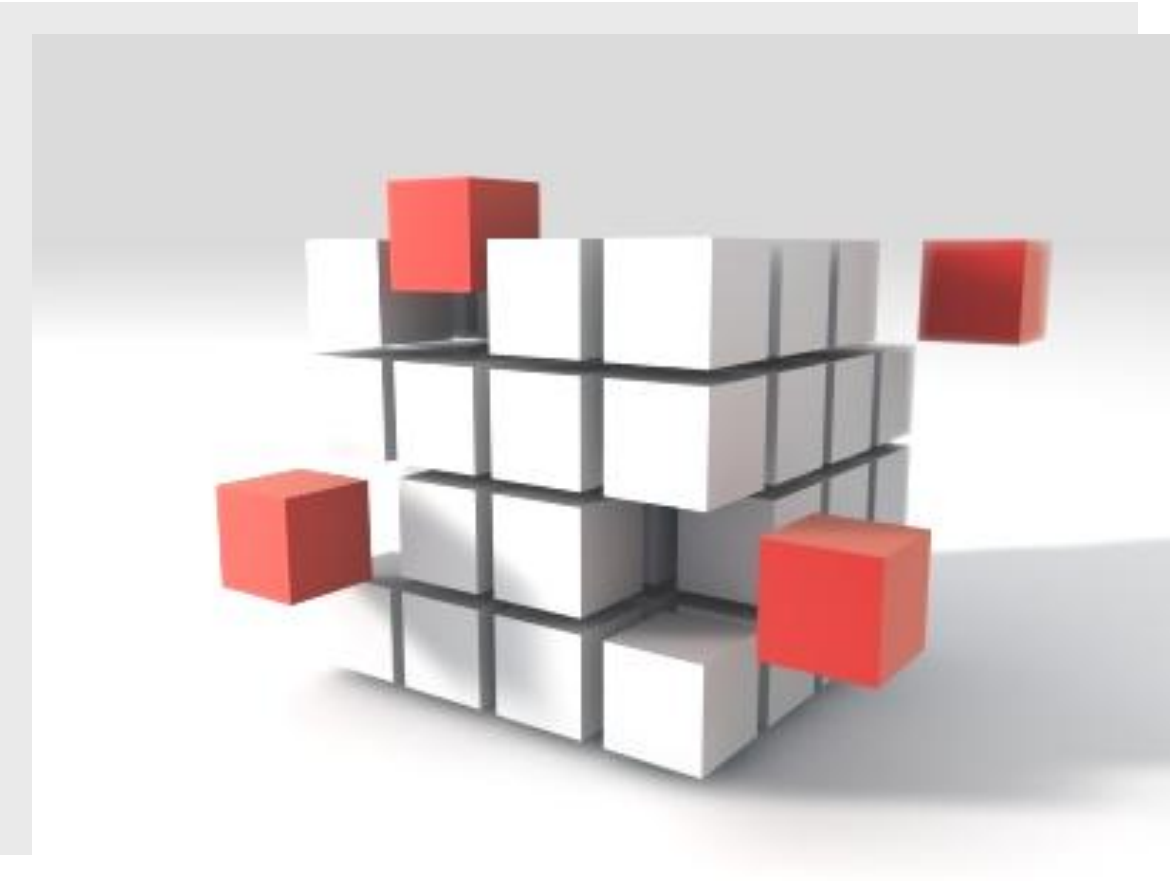
MEMORIA TFC

GESTIÓN DE USUARIOS



Pau Ignasi Gómez Molina
ETIS

Salvador Campo Mazarico
20 de junio de 2011

(Creative Commons)

Aquest treball està subjecte - excepte que s'indiqui el contrari- en una llicència de Reconeixement-NoComercial-SenseObraDerivada 2.5 Espanya de Creative Commons. Podeu copiar-lo, distribuir-los i transmetre'ls públicament sempre que citeu l'autor i l'obra, no es faci un ús comercial i no es faci còpia derivada. La llicència completa es pot consultar en:

http://creativecommons.org/licenses/by-nc-nd/2.5/es/deed.es.

A todos aquellos que se han sacrificado para que pudiera conseguir mi sueño, y especialmente a ti Noemí, sin ti nunca hubiera podido llegar hasta aquí.

RESUMEN

El siguiente documento presenta la memoria del Trabajo Final de Carrera perteneciente al **área J2EE**. En él se describe el desarrollo de la **aplicación** llamada **Gestión de Usuarios** cuyo objeto es el de proveer una *administración de usuarios potente y sencilla*. Ésta debe permanecer aislada de las aplicaciones que requieran su uso, y además, representar un modelo lo más general posible, para que pueda ser utilizada en el mayor número de ámbitos, de manera que será un contenedor de datos cuyo uso y significado vendrá dado por los sistemas clientes que lo consuman.

La estructura de esta memoria pretende mostrar el proceso seguido, siendo cada uno de sus capítulos los pasos cronológicos que se han ido dando hasta la obtención final del producto.

En la primera parte del texto se explicarán cuáles han sido las motivaciones para la elección de este TFC, sus objetivos y la metodología seguida. Al mismo tiempo, se introducirán los conceptos principales de la aplicación. Por otro lado, la segunda parte tiene un carácter más técnico, enfocada a describir las distintas fases del ciclo de vida utilizado.

En ningún momento he pretendido convertir el documento en un manual de J2EE o de las distintas herramientas utilizadas, por lo que muchos conceptos no son tratados en profundidad, si bien, en la mayor parte de los casos incluyo una breve descripción de ellos.

ÍNDICE

ÍNDICE DE CONTENIDOS

1. Introducción 6
- **1.1** *Justificación y Contexto en el que se desarrolla el TFC* 6
- **1.2** *Objetivos generales y específicos* 6
- **1.3** *Metodología* 7
- **1.4** *Planificación del proyecto* 8
- **1.5** *Producto obtenido* 10
- **1.6** *Siguientes capítulos de la memoria* 12

2. Recogida de Documentación 13
- **2.1** *Spring Framework* 13
- **2.2** *Spring Security* 14
- **2.3** *Google Web Toolkit* 14
- **2.4** *ExtJs para GWT (GXT)* 15
- **2.5** *GWT Server Library* 15
- **2.6** *Java Persistence API e Hibernate* 16

3. Análisis y Diseño 18
- **3.1** *Análisis* 18
  - 3.1.1 Actores 18
  - 3.1.2 Casos de Uso 18
  - 3.1.3 Diagrama de Clases 40
  - 3.1.4 Diagrama Entidad-Relación 40
- **3.2** *Implementación* 41
  - 3.2.1 Configuración del Contenedor 41
  - 3.2.2 Configuración de GWT y GXT 43
  - 3.2.3 Modelo, Vista, Controlador 44
  - 3.2.4 Inyección de Dependencias (DI) 49
  - 3.2.5 Capa de Persistencia 49

4. Funcionalidad 51
- **4.1** *Acceso* 51
- **4.2** *Aplicaciones* 52
- **4.3** *Roles* 53
- **4.4** *Agrupadores* 54
- **4.5** *Multiplicadores* 55
- **4.6** *Permisos* 56

5. Futuras Ampliaciones 57

6. Conclusiones 57

7. Glosario 58

8. Bibliografía 59

ÍNDICE DE FIGURAS

Figura 1.	*Planificación del proyecto.*	8
Figura 2.	*Arquitectura MVC de Spring.*	13
Figura 3.	*Diagrama de GWT Server Library.*	16
Figura 4.	*Diagrama de casos de uso del Administrador.*	19
Figura 5.	*Diagrama de casos de uso del Administrados de la Aplicación.*	22
Figura 6.	*Detalle de la Gestión de Agrupadores del Administrador de Aplicación.*	22
Figura 7.	*Detalle de la Gestión de Roles del Administrador de Aplicación.*	24
Figura 8.	*Detalle de la Gestión de Multiplicadores del Administrador de Aplicación.*	27
Figura 9.	*Detalle de la Gestión de Permisos del Administrador de Aplicación.*	29
Figura 10.	*Diagrama de casos de uso del Super Administrador.*	32
Figura 11.	*Diagrama de casos de uso del Servicio Web.*	34
Figura 12.	*Diagrama de Clases.*	40
Figura 13.	*Diagrama Entidad-Relación.*	40
Figura 14.	*Pantalla de Acceso.*	52
Figura 15.	*Pantalla de Aplicación.*	52
Figura 16.	*Pantalla de Roles.*	53
Figura 17.	*Pantalla de Agrupadores.*	55
Figura 18.	*Pantalla de Multiplicadores.*	56
Figura 19.	*Pantalla de Permisos.*	56

1. INTRODUCCIÓN

1.1. JUSTIFICACIÓN Y CONTEXTO EN EL QUE SE DESARROLLA EL TFC

La decisión de escoger esta área del TFC se fundamenta en la necesidad de **adquirir capacidades en el desarrollo de aplicaciones J2EE**.

Desde hace algún tiempo estoy trabajando en una empresa donde realizo tareas de mantenimiento de aplicaciones J2EE ya existentes, sin embargo carezco de experiencia en la creación de aplicaciones desde cero.

Otra de las razones que me impulsaron a realizar este TFC fue el interés por conocer una serie de tecnologías que de otro modo no hubiera tenido ocasión de aprender. Como es bien sabido J2EE es una especificación con multitud de implementaciones diferentes, las cuáles disponen de un **amplio conjunto de frameworks** que las complementan. Esto hace muy difícil especializarse en una tecnología concreta aunque es preciso disponer de conocimientos generales en todos ellos para poder enfrentarse a nuevos proyectos.

En lo que se refiere al proyecto en sí, contemplé diversas opciones pero finalmente me decanté por la gestión genérica de usuarios por varias razones. La primera de ellas es que permite solventar los típicos problemas que surgen en el mantenimiento de un proyecto, cuando un cliente solicita nuevas funcionalidades o la modificación de alguna de ellas en un entorno dónde existen gran cantidad de usuarios con variados niveles de acceso. Además, me permite usar las tecnologías que deseaba aprender a manejar. Tales tecnologías son, *Google Web Toolkit*, para la capa de visualización, *Spring Framework* para el patrón MVC, y *Java Persistence API (JPA)* para la capa de persistencia.

La gestión de usuarios debe ser lo suficientemente *flexible* como para abarcar las necesidades de un amplio abanico de aplicaciones. Desde la más sencilla en la que únicamente precise gestionar un conjunto reducido de roles, a la más compleja en la que se deban asignar permisos a cada uno de los roles. Además, debe estar completamente desacoplada del resto de las aplicaciones que vayan a consumirla, por lo que será simplemente un almacén de la estructura lógica de los usuarios y sus permisos.

1.2. OBJETIVOS GENERALES Y ESPECÍFICOS

El **objetivo principal** de este proyecto es el de **sentar las bases para el desarrollo de una aplicación genérica de gestión de usuarios**. Dicha gestión debe ser capaz de almacenar una serie de datos lo suficientemente generales para ser reutilizado por distintas aplicaciones cliente.

Los **objetivos específicos** del proyecto pueden resumirse en lo siguiente:

- **Analizar los requisitos del proyecto** mediante la utilización de *esquemas UML* y *entidad-relación*.

- **Adquirir habilidades** en la utilización de las siguientes herramientas y tecnologías:
    - *Eclipse* como entorno de desarrollo.
    - *Spring Framework* para implementar el modelo MVC en la aplicación.
    - *JPA* e *Hibernate* con el patrón *DAO* para la capa de persistencia.
    - MySql como gestor de bases de datos, escogido por su gran distribución y recursos disponibles.
    - *GXT* (Google Web Toolkit para ExtJS) y los frameworks de javascript *ExtJS* y *jQuery* para la capa de visualización.
    - Tecnología *AJAX* para que las operaciones repetitivas de edición sean lo más productivas posible.
    - *Apache Tomcat* como contenedor de aplicaciones para el despliegue de la aplicación.
    - *Spring Security* en la implementación de la capa de seguridad.

- Los objetivos personales se centran en la **adquisición de competencias** a la hora de iniciar un proyecto, así como la capacidad de **manejar frameworks** específicos para realizar la tarea.

1.3. METODOLOGÍA

La metodología aplicada al proyecto ha sido **iterativa e incremental**, aplicándose el **ciclo de vida RUP** (*Rational Unified Process*) en el cual pueden distinguirse *cuatro fases*: inicio, elaboración, construcción y transición. Las dos primeras fases están enfocadas a la preparación y análisis del proyecto, mientras que las dos últimas al desarrollo y despliegue.

De las fases por las que ha pasado este TFC puede destacarse:
- **Inicio:** en el que se justifica la necesidad y oportunidad de la aplicación.
- **Elaboración:** dónde se estudia el dominio de la aplicación, las necesidades de información, y además se establece la arquitectura general del software, realizando una planificación del proyecto.
- **Construcción**: fase en que se procede al desarrollo del proyecto de forma iterativa e incremental.
- **Transición**: momento en el que la aplicación pasa a manos del usuario final comenzando el despliegue de la misma.

Según la especificación del *ciclo de vida RUP*, en cada uno de los períodos anteriores se llevan a cabo los siguientes componentes del proceso en distinta medida:
- Recogida de requisitos.
- Análisis y diseño.
- Implementación.
- Prueba.

Así mismo, teniendo en cuenta el ciclo de vida RUP, he decidido dividir el trabajo en distintos módulos atendiendo a su responsabilidad:
- **Integración de la capa de acceso y seguridad**, en la cual se encuentra la implementación del *framework Spring Security* y la inclusión de la capacidad de navegación mediante una *plantilla html*.

- Realización de **plantillas** implementando *GXT* en las distintas páginas dónde sea preciso.
- Incluir **llamadas *Remote Procedure Call (RPC)*** a Spring para integración con GWT.
- Definición de **data source** y **persistencia** utilizando *JPA* e *Hibernate*.
- **Desarrollo de los servicios** que dan funcionalidad a la lógica de la aplicación.

Cada uno de los módulos anteriores ha seguido la metodología iterativa e incremental, lo que me ha permitido ir ajustando el funcionamiento de las distintas partes según iban surgiendo cuestiones no contempladas en la fase de elaboración, según la metodología RUP.

1.4. PLANIFICACIÓN DEL PROYECTO

La planificación del proyecto se ha realizado en base a las **fechas propuestas** para la entrega de las distintas P.A.C. del TFC. Inicialmente se sugirió la siguiente distribución:

Nombre	Duración	Inicio	Terminado
TFC - J2EE	**77 days?**	**4/03/11 8:00**	**20/06/11 17:00**
PAC 1	**9 days?**	**4/03/11 8:00**	**16/03/11 17:00**
Elaboración de Plan de Trabajo	8 days	4/03/11 8:00	15/03/11 17:00
Entrega PAC 1	1 day?	16/03/11 8:00	16/03/11 17:00
PAC 2	**21 days?**	**17/03/11 8:00**	**14/04/11 17:00**
Especificación y análisis	**17 days?**	**17/03/11 8:00**	**8/04/11 17:00**
Definición de Casos de Uso	6 days?	17/03/11 8:00	24/03/11 17:00
Análisis de Entidades	6 days?	24/03/11 8:00	31/03/11 17:00
Diagrama de Clases	6 days?	1/04/11 7:00	8/04/11 17:00
Praparación entorno	**4 days?**	**8/04/11 8:00**	**13/04/11 17:00**
Preparación Entorno	1 day?	8/04/11 8:00	8/04/11 17:00
Establecer esqueleto aplicación	3 days?	10/04/11 8:00	13/04/11 17:00
Entrega PAC 2	1 day?	14/04/11 7:00	14/04/11 17:00
PAC 3	**27 days?**	**15/04/11 7:00**	**23/05/11 17:00**
Implementación	27 days?	15/04/11 7:00	23/05/11 17:00
Entrega PAC 3	0 days?	23/05/11 7:00	23/05/11 8:00
PAC 4	**20 days?**	**24/05/11 8:00**	**20/06/11 17:00**
Revisión	4 days?	24/05/11 8:00	27/05/11 17:00
Presentación Final	16 days?	28/05/11 8:00	20/06/11 17:00
Entrega PAC 4	1 day?	20/06/11 8:00	20/06/11 17:00

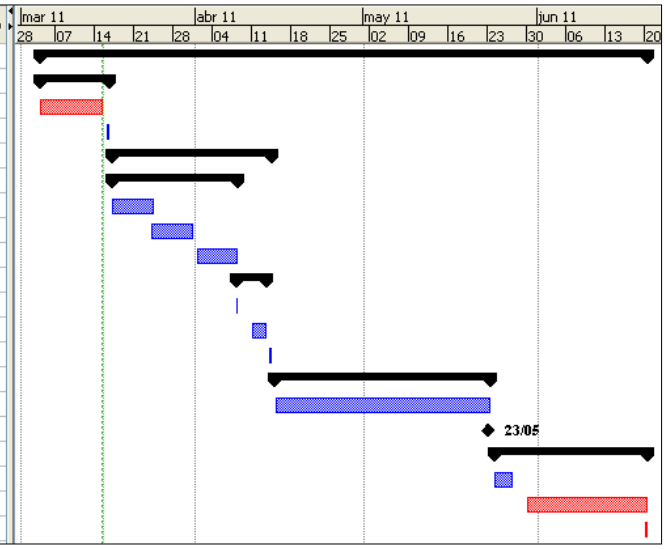


Figura 1. *Planificación del proyecto.*

Al seguir una metodología RUP se ha pasado por sus diferentes fases (descritas anteriormente). Aún así, es posible dividir las tareas realizadas en períodos de recogida de requisitos, análisis y diseño, implementación y pruebas, simplificando de este modo la planificación del proyecto.

A continuación incluyo la **planificación final** seguida:

Fase de recogida de requisitos

Duración:

2 semanas (4 de Marzo – 18 de Marzo).

Descripción:

Durante esta fase se ha decidido qué *tecnologías* emplear buscando información relativa a cada una de ellas. Si bien desde un principio tenía en mente la utilización de varias opciones, la lectura de la documentación recabada fue esencial para tomar una decisión final.

Objetivos:

Recopilación de información sobre las siguientes tecnologías:

- Spring framework 3.
- Google Web Toolkit (GWT)
- ExtJs para GWT (GXT)
- Spring Security 3.
- Java Persistence Api (JPA)
- Hibernate 3.6

Análisis y diseño

Duración:

4 semanas (17 de Marzo – 14 de Abril).

Descripción:

En esta etapa he desarrollado los distintos *casos de uso* así como los *diagramas UML* y *entidad-relación* en base a los cuales la aplicación será diseñada. Al mismo tiempo he profundizado en las distintas herramientas que van a ser utilizadas para su posterior implementación.

Objetivos:

- Descripción de los casos de uso.
- Diseño del diagrama de clases UML
- Diseño de las entidades de la base de datos.

Implementación y pruebas

Duración:

5 semanas (14 de Abril – 23 de Mayo).

Descripción:

Durante la fase de desarrollo se ha procedido a realizar todas aquellas *tareas necesarias para la implementación* de la aplicación.

Objetivos:

- Instalación del entorno, Spring Tool Suite (entorno eclipse para Spring framework).
- Preparación del esqueleto de la aplicación con la inclusión de spring.
- Integración de las páginas HTML a utilizar.
- Integración de Spring Security 3.0.
- Implementación inicial de componentes GXT.
- Integración de JPA y definición de entidades.
- Integración de GWT con spring para la gestión de llamadas RPC.
- Implementación de GXT para cada uno de los casos de uso, así como de los servicios necesarios para su correcta ejecución.
- Realización de pruebas y depuración.
- Empaquetado y entrega del producto.

Fase de Documentación y presentación

Duración:

4 semanas (24 de Mayo – 20 de Junio).

Descripción:

Período establecido en el que se realizará *la memoria* y la *presentación*, que han de ser presentados junto a la aplicación del TFC.

Objetivos:

- Creación de la memoria del proyecto.
- Creación de la presentación del proyecto.
- Realización de los últimos retoques al producto para su presentación.

1.5. PRODUCTO OBTENIDO

El producto final obtenido forma parte de un proyecto mucho más ambicioso y de mayor dimensión para la gestión genérica de usuarios. Así que, me he centrado en el desarrollo de la parte más importante, que es la correspondiente a la gestión de los principales aspectos que componen la administración global de usuarios, y que corresponde a los siguientes aspectos:

- **Acceso de administradores:** Se ha introducido parte de la lógica de acceso de administradores a la aplicación. Si bien existe una página de login, al no existir aún el mantenimiento de usuarios no es posible completarlo. En cualquier caso, la integración de *Spring security* se ha llevado a cabo, controlando las *páginas que pueden ser visitadas* por aquellos usuarios no identificados, así como la *parte privada* accesible únicamente para los administradores.

- **Gestión de Aplicaciones:** Esta gestión de usuarios ha de ser capaz de *administrar los recursos de varias aplicaciones clientes*, por lo tanto se ha procedido a implementar este apartado. En él es posible crear y modificar las aplicaciones que van a ser clientes de la gestión de usuarios. A su vez, se permite *indicar qué aplicaciones compartirán usuarios y/o agrupadores*.

- **Gestión de Agrupadores:** Los usuarios deberán pertenecer a un agrupador al ser dados de alta en la aplicación. Sería válido decir que un usuario ha de existir en un grupo, como por ejemplo, un centro, un despacho o cualquier otra *unidad lógica* en la que deseemos incluirle. Por lo tanto se ha añadido la administración de este apartado. Los agrupadores pueden ser compartidos entre aplicaciones, de esta manera, si dos aplicaciones son complementarias, y existen dentro de una misma organización, podremos reutilizarlos. Pueden ser *compartidos* en la gestión de aplicaciones y *consultados*, *editados* y *modificados* en la gestión de agrupadores, esto podemos hacerlo si los usuarios dados de alta en un agrupador les asignamos un conjunto limitado de *roles*. Dichos roles disponibles en el agrupador también son añadidos en esta gestión, por lo que es posible determinar cuáles se encuentran disponibles para ser asignados a los usuarios pertenecientes a un agrupador dado.

- **Gestión de Roles:** Una vez se proceda al alta de un usuario, se le asignará obligatoriamente un rol. Un rol no es otra cosa que un *nombre identificativo* que tiene implícitos un *conjunto de capacidades* dentro de la aplicación cliente. Desde este apartado nos es posible gestionarlos.

- **Gestión de Multiplicadores:** Los multiplicadores son un aspecto importante dentro del concepto de *permisos*. En esta gestión genérica, como ya ha sido explicado, se le deben asignar roles a los usuarios. Éstos roles pueden ser suficientes por sí mismos para la administración de una aplicación, aunque pueden darse casos en los que se precise un

control más fino de las capacidades de un usuario. Para ese caso, se han añadido los permisos, los cuales facilitan identificar para cada rol si una acción le está permitida o no a un usuario. Los multiplicadores son capaces de "multiplicar" dichos *permisos según un instante* en el proceso de la aplicación cliente.
Por ejemplo, una aplicación de ventas pasará por distintas fases a lo largo de una venta. Dos de estas fases podrían ser "Pedido" y "Envío". El administrador de dicha aplicación ha añadido, entre otros, el permiso "Consultar estado del envío" que puede valer Sí o No. Al permiso se le ha asignado un multiplicador llamado "Estado". Dicho estado identificará todas las etapas por el que el proceso de venta pase, entre ellos "Pedido" y "Envío". Por lo tanto, tendremos que el permiso inicial se ha multiplicado por el número de etapas existentes. Si sólo tuviéramos los dos expuestos en el ejemplo, nos encontraríamos con que el permiso se duplicaría, tendríamos "Consultar estado del envío" a "NO" para el estado "Pedido" y "Consultar el estado del envío" a "SÍ" para el estado "Envío". El multiplicador en esta gestión se compone de dos partes, una primera que lo identifica, y una segunda, o detalle, en la que se desglosa cada uno de los elementos que lo componen. Concretando en el ejemplo anterior, tendríamos un multiplicador llamado "Estado" con los detalles "Pedido" y "Envío".

- **Gestión de Permisos:** La gestión de permisos ha sido implementada, permitiendo un mayor *control en las operaciones que los usuarios pueden realizar*. Los permisos pueden ser binarios, del tipo "Sí/No" o ternarios, "R/W/RW" (Lectura/Escritura/Lectura-Escritura). Además a cada permiso se le puede asignar uno o varios multiplicadores, comentados en el apartado anterior.

El producto se ha entregado en un fichero llamado *paugm-final.zip* que contiene los archivos *paugm-memoria.zip*, *paugm-presentacion-zip* y *paugm-producto.zip*:

- El archivo *paugm-memoria.zip* contiene una carpeta con el mismo nombre cuyo contenido es el siguiente:
  - **paugm-memoria.pdf**: Documento pdf con esta memoria.

- El archivo *paugm-presentacion.zip* contiene una carpeta con el mismo nombre cuyo contenido es el siguiente:
  - **paugm-presentacion.pdf**: Documento pdf con las diapositivas y notas adjuntas.
  - **paugm-presentacion.pptx**: Presentación en formato Power Point 2007.

- El archivo paugm-producto.zip contiene una carpeta con el mismo nombre cuyo contenido es el siguiente:
  - **paugm-manual-despliegue.pdf**: Documento pdf con las instrucciones para el despliegue de la aplicación.
  - Carpeta "*Script Base de Datos*" que contiene el fichero "**ScriptUserAdmin.sql**" que incluye el script para la base de datos.
  - Carpeta "*Fuentes*" en cuyo interior se encuentra el proyecto de Spring ToolSuite con el código fuente de la aplicación.
  - Carpeta "*Producto*" en donde se encuentra el producto del TFC.

1.6. SIGUIENTES CAPÍTULOS DE LA MEMORIA

A continuación, paso a enunciar los siguientes capítulos de la memoria realizando una breve descripción de los mismos, con el objetivo de formar una visión global de la estructura del documento:

- **Capítulo 2: Recogida de documentación.**
  En este capítulo se realiza una introducción a las tecnologías empleadas a lo largo del proyecto que incluye explicaciones de las mismas, durante la fase de recogida de información.

- **Capítulo 3: Análisis y Diseño.**
  Detalla la fase de análisis mediante los casos de uso, diagrama de clases y diagrama de entidad relación, así como el diseño de la aplicación junto a la justificación de las decisiones tomadas en el proceso de desarrollo.

- **Capítulo 4: Funcionalidad.**
  Manual de funcionamiento del producto. Se adjuntan capturas de pantalla para el seguimiento de las explicaciones.

- **Capítulo 5: Futuras ampliaciones.**
  Breve descripción con las tareas que se realizarán para ampliar y mejorar la funcionalidad de la Gestión de Usuarios.

- **Capítulo 6: Conclusiones.**
  Conclusiones personales tras la finalización del TFC.

- **Capítulo 7: Bibliografía.**
  Documentos consultados para la elaboración del trabajo.

2. RECOGIDA DE DOCUMENTACIÓN

2.1. SPRING FRAMEWORK

La aplicación está diseñada utilizando Spring Framework. Dicho framework es un contenedor ligero, no intrusivo con la lógica de la aplicación que, entre otras cosas, facilita la *gestión y creación de beans*. Spring implementa la *inversión de control* (Inversion of Control), por la que es posible *inyectar dependencias* (Dependency Inyection) a las clases que lo requieran. Esto significa que el contexto de Spring creará los beans que precisemos y los inyectará en nuestras clases, siempre mediante la resolución de interfaces, evitando de este modo el acoplamiento entre objetos de nuestra aplicación. A partir de la versión 3 de Spring, se recomienda que la inyección de beans se realice a través de la resolución de anotaciones, de modo que ya no es preciso definir un gran fichero XML en el que se indique explícitamente qué dependencias deben ser inyectadas a qué beans.

A parte de la Inyección de Dependencias (DI), Spring ofrece *soporte para MVC* (Model View Controller). Este framework de MVC, mediante un Dispatcher Servlet, facilita el *mapeo de peticiones web a manejadores* (handlers) anotados con la etiqueta @Controller. También es posible *configurar la resolución de las vistas* (Views) y la *resolución de la localización* para internacionalizar la aplicación.

Un esquema extraído de la documentación oficial [1] permite ilustrar lo comentado sobre el soporte MVC:

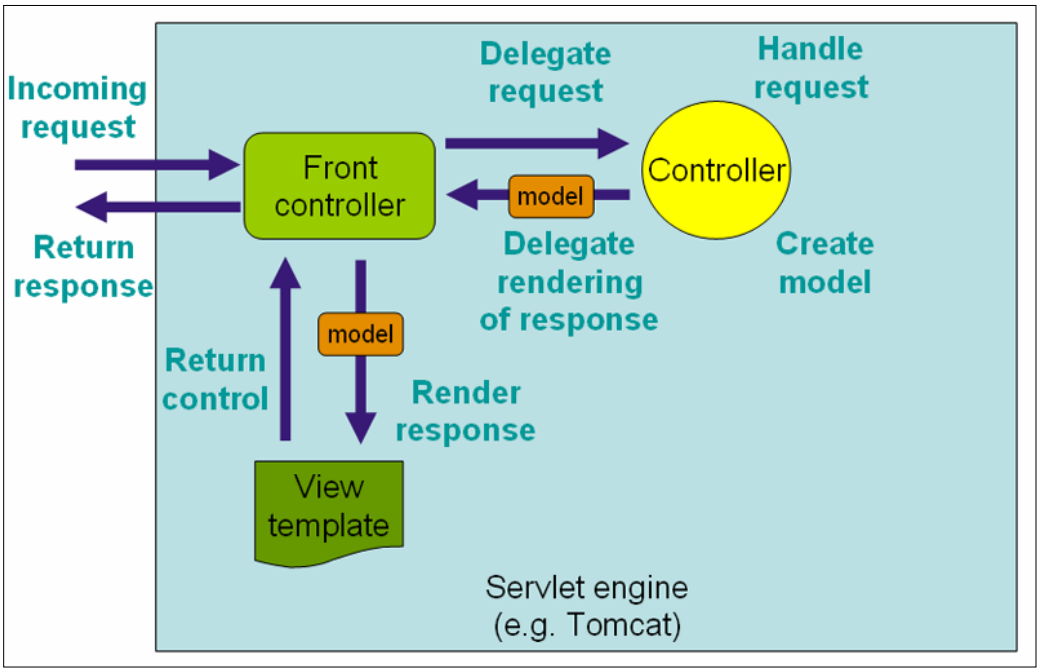


***Figura 2**. Arquitectura MVC de Spring.*

[1] http://static.springsource.org/spring/docs/3.1.0.M1/spring-framework-reference/html/mvc.html#mvc-servlet

Como puede observarse, tras la llegada de una petición, el front controller de Spring transfiere el mando al Controlador para que procese la petición. Después, éste devuelve un modelo, se resuelve una vista a la que se le pasa el modelo y, finalmente, se devuelve la petición.

Según sea la petición, se procederá a seleccionar un controlador adecuado. El contexto de Spring se encarga de instanciarlo e inyectarle aquellas dependencias que se precisen.

2.2. SPRING SECURITY

Spring Security es un proyecto auspiciado por Springsource cuya finalidad es la de proveer de un *servicio completo de seguridad* para proyectos J2EE, principalmente para aquellos basados en Spring Framework. Su funcionamiento se basa en el *encadenamiento de filtros* (FilterChain) que interceptan las peticiones y verifican que las reglas establecidas se cumplen.

Su integración en el TFC busca utilizar las características que ofrece para *autentificación y control de acceso*. La primera de ellas nos permite discernir entre usuarios válidos y no válidos, y la segunda determinar quiénes pueden acceder y a qué áreas.

La configuración e integración del framework de seguridad, se realiza mediante ficheros XML, definiendo beans que serán manejados por Spring.

He empleado la documentación oficial de su dirección web [(2)].

2.3. GOOGLE WEB TOOLKIT

Google Web Toolkit (GWT) es un framework que facilita la *creación de aplicaciones internet ricas* (rich internet applications) sobre JavaScript.

Una característica básica para mí a la hora de desarrollar la aplicación era la de utilizar la *tecnología AJAX en los formularios* para ofrecer una *mejor experiencia de usuario*, eludiendo largas esperas y recargas de páginas innecesarias. Para llevar a cabo esta tarea barajé varias posibilidades, pero finalmente me decanté por el uso de Google Web Toolkit (GWT) y GXT, que más adelante comentaré.

AJAX, cuyas siglas significan JavaScript Asíncrono y XML (Asynchronous JavaScript And XML) es un conjunto de tecnologías que permiten hacer *peticiones asíncronas al servidor*. El modelo clásico de las aplicaciones web funciona mediante el envío de una petición HTTP al servidor, el cual la procesa y retorna una página HTML completa que es recargada en el cliente, el cual, durante todo el proceso se mantiene inactivo. Con AJAX, mientras la petición está siendo procesada por el servidor, el cliente puede continuar respondiendo a las acciones del usuario para que, finalmente, la respuesta sea tratada del modo adecuado, por ejemplo, *mostrando un conjunto de datos en un área concreta de la página sin recargarla por completo*, y manteniendo el estado previo a la recepción de la respuesta. Esto es posible gracias a un motor AJAX incluido en el navegador basado

[(2)] http://static.springsource.org/spring-security/site/docs/3.1.x/reference/springsecurity-single.html

en JavaScript, encargado de la realización de las peticiones y procesado de respuestas por parte del servidor. Por lo tanto, podemos inferir que el corazón de esta tecnología es JavaScript. Precisamente eso es lo que GWT ofrece, un compilador de JavaScript a partir de código Java que incluye una gestión de llamadas asíncronas AJAX.

Si bien en el contexto de la aplicación los controles mostrados serán los renderizados vía GXT, GWT nos ofrece toda la infraestructura sobre la que GXT se apoya. Principalmente nos interesa la capacidad de realizar llamadas asíncronas a servicios expuestos en el servidor. Dichas llamadas se realizan mediante RPC (Remote Procedure Calls), es decir, *llamadas a procedimientos remotos*.

GWT [(3)] ofrece las siguientes funciones:

- Comunicación con el servidor mediante *llamadas RPC*.
- Compilado y optimización de código JavaScript para *distintas versiones de navegadores*.
- Permite *reutilizar componentes* de la interfaz de usuario.

2.4. EXTJS PARA GWT (GXT)

El conjunto de controles para la interfaz de usuario ofrecidos por GWT son bastante reducidos, por lo que decidí utilizar Ext GWT [(4)] (o comúnmente conocido como GXT). GXT dispone de gran cantidad de *controles para la interfaz de usuario* así como multitud de recursos y documentación para su implementación. Al tratarse de un wrapper de GWT, incluye toda la funcionalidad de este, facilitando las llamadas asíncronas mediante RPC.

2.5. GWT SERVER LIBRARY

Las llamadas RPC que realiza GWT requieren que para cada servicio expuesto se añada un servlet. Dejar en manos de Tomcat la administración de servicios provocaba una serie de inconvenientes, como por ejemplo, la gestión de transacciones o la inyección de dependencias. Para poder utilizar todo el potencial de Spring, este acercamiento no era el adecuado, por lo tanto busqué una solución que permitiera exponer los servicios como beans manejados por Spring. La librería GWT Server Library es capaz de *analizar las peticiones RPC y transferir su procesamiento a un controlador de Spring*.

El flujo completo puede ser visto en el siguiente gráfico extraído de la documentación [(5)]:

[(3)] http://code.google.com/intl/es-ES/webtoolkit/overview.html
[(4)] http://www.sencha.com/products/extgwt/
[(5)] http://gwt-widget.sourceforge.net/gwt-sl/reference-1.0/index.html

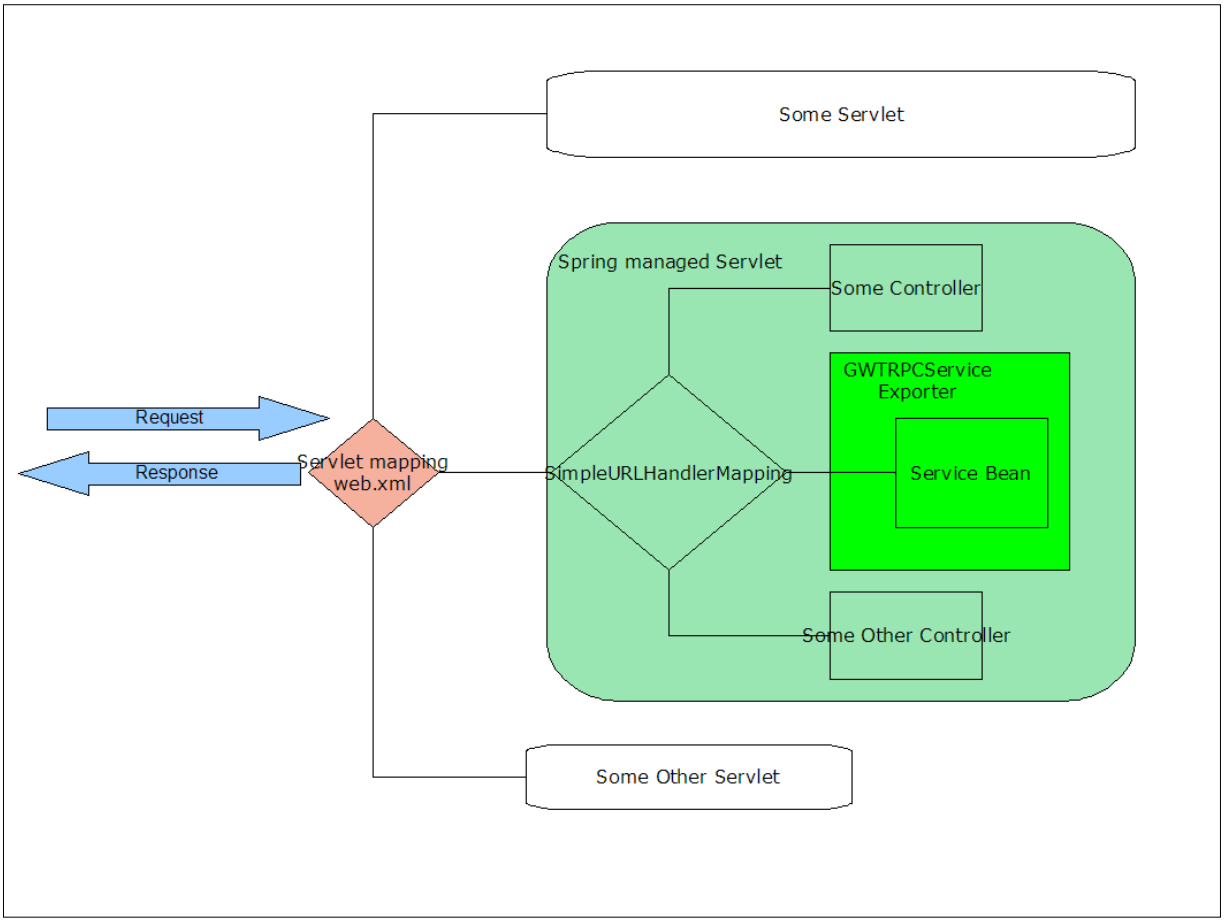


Figura 3. *Diagrama de GWT Server Library.*

A la llegada de una petición RPC, GWT Server Library (SL) es capaz de identificarla como tal y traducirla a una ruta que pueda ser manejada por un controlador de Spring. Una vez que el controlador que ofrece el servicio finaliza, SL se encarga de devolver la respuesta RPC.

Esta aproximación se integra perfectamente con Spring, de modo que los servicios RPC son como cualquier otro controlador al cual se le pueden inyectar dependencias tales como DAO (Data Access Objects) y ejecutar operaciones con la base de datos dentro de una transacción. También *reduce el número de Servlets* y *reduce sensiblemente la complejidad, configuración y mantenimiento de los aspectos relativos a los servicios* que GXT consume.

2.6. JAVA PERSISTENCE API E HIBERNATE

Java Persistence API (JPA) es un estándar incluido en la especificación JSR-220 [(6)] de Enterprise Java Beans 3.0 (EJB), que define la API de persistencia de java. En mi caso he utilizado JPA 2.0, definida en la especificación JSR-317 [(7)].

JPA permite el *mapeo de objetos a modelos relacionales*. Las clases que representan a objetos de la base de datos se llaman entidades y son objetos ligeros y persistentes del dominio de

[(6)] http://www.jcp.org/en/jsr/detail?id=220
[(7)] http://www.jcp.org/en/jsr/detail?id=317

la aplicación. Una de las grandes ventajas de JPA es la *independencia del gestor de bases de datos utilizado*, ya que todas las consultas se realizan con un lenguaje propio, JQL, permitiendo un cambio sencillo a otro gestor en un futuro despliegue. Otra de las ventajas es que se trabaja con POJOs (Plain Old Java Objects), haciendo las *consultas y accesos a la base de datos más naturales* al trabajar directamente con clases que siguen el paradigma de la programación orientada a objetos.

Por su parte, la utilización de Spring da la posibilidad de utilizar un *contenedor de aplicaciones*, Tomcat en este caso, en lugar de un servidor de aplicaciones, como por ejemplo JBoss. Esto significa que no se precisan ciertos aspectos de la especificación J2EE, lográndose un *entorno mucho más ligero y con un menor consumo de memoria*. Como contrapartida, perdemos la posibilidad de dejar al servidor de aplicaciones la administración de la capa de persistencia.

Hay que recordar que la Java Persistence API (JPA) es una especificación dentro de J2EE y los servidores de aplicaciones la implementan. Esto nos obliga a utilizar una implementación de JPA dentro del contenedor de Spring. A este respecto, y tras evaluar otras opciones tales como OpenJPA, EclipseLink o TopLink, me decanté por la implementación que Hibernate [(8)] hace del estándar. La razón de esta decisión se fundamente en que se trata del framework más extendido y documentado, además de haber sido durante mucho tiempo un estándar de-facto, hasta la inclusión de JPA en las especificaciones de J2EE. La versión escogida fue la 3.6.0, última disponible en el momento de la descarga.

Con Spring pude incluir la capa de persistencia, pero fue necesario seleccionar una implementación de Data Source que manejara pools de conexiones. Para esta tarea seleccioné el proyecto C3P0 [(9)], que tras revisar varias alternativas, resultó ser de los más eficientes y configurables en la actualidad.

[(8)] http://www.hibernate.org/
[(9)] http://sourceforge.net/projects/c3p0/

3. ANÁLISIS Y DISEÑO

A lo largo de éste capítulo desarrollaré el análisis de la aplicación, tanto de los aspectos implementados como de los no implementados. En cada caso se hará saber cuál es el estado.

Comenzaré con los *casos de uso* contemplados para luego seguir con el *diagrama de clases* y finalmente incluiré el *esquema de E/R*.

3.1. ANÁLISIS

En este apartado se describen los aspectos relativos al análisis de la aplicación. En primer lugar los actores reconocidos, luego se desglosarán los casos de uso con notación textual, a continuación el diagrama de clases y finalmente el esquema de entidad / relación.

3.1.1. ACTORES

La aplicación reconocerá a los 4 actores descritos a continuación:

- **Administrador:** será aquel usuario con la única responsabilidad de gestionar los usuarios de una aplicación, agregándolos, modificándolos y manteniéndolos dentro de grupos, en caso de ser preciso, además de asignarles roles.

- **Administrador de la Aplicación:** podrá realizar las tareas del administrador pero además, será capaz de gestionar completamente el conjunto de agrupadores, roles, permisos y multiplicadores que estén definidos para una aplicación en concreto.

- **Super Administradores:** deberán ser pocos en número, ya que serán capaces de realizar las tareas previas, y además dar de alta aplicaciones, así como gestionar cualquiera de ellas.

- **Servicio Web:** debe permitir realizar operaciones remotas por parte de los clientes de la gestión de usuarios. Sus casos de uso no heredan de los actores descritos anteriormente al tratarse de funciones propias del servicio dado.

3.1.2. CASOS DE USO

➢ **ADMINISTRADOR**

La aplicación admite cuatro perfiles distintos a la hora de acceder a ella, siendo el primero de ellos el Administrador, actor con la responsabilidad de *gestionar los usuarios de una aplicación cliente, organizándolos por Agrupadores*.

El **diagrama de casos de uso** de este actor es el que sigue:

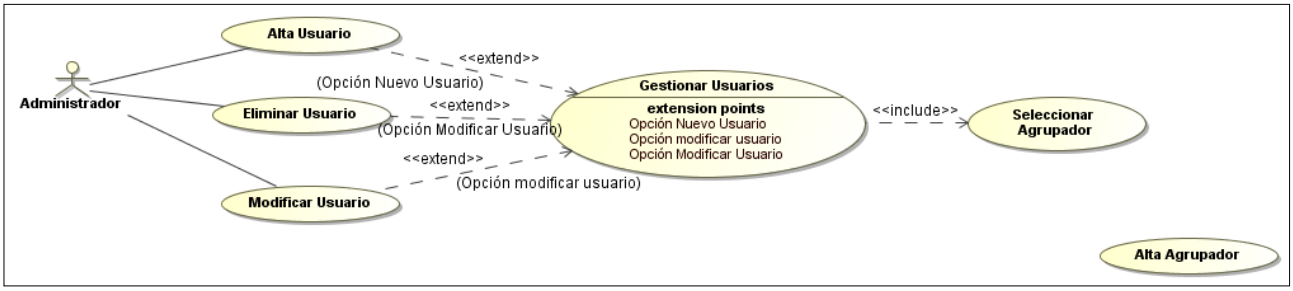


Figura 4. *Diagrama de casos de uso del Administrador.*

Aunque no se ha implementado ningún caso de uso para este actor, a continuación describo cuál debería ser la **funcionalidad** de cada uno de ellos.

Caso de uso: Gestionar Usuarios *(No se implementará)*

Resumen de la funcionalidad:
Permite a los administradores que den de alta, baja y modifiquen a los usuarios de una aplicación determinada.

Actores:
Administrador, Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Alta usuario, Baja usuario, Modificar usuario

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
Ninguna.

Escenario principal de éxito:
1. El administrador selecciona una de las opciones posibles.
2. El administrador realiza la operación actualizándose el conjunto de usuarios de la aplicación.
3. Finaliza el caso de uso.

Alternativas de proceso y excepciones:
No hay alternativas.

Caso de uso: Alta Usuario *(No se implementará)*

Resumen de la funcionalidad:
Permite a los administradores dar de alta un usuario en la aplicación.

Actores:
Administrador, Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de usuarios

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha agregado un usuario a la base de datos.

Escenario principal de éxito:
1. El Administrador pulsa en la Gestión de Usuarios un botón para proceder al alta de un nuevo usuario.
2. Son rellenados todos los campos del formulario.
3. Se validan los campos del formulario.
4. Se pulsa sobre el botón Guardar y se añade el nuevo usuario al sistema.

Alternativas de proceso y excepciones:
1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al usuario.

Caso de uso: Eliminar Usuario *(No implementado)*

Resumen de la funcionalidad:
Permite a los administradores quitar a un usuario de una aplicación.

Actores:
Administrador, Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de usuarios

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador y que la aplicación en la que esté dado de alta se encuentre activa. El usuario a eliminar debe existir en la aplicación.

Postcondición:
El sistema ha quitado al usuario de la aplicación, pero lo mantiene en la base de datos.

Escenario principal de éxito:
1. El administrador selecciona un usuario del listado de usuarios dados de alta en la aplicación y/o agrupador.
2. El sistema desasocia al usuario de la aplicación y desaparece del listado.

Alternativas de proceso y excepciones:
1 Se produce un error en el proceso de actualización.
1.1 El sistema informa al administrador.

Caso de uso: Modificar Usuario *(No implementado)*

Resumen de la funcionalidad:
Permite a los administradores modificar a un usuario existente en la aplicación.

Actores:
Administrador, Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de usuarios

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha agregado un usuario a la base de datos.

Escenario principal de éxito:
1. El Administrador pulsa sobre un usuario desde la Gestión de Usuarios.
2. Se muestra un formulario con los datos del usuario.
3. El Administrador modifica los campos que considera oportunos.
4. El Administrador pulsa sobre el botón Guardar.
5. Se procede a la validación del formulario.
6. Los datos del usuario son actualizados en la base de datos.

Alternativas de proceso y excepciones:
1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al usuario.

- **ADMINISTRADOR DE LA APLICACIÓN**

El Administrador de la aplicación ha de poder realizar las tareas necesarias para *mantener los agrupadores, roles, multiplicadores y permisos de la aplicación*. Además, heredará todas las capacidades asignadas al Administrador.

El siguiente **diagrama** muestra sus **casos de uso**:

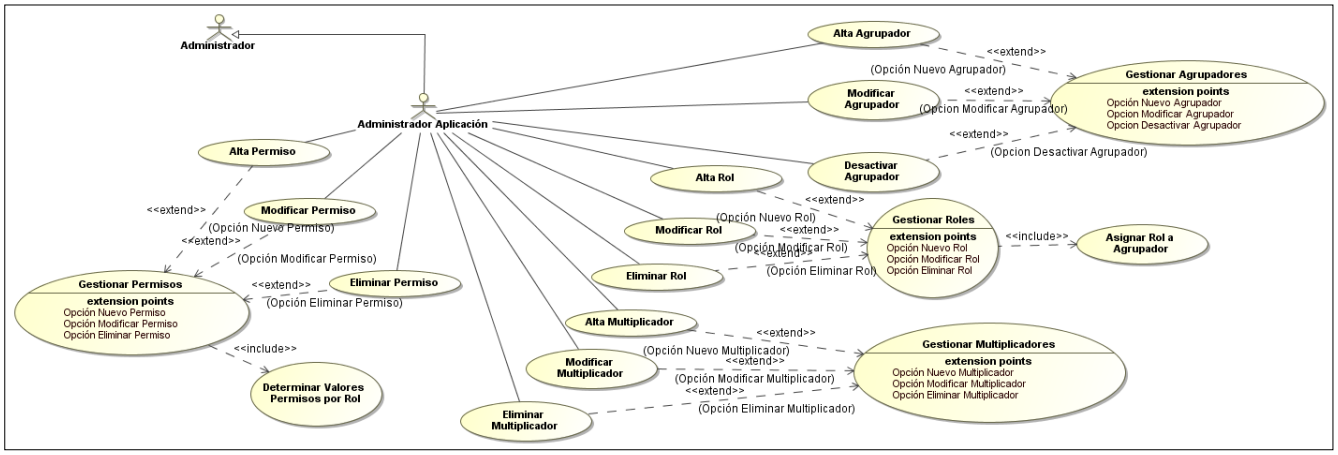


Figura 5. *Diagrama de casos de uso del Administrados de la Aplicación.*

La **funcionalidad** de cada uno de ellos es la siguiente:

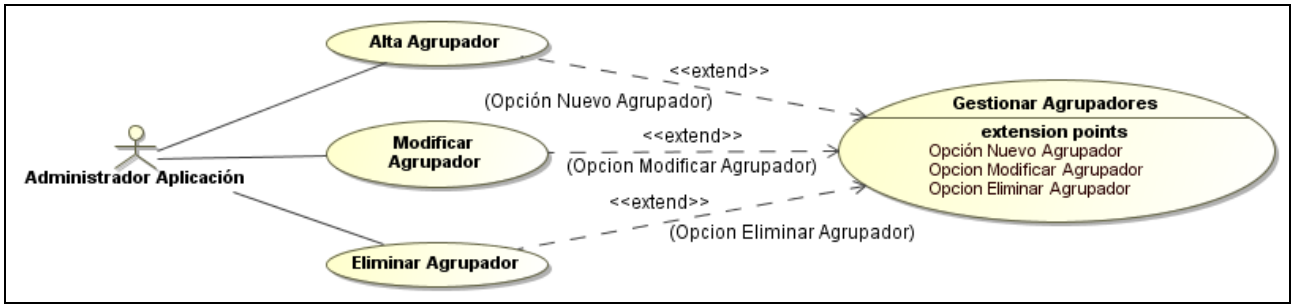


Figura 6. *Detalle de la Gestión de Agrupadores del Administrador de Aplicación.*

Caso de uso: Alta Agrupador
Resumen de la funcionalidad: Permite a los administradores dar de alta un agrupador en la aplicación. **Actores**: Administrador de la aplicación, Super Administrador. **Casos de uso relacionados**: Gestión de agrupadores **Precondición**: Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha agregado un agrupador a la base de datos.

Escenario principal de éxito:
1. El Administrador pulsa en la Gestión de Usuarios un botón para proceder al alta de un nuevo agrupador.
2. El Administrador rellena los campos del formulario pertinente.
3. Se validan los campos del formulario.
4. Se pulsa sobre el botón Guardar y se añade el nuevo agrupador al sistema.

Alternativas de proceso y excepciones:
1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al usuario.

Caso de uso: Modificar Agrupador

Resumen de la funcionalidad:
Permite a los administradores modificar un agrupador existente en la aplicación.

Actores:
Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de agrupadores

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha actualizado un agrupador en la base de datos.

Escenario principal de éxito:
1. El Administrador pulsa sobre un agrupador desde la Gestión de Agrupadores.
2. Se muestra un formulario con los datos del agrupador.
3. El Administrador modifica los campos que considera oportunos.
4. El Administrador pulsa sobre el botón Guardar.
5. Se procede a la validación del formulario.
6. Los datos del agrupador son actualizados en la base de datos.

Alternativas de proceso y excepciones:
1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.

2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al administrador.

Caso de uso: Desactivar Agrupador

Resumen de la funcionalidad:
Permite a los administradores desactivar un agrupador de una aplicación.

Actores:
Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de agrupadores, Eliminar Usuario

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa. El agrupador a desactivar debe existir en la aplicación.

Postcondición:
El sistema ha desactivado el agrupador de la aplicación.

Escenario principal de éxito:
1. El administrador selecciona un agrupador del listado de agrupadores dados de alta en la aplicación.
2. El administrador pulsa sobre el botón Desactivar.
3. El sistema marca como desactivado al agrupador de la aplicación.

Alternativas de proceso y excepciones:
1 Se produce un error en el proceso de actualización.
1.1 El sistema informa al administrador.

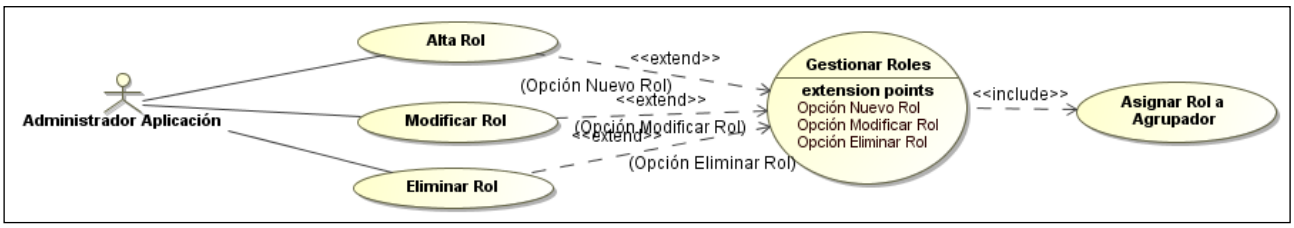


Figura 7. *Detalle de la Gestión de Roles del Administrador de Aplicación.*

Caso de uso: Alta Rol

Resumen de la funcionalidad:
Permite a los administradores dar de alta un nuevo rol en la aplicación.

Actores:
Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de roles

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha agregado un rol a la base de datos.

Escenario principal de éxito:
1. El Administrador pulsa en la Gestión de Usuarios un botón para proceder al alta de un nuevo rol.
2. El Administrador rellena los campos del formulario pertinente.
3. Se validan los campos del formulario.
4. Se pulsa sobre el botón Guardar y se añade el nuevo rol al sistema.

Alternativas de proceso y excepciones:
1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al usuario.

Caso de uso: Modificar Rol

Resumen de la funcionalidad:
Permite a los administradores modificar un rol existente en la aplicación.

Actores:
Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de agrupadores

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:

El sistema ha actualizado un rol en la base de datos.

Escenario principal de éxito:

1. El Administrador pulsa sobre un rol desde la Gestión de Roles.
2. Se muestra un formulario con los datos del rol.
3. El Administrador modifica los campos que considera oportunos.
4. El Administrador pulsa sobre el botón Guardar.
5. Se procede a la validación del formulario.
6. Los datos del rol son actualizados en la base de datos.

Alternativas de proceso y excepciones:

1 El Administrador pulsa sobre Cancelar.
  1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
  2.1 Se muestra información sobre los errores encontrados.
  2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
  3.1 El sistema informa al administrador.

Caso de uso: Eliminar Rol

Resumen de la funcionalidad:

Permite a los administradores quitar un rol de una aplicación.

Actores:

Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:

Gestión de agrupadores, Eliminar Usuario

Precondición:

Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa. El usuario a eliminar debe existir en la aplicación.

Postcondición:

El sistema ha quitado el rol de la aplicación y a todos los usuarios que lo tuvieran asignado.

Escenario principal de éxito:

1. El administrador selecciona un rol del listado de roles dados de alta en la aplicación.
2. El administrador pulsa sobre el botón Eliminar.
3. El sistema solicita confirmación al usuario y le informa de que todos los usuarios que se encuentran dados de alta con dicho rol serán eliminados.
4. El sistema elimina el rol de la base de datos y desaparece del listado.

Alternativas de proceso y excepciones:

1 Se produce un error en el proceso de actualización.
  1.1 El sistema informa al administrador.

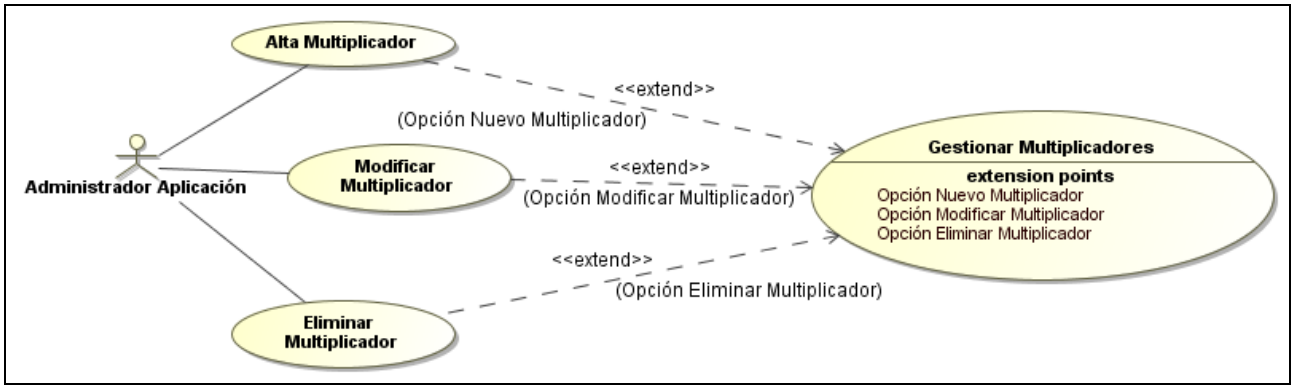


Figura 8. *Detalle de la Gestión de Multiplicadores del Administrador de Aplicación.*

Caso de uso: Alta Multiplicador

Resumen de la funcionalidad:

Permite a los administradores dar de alta un nuevo multiplicador de permisos en la aplicación.

Actores:

Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:

Gestión de multiplicadores

Precondición:

Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:

El sistema ha agregado un multiplicador a la base de datos.

Escenario principal de éxito:

1. El Administrador pulsa en la Gestión de Multiplicadores un botón para proceder al alta de un nuevo multiplicador.
2. El Administrador rellena los campos del formulario pertinente.
3. Se validan los campos del formulario.
4. Se pulsa sobre el botón Guardar y se añade el nuevo multiplicador al sistema.

Alternativas de proceso y excepciones:

1 El Administrador pulsa sobre Cancelar.
  1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
  2.1 Se muestra información sobre los errores encontrados.
  2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
  3.1 El sistema informa al usuario.

Caso de uso: Modificar Multiplicador

Resumen de la funcionalidad:
Permite a los administradores modificar un multiplicador existente en la aplicación.

Actores:
Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de multiplicadores

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha actualizado un multiplicador en la base de datos.

Escenario principal de éxito:
1. El Administrador pulsa sobre un multiplicador desde la Gestión de Multiplicadores.
2. Se muestra un formulario con los datos del multiplicador.
3. El Administrador modifica los campos que considera oportunos.
4. El Administrador pulsa sobre el botón Guardar.
5. Se procede a la validación del formulario.
6. Los datos del multiplicador son actualizados en la base de datos.

Alternativas de proceso y excepciones:
1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al administrador.

Caso de uso: Eliminar Multiplicador

Resumen de la funcionalidad:
Permite a los administradores quitar un multiplicador de una aplicación.

Actores:
Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de multiplicadores, Gestión de permisos.

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y

que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:

El sistema ha quitado el multiplicador de la aplicación y de todos los permisos que lo tuvieran asignado.

Escenario principal de éxito:

1. El administrador selecciona un multiplicador del listado de multiplicadores dados de alta en la aplicación.
2. El administrador pulsa sobre el botón Eliminar.
3. El sistema solicita confirmación al usuario y le informa de que todos los permisos que se encuentran dados de alta con dicho multiplicador serán eliminados.
4. El sistema elimina el multiplicador de la base de datos y desaparece del listado.

Alternativas de proceso y excepciones:

1 Se produce un error en el proceso de actualización.

1.1 El sistema informa al administrador.

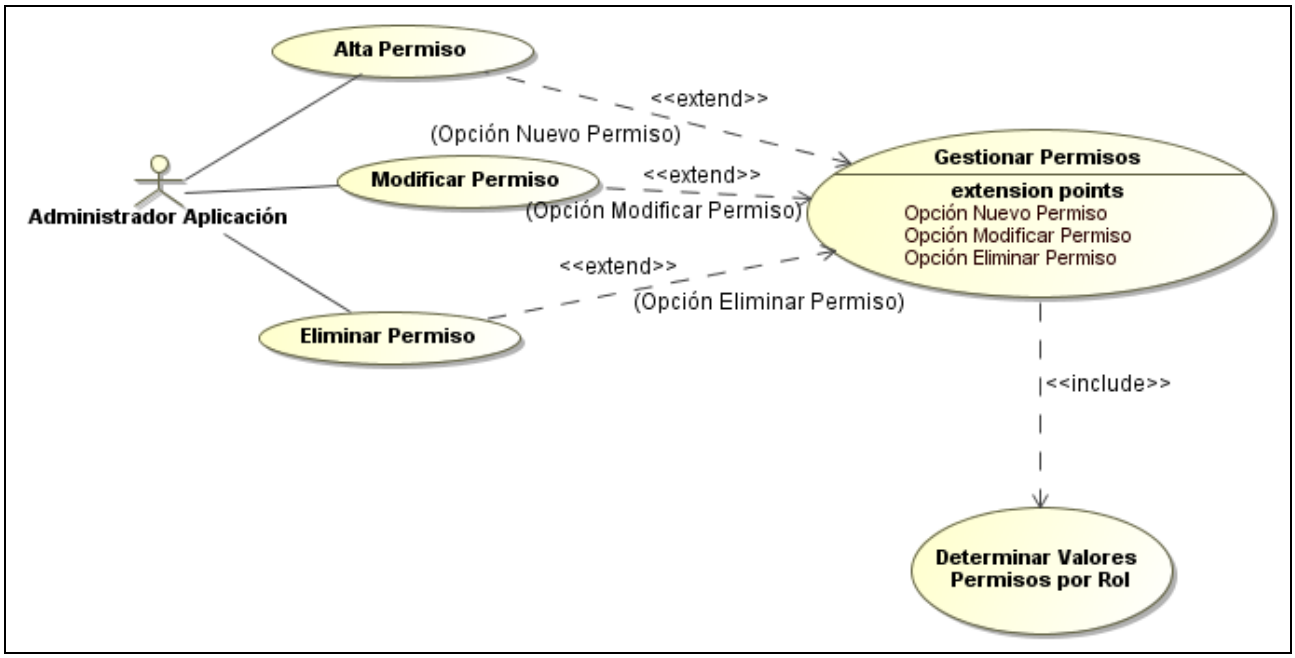


Figura 9. *Detalle de la Gestión de Permisos del Administrador de Aplicación.*

Caso de uso: Alta Permisos

Resumen de la funcionalidad:

Permite a los administradores dar de alta un nuevo permiso en la aplicación.

Actores:

Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:

Gestión de permisos

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha agregado un permiso a la base de datos.

Escenario principal de éxito:
1. El Administrador pulsa en la Gestión de Permisos un botón para proceder al alta de un nuevo permiso.
2. El Administrador rellena los campos del formulario pertinente.
3. Se validan los campos del formulario.
4. Se pulsa sobre el botón Guardar y se añade el nuevo permiso al sistema.

Alternativas de proceso y excepciones:
1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al usuario.

Caso de uso: Modificar Permisos

Resumen de la funcionalidad:
Permite a los administradores modificar un permiso existente en la aplicación.

Actores:
Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de permisos

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha actualizado un permiso en la base de datos.

Escenario principal de éxito:
1. El Administrador pulsa sobre un multiplicador desde la Gestión de permisos.
2. Se muestra un formulario con los datos del permiso.
3. El Administrador modifica los campos que considera oportunos.
4. El Administrador pulsa sobre el botón Guardar.
5. Se procede a la validación del formulario.
6. Los datos del permiso son actualizados en la base de datos.

Alternativas de proceso y excepciones:

1 El Administrador pulsa sobre Cancelar.
  1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
  2.1 Se muestra información sobre los errores encontrados.
  2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
  3.1 El sistema informa al administrador.

Caso de uso: Eliminar Permisos

Resumen de la funcionalidad:
Permite a los administradores quitar un permiso de una aplicación.

Actores:
Administrador de la aplicación, Super Administrador.

Casos de uso relacionados:
Gestión de permisos.

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Administrador de Aplicación y que la aplicación en la que esté dado de alta se encuentre activa.

Postcondición:
El sistema ha quitado el permiso de la aplicación.

Escenario principal de éxito:
1. El administrador selecciona un permiso del listado de permisos dados de alta en la aplicación.
2. El administrador pulsa sobre el botón Eliminar.
3. El sistema solicita confirmación al usuario.
4. El sistema elimina el multiplicador de la base de datos y desaparece del listado.

Alternativas de proceso y excepciones:
1 Se produce un error en el proceso de actualización.
  1.1 El sistema informa al administrador.

- **SUPER ADMINISTRADOR**

Los Super Administradores pueden *dar de alta nuevas aplicaciones*, así como *modificar* cualquier aspecto de las mismas, por lo tanto, van a tener las capacidades descritas anteriormente más las propias.

El **diagrama de caso de uso** de este actor es el que sigue:

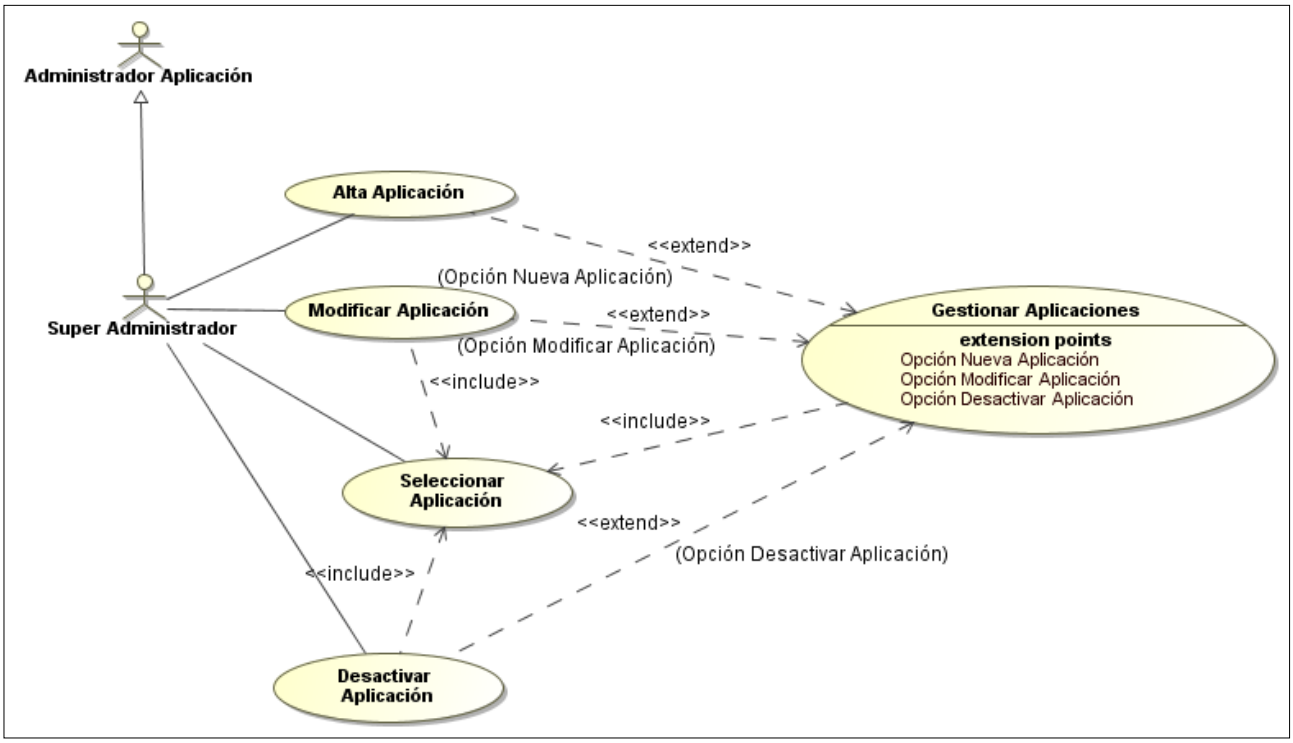


Figura 10. *Diagrama de casos de uso del Super Administrador.*

A continuación describo es la **funcionalidad** de cada uno de ellos.

Caso de uso: Alta Aplicaciones
Resumen de la funcionalidad: Permite a los administradores dar de alta un nuevo permiso en la aplicación. **Actores**: Super Administrador. **Casos de uso relacionados**: Ninguno **Precondición**: Haberse logueado, tener asignado como mínimo el rol de Super Administrador de Aplicación. **Postcondición**: El sistema ha agregado una aplicación a la base de datos. **Escenario principal de éxito**: 1. El Administrador pulsa en la Gestión de Aplicaciones un botón para proceder al alta

de una nueva aplicación.
2. El Administrador rellena los campos del formulario pertinente.
3. Se validan los campos del formulario.
4. Se pulsa sobre el botón Guardar y se añade la nueva aplicación al sistema.

Alternativas de proceso y excepciones:

1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al usuario.

Caso de uso: Modificar Aplicaciones

Resumen de la funcionalidad:
Permite a los administradores modificar una aplicación existente.

Actores:
Super Administrador.

Casos de uso relacionados:
Ninguno.

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Super Administrador.

Postcondición:
El sistema ha actualizado la aplicación en la base de datos.

Escenario principal de éxito:

1. El Administrador pulsa sobre una aplicación desde la Gestión de Aplicaciones.
2. Se muestra un formulario con los datos de la aplicación.
3. El Administrador modifica los campos que considera oportunos.
4. El Administrador pulsa sobre el botón Guardar.
5. Se procede a la validación del formulario.
6. Los datos de la aplicación son actualizados en la base de datos.

Alternativas de proceso y excepciones:

1 El Administrador pulsa sobre Cancelar.
1.1 Se cierra el formulario sin llevar a cabo ninguna operación.
2 Existen errores en la validación del formulario.
2.1 Se muestra información sobre los errores encontrados.
2.2 El sistema espera a que el Administrador rellene correctamente los campos erróneos del formulario.
3 Se produce un error en el proceso de grabación.
3.1 El sistema informa al administrador.

Caso de uso: Desactivar Aplicaciones

Resumen de la funcionalidad:
Permite a los administradores marcar como inactiva una aplicación.

Actores:
Super Administrador.

Casos de uso relacionados:
Ninguno.

Precondición:
Haberse logueado, tener asignado como mínimo el rol de Super Administrador.

Postcondición:
El sistema marca como inactiva la aplicación.

Escenario principal de éxito:
1. El administrador selecciona una aplicación del listado de aplicaciones dadas de alta en la aplicación.
2. El administrador pulsa sobre el botón Desactivar.
3. El sistema solicita confirmación al usuario.
4. El sistema marca como inactiva la aplicación.

Alternativas de proceso y excepciones:
1 Se produce un error en el proceso de actualización.
1.1 El sistema informa al administrador.

- **SERVICIO WEB**

El último de los actores es el Servicio Web que será consumido por las aplicaciones cliente de la gestión de usuarios. Este servicio debe ofrecer un conjunto de métodos suficientes para que *las aplicaciones* que lo consuman *puedan acceder a la información* necesaria para realizar sus tareas en función de los usuarios manejados por la gestión genérica de usuarios. Para la utilización del servicio, debe existir un usuario tipo administrador cuyas credenciales serán transmitidas con la intención de validar al usuario y habilitarle para llevar a cabo las operaciones ofertadas.

Podemos ver el **diagrama** de sus **casos de uso**.

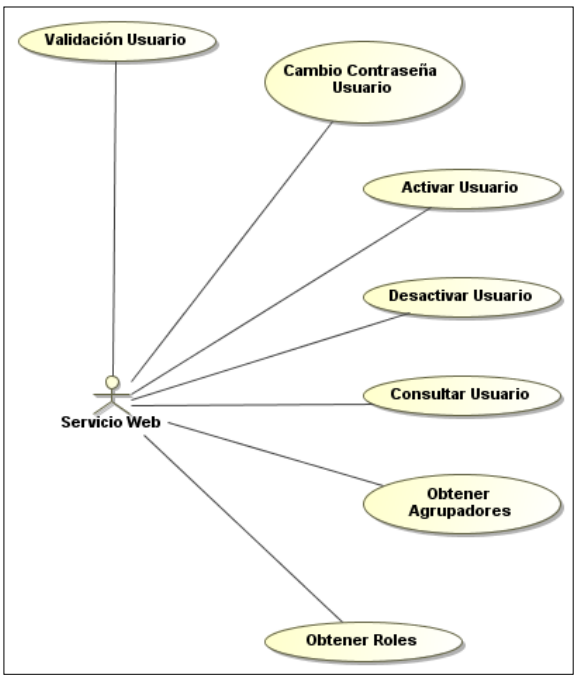


Figura 11. *Diagrama de casos de uso del Servicio Web.*

La **funcionalidad** de cada uno de ellos es la siguiente:

Caso de uso: Validación de Usuario (*No implementado*)

Resumen de la funcionalidad:
Permite a la aplicación cliente verificar las credenciales de un usuario.

Actores:
Servicio Web, Administrador.

Casos de uso relacionados:
Ninguno

Precondición:
Debe existir un Administrador dado de alta en la aplicación a la que pertenece el usuario que deseamos validar.

Postcondición:
El sistema indica si las credenciales del usuario son válidas.

Escenario principal de éxito:
1. El servicio web verifica que las credenciales del usuario administrador que consume el servicio son válidas.
2. El servicio web comprueba las credenciales del usuario que se desea validar perteneciente a una aplicación determinada.
3. El servicio web informa mediante un booleano si el usuario se ha validado con éxito o no.

Alternativas de proceso y excepciones:
1 Las credenciales suministradas del administrador no son válidas para la aplicación.
  1.1 Se lanza una excepción indicando el suceso.
2 La aplicación no existe.
  2.1 Se lanza una excepción indicando el suceso.
3 El usuario a validar está desactivado.
  3.1 Se lanza una excepción indicando el suceso.
4 Se produce un error no controlado.
  4.1 Se lanza una excepción indicando el suceso.

Caso de uso: Cambio Contraseña Usuario (*No implementado*)

Resumen de la funcionalidad:
Permite a la aplicación cliente cambiar la contraseña de un usuario.

Actores:
Servicio Web, Administrador.

Casos de uso relacionados:
Ninguno

Precondición:

Debe existir un Administrador dado de alta en la aplicación a la que pertenece el usuario al que deseamos cambiar la contraseña.

Postcondición:
El sistema actualiza la contraseña del usuario.

Escenario principal de éxito:
1. El servicio web verifica que las credenciales del usuario administrador que consume el servicio son válidas.
2. El servicio web verifica la existencia del usuario al cuál se desea cambiar la contraseña en la aplicación indicada.
3. El servicio web actualiza la contraseña del usuario.

Alternativas de proceso y excepciones:
1 Las credenciales suministradas del administrador no son válidas para la aplicación.
  1.1 Se lanza una excepción indicando el suceso.
2 La aplicación no existe.
  2.1 Se lanza una excepción indicando el suceso.
3 El usuario no existe.
  3.1 Se lanza una excepción indicando el suceso.
4 Se produce un error no controlado.
  4.1 Se lanza una excepción indicando el suceso.

Caso de uso: Activar Usuario (*No implementado*)

Resumen de la funcionalidad:
Permite a la aplicación cliente marcar como activo un usuario de la aplicación.

Actores:
Servicio Web, Administrador.

Casos de uso relacionados:
Ninguno

Precondición:
Debe existir un Administrador dado de alta en la aplicación a la que pertenece el usuario al que deseamos activar.

Postcondición:
El sistema marca como activo al usuario.

Escenario principal de éxito:
1. El servicio web verifica que las credenciales del usuario administrador que consume el servicio son válidas.
2. El servicio web verifica la existencia del usuario el cuál se desea activar.
3. El servicio web activa al usuario.

Alternativas de proceso y excepciones:
1 Las credenciales suministradas del administrador no son válidas para la aplicación.
  1.1 Se lanza una excepción indicando el suceso.
2 La aplicación no existe.

2.1 Se lanza una excepción indicando el suceso.
3 El usuario no existe.
3.1 Se lanza una excepción indicando el suceso.
4 Se produce un error no controlado.
4.1 Se lanza una excepción indicando el suceso.

Caso de uso: Desactivar Usuario (*No implementado*)

Resumen de la funcionalidad:
Permite a la aplicación cliente marcar como inactivo un usuario de la aplicación.

Actores:
Servicio Web, Administrador.

Casos de uso relacionados:
Ninguno

Precondición:
Debe existir un Administrador dado de alta en la aplicación a la que pertenece el usuario al que deseamos desactivar.

Postcondición:
El sistema marca como inactivo al usuario.

Escenario principal de éxito:
1. El servicio web verifica que las credenciales del usuario administrador que consume el servicio son válidas.
2. El servicio web verifica la existencia del usuario el cuál se desea desactivar.
3. El servicio web desactiva el usuario.

Alternativas de proceso y excepciones:
1 Las credenciales suministradas del administrador no son válidas para la aplicación.
1.1 Se lanza una excepción indicando el suceso.
2 La aplicación no existe.
2.1 Se lanza una excepción indicando el suceso.
3 El usuario no existe.
3.1 Se lanza una excepción indicando el suceso.
4 Se produce un error no controlado.
4.1 Se lanza una excepción indicando el suceso.

Caso de uso: Consultar Usuario (*No implementado*)

Resumen de la funcionalidad:
Permite a la aplicación cliente obtener información sobre un usuario.

Actores:
Servicio Web, Administrador.

Casos de uso relacionados:

Ninguno

Precondición:

Debe existir un Administrador dado de alta en la aplicación a la que pertenece el usuario al que deseamos consultar.

Postcondición:

El sistema retorna un objeto que representa al usuario consultado incluyendo información sobre sus roles y permisos.

Escenario principal de éxito:

1. El servicio web verifica que las credenciales del usuario administrador que consume el servicio son válidas.
2. El servicio web intenta obtener los datos del usuario consultado.
3. El servicio web retorna los datos del usuario.

Alternativas de proceso y excepciones:

1 Las credenciales suministradas del administrador no son válidas para la aplicación.
  1.1 Se lanza una excepción indicando el suceso.
2 La aplicación no existe.
  2.1 Se lanza una excepción indicando el suceso.
3 El usuario no existe.
  3.1 Se lanza una excepción indicando el suceso.
4 Se produce un error no controlado.
  4.1 Se lanza una excepción indicando el suceso.

Caso de uso: Obtener Agrupadores (*No implementado*)

Resumen de la funcionalidad:

Permite a la aplicación cliente obtener una colección con todos los agrupadores dados de alta.

Actores:

Servicio Web, Administrador.

Casos de uso relacionados:

Ninguno

Precondición:

Debe existir un Administrador dado de alta en la aplicación a la que pertenecen los agrupadores que deseamos consultar.

Postcondición:

El sistema retorna una colección con todos los agrupadores de la aplicación.

Escenario principal de éxito:

1. El servicio web verifica que las credenciales del usuario administrador que consume el servicio son válidas.
2. El servicio web intenta obtener los agrupadores de la aplicación.
3. El servicio web retorna los agrupadores.

Alternativas de proceso y excepciones:

1 Las credenciales suministradas del administrador no son válidas para la aplicación.
  1.1 Se lanza una excepción indicando el suceso.
2 La aplicación no existe.
  2.1 Se lanza una excepción indicando el suceso.
3 Se produce un error no controlado.
  3.1 Se lanza una excepción indicando el suceso.

Caso de uso: Obtener Roles (*No implementado*)

Resumen de la funcionalidad:
Permite a la aplicación cliente obtener una colección con todos los roles dados de alta.

Actores:
Servicio Web, Administrador.

Casos de uso relacionados:
Ninguno

Precondición:
Debe existir un Administrador dado de alta en la aplicación a la que pertenecen los roles que deseamos consultar.

Postcondición:
El sistema retorna una colección con todos los roles de la aplicación.

Escenario principal de éxito:
1. El servicio web verifica que las credenciales del usuario administrador que consume el servicio son válidas.
2. El servicio web intenta obtener los roles de la aplicación.
3. El servicio web retorna los roles.

Alternativas de proceso y excepciones:
1 Las credenciales suministradas del administrador no son válidas para la aplicación.
  1.1 Se lanza una excepción indicando el suceso.
2 La aplicación no existe.
  2.1 Se lanza una excepción indicando el suceso.
3 Se produce un error no controlado.
  3.1 Se lanza una excepción indicando el suceso.

3.1.3. DIAGRAMA DE CLASES

Las clases identificadas se refieren a aquellas relativas a las *entidades de la base de datos*. El diagrama que las representa es el que sigue:

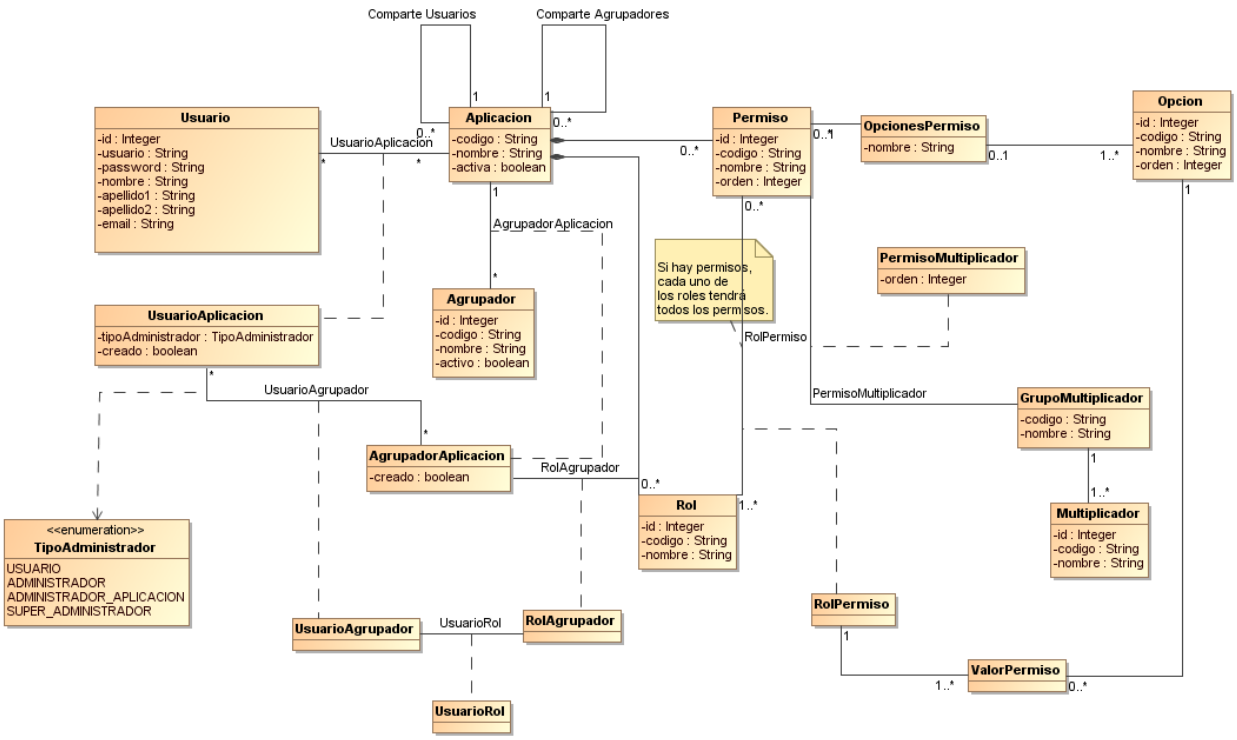


Figura 12. Diagrama de Clases.

3.1.4. DIAGRAMA ENTIDAD-RELACIÓN

El esquema de la base de datos es el siguiente:

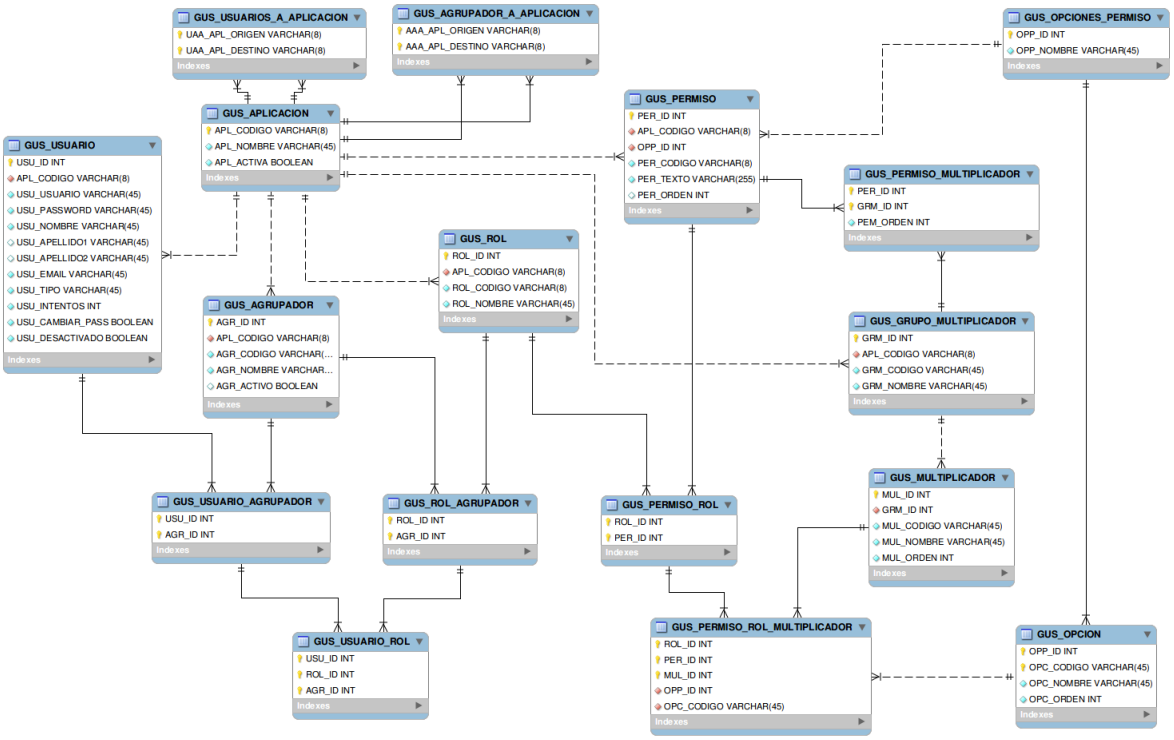


Figura 13. Diagrama Entidad-Relación.

3.2.1. CONFIGURACIÓN DEL CONTENEDOR

La aplicación utiliza el framework de Spring, de modo que es necesario integrarlo en el descriptor de despliegue de la aplicación web. Para realizar esta tarea se ha incluido en el fichero web.xml, que se encuentra en WEB-INF, un listener propio del framework que inicializa el contexto de Spring y un servlet que se encarga de que las peticiones sean transferidas al contenedor de Spring. También se ha añadido soporte para Spring Security, por lo tanto, hay un filtro que se ocupa de *interceptar todas las peticiones web y pasarlas por los distintos filtros de seguridad* antes de permitir su ejecución y procesado.

El **contexto de Spring** se ha definido en un archivo llamado *applicationContext.xml*, en él se definen el conjunto de beans que van a estar disponibles para la aplicación web. Realiza la siguiente configuración:

- Importa el *contexto de seguridad* definido en security.xml (será tratado más adelante).

- Determina que la configuración se realice mediante *anotaciones*.
  En versiones anteriores de Spring era preciso definir en los archivos de contexto xml todos los componentes pertenecientes al framework. Actualmente sólo se precisa que dichos componentes se encuentren anotados con etiquetas propias de Spring. Al inicializar el contexto, recorrerá las clases de la aplicación en busca de estas etiquetas y se auto-configurará.

- Definición del bean "dataSource". Este bean representa una *conexión con la base de datos* mediante una implementación de javax.sql.DataSource. La implementación seleccionada es la ofrecida por la librería C3P0, que incluye un pool de conexiones.

- Definición del bean "entityManagerFactory". Tal y como su nombre indica se trata de una *factoría de Entity Manager*. La implementación del factory corre a cuenta de Spring. Entre los parámetros de configuración de este bean indicamos la implementación de JPA que deseamos utilizar, en este caso, Hibernate.

- Definición del bean "transactionManager" mediante la instanciación de una clase suministrada por spring. Las *transacciones* serán manejadas mediante anotaciones en los DAO y en las clases de servicio.

- Por último, existe un conjunto de beans que son los DAO. Son las siglas de "Data Access Object" u *Objetos de Acceso a Datos*. Implementan la funcionalidad de acceso a la base de datos y serán inyectados como dependencias en las clases de servicio que los consumirán.

El archivo de configuración *security.xml* importando en applicationContext.xml se ocupa de definir **los bean de Spring Security**. La seguridad no se ha implementado completamente a causa de no existir mantenimiento de usuarios, en cambio sí se han sentado las bases para determinar cuáles son las páginas accesibles de forma anónima, cuales son aquellas que han de ser sólo usuarios autenticados los que tengan acceso a ellas,

y cuál es el sistema de login utilizado. Por lo tanto, la configuración es la que sigue:

- Definición del bean "loginUrlAuthenticationEntryPoint" que *identifica la URL de la página de login*, "login.htm".

- Definición del bean "successHandler" que *indica a qué URL se procederá* en el caso de una *autentificación con éxito*, "/private/application.htm".

- Definición del bean "failureHandler", que indica la *URL en caso de error de autentificación*, "/login.htm?login_error=1".

- Definición del bean "authenticatorManager", que determina cuál es el *método de autentificación*. Al no existir gestión de usuarios, se ha creado una lista en memoria que incluye un único usuario llamado "admin" con password "admin".

- Bean "loginUrlAuthenticationEntryPoint", que lleva a cabo la *configuración de las rutas accesibles* de forma anónima y aquellas que necesitan como mínimo un rol para poder acceder a ellas. En principio, se ha establecido que la ruta "/login.htm" sea pública, mientras que las rutas "/private/*.htm" son privadas y el usuario que desee acceder debe estar autentificado.

El servlet incluido en el web.xml se ha mapeado para todas las peticiones *.htm y *.do. Las primeras de ellas responderán devolviendo vistas, mientras que las segundas son las extensiones para las peticiones RPC de GXT. El servlet se llama *userAdmin*, y la clase que lo implementa es el ***DispatcherServlet*** de spring. Es por ello que spring espera encontrar un fichero de configuración llamado userAdmin-servlet.xml en WEB-INF. Dicho archivo configura las peticiones y contiene la definición de los siguientes beans:

- En primer lugar se configura como annotation-driven el manejo del modelo MVC. De este modo, mediante etiquetas propias del framework, *se configurará la aplicación*.

- Se define el bean "messageSource" para indicar el archivo .properties que contendrá los *mensajes internacionalizados* de la aplicación.

- Se define el bean "localeChangeInterceptor" que implementa una clase que revisará la url y buscará en ella un parámetro que indique el *cambio de locale*, y por lo tanto, el *cambio del idioma* de la página. Este parámetro se ha establecido como "lang".

- Se define el bean "localeResolver" que determina cuál es el *locale por defecto*, "es" y el método de descubrimiento del mismo. La implementación seleccionada se aprovecha de las cookies en el cliente para determinarlo.

- Al utilizar spring no disponemos de servlets con un mapeo para *relacionar URLs con controladores que procesen las peticiones*. Esta tarea se lleva a cabo mediante el mapeo en los ficheros de configuración de spring. Por lo tanto es preciso definir un bean, llamado en este caso "handlerMapping", que se encargue de descubrir, en función de la petición, qué controlador debe invocarse. Este bean tiene un atributo que será el bean "localeChangeInterceptor" que buscará en la url el parámetro de cambio de locale. Por su

parte, el mapeo se llevará a cabo mediante anotaciones dentro de los mismos controladores.

- Si bien los controladores son descubiertos mediante el bean anterior, las *vistas del model* MVC lo serán a través de los beans "InternalViewResolver" y "urlViewResolver". Estos beans intentarán, mediante dos estrategias distintas, resolver la vista a renderizar tras la ejecución del controlador.

- Por último, se configuran dos beans no tienen id identificativo, pero es necesario instanciarlos para *permitir la utilización de la GWT Server Library*. Especialmente el bean que instancia la clase GWTHandler, será el encargado de traducir las peticiones RPC al modelo de spring para ser procesado por un componente anotado de este.

3.2.2. CONFIGURACIÓN DE GWT Y GXT

GWT (Google Web Toolkit) es un framework desarrollado por Google que permite la *utilización de JavaScript mediante la implementación de código Java*. Este código es posteriormente compilado por el propio framework y optimizado para el conjunto de navegadores definidos.

Una de las dificultades al trabajar con GWT es que en realidad no se está escribiendo código Java, todo el código que implementemos salvo aquellas partes pertenecientes a los servicios RPC, serán ejecutados en el navegador del cliente tras su compilación como JavaScript. Por lo tanto nos encontramos con una serie de limitaciones, tales como el número de clases nativas a las que podemos acceder, ya que deben tener su representación en JavaScript, o qué clases y/o paquetes estarán disponibles desde la aplicación cliente. Para permitir definir estas cuestiones y muchas otras, GWT precisa de al menos un fichero de configuración, o módulo, que indique cuál es el *paquete a partir del cual todas sus clases serán traducidas a JavaScript*.

En esta aplicación se ha creado un único módulo en el **paquete info.wikipau.useradmin**, que por requerimiento de la API de GWT, ha de llamarse userAdmin.gwt.xml. En este archivo los aspectos más importantes a tener en cuenta son:

- *Rutas para el código traducido*. Estas rutas hacen referencia a los paquetes que deberán ser convertidos a JavaScript. Por defecto se incluyen "client", "server" y "shared". Es decir, si la ruta de nuestro módulo es "info.wikipau.useradmin", el paquete "client" se encontrará en "info.wikipau.useradmin.client", etc. El paquete "client" contiene todas las clases que se convertirán en el código JavaScript que se ejecutará en el cliente. En el "server" estarán las definiciones de los servicios RPC, mientras que su implementación se encontrará fuera del módulo. Finalmente, "shared" contiene todos los objetos DTO que van a ser utilizados a lo largo de la aplicación, principalmente para la conversión de entidades a objetos entendibles por GWT.

- *Punto de entradas* (Entry Point). El punto de entrada es la clase que ha de ejecutarse en un módulo. Cuando una página que contiene código GWT lanza el JavaScript, lo hará a partir de la clase definida como Entry Point.
  Nuestra clase es "info.wikipau.useradmin.client.UserAdminEntryPoint".

- *Clases incluidas*. Se trata de las clases que son importadas y se encuentran disponibles para el módulo en cuestión. Aquí es muy importante reseñar que se han importado, o heredado (según la nomenclatura dada por GWT), las clases pertenecientes GXT. Estas clases implementan los controles que serán utilizados en las diferentes páginas de la aplicación.

- Por último, un dato importante en la configuración es el llamado "user.agent". Nos permite decidir *para qué navegadores* se compilarán los componentes GWT. A mayor número de clientes, más tiempo de compilación, por lo que durante el proceso de desarrollo se ha manejado un único cliente, mientras que el producto resultante está preparado para su ejecución en los navegadores de Internet Explorer desde la versión 6, Firefox, Google Chrome y Safari.

Por su parte, GXT es una extensión de GWT que genera *código JavaScript perteneciente a la librería ExtJS*. Al tratarse de una extensión incluye todas las características de GWT y ampliándolas con controles propios. Estos controles ofrecen funcionalidades más avanzadas que las incluidas por defecto en la implementación de Google, lo que facilita la creación de entornos más funcionales y vistosos, repercutiendo directamente en la experiencia del usuario. GXT no necesita de configuración extra para ser utilizado, únicamente ha de ser importado en la definición del módulo.

3.2.3. MODELO, VISTA, CONTROLADOR

El framework de Spring facilita enormemente la *utilización del modelo MVC* (Modelo, Vista, Controlador). El MVC es un patrón de diseño que propone la separación de las siguientes **capas**:

- **Modelo**: es el conjunto de *información* que la aplicación maneja. Este conjunto incluye tanto las entidades de la base de datos como la capa de negocio.
- **Vista**: hace referencia normalmente al *interface de usuario* y representa el modelo sobre el que el usuario puede interactuar.
- **Controlador**: se corresponde con la *lógica* de la aplicación, y tiene la responsabilidad de responder a eventos y preparar el modelo para su visualización.

Para implementar este modelo, Spring ofrece un "Front Controller" que se encarga de analizar las peticiones y distribuirlas al controlador correspondiente. Además, incluye una serie de clases que permiten la resolución de la vista a devolver. En esta aplicación, se ha respetado este modelo, manteniendo separadas las responsabilidades entre los distintos elementos de este patrón, los cuales se explican a continuación.

- **MODELO**

El modelo está dividido en dos partes. La primera de ellas se trata de las *entidades de la base de datos* que pueden encontrarse en el paquete "info.wikipau.model.entity". Estas entidades son objetos relacionales, mapeados para que puedan ser manejados por el API de persistencia de Java (JPA), pero *no son adecuadas para su manejo desde GWT* por varias razones:

- Se encuentran fuera del ámbito de clases que GWT puede transformar en objetos

JavaScript.
- Debido a que son un mapeo de objetos relacionales, es posible que encontremos un cierto grado de recursividad en ellas lo que provocaría errores e incluso dificultades a la hora de convertirlas y/o manejarlas como JavaScript.
- GWT, y por lo tanto GXT, necesitan que las clases que representan los objetos de la capa de negocio extiendan o implementen clases específicas y propias del framework de visualización (dependiendo del método utilizado a la hora de trabajar con ellas en la vista).

Para solucionar los aspectos anteriores, en segundo lugar se han creado un *conjunto de clases DTO* (Data Transfer Objects) pensadas *para interactuar con GXT*. Estas clases implementan el modelo que la infraestructura de GXT requiere en cada caso, permitiendo el uso de las mismas en los controles que posteriormente serán construidos. Además, se incluyen en el ámbito de las clases reconocidas por GWT, siendo transformados por el compilador a JavaScript. Los DTO pueden encontrarse en el paquete "info.wikipau.shared".

Los DTO son una representación más sencilla y manejable de las entidades de la base de datos, por lo que, para su instanciación se ha recurrido a la utilización de un conjunto de factorías que a partir de una entidad devuelven su correspondiente DTO. Estas "Factories" son manejadas por los controladores que responden a las llamadas RPC, por lo tanto están fuera del contexto GWT y se encuentran en el paquete "info.wikipau.model.dto".

- **VISTA**

La vista es una *combinación de documentos JSP*, cuya resolución corre a cargo de Spring mediante los "View Resolvers" configurados en el "Front Controller", y los componentes GXT que se incrustan en dichos JSP.

Las páginas JSP se encuentran dentro de WEB-INF/jsp. Aquí puede verse como, por un lado tenemos la página pública "login", y por otro, la parte privada incluida en la carpeta "private". Recordemos que durante la definición del contexto de seguridad con Spring Security, habíamos indicado que sólo existiría una página accesible por visitantes anónimos, "login", mientras que todas las demás, cuyo destino sea */private/* sólo pueden ser visitadas por usuarios registrados.

Salvo la página de acceso, todas incluyen contenido GXT, por lo que en ellas será posible encontrar una etiqueta como la que a continuación se indica:

```
<script type="text/javascript" language="javascript"
src="../useradmin/useradmin.nocache.js"></script>
```

El código anterior incorpora, a la página JSP, el módulo renderizado de GXT. Dicho módulo, una vez compilado, se despliega en la carpeta "useradmin" en el directorio raíz del despliegue. Esta carpeta contiene todos los archivos JavaScript que GWT precisa para mostrar los controles y realizar las llamadas a los servicios RPC. Como nota interesante a esta implementación, cabe destacar que se ha desarrollado un único "Entry Point", es decir, un único módulo que deberá ser capaz de mostrar los distintos controles GXT de todas las páginas. Hacerlo así, simplifica enormemente el desarrollo al no tener que crear un nuevo

modulo por cada control GXT mostrado.

Este enfoque necesita, por lo tanto, indicar qué código JavaScript ha de ser incrustado en cada JSP, lo cual se consigue de la manera siguiente:

```
<script type="text/javascript">
     var accion = {
          op : 'application'
     };
...
</script>
```

Se define un objeto JavaScript con la propiedad op. Desde el punto de entrada podemos acceder al objeto "accion" y consultar el valor de la propiedad "op". En ese momento ya somos capaces de escoger entre las distintas clases JavaScript para llamarla y generar la vista correspondiente. Si se desea puede consultarse el código de la clase "info.wikipau.useradmin.client.UserAdminEntryPoint", para ver de qué modo se puede llamar a un componente o a otro.

Una vez que se ha determinado el componente a mostrar, se instancia la clase correspondiente al mismo, que se encontrará en el paquete "info.wikipau.useradmin.client". Estas clases extienden GXT y utilizan los controles del framework. Todo el código escrito en ellas es traducido a JavaScript para que el navegador lo interprete. Aquí también se realizan llamadas a los servicios RPC que dan funcionalidad a la página mediante eventos asíncronos, lo que se traduce en llamadas AJAX al servidor.

- **CONTROLADOR**

El controlador *ejecuta la lógica de la aplicación* manipulando las vistas y el modelo para, finalmente devolver un interface al usuario. Este apartado lo dividiré en dos partes, la primera de ellas tratará de los controladores que prepararán las páginas JSP, mientras que la segunda hablará de los controladores responsables de responder las peticiones RPC .

CONTROLADORES JSP

Cada vez que se realiza la petición de una página JSP, Spring busca entre los controladores anotados con la etiqueta @Controller, y que se encuentran en el paquete "info.wikipau.controller", aquél cuyo mapeo se corresponda con la página solicitada.

Pongamos de ejemplo el siguiente controlador:

```
@Controller
@RequestMapping(value="/private/application")
public class ApplicationController {
     @RequestMapping(method=RequestMethod.GET)
     public String entradaHandler(HttpServletRequest request,
Model model) {
          ...
     }
     @RequestMapping(method=RequestMethod.POST)
     public void formSubmit(HttpServletRequest request,
HttpServletResponse response) throws IOException {
```

```
            ...
        }
    }
```

Se trata del controlador de la página “application.jsp”. En él puede observarse que la página está mapeada mediante la anotación @RequestMapping a la URL “/private/application”.

Cuando intentemos acceder a “http://SERVIDOR/private/application.htm”, Spring ejecutará esta clase. El método que llamará será aquél que se corresponda con el tipo de “Request”. Si se realiza una petición GET, será el método “entradaHandler” el responsable de su procesado, mientras que para POST, se ejecutará el método formSubmit.

La documentación de Spring es muy extensa sobre el formato que cada uno de estos métodos puede tener, permitiendo diversos atributos dependiendo de nuestras necesidades en cada momento. Habitualmente nos encontraremos con que se retorna un objeto de tipo String. Este objeto será la referencia a una vista cuya resolución estará a cargo del “Front Controller” mediante sus “View Resolvers”.

Tal y como se ha configurado, lo que sucederá es que si la cadena devuelta es “application” (por escoger un ejemplo), buscará la página jsp en “WEB-INF/jsp/private/application.jsp”, luego renderizará la vista con el modelo suministrado (en este caso de tipo Model, que no es más que un HashMap), y lo devolverá al cliente.

CONTROLADORES RPC

Los servicios RPC que GWT utiliza, son normalmente implementados mediante un servlet por servicio, pudiendo complicar tanto la configuración como el mantenimiento, por lo que se ha decidido incluir la librería GWT Server Library que traduce llamadas RPC a controladores de Spring y viceversa.

GWT, para implementar los servicios, necesita que se defina por un lado un interface para la llamada asíncrona, dónde se determina el tipo de dato devuelto, y por otro el interface de la llamada al servicio, qué será el implementado. Estos dos interfaces han de encontrarse dentro del contexto de GWT, de modo que están incluidos en el paquete “info.wikipau.useradmin.server”, en cambio, la implementación puede estar fuera de dicho contexto. Esto es muy útil, ya que al colocarlo fuera del contexto de GWT, el servicio tiene acceso a todas las clases del proyecto, sin las limitaciones que GWT impone a este respecto.

Una vez que definimos los interfaces, es hora de decidir qué peticiones atenderá dicho servicio. Como pretendemos utilizar la librería comentada anteriormente, hemos tenido que realizar un par de pasos previos. Estos pasos son:

- Indicar qué peticiones vamos a considerar llamadas a los servicios. Lo hacemos al configurar el descriptor de despliegue web.xml agregando en el servlet de Spring el mapeo *.do.
- Habilitar en la configuración del servlet de Spring los siguientes beans:

```
<bean class =
"org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapte
r" />
<bean class="org.gwtwidgets.server.spring.GWTHandler" />
<context:component-scan base-package="info.wikipau"/>
<context:annotation-config />
```

Los dos primeros beans configuran la librería GWT Server Library, siendo el más importante el segundo de ellos, GWTHandler, que se encargará de transferir las llamadas al componente adecuado.

Los dos últimos beans, habilitan la búsqueda en las clases de anotaciones. El interface del servicio, por su parte, deberá incluir un mapeo para que sea posible identificar qué solicitudes atender.

Para ilustrar este caso introduciré un ejemplo:

```
@GWTRequestMapping("/useradmin/async/application.do")
@RemoteServiceRelativePath("async/application.do")
public interface ApplicationFormService extends RemoteService {
     // FIRMA DE LOS MÉTODOS DEL INTERFACE.
     public static class Util {
          private static ApplicationFormServiceAsync instance;
          public static ApplicationFormServiceAsync getInstance(){
               if (instance == null) {
                    instance =
GWT.create(ApplicationFormService.class);
               }
               return instance;
          }
     }
}
```

Los servicios RPC incorporan la anotación @RemoteServiceRelativePath para indicar la ruta relativa, a partir de ../useradmin, de las peticiones que atenderá el servlet asociado. Por desgracia no es suficiente a la hora de integrarlo con Spring, por lo que la librería GWT Server Library da la posibilidad de incluir la anotación @GWTRequestMapping. Por esa razón, en esta anotación escribiremos la ruta absoluta, no relativa, de la petición RPC a atender. Cuando se reciba un “Request” a la ruta del ejemplo, se transferirá el control al componente que implemente este interface. Esta clase deberá ser un @Controller o un @Component. En realidad @Controller es un tipo de componente, así que es indiferente.

Como puede verse, esta aproximación es extremadamente ágil ya que facilita la continuidad y la integración en el esquema de Spring al seguir su filosofía.

Para terminar, la implementación del servicio anterior sería algo así:

```
@Component
public class ApplicationFormServiceImpl implements
ApplicationFormService {
     @Autowired
     ApplicationDao applicationDao;
     private @Autowired HttpServletRequest request;
     // IMPLEMENTACIÓN DEL INTERFACE.
```

```
}
```

Las implementaciones de los servicios pueden encontrarse en el paquete "info.wikipau.component".

3.2.4. INYECCIÓN DE DEPENDENCIAS (DI)

Tanto los controladores de las páginas JSP, como los componentes de los servicios, necesitan ciertos objetos para realizar sus tareas. En la mayoría de los casos se deberá proveer de *acceso a los beans de la capa de persistencia*, o DAO's (Data Access Objects). Con Spring 3 esta tarea resulta trivial. Sólo se precisa configurar los beans que deseamos poder inyectar en el archivo de contexto de Spring (applicationContext.xml) e incluirlos como propiedades en las clases que vayan a consumirlos. Estas clases deben ser manejadas por el contenedor de Spring, como por ejemplo los controladores. Es aconsejable que los atributos a inyectar hagan referencia al interface en lugar de a la implementación. Por último, etiquetamos estas propiedades con la etiqueta @Autowired, y la próxima vez que se entregue el control al componente, se le inyectarán las dependencias anotadas.

Puede verse un ejemplo en el apartado anterior, dónde a la implementación del servicio se le inyecta un objeto de tipo ApplicationDao, instanciado por el contenedor de Spring.

3.2.5. CAPA DE PERSISTENCIA

Para la capa de persistencia se ha utilizado JPA 2.0 (Java Persistence API). JPA es una especificación que *facilita el mapeo de tablas relacionales de una base de datos con clases Java*, conocidas como entidades o POJO's (Plain Old Java Objects).

Tratándose de una especificación ha sido necesario seleccionar una implementación de la misma. He valorado las siguientes opciones:

- OpenJPA.
- EclipseLink.
- Oracle Toplink.
- Hibernate.

La decisión se ha tomado en función de los siguientes parámetros:

- Sencillez de integración.
- Documentación disponible.
- Rendimiento.

Tras probar todas las implementaciones anteriores, escogí Hibernate por integrarse muy fácilmente con Spring, además de la existencia de gran cantidad de código y ejemplos en la red, así como por ser una de las más eficientes en sus últimas versiones.

La persistencia con Spring ha sido definida en el contexto de la aplicación mediante la definición de un DataSource. Con el bean dataSource se crea un objeto de tipo EntityManagerFactory para que sea el propio contenedor de Spring quien se ocupe de inyectar el EntityManager necesario en los objetos de acceso a datos (DAO's).

Estos objetos siguen un patrón AbstractPojo que facilita la reutilización de la funcionalidad básica de la capa de persistencia. Para ello es preciso que las entidades extiendan de la clase AbstractPojo. Los DAO's implementarán la interface GenericDao<T extends AbstractPojo<?>,L>. Esta interface está parametrizada con los AbstractPojo y un tipo genérico L que representa el tipo de dato del identificador de la clase. Por ejemplo, si una clase A tiene un id de tipo Integer, GenericDao estaría parametrizado como GenericDao<A,Integer>.

Finalmente, la implementación de GenericDao incluye los métodos básicos y comunes a todos los managers de las entidades, tales como grabar, actualizar y borrar. Cada uno de los DAO extenderá este GenericDao y las operaciones comunes se realizarán en función del tipo de identificador definido a la hora de instanciar la clase.

Los distintos DAO son instanciados por Spring en el contexto de la aplicación y pueden encontrarse los interfaces en el paquete info.wikipau.dao, así como en info.wikipau.dao.impl las implementaciones de los mismos. Por su parte, las entidades se encuentran en el paquete info.wikipau.model.entity.

4. FUNCIONALIDAD

En este apartado se va a describir la *funcionalidad implementada* de la aplicación. La Gestión de Usuarios es un *almacén de usuarios y permisos asociados*. El *conjunto de datos* que maneja será *consumido por aplicaciones externas mediante un servicio web*.

Cada una de las aplicaciones externas tendrá un conjunto de usuarios agrupados, por ejemplo, en centros de trabajo. A cada usuario se le ha de asignar un rol de entre los que el centro al que pertenece puede otorgar. Opcionalmente, los roles pueden tener asignados permisos atómicos del tipo Sí/No o Lectura/Escritura/Lectura-Escritura (R/W/RW). En el caso de tener permisos, todos los roles compartirán los mismos, pudiéndose además añadir multiplicadores a cada uno de ellos.

Para entender lo que es un multiplicador imaginemos que un proceso pasa por varios estados, por ejemplo, durante el proceso de compra en una tienda on-line vamos a encontrarnos en distintos momentos como son, la fase de selección de artículos durante la cual se irán añadiendo productos al "carrito", la fase de realización del pedido en dónde aún no hemos abonado el producto, la fase de abono sin que se haya confirmado el ingreso, y el envío de los productos. Todas estas fases corresponden a estados del proceso y cada permiso podría tomar un valor diferente para cada uno de ellos. Como muestra, el permiso "PAGAR", existirá en todas las fases pero a partir del estado "PAGADO" ya no deberemos permitir hacerlo. El multiplicador "multiplica" cada uno de los permisos por el número de estados que tenga asociados.

A continuación, y mediante la utilización de capturas de pantalla, introduciré la funcionalidad implementada ampliando los conceptos anteriores.

4.1. ACCESO

Inicialmente se muestra la pantalla de acceso. Dicha pantalla es accesible mediante la ruta *http://HOST:PUERTO/default/login.htm*. La Gestión de Usuarios, al ser un almacén de datos de usuarios y permisos, *determina la aplicación que deseamos gestionar mediante la URL*, por lo que podemos acceder a cualquiera de las que estén dadas de alta utilizando el código de la misma en la dirección web del navegador. Por ejemplo, la aplicación por defecto se llama default. Si queremos pasar a otra, como la aplicación llamada uoc (incluida), para poder gestionarla sólo tendríamos que ir a la url *http://HOST:PUERTO/uoc/login.htm*. En ambos casos, aparecerá una pantalla como la que a continuación se muestra, dónde podemos loguearnos. Ahora mismo el acceso es libre, sólo es preciso pulsar sobre el botón Acceder.

Para corroborar nuestra elección, la pantalla de login *mostrará el nombre de la aplicación que vamos a administrar* debajo del título Gestión de Usuarios, en el caso de la imagen, habremos accedido a la aplicación *uoc*.

Figura 14. *Pantalla de Acceso.*

4.2. APLICACIONES

Al entrar se mostrará directamente la pantalla de Aplicación, dónde podemos *dar de alta* aquellas *aplicaciones que deseamos administrar*. Para dar de alta una nueva aplicación, deberemos pulsar sobre el botón nuevo, que limpiará el formulario. Luego, en "Código", introduciremos una designación para la aplicación de hasta 8 caracteres, ésta será la que la identifique y la que usaremos en la url para acceder a ella. En "Nombre" escribiremos la denominación descriptiva de la misma. Por último, hacemos click en "Guardar" y se habrá agregado, apareciendo en el grid de la izquierda.

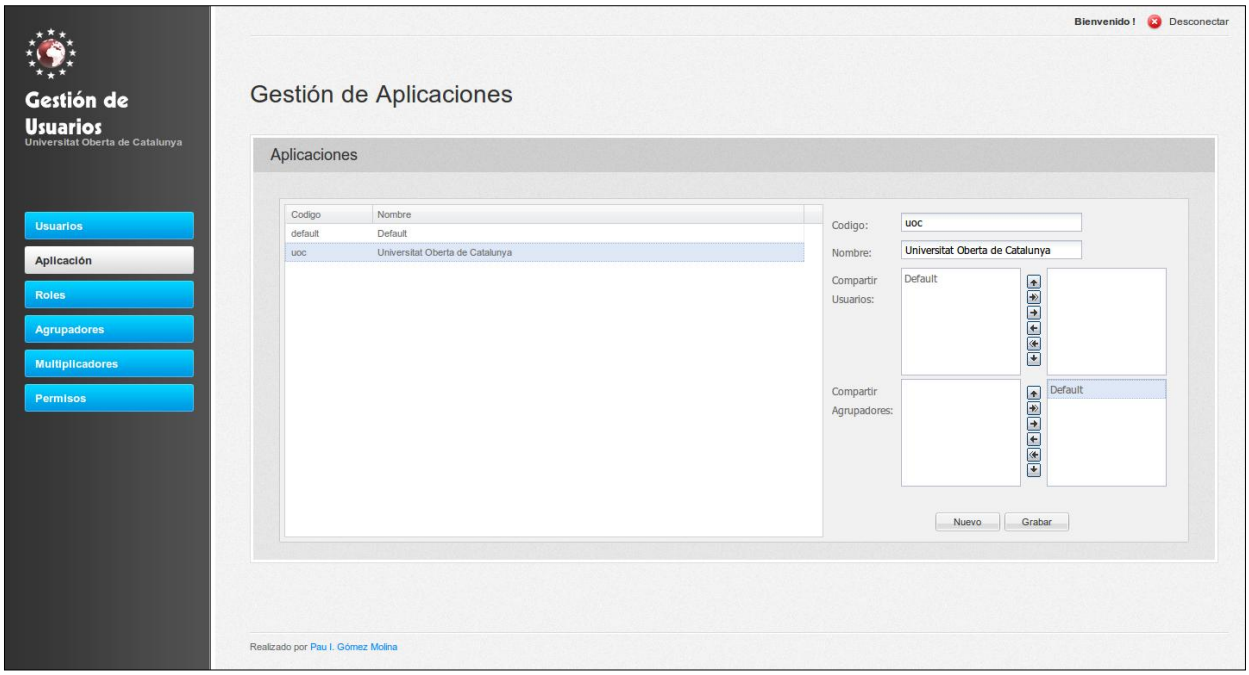


Figura 15. *Pantalla de Aplicación.*

Para aumentar la flexibilidad de la aplicación, permito compartir tanto los usuarios como los agrupadores con otras aplicaciones. Al hacer click en alguna de las aplicaciones mostradas en el grid, podemos determinar entre qué aplicaciones deseamos compartir nuestros usuarios y/o agrupadores. Por ejemplo, hemos accedido a la gestión de la aplicación uoc, y deseamos que los agrupadores creados en ella sean utilizados también por la aplicación default, para ello, en el listbox inferior, seleccionamos default a la izquierda y lo pasamos a la derecha, ya sea arrastrando o mediante alguno de los botones para tal efecto. No debemos olvidar pulsar sobre "Guardar" para que los cambios tengan efecto.

4.3. ROLES

Todos *los usuarios deberán tener un rol asignado*. Un rol le permite a la aplicación cliente determinar qué *capacidades* tiene un usuario determinado. La identificación del rol ha de hacerse a través de un código alfanumérico de hasta 8 caracteres. Este código deberá ser conocido por el cliente, tal vez mediante el uso de constantes o tipos enumerados, y así ser capaz de decidir qué es lo que puede o no puede hacerse.

La gestión de roles es muy sencilla, se trata simplemente de un pequeño formulario asociado a un grid. Para agregar un nuevo Rol, pulsaremos sobre el botón "Nuevo". Automáticamente se añadirá un rol (no persistido aún) con un código aleatorio, y la descripción "ROL". En ese momento puede ser modificado con los datos reales e ir agregando tantos roles como necesitemos. Por último, al pulsar sobre "Grabar", se persistirán todos los roles del grid.

Para la modificación, es necesario que se seleccione un elemento del grid y actualizar tanto el código como el nombre dado al rol. Es posible actualizar cualquier número de roles sin persistir hasta terminar de editarlos, tras lo que, pulsando sobre "Guardar", serán persistidos.

La eliminación funciona siempre y cuando no se hayan asignado roles a Agrupadores. En cuyo caso, informará de que se ha producido un error eliminando el registro. Para borrar se tiene que seleccionar aquello que se desee quitar, y pulsar el botón "Eliminar".

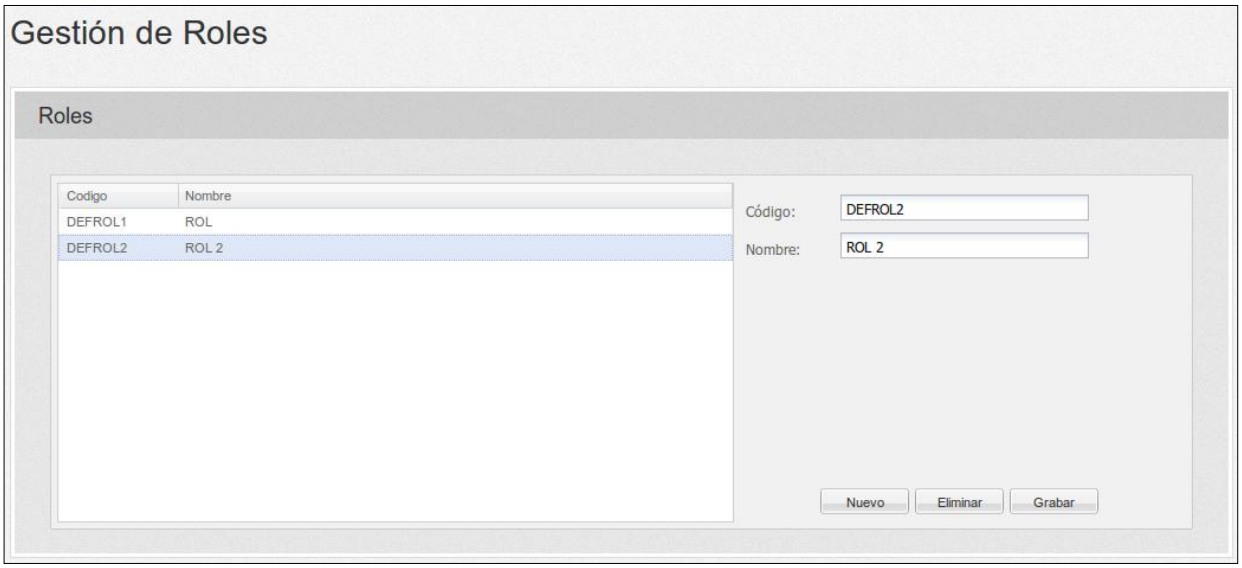


Figura 16. *Pantalla de Roles.*

4.4. AGRUPADORES

Los agrupadores son *unidades lógicas dónde los usuarios son dados de alta*. Se les puede otorgar cualquier significado, como por ejemplo, centros de trabajo, ubicaciones geográficas, responsabilidades, departamentos, etc. Será función de la aplicación cliente decidir de qué modo han de ser organizados los usuarios. También sería válido, en el caso de una aplicación que no requiera este tipo de agrupamientos, dar de alta un único agrupador con un nombre genérico y añadir en él todo el conjunto de usuarios.

En un esfuerzo por generalizar lo más posible la gestión, se ha contemplado la posibilidad de diferenciar los agrupadores estableciendo el conjunto de roles que los usuarios pueden tener por el hecho de existir en ellos. Por ejemplo, si definimos los agrupadores como departamentos dentro de una organización, el departamento “Dirección” tendrá un conjunto de roles diferente a los usuarios pertenecientes al departamento “Ingeniería”, por lo tanto, se puede decidir qué roles, de entre los creados, pueden ser asignados por cada agrupador a sus usuarios.

La gestión de agrupadores funciona exactamente igual que la de los roles, salvo que no tiene botón “Eliminar”, y que además es posible agregarle roles. Si bien, aunque en los casos de uso se dijo que debían poder eliminarse agrupadores, lo cierto es que decidí no hacerlo y dar la ocasión de desactivarlos (no implementado) mediante un check. Este cambio se debe a que todos los usuarios que se den de alta deberán existir en un agrupador por lo que puede ser contraproducente, a la hora de mantener un histórico, si realizamos la eliminación definitiva de este tipo de objetos.

Para crear un nuevo agrupador tan sólo debemos pulsar sobre el botón “Nuevo”. Se añadirá un nuevo agrupador con código y nombre aleatorio. Podemos modificar o crear nuevos (no importa el orden) y los cambios no serán persistidos hasta que el botón “Guardar” se haya pulsado.

Dentro de una misma organización existe la posibilidad de que varias aplicaciones distintas dependan de la información generada entre ellas. En este caso, puede ser interesante disponer de la misma estructura para las mismas. Para facilitar esta situación es por lo que se comparten agrupadores entre aplicaciones en el menú aplicación (comentado anteriormente).

Aquí será posible discernir entre los agrupadores propios y los heredados de otras aplicaciones. A dichos agrupadores los identificaremos porque en el grid aparecerá entre corchetes el nombre de la aplicación a la que pertenecen. Si está renderizado en rojo, significa que se trata de otra aplicación, en cuyo caso sólo podremos modificar los roles que estarán disponibles en esta aplicación para los usuarios dados de alta en ese agrupador. Por el contrario, si fuera de color verde, se tratará de un agrupador creado en esta aplicación y podremos cambiar cualquier aspecto de este. En la captura adjunta puede verse este punto.

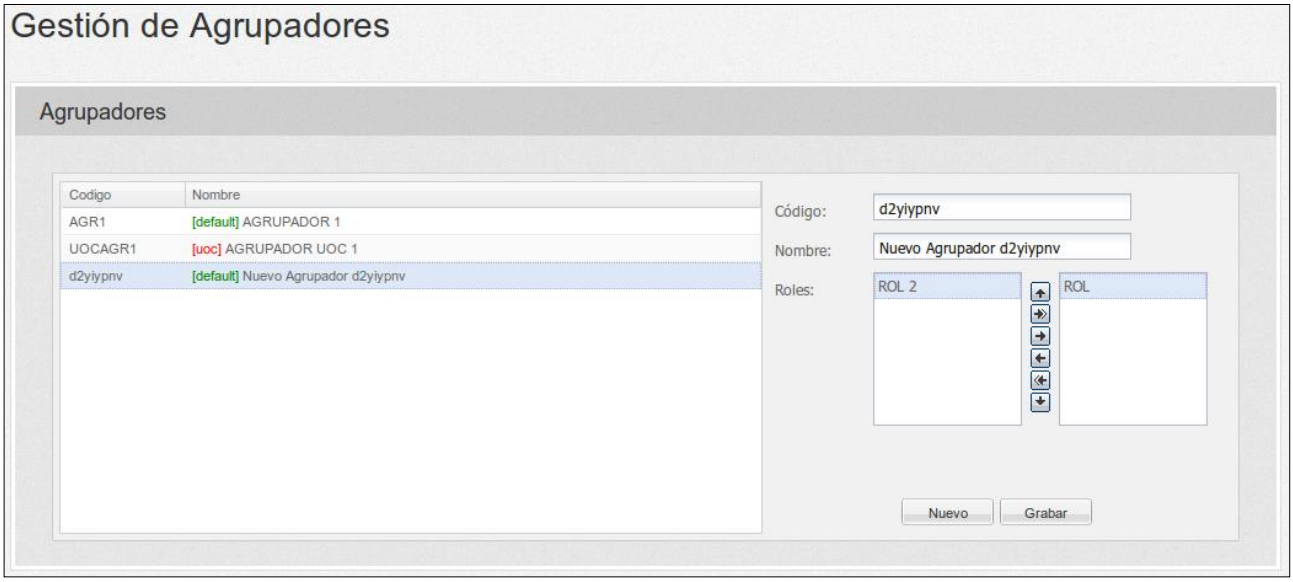


Figura 17. *Pantalla de Agrupadores.*

4.5. MULTIPLICADORES

Los multiplicadores, qué ya han sido introducidos al inicio de este capítulo, son una herramienta extremadamente flexible para dibujar las capacidades de los usuarios, principalmente porque *permiten decidir cómo varía el permiso a lo largo del tiempo.* El multiplicador se define por un código/descripción general y un conjunto de "detalles". Si se crea un multiplicador llamado "ESTADO", podemos, por ejemplo, crear los detalles "INICIADO", "EN PROCESO", "CERRADO" y "FINALIZADO". A continuación, si a un permiso le añadimos el multiplicador "ESTADO", tomará hasta cuatro valores, uno por cada detalle. Siguiendo con el ejemplo planteado, un permiso "P" valdrá "SÍ" para los estados "INICIADO" y "EN PROCESO", mientras que podría valer "NO" para los estados "CERRADO" y "FINALIZADO". Se trataría, por lo tanto, de un cambio en las capacidades que un permiso otorgado a un usuario a lo largo de un proceso.

Cada multiplicador puede tener un número diferente de detalles. Por lo tanto, para poder gestionarlos el formulario se compone de dos grids, y cada uno de ellos con su formulario. El formulario superior es el multiplicador, que le da nombre al conjunto de detalles que irán en el grid inferior. La creación de un nuevo multiplicador sigue el formato de los casos anteriores, pulsamos sobre el botón "Nuevo" y un nuevo multiplicador con código y nombre aleatorio será creado. Para crear el detalle, seleccionamos un multiplicador y, pulsamos sobre el botón "Nuevo Detalle", agregándose el detalle al multiplicador.

La modificación sigue el mismo patrón, seleccionamos un multiplicador y/o detalle, y cambiamos cualquiera de sus propiedades.

La grabación se lleva a cabo mediante el botón "Grabar" que existe únicamente en el formulario superior, debido a que el detalle no tiene sentido sin un multiplicador, y es ahí donde el botón se encuentra habilitado.

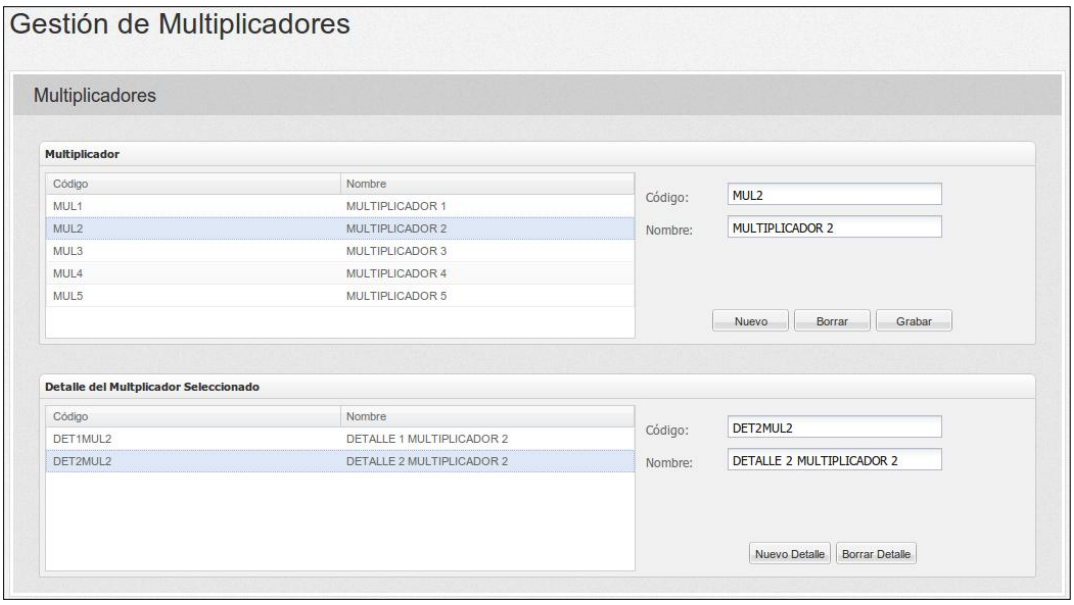


Figura 18. *Pantalla de Multiplicadores.*

4.6. PERMISOS

El último de los casos de uso implementados es la Gestión de Permisos. Un permiso es una unidad atómica que indica *qué puede o no puede hacer un usuario sobre un asunto concreto*. Tal y como se ha explicado en el apartado anterior, a cada permiso se le puede asociar uno o más multiplicadores con la intención de variar el permiso a lo largo del tiempo.

El funcionamiento es igual que en los casos anteriores, para crear un nuevo Permiso hacemos click sobre el botón “Nuevo”, de esa manera se generará un nuevo permiso con código y nombre aleatorio. Podemos modificar y agregar tantos permisos como queramos, pero sólo se persistirán una vez pulsemos el botón “Grabar”.

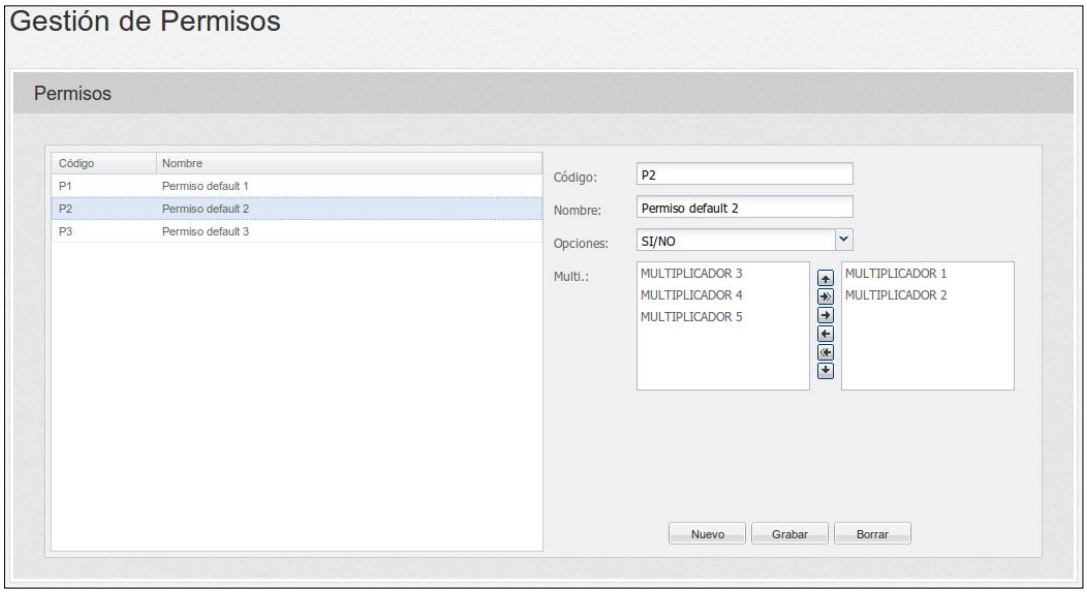


Figura 19. *Pantalla de Permisos.*

5. FUTURAS AMPLIACIONES

El producto entregado *no está cerrado*, aún es preciso implementar parte de la lógica. En primer lugar deberá de habilitarse la posibilidad de administrar usuarios, asignándole a cada uno de ellos el resultado de la configuración de la aplicación cliente. El siguiente paso natural es la creación del servicio web, el cual, debe permitir la gestión remota de los usuarios, facilitando las tareas de validación de los mismos así como su consulta. Por último, debiera integrarse una auditoría, junto a un sistema de alertas, tanto visuales al entrar a la gestión, como notificaciones vía e-mail.

6. CONCLUSIONES

He pretendido que el TFC sea un camino a través del cual recorrer todos los aspectos relacionados con la creación de una nueva aplicación. Lo he enfocado principalmente a adquirir habilidades técnicas. Por lo tanto, en primer lugar quisiera comentar que J2EE es un territorio muy grande que recorrer, un lugar donde convergen gran cantidad de entornos heterogéneos, únicamente enlazados entre sí por las especificaciones del lenguaje. La integración de estas tecnologías supone un gran hándicap, aunque su inclusión favorece enormemente el trabajo a realizar. En el caso de la gestión de usuarios, he querido llevar a cabo una aproximación, no sin dificultades, a los componentes actuales con más difusión a día de hoy.

Por otra parte, ha resultado tremendamente interesante tratar de generalizar el proyecto para aplicarlo en el mayor número de casos posibles. Esto ha sido un gran reto, puesto que se necesita considerar un número muy elevado de aspectos y posibilidades. Al final, en mi opinión, la lógica planteada permite que se cree un entorno muy útil y extremadamente sencillo de gestionar, distinto a productos comerciales de gestión de identidades que, aún dando la opción de afrontar gran cantidad de tareas, en la gran mayoría de las ocasiones son muy costosos por su mantenimiento, tienen una curva de aprendizaje muy grande y se encuentran sobredimensionados para entornos medianos y pequeños.

Todo lo relacionado con el proyecto ha sido muy atrayente, sobre todo el entrar en contacto con un framewok como Spring. Éste permite una configuración tremendamente cómoda a partir de anotaciones a la hora de incorporar el modelo MVC, ya que da soporte para prácticamente cualquier framework o especificación del lenguaje Java. Además, permite desplegar aplicaciones sin la obligatoriedad de utilizar un servidor de aplicaciones, algo que repercute directamente en la cantidad de memoria precisa para su ejecución. Por supuesto, está diseñado para soportar cualquier implementación de JPA, o al menos las más conocidas, las cuales incluso con su inherente complejidad favorecen un desarrollo orientado a objetos, en lugar de orientarlo a tablas, como sucede con JDBC.

Tal vez la parte menos positiva sea la referente a Google Web Toolkit. Aunque parece relativamente sencillo impone ciertas dificultades a la hora de adquirir un cierto grado de comprensión del mismo, por tratarse de JavaScript en un entorno Java. En algunos momentos se

hace difícil comprender por qué está diseñado del modo que lo está o por qué no se pueden consumir clases y paquetes propios de Java cuando lo que estamos escribiendo es código Java. A este respecto, prefiero codificar directamente en JavaScript, es más natural, permitiendo mayor control sobre el comportamiento de los componentes incrustados en la página web. Por otro lado, he de reconocer que me retrasó mucho la integración de Spring con GWT, pero resultó ser un gran acierto el uso de la GWT Server Library ya que supuso que pudiera disponer de todos los recursos que Spring ofrecía, sobre todo, la inyección de dependencias.

Creo firmemente que el trabajo ha contribuido a crecer profesionalmente en el terreno del desarrollo de aplicaciones, por un lado fomentando mi creatividad, y por otro propiciándome una visión más completa del mundo J2EE.

7. GLOSARIO

- **Agrupador**: Determina qué modelo lógico ha de seguirse para ubicar a los usuarios. La Gestión de Usuarios obliga a que aquellos individuos que vayan a ser administrados pertenecen a un grupo que los identifica según sus funciones. Es necesario, por lo tanto, la creación de al menos un agrupador donde agregar usuarios.

- **Aplicación o Aplicación Cliente**: Hace referencia al software externo a la Gestión de Usuarios que recurre a esta para administrar los usuarios, roles y permisos de su dominio.

- **Multiplicador**: Es un componente que permite describir los instantes en el tiempo donde los permisos de los usuarios pueden variar.

- **Permiso**: Determina de forma atómica cada una de las atribuciones otorgadas a un usuario dentro de un rol.

- **Rol**: Tiene la función de identificar cuáles son las funciones, dentro de la aplicación cliente, de cada usuario y que le habilitarán para llevar a cabo un subconjunto de acciones en la organización.

- **Usuario**: Se trata de un individuo administrado por la Gestión de Usuarios y perteneciente a una aplicación.

- **Valor Permiso**: Opción elegida de entre los Valores Permiso.

- **Valores Permiso**: Conjunto de opciones elegibles para asignar a un permiso. En la Gestión de Usuarios existen los Valores de tipo Sí/No y los Valores R/W/RW.

8. BIBLIOGRAFÍA

Libros:

- *Eckel, Bruce.* **Piensa en Java 4ª edición**. Editorial Pearson Educación S.A., Madrid, 2007.
- *Machaek, Vukotic, Chakraborty, Ditt*. **Pro Spring 2.5.** Editorial Apress, Nueva York, 2008.
- *Mak, G.* **Spring Recipes**. Editorial Apress, Nueva York, 2008.
- *Mularien P.* **Spring Security 3**. Editorial Packt Publishing, Birmingham, UK, 2010.
- *Bauer C., King G.* **Java Persistence with Hibernate**. Editorial Manning, Greenwich, CT, 2007.
- *Keith M., Schincariol M.* **Pro JPA 2**. Editorial Apress, Nueva York, 2009.
- *Dwyer J.* **Pro Web 2.0 Application Development with GWT**. Editorial Apress, NY, 2008.
- *Cooper R., Collins C.* **GWT in Practice**. Editorial Manning, Greenwich, CT, 2008.

Documentos:

- **Spring Framework 3.0 Reference Manual**,
  http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf
- **Spring Security Reference Documentation**,
  http://static.springsource.org/spring-security/site/docs/3.0.x/reference/springsecurity.pdf
- *Waldman S.* **C3P0 JDBC3 Connection and Statement Pooling**,
  http://www.mchange.com/projects/c3p0/index.html, Machinery For Charge, Inc. 2006.
- **GWT Server Library - Reference Documentation v1.0**,
  http://gwt-widget.sourceforge.net/gwt-sl/reference-1.0/index.html
- **GWT Developers Guide**,
  http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuide.html
- **GXT API Documentation**,
  http://www.sencha.com/gxtdocs/