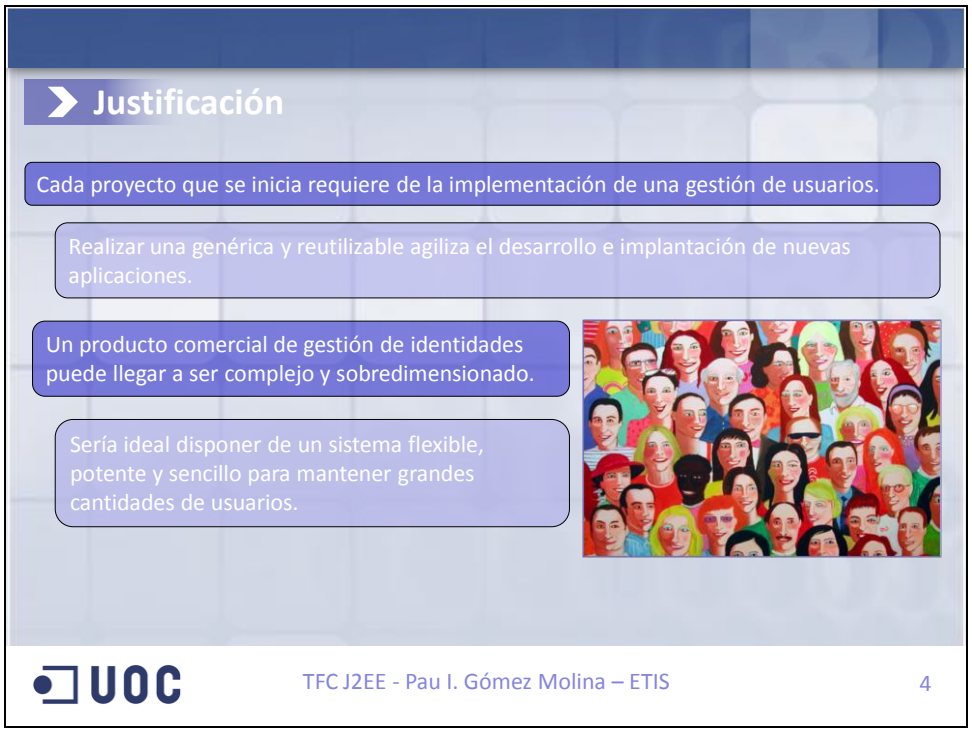
Contenidos

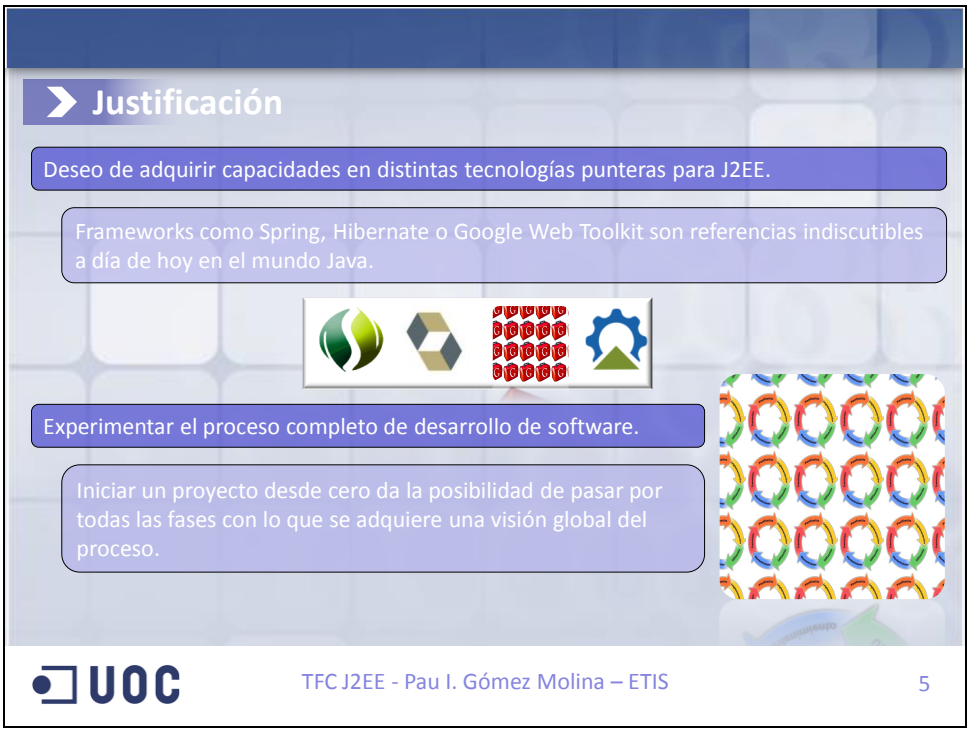
- **Introducción**
  - Justificación
  - Objetivos
  - Planificación
- **El Proyecto**
  - Descripción
  - Estructura Lógica
  - Componentes
  - Actores
- **Diseño**
  - Tecnologías
  - Integración
- **Producto**
  - Escenario
  - Producto
- **Conclusiones**

Introducción



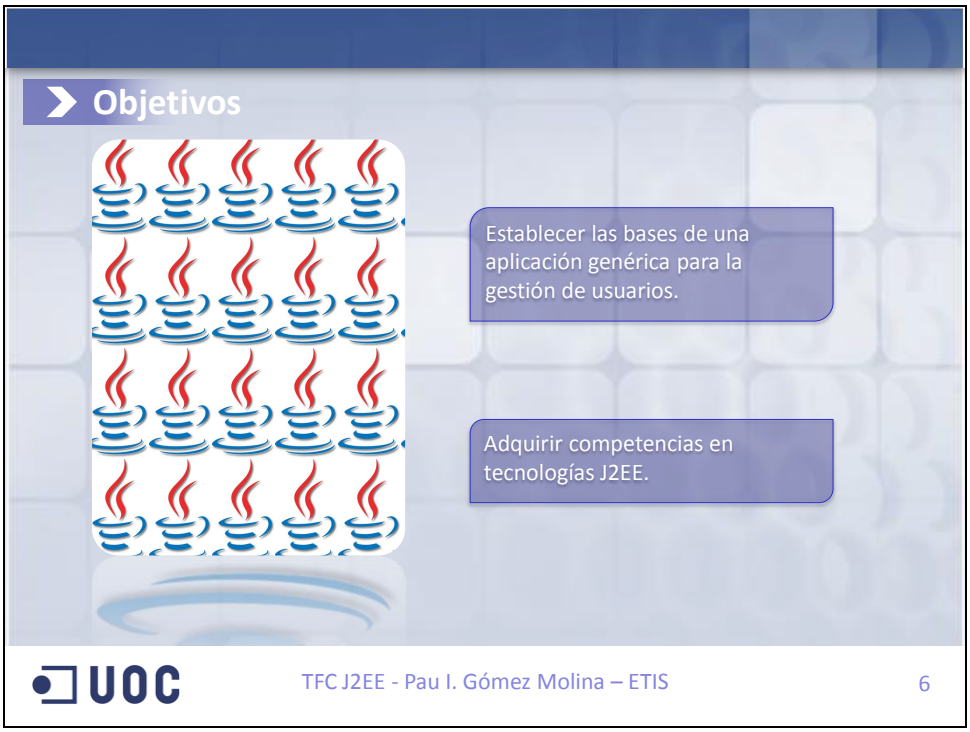
El desarrollo de aplicaciones requiere un alto grado de reusabilidad para permitir que la productividad sea elevada. Lograr este objetivo no es tarea fácil, más aún cuando los requisitos y funcionalidades de los clientes varían en gran medida de un proyecto a otro. Aún así, hay ciertos aspectos comunes que sí se pueden implementar y aprovechar en distintos ámbitos.

Este TFC pretende desarrollar una parte tan importante y fundamental como la gestión de usuarios. Las gestiones de usuarios pueden llegar a ser entornos complejos y muy atados a exigencias propias del producto en construcción. Esta administración ha de ser lo bastante flexible como para facilitar su integración en entornos heterogéneos.



Otras razones por las que escogí esta área del TFC son la necesidad de **adquirir capacidades en el desarrollo de aplicaciones J2EE** y 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.



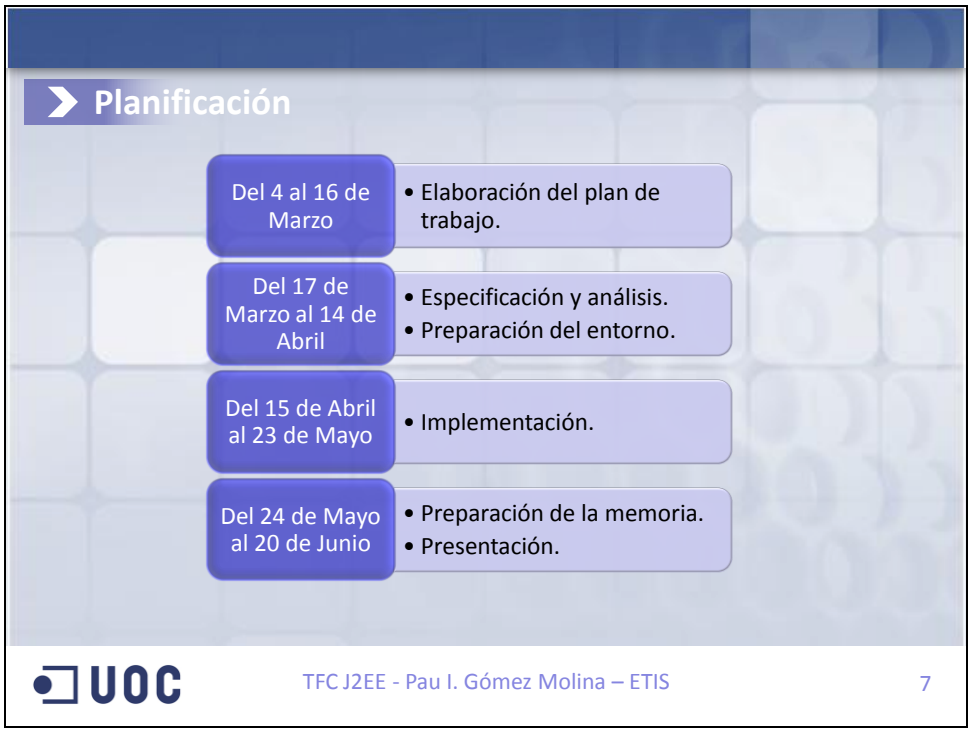
El **objetivo principal** de este proyecto es el de **sentar las bases para el desarrollo de una aplicación genérica de gestión de usuarios**. Dicha gestión debe ser capaz de almacenar una serie de datos lo suficientemente universal para ser reutilizado por distintas aplicaciones cliente.

Los **objetivos específicos** del proyecto son el **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.

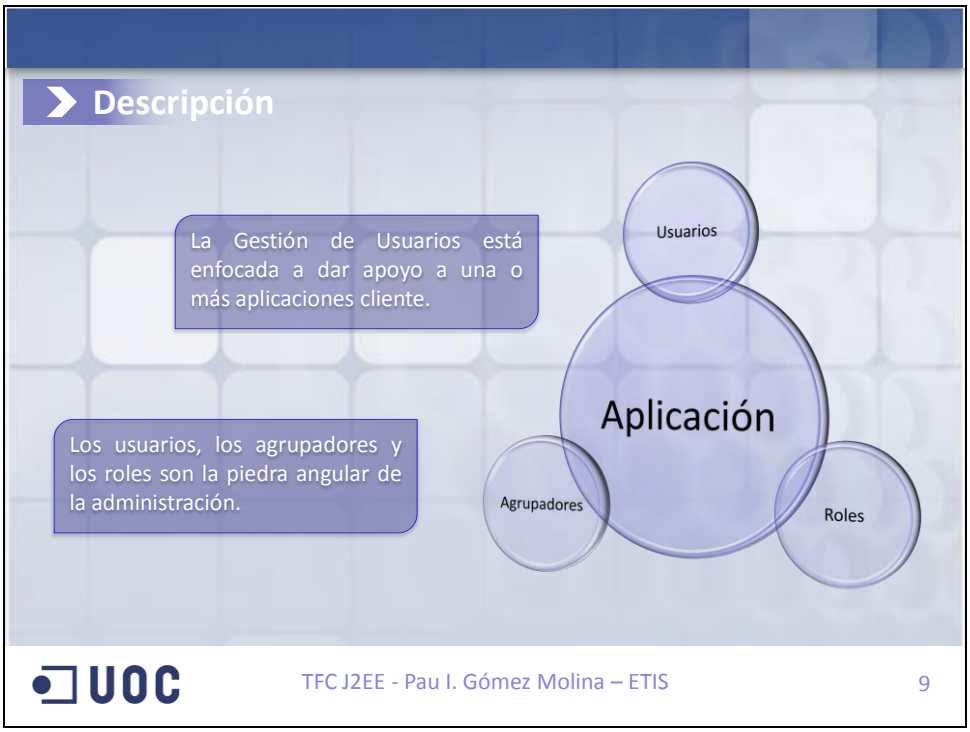
Los objetivos personales se centran en la **adquisición de competencias** a la hora de iniciar un proyecto, así como la capacidad de **manejar frameworks** específicos para realizar la tarea.



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.

Las fechas indicadas han seguido la temporalización marcada en el aula, en dónde cada hito hacía referencia a la entrega de una P.A.C.

El Proyecto



La **Gestión de Usuarios** es en realidad un almacén que organiza los aspectos básicos de la administración de usuarios. Así mismo, su eje central son las aplicaciones cliente que consumirán sus servicios. Es capaz de mantener un número elevado de aplicaciones de forma simultánea, lo que la convierte en una herramienta muy conveniente para dar soporte tanto a desarrollos propios como de terceros.

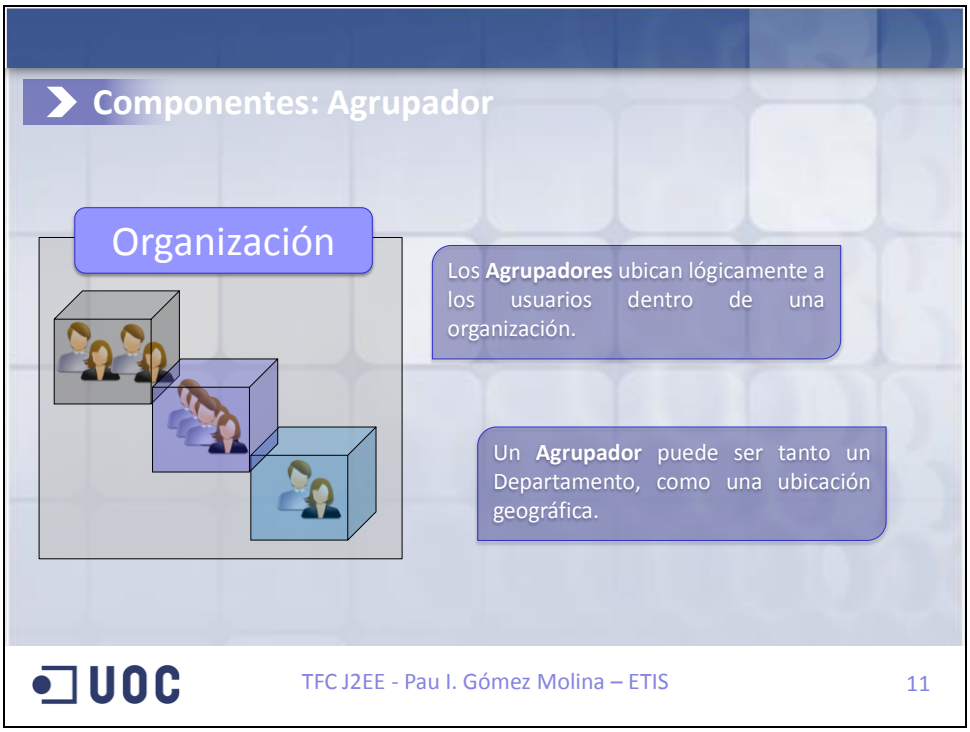
Las piedras angulares en la administración son los Usuarios, los Agrupadores y los Roles :

- Los **Usuarios** serán los individuos que forman parte del dominio de la aplicación cliente y que interactuarán en distintos grados con ella.
- Los **Agrupadores** determinarán el ámbito de cada usuario, cuál es su ubicación dentro de una organización.
- Cada usuario deberá tener asignado al menos un **Rol** que indicará cuáles son sus responsabilidades y, por consiguiente, las capacidades de que dispondrá para interactuar con la aplicación cliente.

El modo en que la aplicación interactúa con los Agrupadores, Roles y Usuarios define la estructura lógica que la Gestión de Usuarios administra.

En el nivel superior tenemos a la Aplicación cliente, dada de alta en la gestión. Normalmente la aplicación se definirá a sí misma con un nombre y un código que la identifique. A continuación, la aplicación dividirá todos los restantes ámbitos en Agrupadores. Cada Agrupador ha de tener un número de roles que pueda asignar a los usuarios que sean incluidos en él.

Para facilitar una mejor integración con distintas aplicaciones, la Aplicación puede compartir sus Agrupadores y Usuarios con otras aplicaciones que existan en la misma gestión de usuarios.

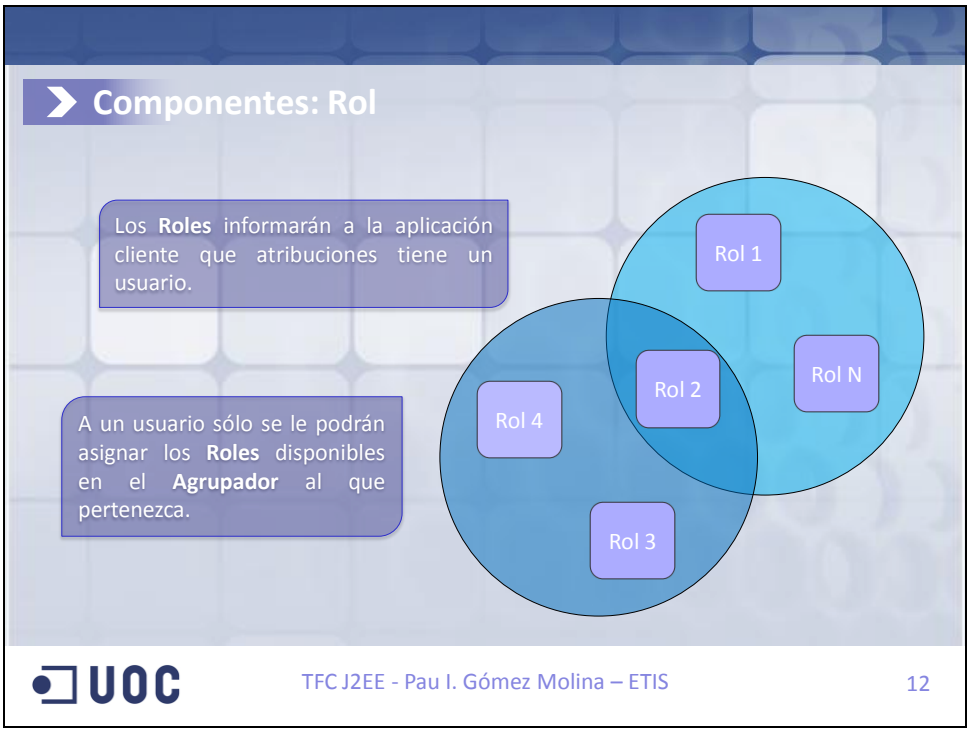


El **Agrupador**, determina qué modelo lógico ha de seguirse para ubicar a los usuarios. Pongamos por ejemplo una empresa comercial con diversas delegaciones a lo largo del país. Sus usuarios podrían ser "Agrupados" según su ubicación geográfica, o tal vez por el departamento al que pertenecen. Es decir, sería lógico que tuviéramos los siguientes Agrupadores:

- Dirección
- Departamento de Ventas
- Departamento de Facturación
- Almacén

En el ejemplo anterior añadiríamos usuarios según su departamento a cada agrupador de tal modo que a un vendedor lo daríamos de alta en el Agrupador "Ventas", mientras que al presidente de la compañía en "Dirección".

Esta forma de organizar usuarios no deja de ser una estructura lógica que dependerá de cada organización o caso.



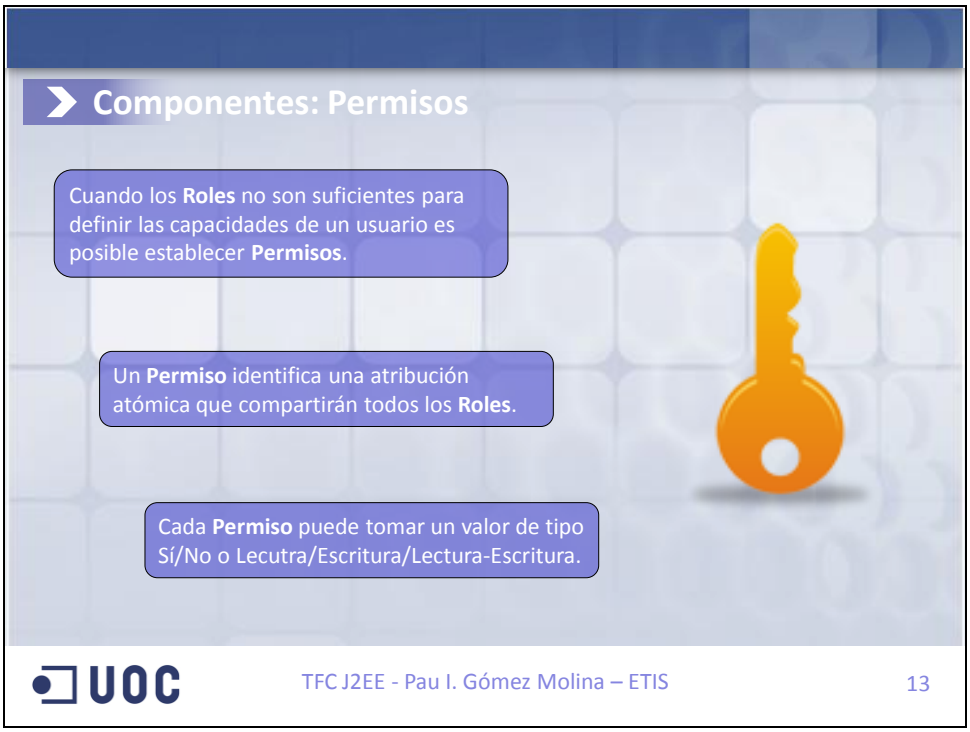
Los **Roles** tienen la función de identificar cuáles son las capacidades dentro de la aplicación cliente de cada usuario. En la Gestión de Usuarios los roles están identificados con un código y un nombre descriptivo.

Los **Agrupadores**, que son los que contendrán los usuarios, tendrán asignados un subconjunto de roles que podrán otorgar a sus usuarios. Con esta aproximación puede evitarse que ciertos usuarios con capacidad de administrar un Agrupador den Roles que los usuarios no debieran tener.

Por ejemplo, supongamos que en la Gestión de Usuarios se han creado los siguientes roles que la aplicación cliente conoce:

- Administrador
- Director
- Gestor
- Lector
- Operador

Entonces podríamos asignar los roles Administrador, Director y Gestor al Agrupador "Dirección", mientras que al "Departamento de Almacén" le damos el rol "Lector". En este caso, a un usuario existente en "Almacén" nunca podría dársele el rol "Director". Tampoco se daría el rol "Lector" a ninguno de los dos Agrupadores ya que no se les ha asignado.

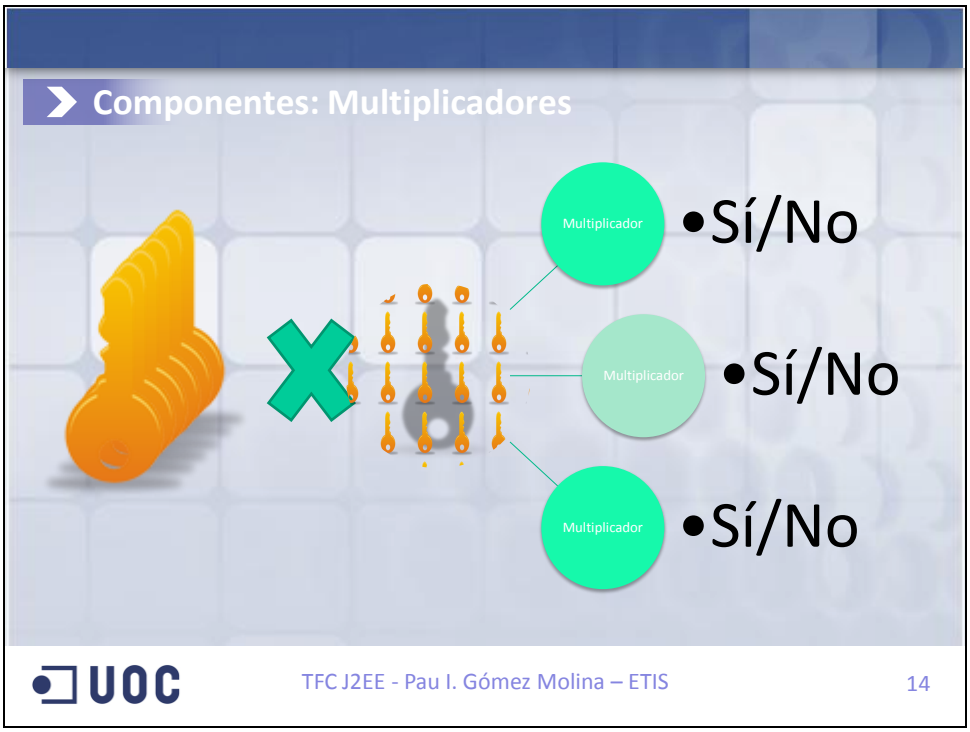


Existen ocasiones en las que las aplicaciones necesitan especificar con detalle cuales son las operaciones atómicas que sus usuarios pueden realizar. Para estos casos pueden crearse **Permisos**. Si bien son opcionales, en caso de existir son comunes a todos los roles.

Se han contemplado dos posibles tipos de permisos según el valor que puedan tomar, estos son los tipos :

- Sí / No
- R/W/RW (Lectura / Escritura / Lectura-Escritura)

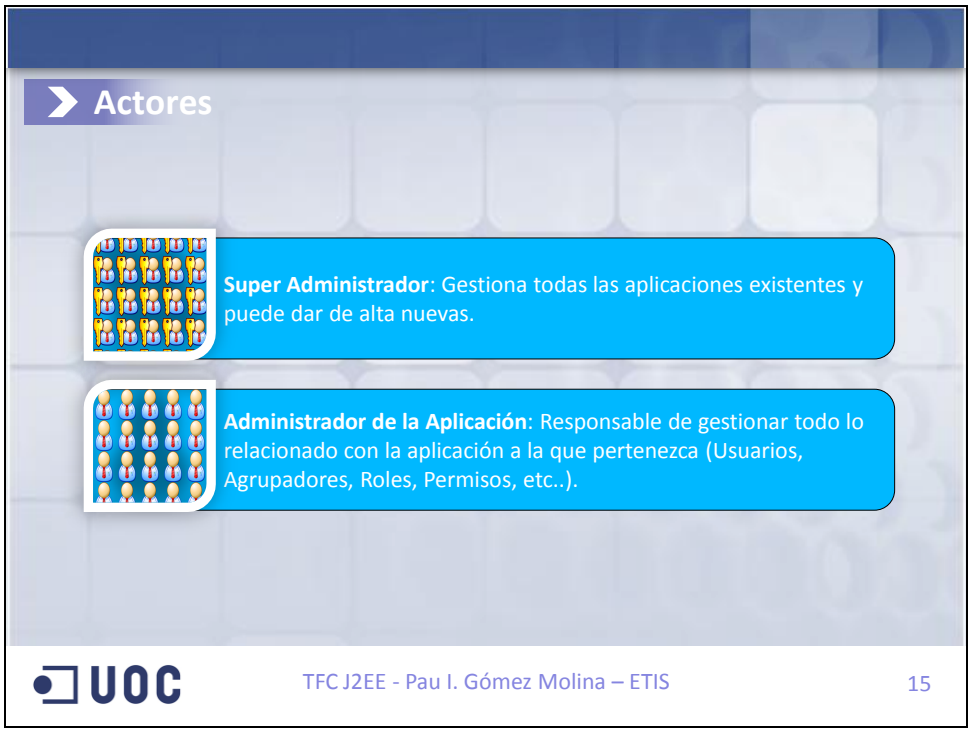
Por ejemplo, podríamos crear un permiso llamado “Enviar e-mail” que pueda valer Sí o No. En el caso de que creáramos uno llamado “Acceso al directorio” es posible hacer que valga Lectura, Escritura o Lectura / Escritura.



En ocasiones puede ser necesario variar el valor de un permiso a lo largo del tiempo. Para tal fin se ha incluido el concepto de **Multiplicador**. Un multiplicador representa un estado en un proceso. Durante el ciclo de vida del proceso pueden existir diversos estados, y en cada uno de ellos asignarle un valor distinto a los permisos afectados.

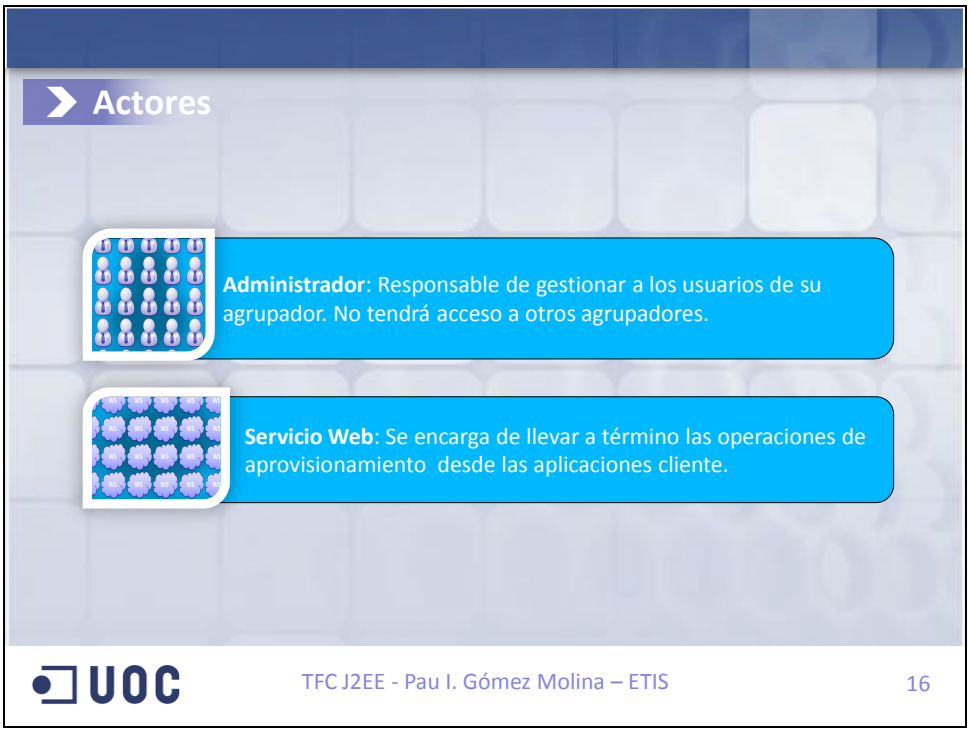
El nombre Multiplicador viene a representar este concepto puesto que “multiplica” un permiso tantas veces como el número de estados por el que el proceso pase. Por ejemplo un permiso “P” en el estado “E1” tendrá el valor “Sí”, el mismo permiso en el estado “E2” tendrá el valor “No” y durante el estado “E3” de nuevo “Sí”. Se ha “multiplicado” el permiso 3 veces, el mismo número que de estados.

Si bien no es obligatorio asignar multiplicadores a los permisos, podemos decidir que un permiso es sensible a dichos estados mientras otro no, por lo que los habrá que se “multipliquen” mientras que otros no lo harán.



La Gestión de Usuarios reconoce un conjunto de Actores que interactuarán con ella:

- **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.
- **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.

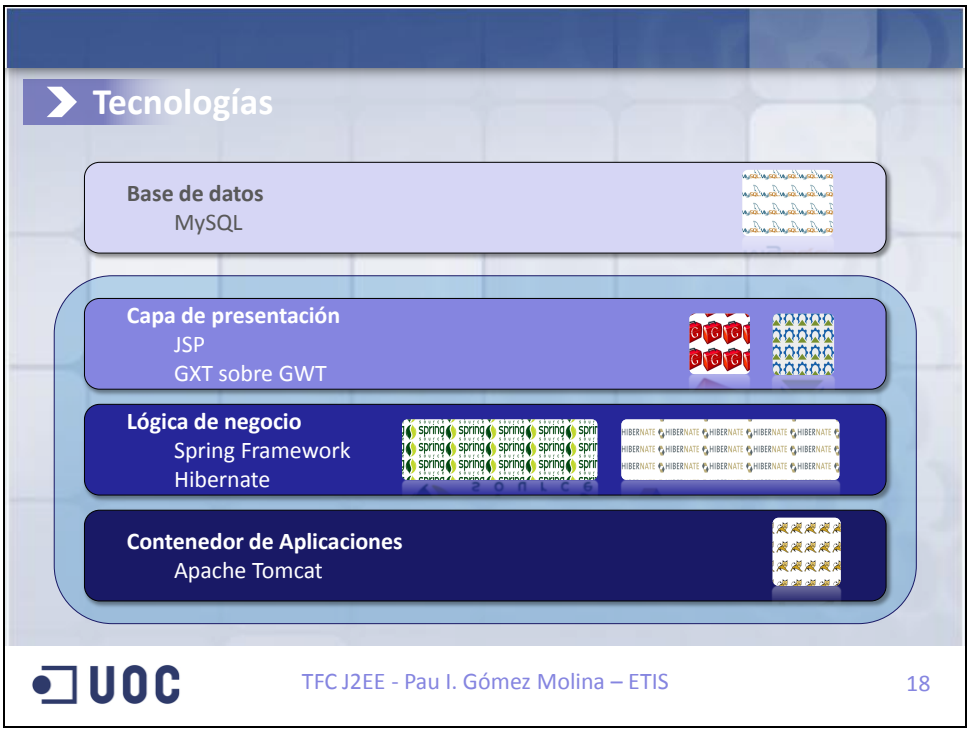


A parte de los dos anteriores, la Gestión de Usuarios contempla los siguientes Actores:

- **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.
- **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.

Al no estar implementada la administración de usuarios, el único actor reconocido es el super administrador.


Diseño



Para el desarrollo de la Gestión de Usuarios se han escogido las siguientes tecnologías:

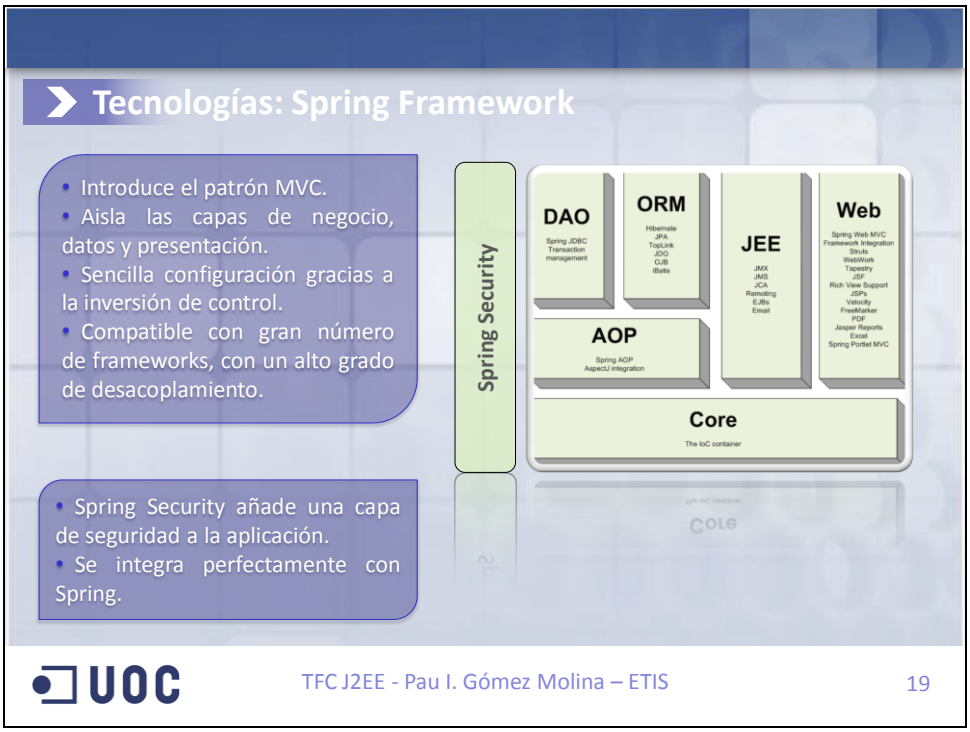
- **MySQL** como Sistema Gestor de Bases de Datos. Se trata de un producto con muchos años de existencia que le confieren mucha fiabilidad y prestaciones iguales o superiores a algunos motores comerciales. Además, dipone de gran cantidad de recursos on-line.

- **Apache Tomcat** es un contenedor de aplicaciones muy ligero y con un consumo de memoria sensiblemente inferior a los servidores de aplicaciones. Además, es muy fácil de configurar y administrar.

- **Spring** es un framework muy ligero que ofrece múltiples ventajas a la hora de afrontar un nuevo desarrollo. Por un lado facilita la inclusión del patrón MVC y por otro permite una estupenda integración con distintas implementaciones de JPA. Además, bajo su auspicio existen multitud de proyectos que se complementan perfectamente con él, como por ejemplo, **Spring Security** que también ha sido incluido para dotar de seguridad a la aplicación.

- **Hibernate** es una de las primeras implementaciones de la especificación JPA y tal vez las más utilizada. Por esta razón ofrece mucha fiabilidad así como recursos on-line que ayudan a su manejo.

- **Google Web Toolkit (GWT) y ExtJS para GWT (GXT)** son dos frameworks utilizados para la capa de visualización que combinan un ágil desarrollo en Java con gran cantidad de controles que posteriormente son traducidos a JavaScript, dando soporte para AJAX lo cual repercute directamente en la experiencia del usuario.



Spring 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.

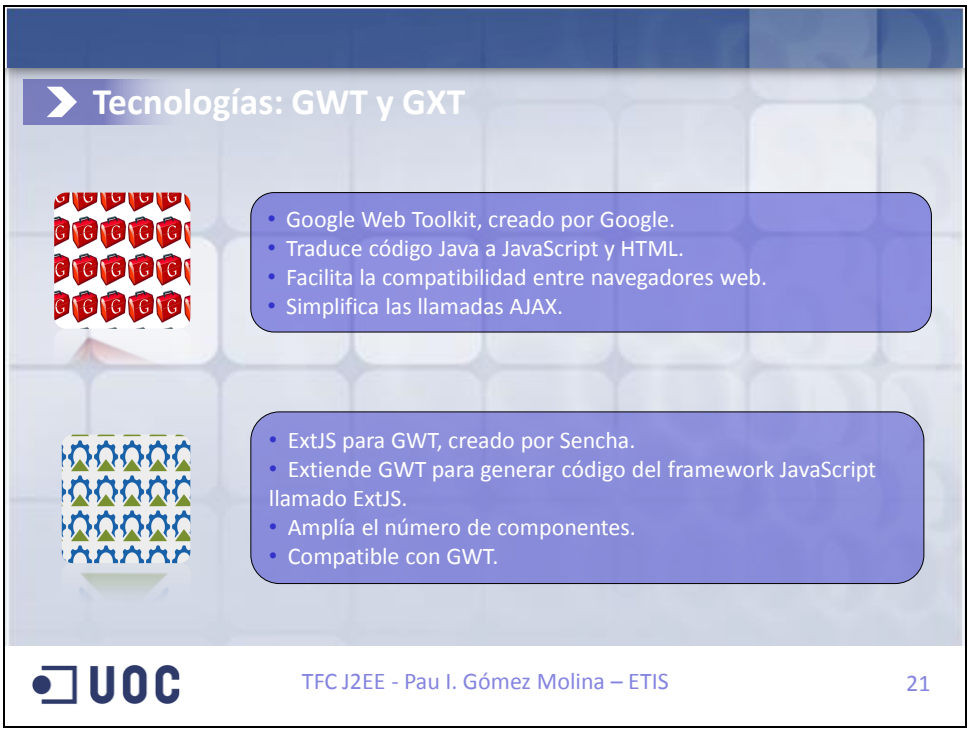
Spring ofrece *soporte para patrón **MVC*** (Model View Controller). De este modo será el propio Spring quién mapee las peticiones a controladores manejados por él. Esta aproximación permite el aislamiento de las capas :

- **Modelo**, describe los elementos del dominio de la aplicación. Normalmente hace referencia a las entidades que son persistidas en la base de datos.
- **Vista**, es la capa de visualización, o aquella que produce los interface de usuarios.
- **Controlador**, es la capa en la que se desarrolla toda la lógica de la aplicación y en un contexto de Spring será la que responda a las peticiones (Requests).

Se ha realizado también la integración del proyecto **Spring Security** cuyo objetivo es el de proveer de seguridad a la Gestión de Usuarios. Su función principal será la de validar a los usuarios, proteger del intento de acceso a áreas privadas por usuarios no autentificados y la de dificultar un ataque a la integridad de la Gestión de Usuarios.

Para la capa de persistencia se va a utilizar Hibernate. Este framewok implementa las especificaciones de JPA por lo que es un gestor de entidades. Una entidad es una clase java o POJO (Plain Old Java Object) que es mapeada a una tabla de una base de datos relacional. Con esto se persigue que todo el desarrollo sea orientado a objetos, eliminando el acceso a la base de datos vía JDBC con lo que se obtienen grandes beneficios a la hora de trabajar con el modelo de negocio.

JPA necesita de un "EntityManager" que debe ser inyectado en cada una de las clases que manejan la persistencia. Estas clases o DAO's (Data Access Objects) serán los servicios que tendrán la lógica encargada de persistir y obtener objetos persistentes. Su creación, administración e inclusión en transacciones puede llegar a ser una tarea complicada y tediosa, por lo que combinando Hibernate con Spring se logra un doble objetivo, primero una configuración sencilla y rápida gracias a que los objetos necesarios (EntityManager, Transacciones, DAO's) son creados y mantenidos por Spring, y segundo se logra desacoplar la capa de persistencia del resto de la aplicación. Esto último nos facilitaría el cambio de implementación en un futuro, por ejemplo a TopLink, sin necesidad de cambiar una sola línea de código. Para que esto último sea posible, hay que tratar de respetar las especificaciones de JPA y evitar utilizar herramientas propias de Hibernate.

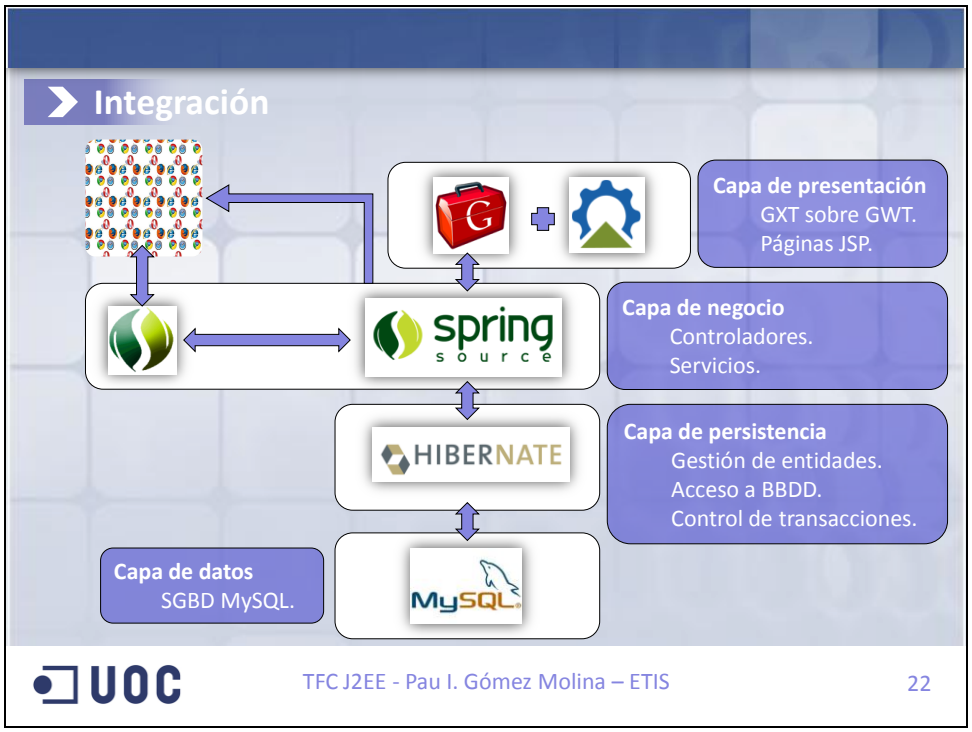


Como cualquier aplicación web actual, se desea utilizar la tecnología AJAX para mejorar tanto la experiencia del usuario, evitando recargas innecesarias de la página cada vez que se actualiza un dato. AJAX permite esto al realizar llamadas asíncronas al servidor. Se trata un componente del lenguaje JavaScript que todos los navegadores modernos soportan.

A día de hoy puede conseguirse este comportamiento de muchas maneras, aunque lo habitual es usar algún framework JavaScript que ayude a su implementación. De entre estas opciones Google Web Toolkit (GWT) ofrece la posibilidad de escribir código java que posteriormente, mediante el uso de su propio compilador, lo convierte a JavaScript. La gran ventaja radica entre otras cosas en la optimización que el compilador de GWT hace del código generado y la garantía de compatibilidad con la mayoría de exploradores web.

GXT (ExtJS para GWT) es un proyecto que extiendo GWT para la utilización de la librería JavaScript ExtJs. Ofrece multitud de nuevos controles y posibilidades que superan en creces los ofrecidos por GWT. Sin embargo, todo el soporte tecnológico sigue siendo el ofrecido por el paquete de Google, por lo tanto las tareas de llamadas asíncronas seguirán su patrón, es decir, serán llamadas a procedimientos remotos java (RPC).

Otro aspecto importante de GWT es que al generar código JavaScript , aun habiendo sido escrito en java, las clases que pueden ser convertidas son muy limitadas. Esto impone una serie de restricciones, como que por defecto no es posible utilizar clases desarrolladas por nosotros que no se encuentren dentro del paquete de GWT. La mayor dificultad la encontramos, no tanto en las clases necesarias para montar la vista, sino en aquellas que deben ser transferidas a través de los servicios RPC. Para solucionarlo, es preciso escribir Data Transfer Objects (DTO's) que serán construidos a partir de las entidades persistentes, para así poder intercambiar datos entre el navegador y el servicio. La información que llegue desde GWT vendrá como clases de tipo DTO y deberá ser convertida a una entidad antes de poder ser persistida.

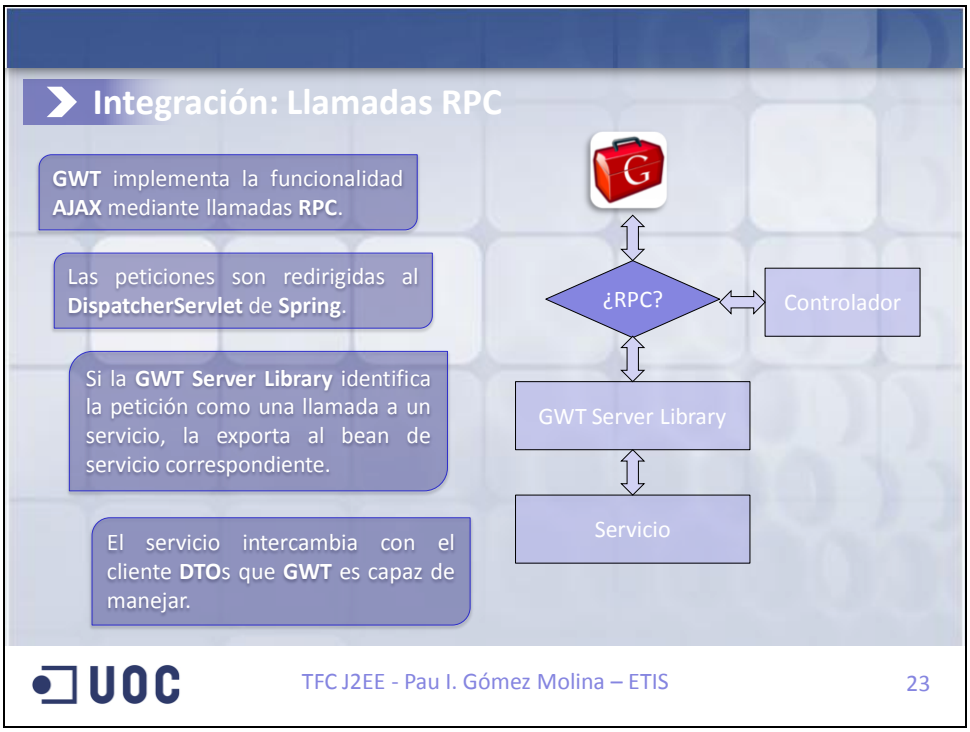


La integración de todos los componentes anteriores es relativamente sencilla gracias al uso de Spring. En primer lugar se configurará la aplicación para que sea Spring el que responda a las peticiones tanto de páginas dinámicas como a aquellas que sean llamadas a servicios RPC.

Por otro lado Spring se encargará de la creación de la capa de persistencia JPA utilizando el framework Hibernate, inyectando los "EntityManager" a los objetos de acceso a datos. Hibernate interaccionará con el gestor de bases de datos (SGBD) MySQL.

La vista será implementada de dos formas distintas. Por páginas JSP internacionalizadas con Spring, y por la utilización de GWT y GXT. Estos últimos obtendrán el modelo gracias a las llamadas asíncronas que realizarán a los servicios RPC que la Gestión de Usuarios oferta.

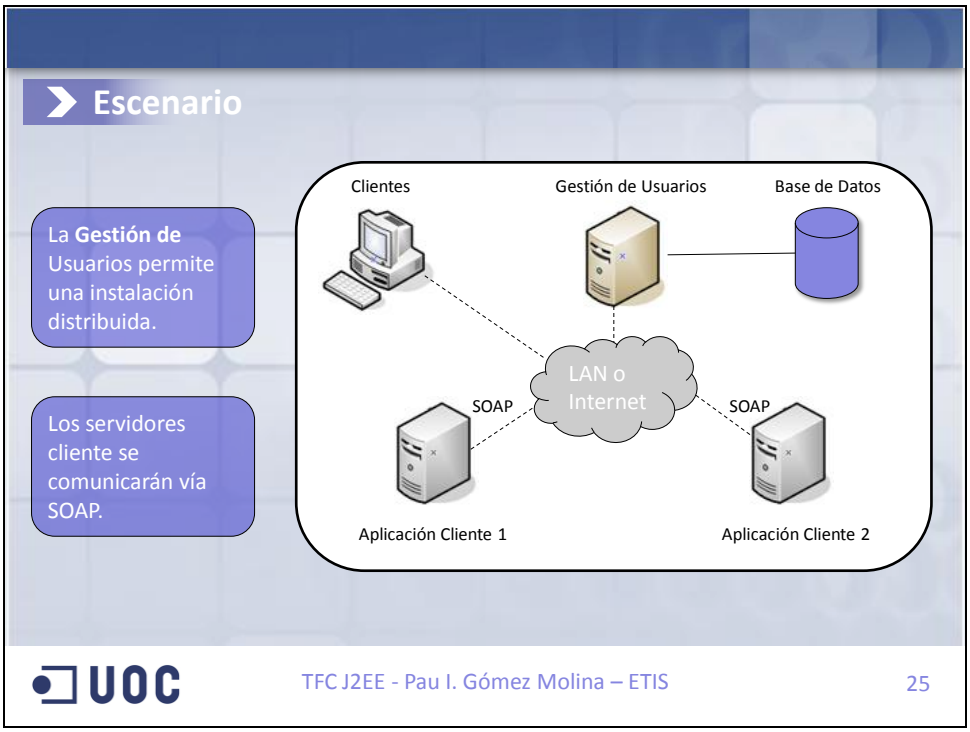
Tal y como puede observarse en el gráfico, Spring es quién se ocupa de interactuar con todas las tecnologías usadas, es un puente entre todas ellas. Eso es debido a la inversión de control que permite al framework "repartir" tareas y responsabilidades entre los integrantes de la aplicación y la inyección de dependencias que le da a estos beans otros que precisan para realizar sus tareas. En realidad Spring se preocupa de mantener el ciclo de vida de todos los objetos de los que es responsable.



En el apartado de la integración hay que detenerse en las llamadas RPC. Estas llamadas son realizadas por GWT y deben ser capturadas por un Servlet por cada servicio que se ofrezca. Si se utiliza este enfoque, las llamadas RPC serían procesadas por el Servlet fuera del contexto de Spring y perderíamos todas las ventajas de trabajar con él. Afortunadamente existe un proyecto llamado **GWT Server Library** que soluciona este problema. Básicamente su función es la de escuchar las peticiones entrantes y, cuando detecta que alguna de ellas solicita un servicio, reenvía dicha petición al contexto de Spring quien la gestionará como cualquier otra llamada, entregando en este caso del control a la implementación del servicio.

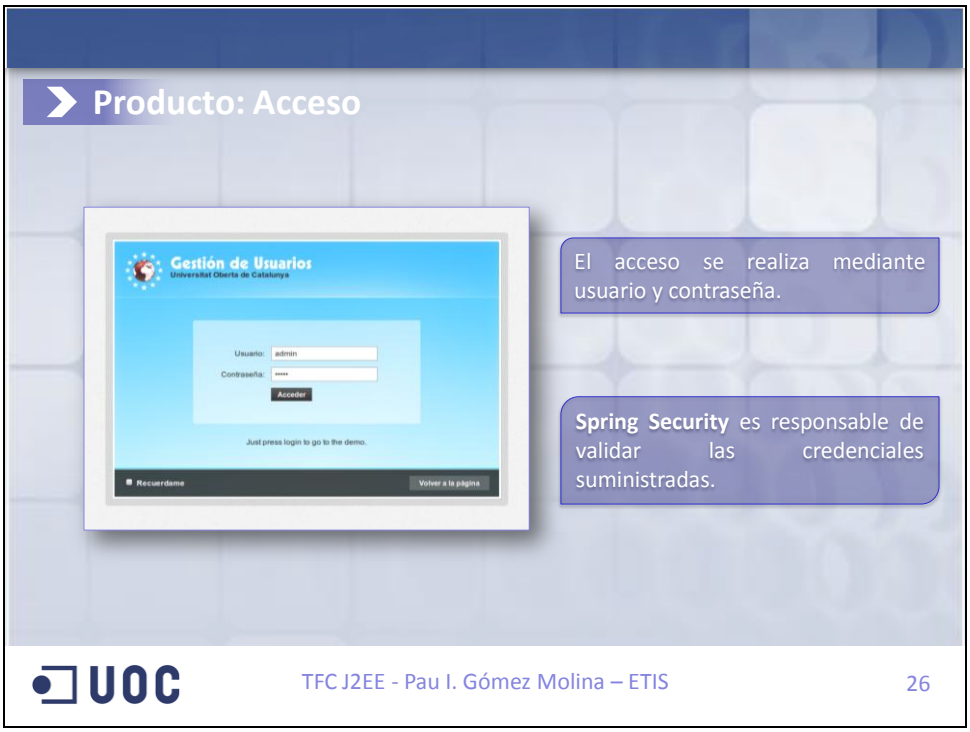
Así mismo, los datos que son transferidos mediante las llamadas RPC serán objetos de transferencia de datos (DTO's) comentados en la diapositiva anterior.

Producto

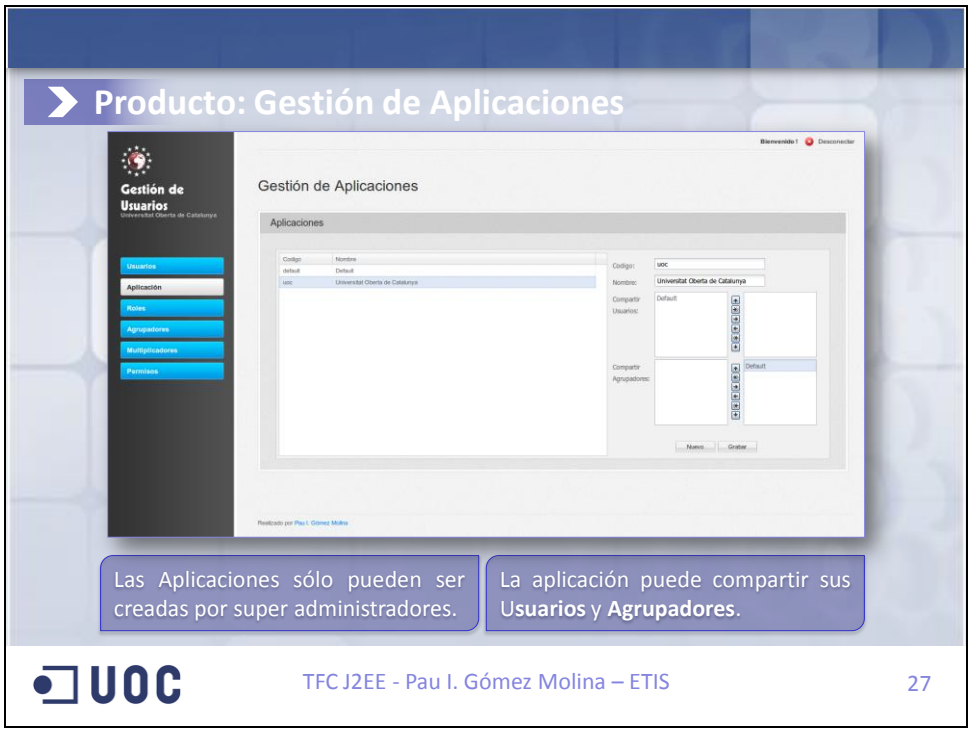


La Gestión de Usuarios está pensada para ser desplegada en un entorno distribuido. Todas las peticiones de aprovisionamiento de usuarios deberán ser realizadas a través del servicio web, por lo tanto sólo se precisa que el servicio se encuentre accesible desde las aplicaciones que deseen consumirlo.

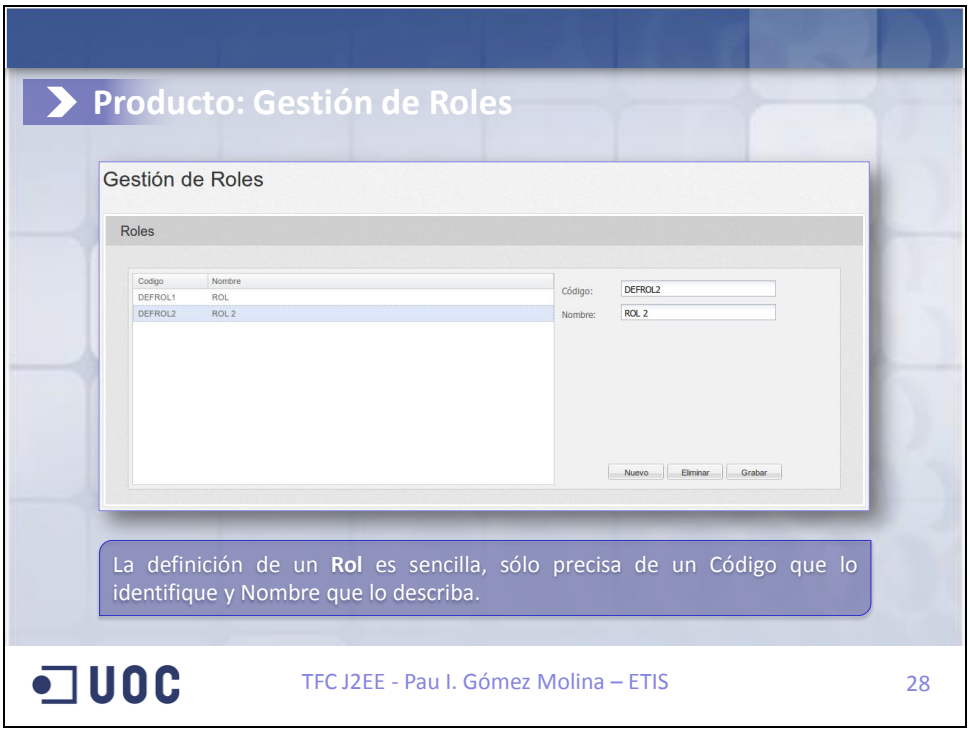
Esta arquitectura ofrece la posibilidad de dar soporte a aplicaciones no desarrolladas por nosotros y que puedan requerir de una gestión genérica como la que aquí se está presentando.



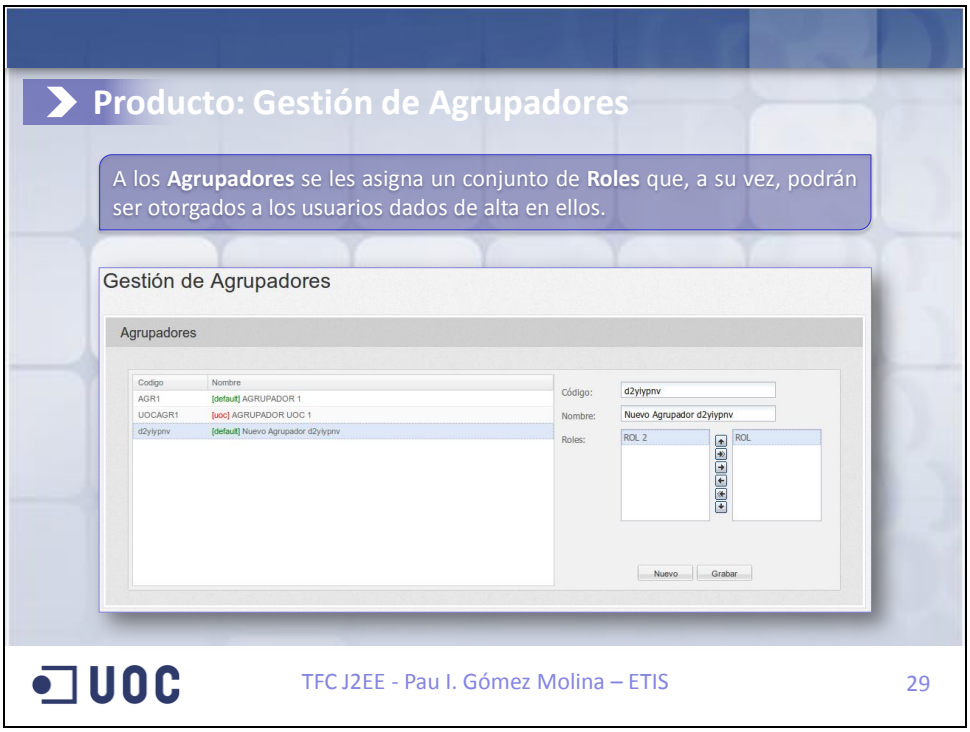
La Gestión de Usuarios al ser un almacén de datos de usuarios y permisos, determina la aplicación que deseamos gestionar mediante la url, por lo que podemos acceder a cualquiera de las que estén dadas de alta utilizando el código de la misma. Por ejemplo, si tuvieramos una aplicación llamada uoc, para poder gestionarla sólo tendríamos que ir a la url *http://HOST:PUERTO/uoc/login.htm*. Aparecerá una pantalla como la de la diapositiva, en dónde podremos loguearnos. Ahora mismo el acceso es libre, sólo es preciso pulsar sobre el botón Acceder y entraremos. La pantalla de login nos 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*.



La Gestión de Aplicaciones permite a un super administrador crear o modificar aplicaciones cliente. Además, a favor de una gran flexibilidad, aquí es posible compartir Usuarios y Agrupadores con otras aplicaciones cliente.

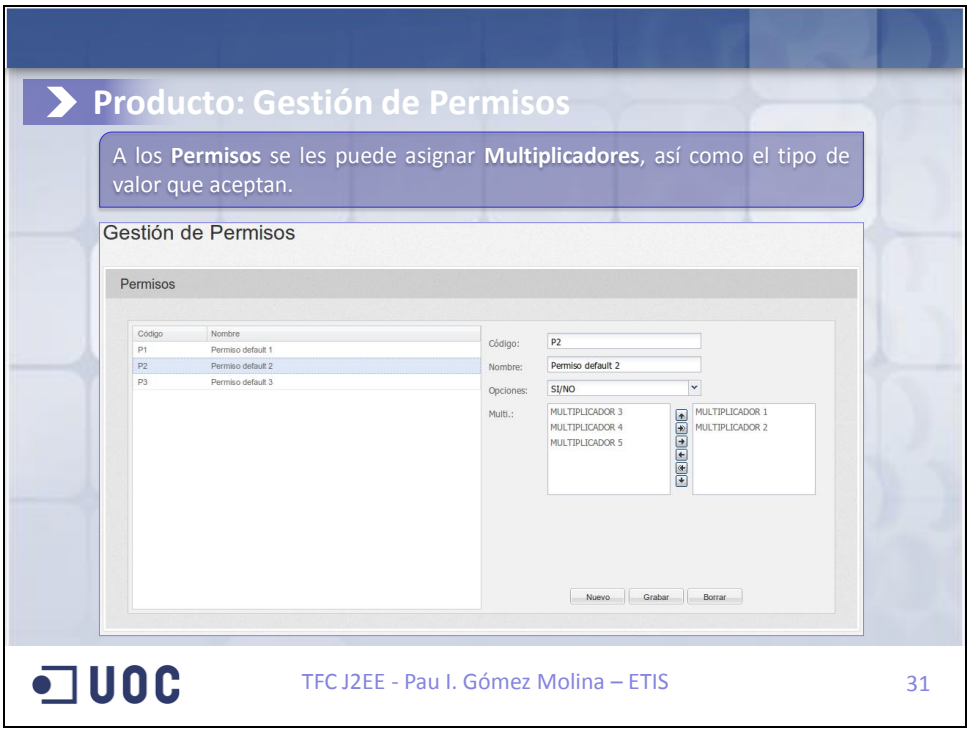


El mantenimiento de roles es muy simple y sólo precisa de un código que lo identifique y un nombre que lo describa.



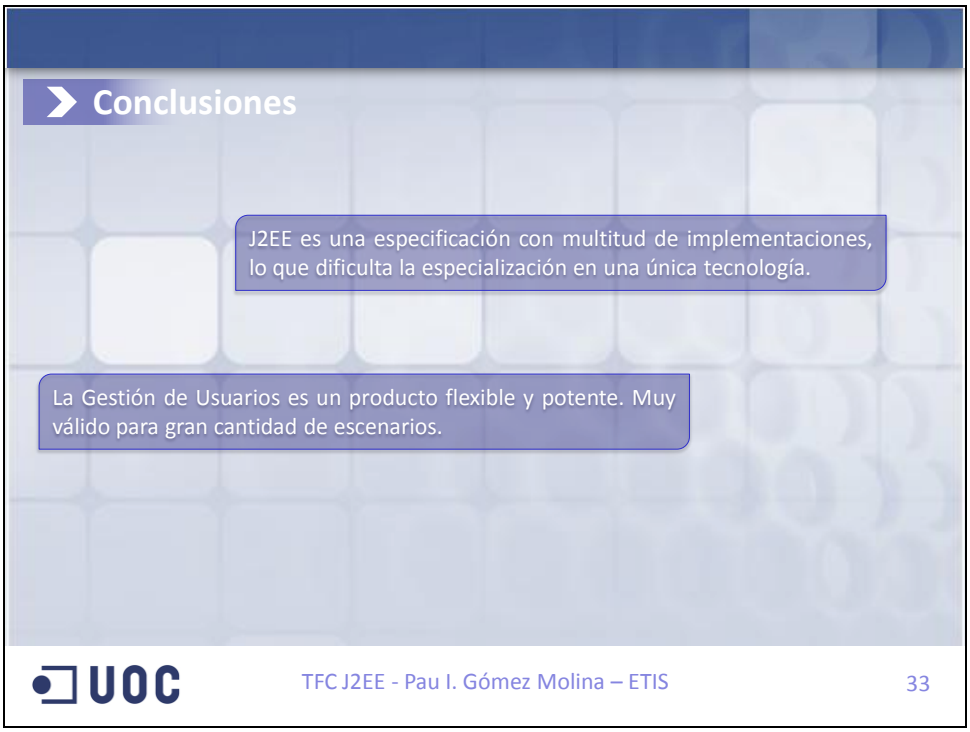
Recordemos que a los Agrupadores se les asignaban los roles que podían otorgarse a sus usuarios, por lo tanto lo más importante de este apartado es que aquí se seleccionan dichos roles para el Agrupador. Además, es interesante resaltar que aquellos agrupadores heredados de otras aplicaciones cliente distintas a la que estemos administrando, aparecerán con su nombre en rojo en el grid. Al no poseer la aplicación que estamos gestionando dichos agrupadores, no nos será posible modificar su código ni descripción, pero sí podremos asignar roles al Agrupador que sólo serán válidos en esta aplicación, sin interferir con los que tengan asignados en la aplicación que lo creó. Aquellos que aparezcan en verde serán Agrupadores propios de la aplicación que gestionamos y por ello podremos realizar cualquier cambio en ellos.

Cada multiplicador tendrá un conjunto de “Detalles” que serán los “multiplicandos”. Por ejemplo, un Multiplicador llamado “Comida” puede tener los detalles “Desayuno”, “Comida” y “Cena”, por lo tanto los permisos a los que se le asigne este Multiplicador tomarán un valor distinto por cada detalle, es decir, 3.



Por último, en la Gestión de Permisos podemos dar de alta nuevos permisos indicando un código que lo identifique, un nombre que lo describa, un tipo de valor (Sí / No o R/W/RW) y qué multiplicadores le afectarán. Siguiendo el ejemplo de la diapositiva anterior, si definimos un permiso llamado “Tomar Café” y le asociamos el Multiplicador “Comida”, tendremos la posibilidad de asignarle un valor al permiso para el “Desayuno”, otro para la “Comida” y un tercero para la “Cena”.

Conclusiones



El desarrollo con herramientas como las utilizadas en este proyecto, favorecen la productividad, ayudan a un mantenimiento mucho más simple y fomentan la reutilización de código. En general J2EE dispone de multitud de especificaciones con un número muy elevado de distintas implementaciones y puede llegar a ser una dura tarea la elección de la opción adecuada. En este caso se ha trabajado con algunas de las más representativas en el panorama actual, lo cual ha resultado altamente gratificante. Mencionaría especialmente a Spring que ha resultado ser una herramienta altamente configurable y con un rendimiento muy bueno.

Aún así, es complicado conocer con detalle todos los frameworks disponibles para J2EE ya que cada uno tiene una considerable curva de aprendizaje, por lo tanto opino que resulta difícil especializarse en una única tecnología cuando ante un nuevo proyecto podemos encontrarnos con un entorno completamente diferente.

En cuanto a la Gestión de Usuarios, he implementado aquellas partes que el tiempo del TFC me ha permitido, pero creo firmemente que es una herramienta extremadamente útil. Sería posible utilizarla en la mayoría de aplicaciones medianas y pequeñas que actualmente se desarrollan. Para finalizar decir que me hubiera gustado implementar, aunque fuera una parte, el servicio web, pero creo que sin él todavía es posible apreciar el objeto de la gestión.