finalmente si todo ha ido de forma correcta se actualizan los datos del usuario en la aplicación, se contrastarán los datos con el servidor y se creará la actividad principal. Por el contrario, si la operación de descifrado ha fallado, se mostrará un mensaje al usuario indicando este error y se cerrará la aplicación.

3.2.4.4. ListadoActivity

ListadoActivity es lanzada desde *ConexiónServidorListaServicios*, tras haber pulsado el botón “Listar servicios” del menú principal y haber obtenido una respuesta del servidor con el listado de los servicios disponibles. *ConexiónServidorListaServicios* es la clase encargada de realizar la petición de los servicios al servidor, devolviendo este listado a la actividad *ListadoActivity* que es la encargada de mostrarlos.

Para poder listar los servicios correctamente en Android se ha creado una clase llamada *ServiciosAdapter*. Esta clase hereda de *ArrayAdapter* y se ocupa de recibir un array de servicios e ir añadiéndolos a un *ListView* creando una vista por cada servicio. Esta vista la se ha definido en un archivo llamado *fila_listado.xml* y se encuentra dentro del directorio *res/layout*.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_listado);

   Intent intent = getIntent();
   Bundle bundle = intent.getExtras();
   ArrayList<Servicio> servicios = (ArrayList<Servicio>)
       bundle.getSerializable("arrayServicios");

   adaptador = new ServiciosAdapter(this,R.layout.fila_listado,
       servicios);
   listadoServicios = (ListView)findViewById(R.id.listaServicios);
   listadoServicios.setAdapter(adaptador);

   listadoServicios.setOnItemClickListener(new OnItemClickListener() {
   @Override
   public void onItemClick(AdapterView<?> av, View v, int position,
       long id) {
       Servicio servicio = (Servicio)av.getItemAtPosition(position);
       if(servicio == null) return;
       Intent i = new Intent(ListadoActivity.this,
              ServicioActivity.class);
       i.putExtra("posicion", position);
       i.putExtra("servicio", servicio);
       startActivityForResult(i, servicio.getId());
       }
    });

    setupActionBar();
}
```

En el método anterior se han eliminado algunos detalles relacionados con la implementación, irrelevantes la comprensión de la funcionalidad del método.

Dejando a un lado las dos instrucciones primeras que permiten la creación de la vista, las tres instrucciones siguientes se encargan de obtener desde el *intent* los servicios devueltos por la clase *ConexiónServidorListaServicios* y crear el array `servicios`.

Justo después se añade el *adapter* que comentado anteriormente y se crea un *listener* por cada elemento fila del *ListView*. Se puede apreciar cómo si se llegara a pulsar en cualquier elemento de la lista, se creará una nueva actividad *ServicioActivity* pasándole, en el mismo *intent,* el servicio que ha pulsado y la posición en la que se encuentra. Con ello la nueva actividad podrá mostrar los detalles de un servicio concreto.

ServicioActivity es una actividad de la cual se espera un resultado, por ello se ha de sobrescribir el método *onActivityResult* tal como se puede ver en el siguiente fragmento de código:

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
      Intent data){
Servicio servicio = null;
if(data==null) return;
if((servicio = (Servicio) data.getExtras().getSerializable(
      "servicio"))!=null)
    if (requestCode==servicio.getId() && resultCode==RESULT_OK &&
      servicio.isPagado()){
            adaptador.servicios.remove(data.getExtras().getInt(
                  "posicion")-1);
            adaptador.notifyDataSetChanged();
      }
}
```

Con este método se elimina un servicio de la lista de servicios si éste ha sido consumido por el usuario.

Otra actividad que no se analizará en profundidad es *MiListadoActivity* puesto que es similar a *ListadoActivity*, la única diferencia radica en los servicios que aparecen en el listado. Mientras que *ListadoActivity* muestra todos los servicios que pueden ser consumidos por los usuarios, *MiListadoActivity* muestra un listado de servicios que han sido creados por el usuario registrado en la aplicación.

3.2.4.5. ServicioActivity

Mediante esta actividad un usuario puede consumir un servicio seleccionado previamente de *ListadoActivity*. Contiene como atributo una clase llamada *ServicioPagoFake* que realiza las funciones de un servicio de pago falso. Con *ServicioPagoFake* se puede emular una compra y obtener un resultado de la misma.

Esta actividad también contiene dos botones “Comprar” y “Descargar”. El método *onClick()* de “Comprar” es el siguiente:

```
public void onClick(View v) {
      if(notificarServidorCompra()){
            realizarCompra();
            notificarServidorCompraRealizada(confirmarCompra());
      }else{
            MenuPrincipalActivity.lanzarToast("Compra no realizada con
                  éxito, no se ha podido notificar al servidor",
                  ServicioActivity.this);
      }
}
```

El primer método llamado es *notificarServidorCompra()* que se encarga de comunicar al servidor que se quiere realizar una compra de un servicio determinado. Lo realiza mediante una clase que también hereda de *ConexionServidor* llamada *ConexionServidorPagosFake*.

realizarCompra() es el segundo método que se puede ver en el código. Se encarga de emular la compra mediante la clase *ServicioPagoFake* comentada con anterioridad.

Después se llama al método *confirmarCompra()*, encargado de comprobar la respuesta del servicio de pago emulado, y su resultado es pasado directamente al método *notificarServidorCompraRealizada()*. Este último se encarga de notificar al servidor del resultado de la compra emulada a través de la clase *ConexionServidorPagosFake*.

El otro botón que posee la clase, "Descargar", sólo estará visible si el servicio lleva adjunto algún fichero. Se podrá pulsar sobre él cuando un usuario compre ese servicio, mientras tanto el botón aparece, pero desactivado. Se puede ver a continuación el fragmento del método *confirmarCompra()* que activa este botón:

```
if(((ServicioPagoFake)servicioPago).getResultadoCompra()
        ==ResultadoCompra.OK){
        if(btnDescargarArchivo!=null)
                btnDescargarArchivo.setEnabled(true);
```

Este botón lleva asociado el evento siguiente:

```
@Override
public void onClick(View v) {
        ConexionServidorDescargaArchivo descargaArchivo =
                New ConexionServidorDescargaArchivo(
                (Activity)ServicioActivity.this);
        descargaArchivo.execute(servicio.getArchivo());
}
```

Este fragmento de código crea una instancia de la clase *ConexiónServidorDescargaArchivo* y llama al método *execute()*. Con esta clase, que hereda también de *ConexionServidor*, se encarga de realizar la llamada al servidor para obtener el archivo asociado al servicio en cuestión.

Esta clase tiene otro método a destacar y es el siguiente:

```
@Override
public void onBackPressed() {
        Intent i = new Intent();
        i.putExtra("servicio", servicio);
        i.putExtra("posicion", posicion);
        setResult(Activity.RESULT_OK, i);
        super.onBackPressed();
}
```

Es llamado cuando desde la actividad se pulsa el botón "atrás" del dispositivo. Se le pasa a la actividad que llamó a *ServicioActivity* el resultado de la misma, por si la actividad padre (*ListadoActivity*) ha de eliminar el servicio de su lista o no realizar modificación alguna.

La actividad *MiServicioActivity* se detallará puesto que es análoga a *ServicioActivity,* salvo por el tratamiento de los servicios. Como *MiServicioActivity* es llamada desde *MiListadoActivity* sólo podrán aparecer servicios que sean propiedad del usuario registrado en la aplicación. En ella podrán aparecer servicios que hayan sido consumidos por otros usuarios o todavía disponibles para la comunidad. El usuario registrado podrá únicamente eliminar el servicio si así lo cree conveniente.

3.2.4.6. ConexionServidor

Esta clase es la clase padre de todas aquellas que realizan alguna conexión con el servidor de datos. Como vemos en la imagen de abajo *ConexiónServidorPagosFake, ConexiónServidorListaServicios, ConexiónServidorServicio, ConexiónServidorUsuario* y *ConexiónServidorDescargaArchivo* heredan de ésta.

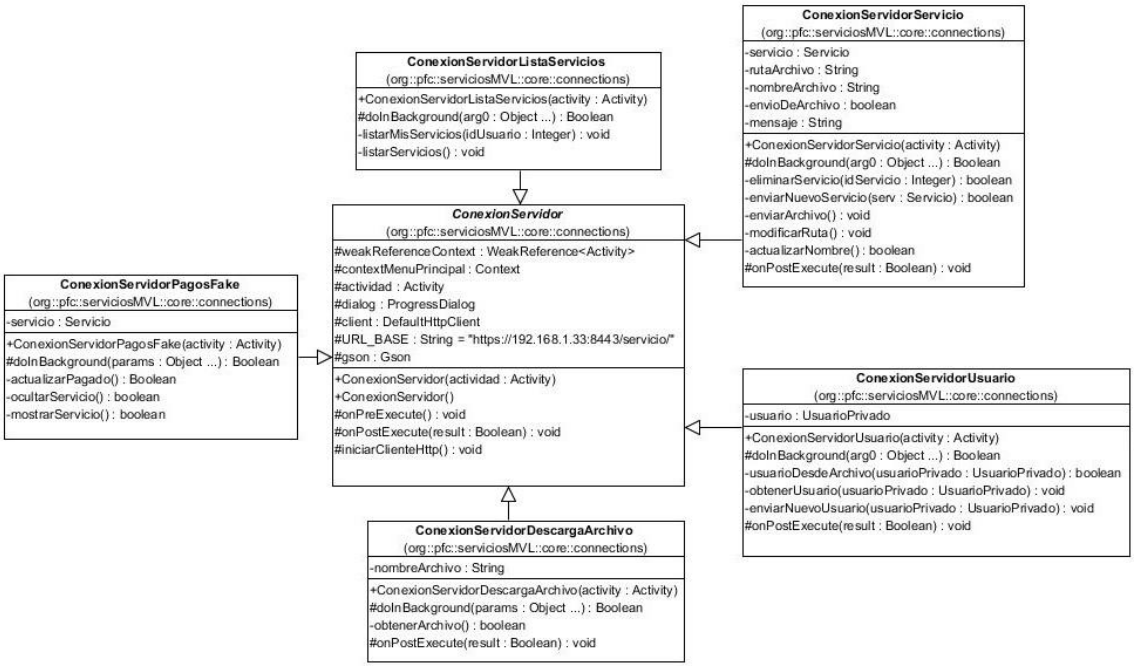


Diagrama de clase 12

ConexionServidor hereda de *AsyncTask* que es una clase que permite realizar operaciones costosas (por ejemplo, descargar un archivo) en segundo plano, librando el hilo principal de bloqueos indebidos. *AsyncTask* también nos permite comunicar al hilo de la interfaz de usuario el estado de la tarea, por ejemplo con una barra de descarga.

Al heredar de *AsyncTask* se deben sobrescribir algunos de sus métodos, el único obligatorio es *doInBackground()* aunque se recomienda sobrescribirlos todos para mantener el control de la tarea que se está ejecutando en segundo plano. Estos son los métodos disponibles para ser sobrescritos:

- *onPreExecute()*: Se ejecuta antes del método *doInBackground()*. Se suele utilizar para preparar la ejecución de la tarea, inicializar algún dialog o barra de descarga, etc.

- *doInBackground()*: Este método contiene el código principal de la tarea. En él se ejecutará todo aquello que implique una sobrecarga al hilo de la interfaz gráfica y pueda quedar bloqueada. Desde este método no se puede acceder al hilo de la interfaz de usuario.

- *onProgressUpdate()*: Se ejecuta cada vez que llamemos al método *publishProgress()* desde el método *doInBackground()*. Es usado para informar a la interfaz de usuario del estado actual de la tarea.

- *onPostExecute()*: Se ejecuta una vez finalizada la tarea implementada en el método *doInBackground()*.

- *onCancelled()*: Si se cancela la tarea y finaliza antes de lo normal se ejecuta este método.

A continuación se mostrarán los atributos más importantes de la clase *ConexiónServidor*:

- `ProgressDialog dialog`: Es una ventana que mostrará información sobre las distintas tareas que se hagan en segundo plano.

- `DefaultHttpClient client`: Cliente HTTP que se usará para la comunicación con el servidor REST. Se hablará más detenidamente sobre la configuración de este cliente.

- `String URL_BASE = "https://192.168.1.33:8443/servicio/"`: Es el string base de las distintas URLs aceptadas por el servidor. A este string base se le concatenarán otras cadenas de caracteres que completen una dirección válida. En este caso se puede ver cómo la IP corresponde con la de una red privada, este valor sufrirá modificaciones cuando la aplicación se lleve a producción.

- `Gson gson`: Objeto de la librería *google-gson* que permite convertir objetos de tipo Java a su representación en JSON y viceversa, para ser transmitidos a través de internet.

- `Context contextMenuPrincipal`: Contexto desde el cual se llama a la tarea asíncrona.

- `Activity actividad`: Actividad desde la cual se llama a la tarea asíncrona.

A continuación se analizarán los métodos que tiene *ConexionServidor,* comenzando por *onPreExecute()*:

```
@Override
 protected void onPreExecute() {
          dialog = new ProgressDialog(contextMenuPrincipal);
          dialog.setMessage("Cargando...");
          dialog.setCanceledOnTouchOutside(false);
          dialog.show();
          iniciarClienteHttp();
    }
```

Lo más importante de este método es la llamada a *iniciarClienteHttp()* puesto que el código anterior sólo crea un diálogo en el hilo de la interfaz de usuario con el mensaje "Cargando…". *iniciarClienteHttp()* contiene la siguiente línea de código `client = (DefaultHttpClient) HttpsClient.getNewHttpClient(contextMenuPrincipal);` que como se puede apreciar hace uso de un método estático de la clase *HttpsClient* que se verá más adelante. La función de este método es devolver un cliente HTTP configurado para la comunicación HTTPS usando el certificado creado por el servidor para cifrar la conexión.

Otro de los métodos sobrescritos es *onPostExecute()*:

```
@Override
protected void onPostExecute(Boolean result) {
       super.onPostExecute(result);
       if(dialog!=null) dialog.cancel();
}
```

La única misión de este método es cancelar el diálogo si ha sido creado previamente en la interfaz de usuario una vez haya finalizado la tarea.

Se verá a continuación una de las clases hijas más completas de *ConexionServidor* para estudiar la comunicación a través de la red con el servidor.

3.2.4.7. ConexionServidorServicio

Esta es la clase encargada del envío o el borrado de un servicio en el servidor. Como se ha visto, hereda de *ConexionServidor* y tiene tres métodos principales: *eliminarServicio(), enviarNuevoServicio()* y *enviarArchivo()*.

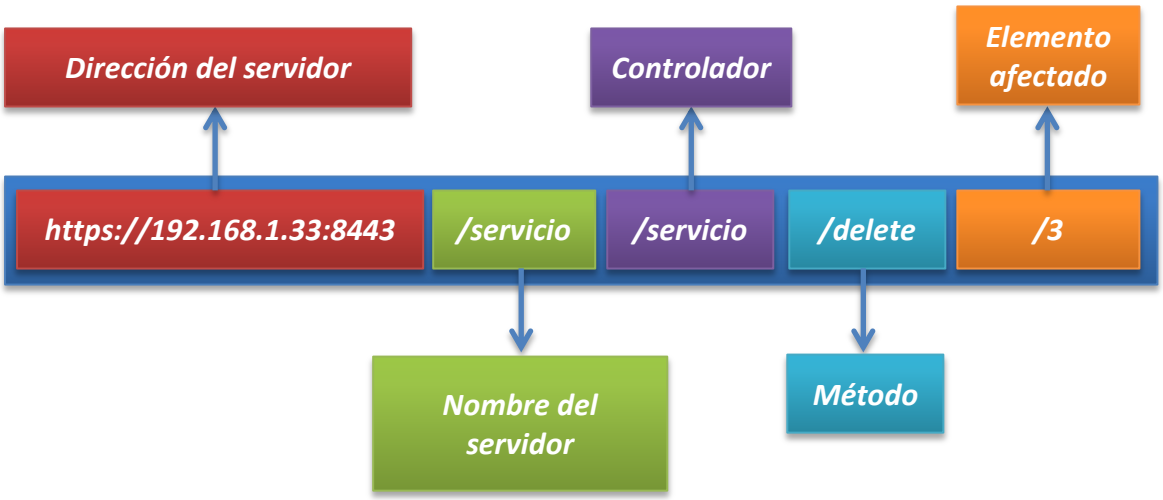
Se puede apreciar en el código que *eliminarServicio()* y *enviarNuevoServicio()* son métodos muy similares entre sí. En una posible mejora del código, se estudiará la posibilidad de crear un método genérico que, por medio de unos atributos obtenidos como parámetros, se consiga realizar una u otra tarea pero en con el mismo bloque de código.

Comenzaremos viendo el método *eliminarServicio()*:

eliminarServicio()

```
private boolean eliminarServicio(Integer idServicio) {
	String url = URL_BASE + "servicio/delete/"+idServicio;

	HttpDelete delete = new HttpDelete(url);
	String encoding = Base64.encodeToString(
		(MenuPrincipalActivity.usuario.getEmail() + ":" +
		MenuPrincipalActivity.usuario.getClaveAcceso()).getBytes()
		, Base64.NO_WRAP);
	delete.setHeader("Authorization", "Basic " + encoding);

	try {
		HttpResponse response = client.execute(delete);
		boolean eliminado = gson.fromJson(EntityUtils.toString(
		response.getEntity()), Boolean.class);
		if (eliminado) {
			mensaje = "Servicio eliminado";
			return true;
		}
		mensaje = "No se ha podido eliminar el servicio";
		return false;
	} catch (Exception e) {
		e.printStackTrace();
		return false;
	}
```

La primera acción es crear la URL completa que será la dirección a la cual le enviaremos la petición de borrado del servicio. Se usa el atributo `URL_BASE` heredado de la clase *ConexionServidor* y se le añade `"servicio/delete/"+idServicio`, donde `idServicio` es un número que identifica de forma única a un servicio en la base de datos. Con esto la URL quedaría ***https://192.168.1.33:8443/servicio/servicio/delete/3*** (se ha colocado el número 3 como ejemplo, refleja el identificador de un servicio):



Se define el método HTTP usado con *HttpDelete* pasándole como parámetro la URL creada. A continuación, se crea la cabecera con los credenciales del usuario para que logue identificarse en el servidor y le autorice la ejecución del método correspondiente

a la petición realizada en el servidor. Seguidamente se le ordena al cliente HTTP que ejecute la petición y se debe obtener un booleano como respuesta.

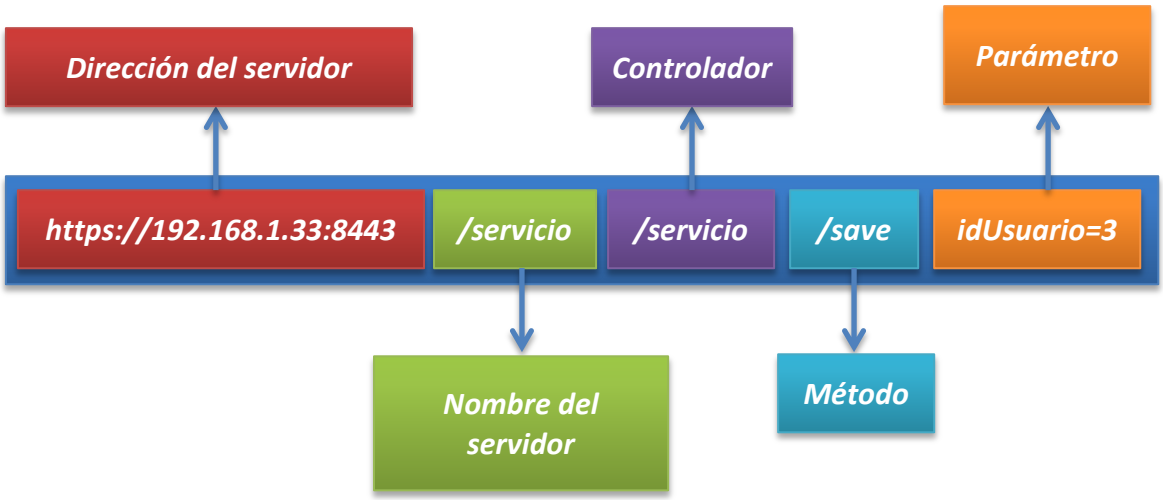
Se puede ver en el código un atributo (`mensaje`) que contiene un string. Este mensaje sirve para informar al usuario de la aplicación lo que ocurre en la tarea que se ejecuta en segundo plano pero, como no se permite acceder al hilo de la interfaz de usuario desde el método *doInBackground()*, el mensaje se mostrará desde *onPostExecute()*.

enviarNuevoServicio()

```
private boolean enviarNuevoServicio(Servicio serv) {
	servicio = serv;
	if(actualizarNombre())servicio.setArchivo(nombreArchivo);
	Usuario usuario = servicio.getPropietario();
	if(usuario!=null && MenuPrincipalActivity.usuario.getId() ==
		usuario.getId()){
		String url = URL_BASE + "servicio/save?idUsuario=" +
			usuario.getId();
		String encoding = Base64.encodeToString(
			(MenuPrincipalActivity.usuario.getEmail() + ":" +
			MenuPrincipalActivity.usuario.getClaveAcceso()).
			getBytes(), Base64.NO_WRAP);
		HttpPost post = new HttpPost(url);
		post.setHeader("Authorization", "Basic " + encoding);
		try {
			StringEntity stringEntity = new StringEntity(
				gson.toJson(servicio), HTTP.UTF_8);
			stringEntity.setContentType("application/json");
			post.setEntity(stringEntity);
			HttpResponse response = client.execute(post);
			servicio = gson.fromJson(EntityUtils.toString(
				response.getEntity()), Servicio.class);
			if(servicio!=null && envioDeArchivo)
				enviarArchivo();
			if(servicio==null) {
				mensaje = "No se ha podido crear el
					servicio";
				return false;
			}
		mensaje = "Servicio creado";
		return true;
		} catch (Exception e1) {
			e1.printStackTrace();
			servicio=null;
			return false;
		}
	} else{
		servicio=null;
		return false;
	}
}
```

Analizando el código se puede observar la similitud entre este método y el anterior, salvo por las comprobaciones, llamadas a métodos iniciales, URL a la que va dirigida la petición y método usado en la petición (*HttpPost*). En este caso se pretende

almacenar un servicio en el servidor, por ello se realiza la petición mediante el método *post* y a la URL ***https://192.168.1.33:8443/servicio/servicio/save/idUsuario=3*** (se ha indicado 3 como ejemplo).



Con esta URL se está indicando que el dueño del servicio será el usuario con identificador en la base de datos igual a tres.

Al igual que el método anterior, se crea la cabecera con los credenciales del usuario y, en este caso, para la petición se usará *HttpPost*. Se añade el servicio que se desea almacenar en formato JSON gracias a la librería *google-gson*. Seguidamente se ejecuta la petición al servidor y, si el servicio fuera acompañado de un archivo asociado, se llamaría al siguiente método:

enviarArchivo()

```
private void enviarArchivo() {
	String url = URL_BASE + "servicio/saveFile?nombreArchivo=" +
		nombreArchivo;
	modificarRuta();
	File file = new File(Environment.getExternalStorageDirectory(),
		rutaArchivo);
	try {
	    HttpPost httppost = new HttpPost(url);
	    MultipartEntityBuilder mp = MultipartEntityBuilder.create();
	    mp.addPart("archivo", new FileBody(file));
	    httppost.setEntity(mp.build());
	    HttpResponse response = client.execute(httppost);
	} catch (Exception e) {
	    e.printStackTrace();
	}
}
```

Se encarga de enviar un archivo al servidor a través de una petición *post*. La creación de la petición se realiza del mismo modo, a través de *HttpPost,* con la diferencia de añadirle el archivo en el cuerpo de la petición. Esto lo se consigue gracias

al método *addPart()* de *MultipartEntityBuilder* y el método *setEntity()* de la clase *HttpPost*.

La petición para almacenar el archivo en el servidor se realiza mediante la URL ***https://192.168.1.33:8443/servicio/servicio/saveFile?nombreArchivo=canción.mp3***. *canción.mp3* es un ejemplo de nombre de archivo que se pasa como parámetro en la misma URL.

3.2.4.8. HttpsClient

Anteriormente se ha comentado que en *ConexiónServidor* se configuraba un cliente HTTP para cifrar la conexión a través de HTTPS usando el certificado creado para el servidor. Este cliente se obtiene al llamar a *getNewHttpClient()* esta clase:

```
public class HttpsClient {

	private final static String CLAVE_KEYSTORE =
		"almacenandroidservicios";

	public static HttpClient getNewHttpClient(Context
		contextMenuPrincipal) {
		try {
			KeyStore trustStore = KeyStore.getInstance("BKS");
			if (contextMenuPrincipal == null)
				trustStore.load(null, null);
			else {
				InputStream in = (InputStream)contextMenuPrincipal.
					getResources().openRawResource(
					R.raw.keystoreandroid);
				trustStore.load(in, CLAVE_KEYSTORE.toCharArray());
				in.close();
			}
			SSLSocketFactory sf = new SpringSSLSocketFactory(
				trustStore);
			sf.setHostnameVerifier((X509HostnameVerifier)
				SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
			HttpParams params = new BasicHttpParams();
			HttpProtocolParams.setVersion(params,
				HttpVersion.HTTP_1_1);
			HttpProtocolParams.setContentCharset(params, HTTP.UTF_8);
			SchemeRegistry registry = new SchemeRegistry();
			registry.register(new Scheme("http", PlainSocketFactory
					.getSocketFactory(), 8080));
			registry.register(new Scheme("https", sf, 8443));
			ClientConnectionManager ccm = new
				ThreadSafeClientConnManager(params, registry);
			return new DefaultHttpClient(ccm, params);
		} catch (Exception e) {
			return new DefaultHttpClient();
		}
	}
}
```

El fragmento más relevante del código anterior es el siguiente:

```
InputStream in = (InputStream)contextMenuPrincipal.
      getResources().openRawResource(R.raw.keystoreandroid);
trustStore.load(in, CLAVE_KEYSTORE.toCharArray());
in.close();
```

Así, se abre el almacén de claves situado en *res/raw* usando la contraseña almacenada en la constante *CLAVE_KEYSTORE* y lo se carga en el *trustStore*.

El truststore se utilizará para negociar una conexión segura entre un cliente y el servidor a través de HTTPS. Para ello es necesario establecer una relación de confianza entre tal cliente y el servidor que se crea teniendo descargado el certificado con la clave pública del servidor en la aplicación y añadiéndose al truststore.

Test de conexión HTTPS

Para comprobar si conexión se cifra correctamente se han usado dos herramientas para la captura de datos de la red y para el estudio de los paquetes que viajan por ella, *Wireshark* y *RawCap*. Con *RawCap* se capturaron los paquetes de las primeras conexiones ya que se éstas se realizaban sobre localhost, el emulador del dispositivo móvil Android (AVD) y el servidor se encontraban en el mismo PC. *Wireshark* no era capaz de capturar los datos que viajan desde localhost a otro punto situado también en localhost, por este motivo se hizo uso de la herramienta *RawCap*.

Para mostrar los datos capturados de forma gráfica se hacía uso de *Wireshark* puesto que *RawCap* no dispone de interfaz gráfica, su uso se restringe a línea de comandos. En la Imagen 21 se puede ver una captura del uso de RawCap:

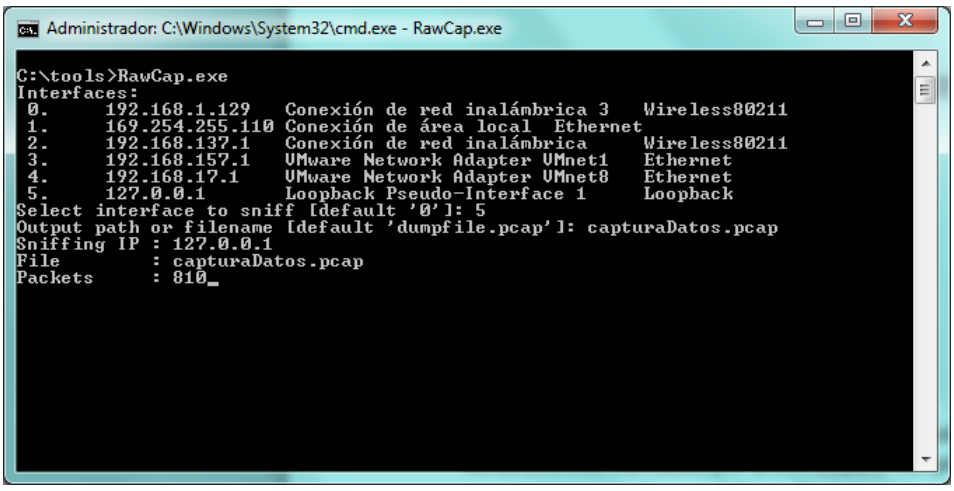


Imagen 21

Para evitar cualquier tipo de problemas, la consola de comandos la se abría en modo "Administrador". Una vez abierta se tiene que buscar la carpeta en la que se encuentra el programa *RawCap.exe* y se ejecuta. En la imagen anterior se puede ver cómo se ha de seleccionar la interfaz de red sobre la que se desean capturar los datos y, justo después, se debe dar un nombre al fichero de salida. Al pulsar sobre la tecla

"intro" el programa capturará los paquetes que salgan o lleguen a la interfaz previamente seleccionada.

El fichero obtenido se abrirá con *Wireshark* para analizar de una mejor forma los datos obtenidos. Los paquetes de la conexión que se pueden ver a continuación fueron capturados antes de permitir las conexiones cifradas en el servidor y en el dispositivo móvil:

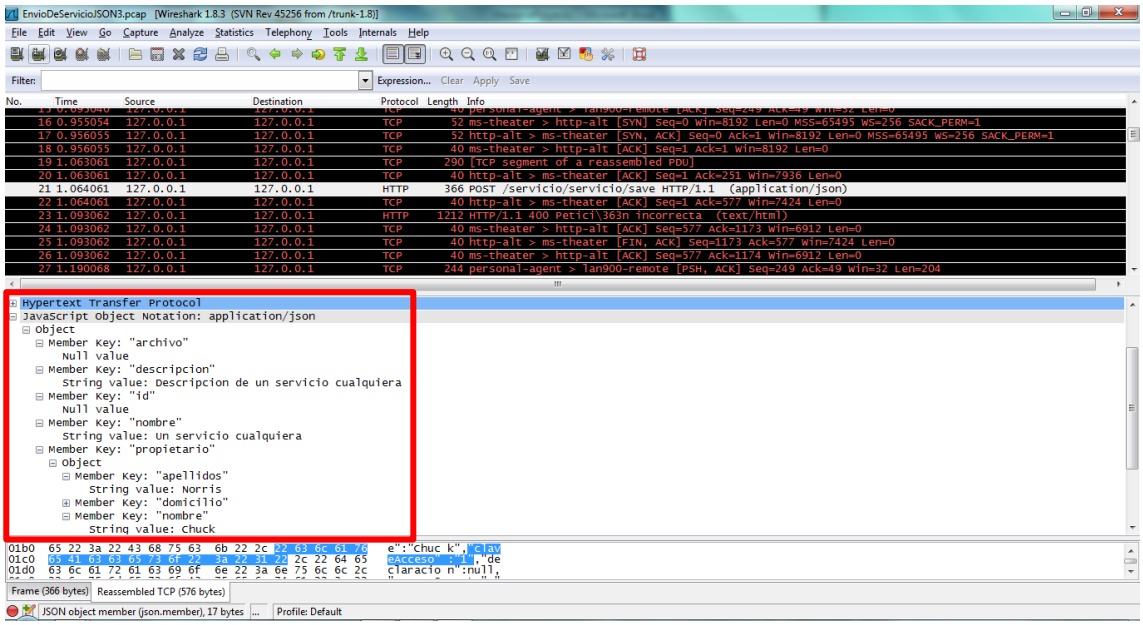


Imagen 22

En la Imagen 22 se puede observar cómo el paquete seleccionado viaja por **HTTP**, el método usado es **POST**, la URL es ***/servicio/servicio/save*** y el tipo de datos ***application/json***. En el área resaltada de color rojo se pueden ver los objetos JSON incluidos en el paquete, pudiendo acceder incluso a los valores de cada uno de los atributos de estos objetos.

Una vez implementada la seguridad de las comunicaciones, la aplicación final se instaló en el dispositivo móvil para realizar las pruebas sobre un terminal físico. Los datos viajan ahora a través de la red por lo que ahora sí se podían hacer las capturas de los paquetes y la visualización de los datos capturados desde el propio *Wireshark*, evitando así el uso de la herramienta *RawCap*.

La captura de paquetes en *Wireshark* resulta más sencilla puesto que, desde la interfaz gráfica del programa, puedes acceder a toda su funcionalidad. El programa también se debe de arrancar como "Administrador", al igual que se hacía en *RawCap,* para que permita realizar capturas desde las interfaces de red. Una vez iniciado se pulsará sobre el botón "Lista de interfaces disponibles", y se seleccionará la interfaz por la cual circularán los datos. A continuación se pulsará sobre el botón "Start". En la siguiente imagen se puede ver de forma más gráfica lo descrito en este párrafo.

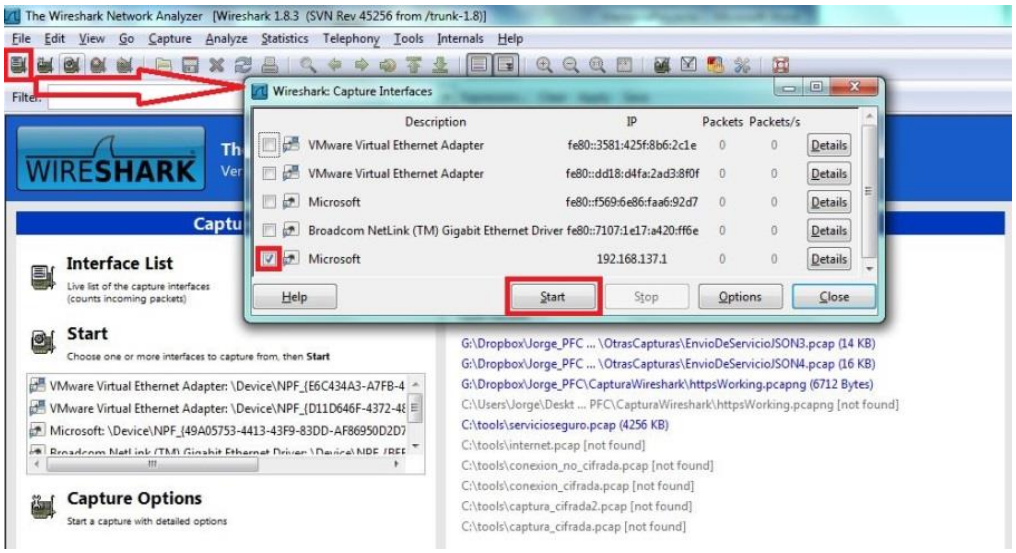


Imagen 23

La captura que se mostrará a continuación fue realizada cuando la conexión entre el servidor y el cliente usaba finalmente el protocolo HTTPS:

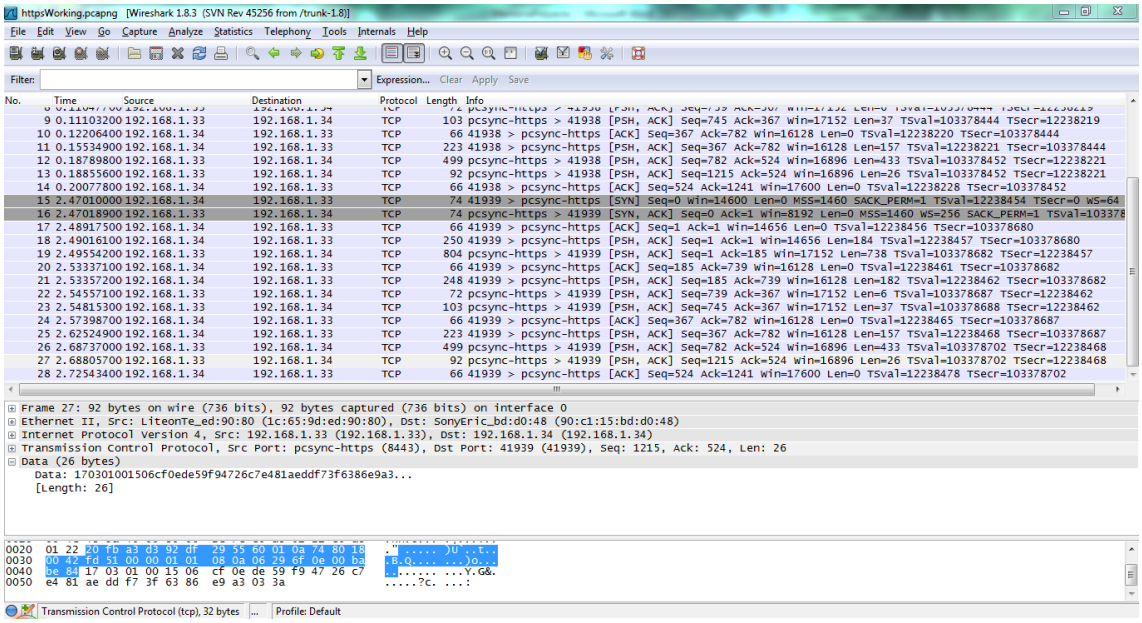


Imagen 24

Se puede apreciar en la Imagen 24 que la comunicación efectivamente va sobre HTTPS y los datos no son visibles como en la anterior captura (Imagen 22).

Se ha implementado todo lo necesario para mantener seguros los datos de los usuarios del sistema. La base de datos ha sido cifrada con *Jasypt* y con el cifrado de claves propio de Spring y las comunicaciones se han cifrado gracias al certificado de seguridad creado, a la configuración establecida en el servidor *Tomcat* y a la creación de la clase *HttpsClient* en la aplicación móvil para el uso de este certificado. Por este motivo podemos afirmar que el sistema al completo, servidor, comunicaciones y móvil, cumple con el Artículo 9 de la LOPD.

4. Conclusiones y trabajos futuros

Este proyecto ha cumplido todos los Objetivos marcados en el inicio del mismo:

1. Se ha creado una plataforma móvil segura con la cual se puedan se pueden crear y consumir cualquier tipo de servicio.
2. Se ha creado un servidor seguro encargado de gestionar las peticiones de los clientes móviles.
3. Se ha realizado un profundo estudio de la Ley Orgánica de Protección de datos y se ha ido aplicando a cada una de las etapas del desarrollo del proyecto y a cada uno de los módulos del mismo. Con ello hemos conseguido cumplir los requisitos marcados por dicha ley, tanto en la parte relacionada con el servidor como en el dispositivo móvil.

Se ha seguido la planificación marcada en el apartado 1.3 de este documento, y se han usado todos los recursos indicados en el apartado 1.2.

Gracias a la implementación de este proyecto y a la creación de este documento, se han afianzado numerosos de los conocimientos aprendidos a lo largo de la carrera. Además se han adquirido otros conceptos bastante novedosos y demandados en el desarrollo del software en la actualidad.

El empleo de una metodología ágil en el desarrollo, concretamente Scrum, mediante el uso de la herramienta web como Scrumdo, ha hecho que me familiarice con este método de desarrollo. Aunque no se haya podido cumplir estrictamente, puesto que el equipo de desarrollo ha estado compuesto por una sola persona, se ha intentado seguir de forma rigurosa en la medida de lo posible.

Se ha implementado una aplicación completa en Android que, como se ha comentado en la introducción del documento, es uno de los sistemas operativos dominantes y con una gran demanda de desarrolladores. Aunque previamente conocía la programación en este entorno, el proyecto ha servido para sentar las bases y conocer otros aspectos novedosos del ecosistema Android.

El servidor encargado de gestionar las peticiones de los clientes móviles ha sido desarrollado en Spring Framework, creado en el lenguaje java. Con ello he conseguido adquirir una perspectiva distinta y una forma de programación alternativa como es el desarrollo de software del lado del servidor.

Entre los requerimientos del sistema se encontraba la seguridad de los datos en ambos módulos, móvil y servidor. La implementación del protocolo HTTPS para la comunicación entre servidor y dispositivos móviles, el uso de certificados de seguridad y la utilización de diversas técnicas de cifrado, nos aseguran que los datos de los usuarios de nuestro sistema permanecerán seguros. Esta parte quizá haya sido la más costosa debido a los numerosos conceptos utilizados y la complejidad de los mismos.

La decisión de cumplir la Ley Orgánica de Protección de datos en el proyecto presentó nuevo reto, puesto que nunca había tenido que enfrentarme al estudio e implantación de ninguna ley en el desarrollo del software previamente. Se ha realizado un profundo análisis de la misma y se han ido implementando en cada caso las medidas necesarias para que el proyecto se ajuste a los requisitos marcados por la LOPD.

4.1. Requisitos finales LOPD

Aunque el software del proyecto ha sido diseñado e implementado siguiendo las directrices marcadas por la LOPD, para cumplir con esta ley, todavía hay que realizar tres pasos antes de que dicho software pueda lanzarse a producción.

1. Se debe crear el documento de seguridad. Este documento recogerá las medidas de índole técnica y organizativa acorde a la normativa de seguridad vigente que será de obligado cumplimiento para el personal con acceso a los datos de carácter personal. Como mínimo este documento ha de incluir:

    a. Ámbito de aplicación del documento con especificación detallada de los recursos protegidos.

    b. Medidas, normas, procedimientos de actuación, reglas y estándares encaminados a garantizar el nivel de seguridad exigido en este reglamento.

    c. Funciones y obligaciones del personal en relación con el tratamiento de los datos de carácter personal incluidos en los ficheros.

    d. Estructura de los ficheros con datos de carácter personal y descripción de los sistemas de información que los tratan.

    e. Procedimiento de notificación, gestión y respuesta ante las incidencias.

    f. Los procedimientos de realización de copias de respaldo y de recuperación de los datos en los ficheros o tratamientos automatizados.

    g. Las medidas que sea necesario adoptar para el transporte de soportes y documentos, así como para la destrucción de los documentos y soportes, o en su caso, la reutilización de estos últimos.

Si fuera necesario aplicar el nivel de seguridad medio o el nivel de seguridad avanzado a los ficheros de datos personales (nuestro proyecto no hay que aplicar estos dos últimos niveles) se deberían añadir además los siguientes apartados:

    h. Procedimiento de registro de accesos.

i. Procedimiento de registro de entrada y salida de soportes.
j. Criterios de archivo.
k. Almacenamiento de la información.
l. Custodia de soportes.
m. Mecanismos de cifrado.
n. Traslado de documentación.
o. Copias de respaldo y recuperación.
p. Responsable de seguridad.
q. Procedimiento de auditoría.

2. El documento anterior ha de ser entregado en la Agencia de Protección de Datos.

3. Esperar una notificación de dicha entidad comunicándonos la inscripción del fichero. Si transcurrido un mes desde la entrega de nuestro documento no hemos recibido notificación alguna, entenderemos que el fichero ha sido inscrito con normalidad.

Para crear el documento de seguridad es recomendable seguir la Guía de Seguridad de Datos[7], el modelo de seguridad editable [8] y revisar la página web de la Agencia Española de Protección de Datos[9] la cual contiene a los documentos anteriores.

4.2. El libro naranja de la NSA

En 1983 el Departamento de Defensa de los Estados Unidos publicó el Orange Book. En él se especifican unos requisitos básicos para la seguridad en un sistema de computadores. Estos requisitos se denominan TCSEC (Trusted Computer System Evaluation Criteria) y formaron un estándar en el que se clasifica el nivel de seguridad de un sistema informático.

4.2.1. Fundamentos de los TCSEC y clasificación del sistema implementado

Para estos sistemas se definen seis requisitos fundamentales: cuatro de ellos indican qué se necesita para proporcionar un control de acceso a los datos; los otros

[7] http://www.agpd.es/portalwebAGPD/canaldocumentacion/publicaciones/common/Guias/GUIA_SEGURIDAD_2010.pdf
[8] https://www.agpd.es/portalwebAGPD/canalresponsable/guia_documento/index-ides-idphp.php
[9] https://www.agpd.es/portalwebAGPD/canalresponsable/guia_documento/index-ides-idphp.php

dos tratan sobre cómo se podría asegurar su cumplimiento en un sistema de confianza:

- **Requisito 1 - Política de seguridad**: Debe haber un conjunto de reglas que sean usadas por el sistema para determinar si un sujeto dado puede acceder o no a un objeto específico.

- **Requisito 2 - Marcado**: Se deben asociar etiquetas de control de acceso con objetos.

- **Requisito 3 - Identificación**: Cada acceso a la información debe ser moderado, teniendo en cuenta quién accede a la información y a qué tipo de información puede acceder.

- **Requisito 4 - Responsabilidad**: Un sistema de confianza debe permitir almacenar los eventos de seguridad relevantes en un registro de auditoría. Esto se utilizará para poder realizar un seguimiento de las acciones que afecten a la seguridad del sistema y poder exigir responsabilidades.

- **Requisito 5 - Garantía**: Un sistema informático ha de tener mecanismos hardware y/o software que puedan garantizar que el sistema cumple de forma estricta los requisitos del 1 al 4 vistos anteriormente.

- **Requisito 6 - Protección Continua**: Los mecanismos de confianza que aseguran el cumplimiento de estos requerimientos básicos deben ser protegidos continuamente contra manipulaciones y cambios no autorizados.

La clasificación propuesta para los sistemas se divide en cuatro divisiones: D, C, B y A ordenadas de forma jerárquica, siendo A la división reservada para los sistemas más seguros. Cada división superior representa una mejora con respecto a su predecesora y contiene todos los niveles de seguridad de la misma. Dentro de las divisiones B y C existen unas subdivisiones llamadas clases. Estas clases también son ordenadas de forma jerárquica.

A continuación se presentarán las divisiones de seguridad y sus clases respectivas presentes en el Orange Book:

- **Nivel D:** Este nivel contiene sólo una división y está reservada para sistemas que no cumplen con ninguna especificación de seguridad. Son sistemas no confiables, no hay protección para el hardware, el sistema operativo es inestable y no hay autentificación con respecto a los usuarios y sus derechos en el acceso a la información.

- **Nivel C1 (Protección Discrecional):** En estos sistemas se requiere identificación de los usuarios para el acceso a la información. Cada usuario dispone de su información privada y se hace distinción entre los usuarios y el administrador del sistema, quien tiene control total de acceso. Es común que la tarea del

administrador la desempeñen dos o tres personas. Esto supone un problema, pues no hay forma de distinguir entre los cambios que realiza cada "super usuario". A continuación se enumeran los requerimientos mínimos que debe cumplir un sistema de la clase C1:

  - Acceso de control discrecional: distinción entre usuarios y recursos. Se podrán definir grupos de usuarios (con los mismos privilegios) y grupos de objetos (archivos, directorios, disco) sobre los cuales podrán actuar usuarios o grupos de ellos.

  - Identificación y Autentificación: se requiere que un usuario se identifique antes de comenzar a ejecutar acciones sobre el sistema.

- **Nivel C2 (Protección de Acceso Controlado):** Cuenta con características adicionales al nivel C1 que crean un ambiente de acceso controlado. Se debe llevar una auditoria de accesos e intentos fallidos de acceso a objetos. Restringe aún más la funcionalidad de los usuarios en cuanto a ejecutar ciertos comandos o tener acceso a ciertos archivos, permitir o denegar datos a usuarios en concreto, etc. Requiere una auditoría del sistema que será utilizada para llevar registros de todas las acciones relacionadas con la seguridad. La auditoría requiere de autenticación adicional para estar seguros de que la persona que ejecuta el comando es quien dice ser. Su mayor desventaja reside en los recursos adicionales requeridos por el procesador y el subsistema de discos. Los usuarios de un sistema C2 tienen la autorización para realizar algunas tareas de administración del sistema sin necesidad de ser administradores.

- **Nivel B1 (Seguridad Etiquetada):** Este subnivel, es el primero de los tres con que cuenta el nivel B. Soporta una seguridad multinivel, como la secreta y ultrasecreta. Se establece que el dueño del archivo no puede modificar los permisos de un objeto que está bajo control de acceso obligatorio. A cada objeto del sistema (usuario, dato, etc.) se le asignará una etiqueta, con un nivel de seguridad jerárquico (alto secreto, secreto, reservado, etc.) y con unas categorías (contabilidad, nóminas, ventas, etc.). Cada usuario que accede a un objeto debe poseer un permiso expreso para hacerlo y viceversa. También se establecen controles para limitar la propagación de derecho de accesos a los distintos objetos.

- **Nivel B2 (Protección Estructurada):** Requiere que se etiquete cada objeto de nivel superior por ser padre de un objeto inferior. La Protección Estructurada es la primera que empieza a referirse al problema de un objeto a un nivel más elevado de seguridad en comunicación con otro objeto a un nivel inferior. El sistema es capaz de alertar a los usuarios si sus condiciones de accesibilidad y seguridad son modificadas; y el administrador es el encargado de fijar los canales de almacenamiento y ancho de banda a utilizar por los demás usuarios.

- **Nivel B3 (Dominios de Seguridad):** Refuerza a los dominios con la instalación de hardware. Existe un monitor de referencia que recibe las peticiones de acceso de cada usuario y las permite o las deniega según las políticas de acceso que se hayan definido. Todas las estructuras de seguridad deben ser lo suficientemente pequeñas como para permitir análisis y testeos ante posibles violaciones. Este nivel requiere que la terminal del usuario se conecte al sistema por medio de una conexión segura. Además, cada usuario tiene asignado los lugares y objetos a los que puede acceder.

- **Nivel A (Protección Verificada):** Es el nivel más elevado, incluye un proceso de diseño, control y verificación, mediante métodos formales (matemáticos) para asegurar todos los procesos que realiza un usuario sobre el sistema. El diseño requiere ser verificado de forma matemática y también se deben realizar análisis de canales encubiertos y de distribución confiable. El software y el hardware son protegidos para evitar infiltraciones ante traslados o movimientos del equipamiento. [17] [18] [19]

El sistema descrito en este texto encajaría perfectamente en un nivel C1 o Protección Discrecional puesto que, para el acceso a la información, se requiere identificación de los usuarios y se hace distinción entre los usuarios y el administrador del sistema, quien puede realizar cualquiera de las opciones del sistema.

4.2.2. Estudio de la LOPD frente a los niveles de seguridad

Se han visto unos niveles bien especificados en el Orange Book que clasifican los sistemas informáticos según el nivel de seguridad. Por otra parte, la Ley Orgánica de Protección de Datos también incluye unos ciertos niveles de seguridad en los cuales se aplican una serie de normas específicas de cada nivel.

Un futuro estudio podría centrarse en la aplicabilidad de la Ley Orgánica de Protección de Datos en cada uno de los niveles sugeridos por el Departamento de Defensa de los Estados Unidos. Con ello se podría conseguir un sencillo manual de referencia para aquellas entidades que instalen cualquier nuevo sistema informático y quieran cumplir con esta ley.

5. Referencias y bibliografía

[1] P. UOC, «Aprendizaje móvil (I): un poco de historia,» 21 Octubre 2013. [En línea]. Available: http://informatica.blogs.uoc.edu/2013/10/21/aprendizaje-movil-i-un-poco-de-historia/.

[2] Wikipedia, «Android,» 20 Enero 2013. [En línea]. Available: http://es.wikipedia.org/wiki/Android.

[3] «Introducción al patrón Modelo-Vista-Controlador con Spring MVC,» 22 Noviembre 2012. [En línea]. Available: http://www.e-continua.com.mx/index.php/15-springmvc.

[4] C. Walls, Spring in action, Manning Publications Co, 2011.

[5] T. A. S. Foundation, «tomcat.apache.org,» [En línea]. Available: http://tomcat.apache.org/.

[6] T. A. S. Foundation, «Welcome to Apache Maven,» [En línea]. Available: http://maven.apache.org/.

[7] Wikipedia, «Hibernate,» [En línea]. Available: http://es.wikipedia.org/wiki/Hibernate.

[8] L. D. Seta, «Introducción a los servicios web RESTful,» 13 Noviembre 2008 . [En línea]. Available: http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html.

[9] Json, «Introducción a JSON,» [En línea]. Available: http://www.json.org/json-es.html.

[10] J. C. Moratilla., «Análisis del régimen sancionador de la LOPD,» Octubre 2011. [En línea]. Available: http://noticias.juridicas.com/articulos/15-Derecho-Administrativo/201110-8529523812578644.html.

[11] Wikipedia, «Criptografía asimétrica,» [En línea]. Available: http://es.wikipedia.org/wiki/Criptograf%C3%ADa_asim%C3%A9trica.

[12] Jasypt, «Jasypt, Java Simplified Encryption,» 22 Agosto 2013. [En línea]. Available: http://www.jasypt.org/.

[13] D. Cutanda, «Fundamentos sobre certificados digitales,» 13 mayo 2013. [En línea]. Available: http://www.securityartwork.es/2013/05/13/fundamentos-sobre-certificados-digitales/.

[14] «Certificado de Seguridad,» 2014. [En línea]. Available: http://www.certificadodeseguridad.com/.

[15] L. Castro, «¿Qué es SSL?,» [En línea]. Available:

http://aprenderinternet.about.com/od/ConceptosBasico/a/Que-Es-Ssl.htm.

[16] W. Herrera, «Qué es el protocolo HTTPS, cómo funciona y para que sirve?,» 13 Abril 2011. [En línea]. Available: http://webadictos.com/2011/04/13/que-es-el-protocolo-https-y-como-funciona/.

[17] D. o. D. Standard, Department of Defense Trusted Computer System Evaluation Criteria, 1985.

[18] C. Borghello, «Seguridad Lógica,» [En línea]. Available: http://www.segu-info.com.ar/logica/seguridadlogica.htm.

[19] F. d. I. Universidad Nacional Autónoma de México, «Fundamentos de Seguridad Informática,» [En línea]. Available: http://redyseguridad.fi-p.unam.mx/proyectos/seguridad/TCSEC.php.

[20] Wikipedia, «Modelo Vista Controlador,» [En línea]. Available: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador.

[21] C. Á. Caules, «Spring MVC Configuración (I),» 20 Noviembre 2012. [En línea]. Available: http://www.arquitecturajava.com/spring-mvc-configuracion/.

[22] GoPivotal, «Spring Framework,» [En línea]. Available: http://projects.spring.io/spring-framework/.

[23] GoPivotal, «Spring,» [En línea]. Available: http://spring.io/.

[24] Wikipedia, «Spring Framework,» [En línea]. Available: http://es.wikipedia.org/wiki/Spring_Framework.

[25] http://www.javamexico.org/usuarios/ezamudio, «Contenedores ligeros con Spring,» 5 Mayo 2008. [En línea]. Available: http://www.javamexico.org/blogs/ezamudio/contenedores_ligeros_con_spring.

[26] http://www.genbetadev.com/autor/mangrar, «Spring Framework: Introducción,» 15 Junio 2011. [En línea]. Available: http://www.genbetadev.com/java-j2ee/spring-framework-introduccion.

[27] VORTEXBIRD, «Introduccion springframework,» 8 Septiembre 2011. [En línea]. Available: www.slideshare.net/vortexbird/introducccion-springframework.

[28] Wikipedia, «Tomcat,» [En línea]. Available: http://es.wikipedia.org/wiki/Tomcat.

[29] Wikipedia, «Maven,» [En línea]. Available: http://es.wikipedia.org/wiki/Maven.

[30] L. González, «Curso de Hibernate con Spring,» 1 Enero 2014. [En línea]. Available:

http://cursohibernate.es/doku.php.

[31] Wikipedia, «Representational State Transfer,» [En línea]. Available: http://es.wikipedia.org/wiki/Representational_State_Transfer.

[32] Wikipedia, «X.509,» 2014. [En línea]. Available: http://es.wikipedia.org/wiki/X.509.

[33] Wikipedia, «Certificado digital,» 3 Enero 2014. [En línea]. Available: http://es.wikipedia.org/wiki/Certificado_de_clave_p%C3%BAblica.

[34] Wikipedia, «Infraestructura de clave pública,» 2 Diciembre 2013. [En línea]. Available: http://es.wikipedia.org/wiki/Infraestructura_de_clave_p%C3%BAblica.

[35] Certsuperior, «¿Qué es un certificado SSL?,» [En línea]. Available: http://www.certsuperior.com/QueesunCertificadoSSL.aspx.

[36] Wikipedia, «Trusted Computer System Evaluation Criteria,» [En línea]. Available: http://en.wikipedia.org/wiki/Trusted_Computer_System_Evaluation_Criteria.

[37] A. E. d. P. d. Datos, Guía de Seguridad de Datos, 2008.

[38] A. E. d. P. d. Datos, Guía del Reponsable de Ficheros, 2008.

6. Anexos

Se adjunta información complementaria que creemos conveniente para extender y completar el proyecto.

6.1. Guía de uso de la aplicación móvil

A continuación vamos a crear una pequeña guía que muestre el uso de la aplicación móvil creada incluyendo algunas capturas de pantalla de la misma.

6.1.1. Registro en nuestro sistema

La primera pantalla con la que nos encontramos al iniciar la aplicación es la del registro. Como podemos ver en la Imagen 25 nos aparecen unos simples cuadros de texto para indicar los datos personales de los usuarios de la aplicación. Todos los campos son obligatorios por lo que si un usuario pulsara sobre el botón "Registrarse" pero hubiera algún campo vacío se le notificaría esta incidencia. Una vez tenga todos los campos correctamente cumplimentados en el formato correcto (por ejemplo un número telefónico superior a 9 números) podrá darse de alta en nuestro sistema.

El comportamiento del botón "Aviso legal" ya lo hemos comentado en el apartado *NuevoUsuarioActivity*, por lo que no es necesario volver a explicarlo.

Imagen 25

Una vez registrados en la aplicación nos aparecerá la pantalla de la Imagen 26. Esta es la pantalla principal de la aplicación, desde ésta se podrá acceder a las demás funcionalidades implementadas. Como vemos está compuesta por tres simples

botones que te permitirán crear un nuevo servicio, listar todos los servicios disponibles o listar todos los servicios creados por el usuario de la aplicación respectivamente.

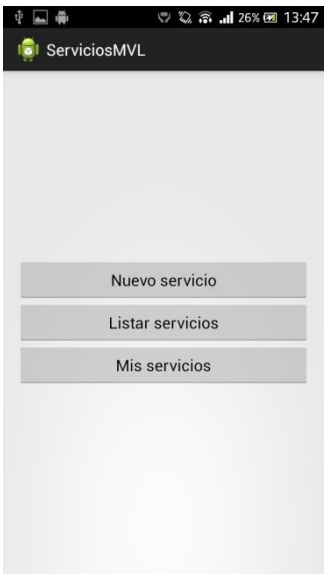


Imagen 26

Antes de comenzar a describir cada una de las funcionalidades de los botones de la anterior imagen vamos mostrar lo que ocurre cuando un usuario ya está registrado en nuestro, tiene la aplicación instalada y la ejecuta desde su terminal.

6.1.2. Inicio de aplicación

Como ya hemos avanzado si un usuario está registrado en nuestro sistema no le aparecerá de nuevo la pantalla de la Imagen 25, en este caso lo que el usuario vería en su dispositivo sería la siguiente imagen:

Imagen 27

Para acceder a la aplicación, el usuario debe introducir el pin secreto que introdujo al registrarse. Este pin se usa para descifrar un archivo almacenado en la memoria del dispositivo que contiene el identificador del usuario, el correo del mismo y una contraseña aleatoria generada por el dispositivo en el momento del registro del usuario de 128 caracteres de longitud. Esta información se envía al servidor para comprobar si el usuario se encuentra en la base de datos y puede acceder a la aplicación.

Si el usuario intenta pulsar el botón “Atrás”, la aplicación le comunicará que, si desea cerrar la aplicación, pulse el botón “Home”. Lo vemos en la siguiente imagen:

Imagen 28

Una vez el usuario ha introducido el pin correcto y el servidor ha confirmado los datos, la aplicación mostrará la siguiente pantalla:

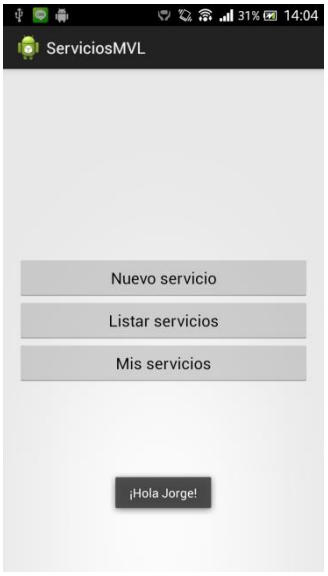


Imagen 29

6.1.3. Crear un servicio

Vamos a comenzar con el primer botón que aparece en la pantalla principal de la aplicación, el botón “Nuevo Servicio”. Al pulsar sobre este botón, se nos abrirá otra sección de la aplicación encargada de obtener los atributos del servicio que queramos crear.

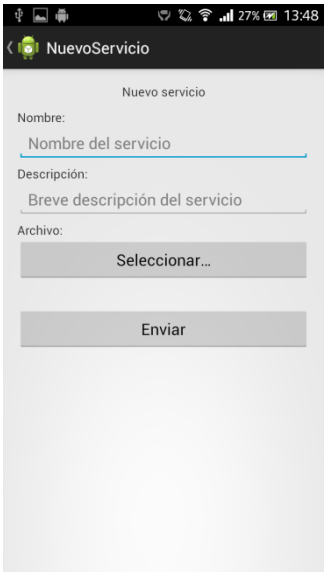


Imagen 30

En la anterior imagen vemos los atributos que tiene el servicio. Al servicio tendremos que darle un nombre. Éste ha de describir el servicio que queremos crear pero también ha de ser lo más breve posible.

En apartado “Descripción” detallamos ampliamente el servicio para evitar confusiones y dejar claro qué es lo que se ofrece o se demanda.

También tenemos la posibilidad de adjuntar un archivo, es probable que el servicio sea un simple archivo multimedia, por lo que ese archivo debería ir adjunto al servicio.

Una vez hemos introducido la información del servicio pulsaremos sobre el botón enviar. Entonces se enviarán los datos del servicio al servidor que lo almacenará en la base de datos. A cualquier usuario que liste los servicios le aparecerá el nuevo servicio creado.

Al finalizar la tarea, la aplicación vuelve a la pantalla que vemos en la Imagen 26, por si el usuario desea realizar alguna otra función.

6.1.4. Listar servicios disponibles

El siguiente botón que vemos en la Imagen 26 es “Listar servicios”. Al pulsar sobre este botón se realizará una petición al servidor esperando como respuesta un listado de todos los servicios disponibles. Los servicios devueltos por el servidor se mostrarán en forma de lista tal y como se muestra en la siguiente imagen:

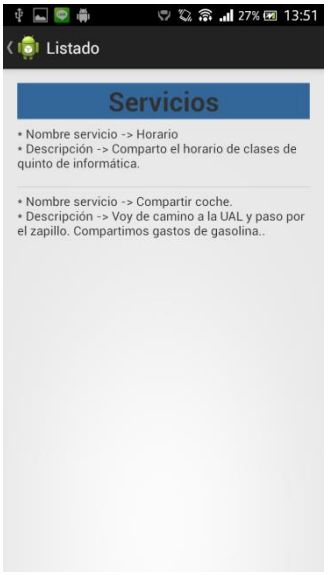


Imagen 31

En esta pantalla se visualizan solamente dos servicios. Pulsando en cada uno de ellos se visitará otra sección en la que se puede ver con más detalle el servicio pulsado.

6.1.5. Comprar servicio

Una vez se han listado los servicios disponibles y se ha pulsado en cualquiera de ellos la aplicación mostrará una ventana en la que se puede ver el servicio más detallado y un botón “Comprar” con el cual puedes obtener dicho servicio.

Imagen 32

La Imagen 32 corresponde al segundo servicio del listado que se mostraba en la Imagen 31. Aparece el botón "Comprar" comentado anteriormente e informa de la no existencia de archivo adjunto al servicio actual. Si se pulsara sobre "Comprar" se envía la información del usuario que ha pulsado el botón y la del servicio en sí. A continuación el servidor creará una relación entre ambas entidades y ocultará este servicio al resto de los usuarios. Una vez finalizada la operación se vuelve a la pantalla mostrada en la Imagen 31.

En la siguiente imagen se puede ver la misma pantalla que estamos describiendo pero con los datos del primer servicio de la lista mostrado en la Imagen 31:

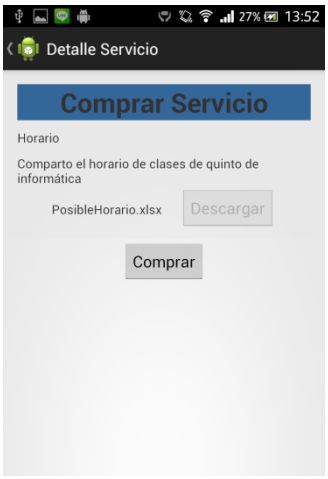


Imagen 33

En este caso el servicio sí lleva adjunto un fichero, por ello se muestra el nombre del archivo y un botón deshabilitado "Descargar". Para habilitar el botón que te permite descargar el archivo primero has de comunicarle al servidor el interés en obtener el servicio, para ello se ha de pulsar el botón "Comprar". Una vez el servidor ha creado la relación entre el usuario interesado y el servicio, se desbloqueará el botón "Descargar" tal y como se muestra en la imagen siguiente:

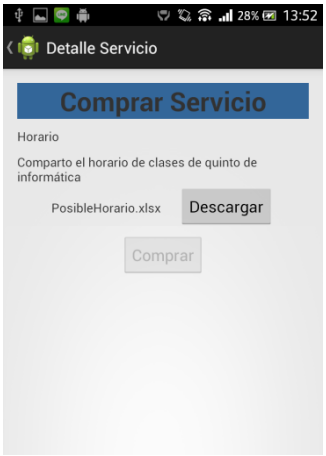


Imagen 34

Para salir de esta ventana se ha de pulsar el botón “Atrás” del dispositivo y como hemos pulsado sobre “Comprar” la lista de servicios se actualiza con los servicios disponibles, en nuestro caso, el servicio disponible:

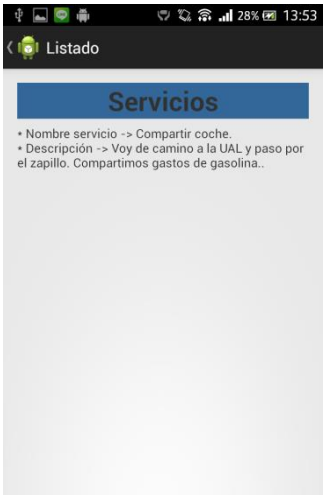


Imagen 35

6.1.6. Listar mis servicios

Volviendo a la pantalla principal de la aplicación, la mostrada en la Imagen 29, analizaremos la funcionalidad del último de sus botones “Listar mis servicios”. Al pulsar sobre dicho botón se realizará una petición al servidor, cuya respuesta será una lista de los servicios creados por el usuario de la aplicación, y abrirá una nueva ventana mostrando estos servicios en un listado tal y como los vemos en la siguiente imagen:

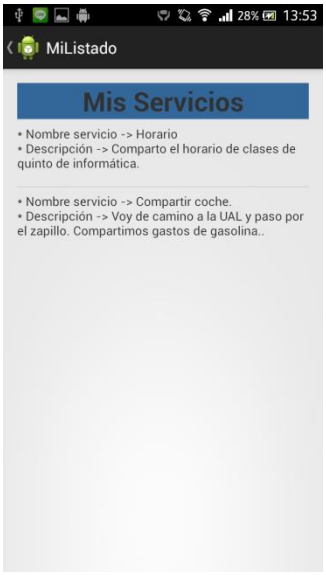


Imagen 36

Esta pantalla es muy similar a la que se aprecia en la Imagen 31 sólo que en esta se mostrarán los servicios creados por el usuario de la aplicación y en la de la Imagen 31 aparecerán todos los servicios disponibles.

Al pulsar sobre cualquiera de los servicios de este listado aparecerá una nueva ventana para poder eliminar el servicio propio del usuario del servidor. Esta ventana se verá en el siguiente apartado.

6.1.7. Eliminar un servicio propio

Esta nueva pantalla muestra el detalle de un servicio propio del usuario de la aplicación y un botón "Eliminar" con el que se puede borrar este servicio del servidor.

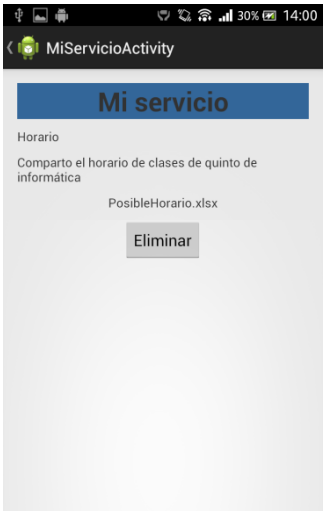


Imagen 37

Si se pulsa sobre "Eliminar" se enviará la petición de borrado al servidor y se retornará a la ventana de la Imagen 36 pero con el listado actualizado, tal y como se muestra en la siguiente imagen:

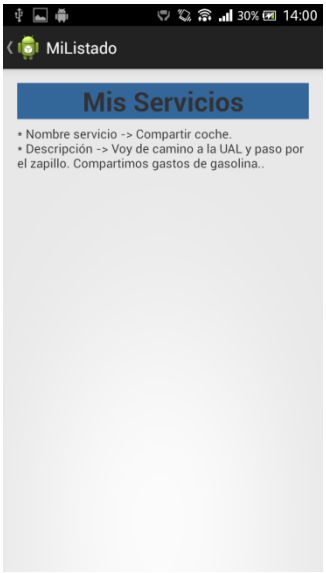


Imagen 38