



Connectin

Aplicaciones distribuidas

Convocatoria de septiembre de 2013.

Alumno: Francisco Campillo Asensio

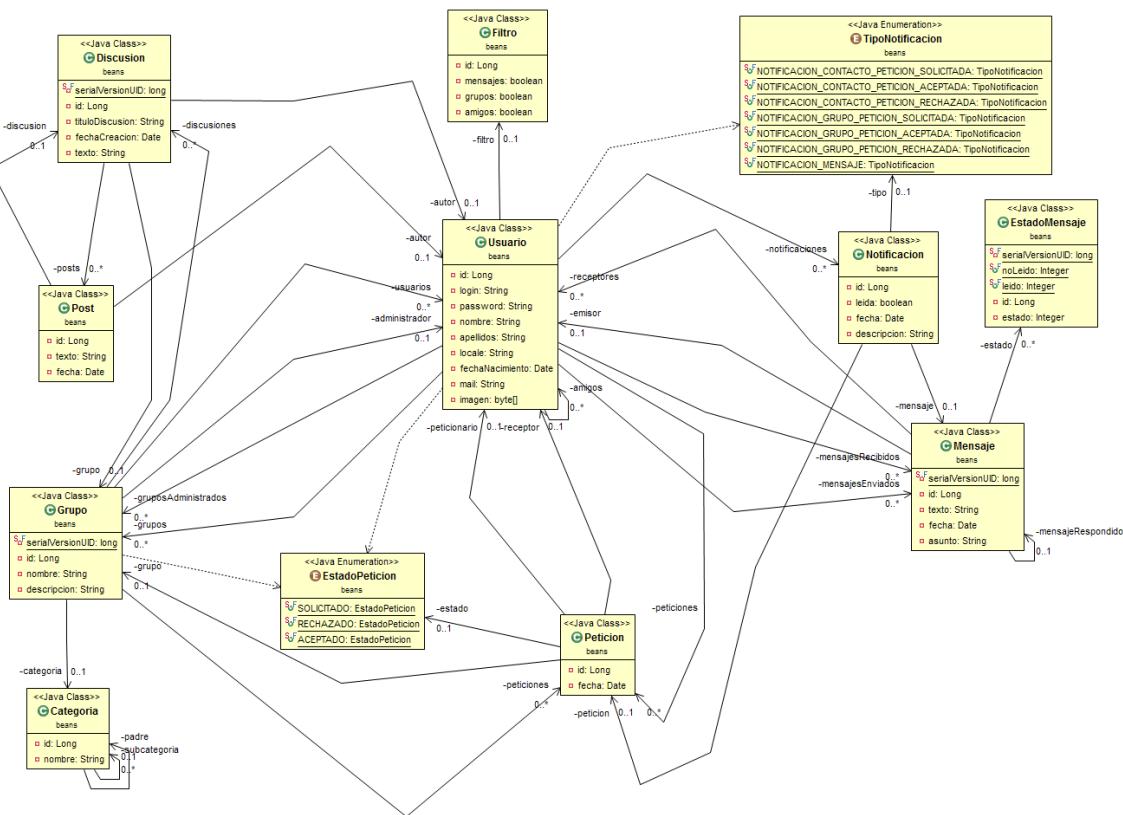
francisco.campillo@um.es

Profesor: Francisco Javier Bermúdez Ruiz

Introducción	3
Instalación y despliegue	5
Descripción técnica de la implementación	7
Capa de presentación y Control (JSF).....	7
Managed Beans (o Backing Beans)	9
Validadores.....	10
Conversores.....	11
Phase Listeners.....	11
Internacionalización	12
Componentes gráficos de PrimeFaces	14
Capa de negocio	18
Data Access Object (DAO)	18
Java Persistence Api (JPA)	19
Enterprise Java Beans (EJB)	21
2 entidades EJB2 CMP2	22
Implementación de un patrón de la capa de negocio: Value Object.....	24
Controlador de sesión para gestionar idioma (EJB3 de sesión)	30
Java Remote Method Invocation (RMI)	33
Visita guiada por la aplicación (desde el punto de vista del usuario)	38
Nuevos aspectos	63
Lambaj	63
Servlet Filters.....	65
Gestión de foto de perfil	67
Requisitos básicos no implementados.....	67
Otros.....	67

Introducción

El problema que se presenta en la práctica es la implementación de una aplicación web para la red social ConnectIn. A través de la aplicación, los usuarios podrán agregar contactos, registrarse en grupos, intercambiar mensajes y postear.



Veamos las entidades que aparecen en el dominio del problema a resolver.

Una discusión es lo que da comienzo a un hilo de post dentro de un grupo y es iniciada por un usuario el cual es considerado autor de la discusión. En respuesta a la discusión los usuarios escriben una serie de Posts a esa discusión.

Un usuario representa a la persona que se conecta a la aplicación, al que la utiliza. Un usuario puede enviar mensajes a otros usuarios, crear grupos, crear discusiones en los grupos a los que pertenece y responder a discusiones creadas por el u otros usuarios con la creación de un post, solicitar entrar en grupos, solicitar amistad a otros usuarios y aceptar estas solicitudes provenientes de otros usuarios.

El usuario recordara la información que quiere ver en su panel de notificaciones, tendrá por lo tanto asociado un filtro.

En el contexto de esta aplicación una categoría nos sirve para etiquetar grupos para su mejor localización y descubrimiento según los intereses de los usuarios. Una categoría tiene un carácter jerárquico en forma de árbol, es decir una categoría tiene una serie de categorías hijas pero solo puede ser hija de una categoría.

Una petición encapsula tanto a las solicitudes para ser amigos entre dos usuarios como solicitudes para entrar en un grupo. Una petición tiene un ciclo de vida de 3 estados posibles, solicitada y aceptada o rechazada.

Un grupo esta categorizado dentro de una categoría tiene un administrador que es el usuario que lo creó. Tiene una serie de discusiones creadas por los usuarios pertenecientes al grupo.

En la aplicación cada evento relevante genera una notificación:

Una notificación puede ser de diversos tipos. Puede originarse cuando un usuario te ha mandado una petición de amistad, cuando una petición de amistad que mandaste ha sido rechazada o aceptada, de la misma forma sucede con las peticiones de ingreso en grupos siendo el administrador del grupo el que recibe las notificaciones de petición. Finalmente también recibimos una notificación al recibir un mensaje.

Una notificación se crea para informar sobre la ocurrencia de eventos como puede ser nuevos mensajes recibidos, solicitudes de amistad o de ingresar en grupos que el usuario administra.

Los usuarios pueden mandarse mensajes entre sí. Un mensaje se compone de un cuerpo, un asunto, un emisor, unos destinatarios y se hace referencia otro mensaje si este es respuesta del otro mensaje.

Cuando se envía un mensaje el mensaje se mantiene en estado no leído. Cuando los destinatarios van leyendo el mensaje este va pasando a respondido.

En realidad un mensaje tiene una lista con los estados para cada destinatario.

Instalación y despliegue

Para desplegar la aplicación necesitamos.

- Servidor mysql y una base de datos creada llamada “aadd20122013b”.
- Un servidor de aplicaciones Jboss 6.

Movemos el fichero suministrado aadd-ds.xml a la ruta:

Path_to_jboss\jboss-6.1.0.Final\server\default\deploy\aadd-ds.xml

Modificamos el fichero standardjbosscmp-jdbc.xml con los siguientes datos.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jbosscmp-jdbc PUBLIC
  "-//JBoss//DTD JBOSSCMP-JDBC 4.0//EN"
  "http://www.jboss.org/j2ee/dtd/jbosscmp-jdbc_4_0.dtd">

<!-- ===== -->
<!-- Standard JBossCMP-JDBC Configuration -->
<!-- ===== -->
<!-- $Id: standardjbosscmp-jdbc.xml 77362 2008-08-22 15:09:04Z
alex.loubyansky@jboss.com $ -->

<jbosscmp-jdbc>

  <defaults>
    <datasource>java:/DataSourceDAD</datasource>
      <!-- optional since 4.0 <datasource-mapping>Hypersonic
SQL</datasource-mapping> -->
    <datasource-mapping>mySQL</datasource-mapping>
    ...
  </defaults>
  ...
</jbosscmp-jdbc>
```

Path_to_jboss\jboss-6.1.0.Final\server\default\conf\standardjbosscmp-jdbc.xml

Colocamos las librerías eclipselink.jar y mysql-connector-java-5.1.21-bin.jar en
Path_to_jboss\jboss-6.1.0.Final\server\default\lib

Movemos la carpeta rmi a un sitio localizable (los siguientes datos corresponderán con C:\rmi)

Si todavía no existe en la variable de entorno PATH abrimos una consola e introducimos el siguiente comando:

```
set PATH=%PATH%;E:\Program Files\Java\jdk1.6.xxxxx\bin
```

jdk1.6.xxxxx sea la carpeta donde tenemos el jdk.

Una vez hecho esto introducimos los siguientes 2 comandos:

```
set CLASSPATH=
rmiregistry
```

Abrimos otra consola he introducimos los siguientes comandos:

```
C:
cd rmi
java -Djava.security.policy=politicas.txt -Djava.rmi.server.codebase=file:///c:/rmi/
mirmi.Servidor
```

Ya tenemos funcionando el proceso servidor que proporciona acceso al objeto distribuido rmi GestorMails.

Arrancamos el servidor mysql.

Copiamos los ficheros ConnectInWeb.war, ConnectinDistEJBCMP2.jar y ConnectInDistEJB3.jar en la carpeta de despliegue Path_to_jboss\jboss-6.1.0.Final\server\default\deploy\

Debemos añadir 2 parámetros al servidor para que funcione la parte de RMI.

En las opciones de launch configurations de jboss:

```
"-Djava.security.policy=c:/rmi/politicas.txt" "-Djava.rmi.server.codebase=file:///c:/rmi/"
```

Finalmente arrancamos el servidor jboss.

Toda esta información y los ficheros necesarios para el despliegue se pueden encontrar en el fichero PracticaFinal.zip

Descripción técnica de la implementación

Las tecnologías utilizadas para el desarrollo del caso práctico han sido:

Vista y MVC: Java Server Faces con la utilización de componentes de PrimeFaces.

Negocio: JPA y EJB CMP2 de entidad (2 entidades aisladas a las del resto del modelo)

1 objeto distribuido RMI.

1 un controlador de sesión EJB3 que cuya misión es recordar el idioma utilizado para representar las cadenas internacionalizadas.

JMS no implementado. Me permitió no repetir esta parte ya que meses antes hice la práctica de JMS en Arquitectura del software.

Esta primera aproximación me refiero a las tecnologías que he elegido para cumplir con las características que se pide en el enunciado del caso práctico, para otras librerías/tecnologías las trataremos en el capítulo Nuevos aspectos.

Capa de presentación y Control (JSF)

Para la vista he utilizado Java Server Faces con Facelets.

JavaServer Faces (JSF) es una tecnología y framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

JSF usa JavaServer Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías.

Facelets es un lenguaje de declaración de páginas, poderoso pero ligero, que es usado para construir vistas de JavaServer Faces usando plantillas de estilo de HTML y construyendo arboles de componentes.

Facelets nos permite crear páginas web mediante el uso de XHTML, soporta librerías de etiquetas Facelets que se suman a las librerías JavaServerFaces y JSTL, soportan EL y nos permite usar plantillas para el diseño de páginas y componentes.

Todo esto conlleva una serie de ventajas como una mayor reutilización del código por medio de las plantillas y componentes compuestos, extensibilidad funcional de componentes , validación EL en tiempo de compilación etc.,

Haciendo uso de estas plantillas he definido un marco común para todas las páginas de forma que el aspecto de toda la aplicación se mantiene homogéneo y el código html que define las partes comunes como cabecera o barra lateral se mantiene desde un solo fichero y no se replica en cada una de las páginas con las consiguientes ventajas.

JSF nos proporciona una serie de mecanismos que ayudan en las tareas cotidianas del desarrollo web. Aparte de las ya mencionadas haremos un repaso de las características usadas.

Las clases que se utilizan para implementar características suelen ir declaradas mediante un fichero de configuración llamado faces-config.xml o bien mediante anotaciones.

Después de probar las dos formas me parece mucho más cómodo y reutilizable las anotaciones.

Managed Beans (o Backing Beans)

En JSF más que JavaBeans se usan Managed Beans, que no son más que JavaBeans especiales a los cuales se les asocia un componente de interfaz de usuario de la siguiente forma.

```
<h:outputText value="#{MiManagedBean.nombre}" />
```

```
<h:inputText value="#{MiManagedBean.nombre}" />
```

El programador de la vista se despreocupa de hacer llegar los datos del servidor al cliente y viceversa y lo hace de manera transparente, además un bean puede definir métodos para ser lanzados cuando ciertos eventos se produzcan (click en un commandButton, cambio de valor en un selectOneMenu...).

Los managed beans pueden tener distintos “scopes” el tipo que más se utiliza es @ViewScoped sobre todo en formularios en los que el valor elegido en algún campo condiciona las opciones a elegir en otro campo.

Un @ViewScoped @ManagedBean se crea cuando accedemos a una página jsf que tiene alguna referencia al bean. En ese momento se ejecuta el constructor del bean y en las posteriores interacciones se hace uso del bean creado en un primer momento.

Un @ RequestScoped @ ManagedBean no permitiría por ejemplo tener un componente selectOneMenu que carga un segundo selectOneMenu con subopciones puesto que las instancias solamente existen durante la petición HTTP. Una vez que termina el ciclo de vida de la petición, también termina su contexto.

Si se puede usar en un formulario de entrada en el que no se produzca esa circunstancia o en una bean que se limite a proporcionar información normalmente recolectada de algún sitio en el momento de su construcción por ejemplo RegistroUsuarioBean.

Un @SessionScoped @ManagedBean lo utilizo para mantener datos entre peticiones, el usuario puede ir visitando una serie de páginas y un bean de sesión retiene esa información que es almacenada en este.

Inicialmente recordar el usuario logueado en la aplicación lo hubiera hecho con un bean de sesión sin embargo por requisitos de características de la práctica esta información se registra de otra forma a través de un Servlet.

También por modificaciones derivadas de exigencias de la práctica un Bean de sesión era LocaleBean que mantenía el lenguaje utilizado actualmente dejó de necesitar ser @SessionScoped pues delegó en un EJB de sesión esta tarea aunque en la primera versión de la práctica este es un @SessionScoped @ManagedBean.

Validadores

JSF dispone de varios controles para la introducción de datos: campos de texto (ocultos, normales o de contraseña), áreas de texto y controles de selección, tanto múltiple como individual. Todos ellos aceptan varias formas de validación de los datos introducidos:

- Existen propiedades autónomas de la etiqueta del control que pueden cargarse como true o false para adaptar el comportamiento del control cuando se realiza el envío (summit) del formulario.
- Es posible que exista un método que realice la validación de los datos introducidos en el control. Este método se encuentra en el bean de respaldo (backbean) o de otro que exista en el alcance de la página. En este caso, el método se indica mediante una expresión EL de JSF en una de las propiedades del control.
- Invocación de un método validate del objeto que implementa el control (binding).
- De igual forma, existen etiquetas específicas para asignar un cierto validador a un control. Esta etiqueta se crea como hija del control, permitiéndose más de una. Esta es la forma de añadir los validadores a los que por defecto proporciona JSF y los que se hayan creado en la aplicación como clases independientes.

Una vez que la página ha sido enviada, los validadores asociados a un control son invocados y el control se considerará válido cuando todos sus validadores sean válidos. Si el proceso de validación falla, los datos no se cargan en el modelo de datos permitiendo a la fase de validación que genere la respuesta al usuario. Cada método de validación, en el caso de que no considere válido el valor del control, debe crear una instancia de FacesMessage en donde incluirá un mensaje alusivo. Este FacesMessage se rodeará de una excepción ValidatorException que será lanzada por el método. Luego será responsabilidad del resto del ciclo de vida mostrar al usuario de forma apropiada los mensajes de error.

Un buen ejemplo de los validadores utilizados lo encontramos en register.xhtml donde se ha puesto más empeño en la robustez en la entrada de datos.

```
<f:validator validatorId="emailExistValidator"/>

<f:validator validatorId="loginNameExistValidator"/>

<f:validateLength minimum="2" maximum="128" />

<f:validateRegex pattern="([A-ZÁÉÍÓÚÑ][a-záéíóúñ]{1,}) ([\s][A-ZÁÉÍÓÚÑ][a-záéíóúñ]{1,})*" />

<p:inputText value="#{registroUsuarioBean.login}" required="true"
validatorMessage="#{msg.login}:#{registroUsuarioBean.login}#{msg.error_login_field}"/>
```

Tenemos dos validadores personalizados, emailExistValidator y loginNameExistValidator.

Para hacer un validador personalizado basta con crear una clase que implemente javax.faces.validator.Validator e implemente su método

```
public void validate(FacesContext arg0, UIComponent arg1, Object arg2)
```

Dentro de este método se harán las operaciones necesarias para validar y en caso de que no se cumplan se lanza una ValidatorException. Lo que comprueban estos dos validadores es evidente dado sus nombres, si el email ya existe o si el login ya existe.

El ejemplo mostrado arriba de validateRegex nos sirve por ejemplo en el campo de nombre para permitir solo nombres que empiezen por 1 Mayúscula luego minúsculas y si es nombre compuesto el resto de nombres de la misma forma. Este validador se aplica de la misma forma en el campo de apellidos.

Conversores

JSF proporciona dos mecanismos separados que nos ayuda a validar los valores introducidos por los usuarios a la hora de submitir los formularios. Acabamos de ver uno de ellos que es el mecanismo de validación.

El otro mecanismo es el de conversión y está definido por el interfaz javax.faces.convert.Converter y sus múltiples implementaciones.

Las conversiones aseguran que el tipo de un dato introducido en un formulario JSF sea el correcto, es decir, que el dato tipo cadena del formulario corresponde con el tipo JAVA esperado, y que está especificado en la propiedad correspondiente del bean.

Los Conversores (implementaciones de la interfaz Converter) son los componentes que se encargan de hacer estas transformaciones (cadena>Tipo JAVA y viceversa). JSF invoca a los Conversores antes de efectuar las validaciones y por lo tanto antes de aplicar los valores introducidos a las propiedades del bean.

En el caso de que un dato tipo cadena no se corresponda con el tipo JAVA apropiado, el Conversor correspondiente lanzará un ConversionException y el componente se marcará como invalidado.

En la práctica se usan converters para dos entidades CategoriaConverter y UsuarioConverter.

Phase Listeners

Un Phase Listener es una clase que se registra como oyente de eventos que se producen en el paso de una fase a otra dentro del ciclo de vida de una página JSF.

Las 6 fases de la vida JSF son:

- Restaurar vista.
- Asignar valores de petición.
- Realizar validaciones.
- Actualizar los valores del modelo.
- Invocar la aplicación.
- Presentar la respuesta.

En un momento dado se presentó la necesidad de mostrar un mensaje en una página a la que se redirige después de finalizar el proceso de registro. Si después de registrar con éxito a un

usuario lanzaba un FacesMessage, el mensaje no se visualizaba al redirigir de register.faces a login.faces por las características del ciclo de vida de páginas JSF.

Para permitir esto lo que hacemos es definir un PhaseListener que haga lo siguiente:

Después de cada fase donde los mensajes pueden ser añadidos, movemos los mensajes del contexto JSF a un mapa registrado como de sesión, de lo contrario al moverlos a otra página estos mensajes se perderían.

Antes de cada fase volvemos de vuelta los mensajes del mapa en el que almacenamos los mensajes al contexto JSF de nuevo.

Sólo movemos los mensajes globales, no los que están relacionados con un componente concreto, estos últimos no pueden ser representados en una página diferente a la que fue originado.

Internacionalización

JSF proporciona un mecanismo de internacionalización para que nuestra aplicación pueda ser vista en distintos idiomas.

Se basa en el uso de ficheros de propiedades .

Estos son una colección de pares clave =valor por ejemplo: error_name_too_long = nombre demasiado largo.

Establecemos un fichero con un determinado nombre por ejemplo MessageResources.properties este fichero almacenará los valores por defecto en caso de no seleccionar ningún idioma.

Para cada uno de los idiomas creamos un fichero con el mismo nombre acabado en _xxx donde xxx es el código ISO del idioma o bien del idioma + zona.

Luego ponemos en nuestros facelets:

```
<f:loadBundle basename="resource.MessageResources" var="msg" />  
<f:view locale="#{languageBean.localeCode}">  
...  
#{msg.unaclave}  
<f:view/>
```

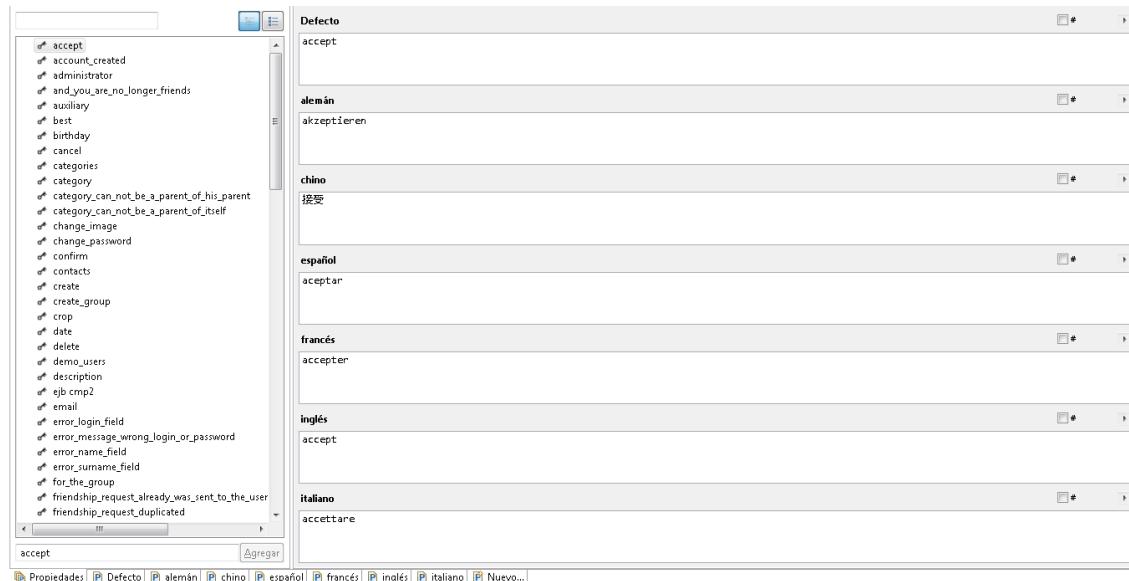
Donde locale toma valores como es, en, de, zh, it...

Para hacer cómodo el proceso de internacionalización mi manera de proceder ha sido no traducir nada hasta el final e ir poniendo mensajes sin traducción por todo el proyecto, se visualizan los mensajes cómo ???clave???.

Las claves las voy guardando en una hoja de cálculo de google docs y cuando decido traducir voy rellenando las columnas de los idiomas que quiera traducir a mano y los que no cojo las traducciones en inglés y las paso al traductor automático, de esta forma la presencia de más idiomas no supone mucho más trabajo. Luego simplemente se copian las distintas columnas para cada idioma con sus pares clave = valor y se pegan en los respectivos ficheros de internacionalización con la ventaja de que se mantiene la codificación de caracteres.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	en	es	de	fr	zh	it		en	es	de	it	fr	zh	it
2	accept	accept	aceptar	akzeptieren	accepter 接受	accettare		accept = accept	accept = acceptar	accept = akzeptieren	accept = acceper	accept = accepter	accept = accettare	
3	and_you_are_no_longer_friends	and you are no longer friends	y tú ya no sois amigos	und Sie sind keine Freunde mehr	et vous n'êtes plus amis	e non si è più amici		and_you_are_no_... = and you are no longer friends	and_you_are_no_... = und Sie sind keine Freunde mehr	and_you_are_no_... = e non si è più amici	and_you_are_no_... = and you are no longer friends	and_you_are_no_... = y tú ya no sois amigos	and_you_are_no_... = 你不再是朋友	and_you_are_no_... = and you are no longer friends
4	auxiliary	auxiliary	auxiliar	auxiliaire	auxiliar 铅笔的	auxiliare		auxiliary = auxiliary	auxiliary = auxiliar	auxiliary = auxiliaire	auxiliary = auxiliar	auxiliary = auxiliare	auxiliary = auxiliar	
5	best	best	la mejor	besten	meilleur	migliore		best = best	best = mejor	best = besten	best = meilleur	best = migliore	best = migliore	
6	birthday	birthday	fecha de nacimiento	Geburtsst.	anniversaire	compleanno		birthday = birthday	birthday = fecha de nacimiento	birthday = Geburtstag	birthday = anniversaire	birthday =生日	birthday = compleanno	
7	cancel	cancel	cancelar	storniere	réšílier	annullare		cancel = cancel	cancel = cancelar	cancel = stornieren	cancel = réšílier	cancel = annullare	cancel = 取消	
8	category	category	categoría	Categorie	类别	categorie		category = category	category = categoría	category = Categorie	category = categorie	category = categoria	category = 类别	
9	change_image	change image	cambiar imagen	Bild ändern	changer l'image	cambiare immagine		change_image = change image	change_image = cambiar imagen	change_image = Bild ändern	change_image = changer l'image	change_image = cambiare immagine	change_image = 改变形象	
10	change_password	change password	cambiar clave	Kenntwort de ändern	修改密 码	modifica la password		change_password = change password	change_password = cambiar clave	change_password = Kenntwort ändern	change_password = changer la password	change_password = modificare la password	change_password = 更改密码	
11	confirm	confirm	confirmar	bestätigen	确认	confirmare		confirm = confirm	confirm = confirmar	confirm = bestätigen	confirm = confirmer	confirm = confermare	confirm = 确认	
12	contacts	contacts	contactos	Kontakte	联系人	contatti		contacts = contacts	contacts = contactos	contacts = Kontakte	contacts = contatti	contacts = 联系人	contacts = 联系人	
13	create_group	create group	crear grupo	Gruppe erstellen	创建组	creare gruppo		create_group = create group	create_group = crear grupo	create_group = Gruppe erstellen	create_group = creare gruppo	create_group = 创建组	create_group = 创建组	
14	crop	crop	recortar	Entfernen	cultures	移除		crop = crop	crop = recortar	crop = entfernen	crop = 去除	crop = cultures	crop = 移除	
15	date	date	fecha	Datum	Date 日	data		date = date	date = fecha	date = Datum	date = Date	date = 日	date = data	

Una herramienta que puede ayudar a hacer más cómodo la gestión de la internacionalización es un plugin para eclipse que se llama [Resource Bundle Editor](#) el cual te permite seleccionar una clave y llenar su valor para todos los idiomas.



Además de los mensajes personalizados también podemos manejar traducciones de mensajes lanzados por otras librerías, para esto basta con localizar la librería en cuestión y revisar los ficheros de internacionalización que contiene para luego definir traducciones para esas claves.

Componentes gráficos de PrimeFaces

PrimeFaces es una librería de componentes enriquecidos JSF de código abierto, hace uso de jquery y jquery ui y contiene una gran cantidad de componentes que facilitan las labores de desarrollo de interfaces de usuario.

Esta es una lista de los principales componentes utilizados:

Algunos realizan la misma tarea que los componentes de la implementación de JSF pero le aplica el aspecto del [tema](#) seleccionado otros componentes son nuevos con respecto a la implementación de JSF o añaden características adicionales a estos. Además unos componentes llevan una representación gráfica y otros no (azul).

outputPanel	commandButton	graphicImage	radioButton
accordionPanel	commandLink	growl	selectBooleanCheckbox
panel	confirmDialog	imageCropper	selectManyMenu
password	dataTable	inputText	outputPanel
autoComplete	dialog	inputTextarea	ajaxStatus
breadCrumb	editor	messages	ajax
calendar	fileUpload	outputLabel	hotkey
tree	hotkey	selectOneMenu	

Veamos algunos de estos:

breadCrumb: Presenta un menú que nos indica donde estamos a modo de barra horizontal con enlaces.

calendar:Este componente nos permite seleccionar una fecha y guardarla directamente en un objeto de tipo Date en el Backing Bean.

fileUpload y imageCropper: Con fileUpload podemos subir un fichero al servidor con distintas opciones para gestionar la subida. ImageCropper nos presenta una imagen y un recuadro que podemos situar y dimensionar como queramos para después obtener una imagen contenida en la selección realizada (requiere programación en el backing bean para realizar esto pero el componente facilita las cosas).

ajaxStatus: Nos permite mostrar lo que queramos mientras estamos procesando peticiones Ajax. En el caso de la práctica lo que se muestra es un diálogo con un gif animado en el centro y una cabecera que reza “Procesando datos”.

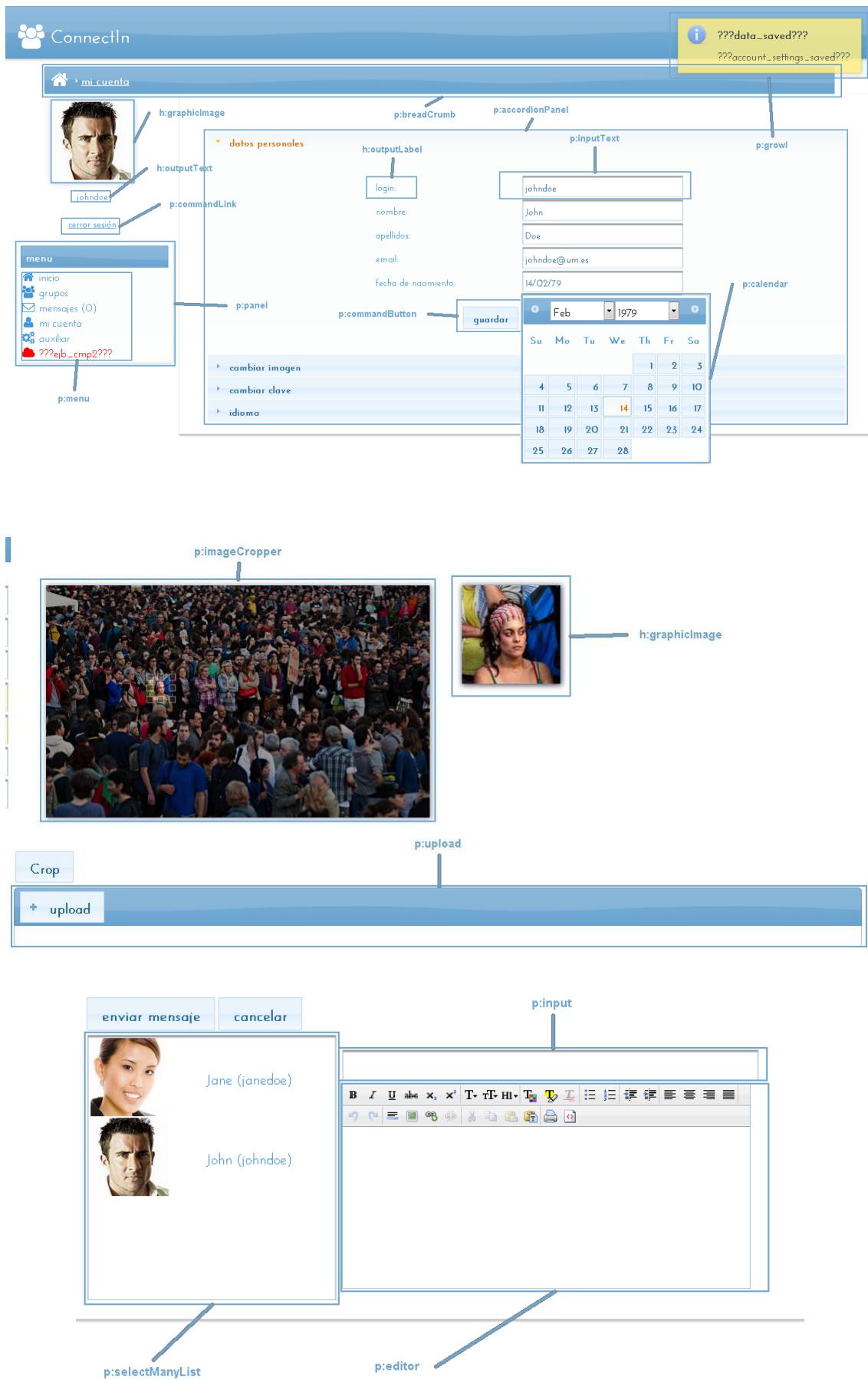
Tree: Este componente nos permite presentar datos en forma de árbol.

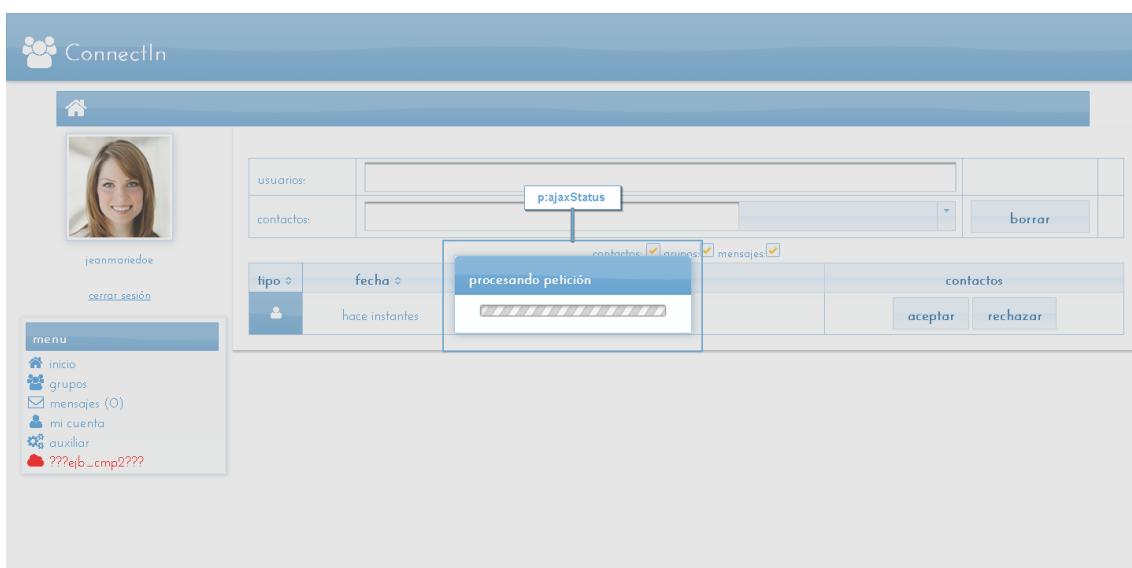
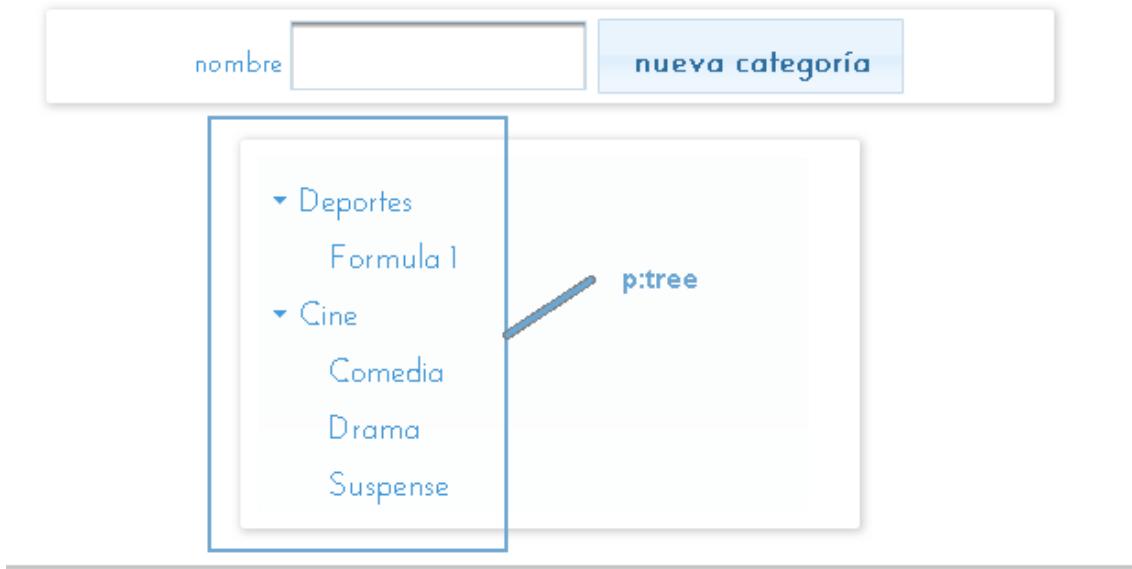
dataTable: Componente mejorado con respecto al ofrecido por la implementación estándar de JSF como por ejemplo el estilizado de la tabla con el tema elegido, filtros Ajax, orden encolumnas etc.

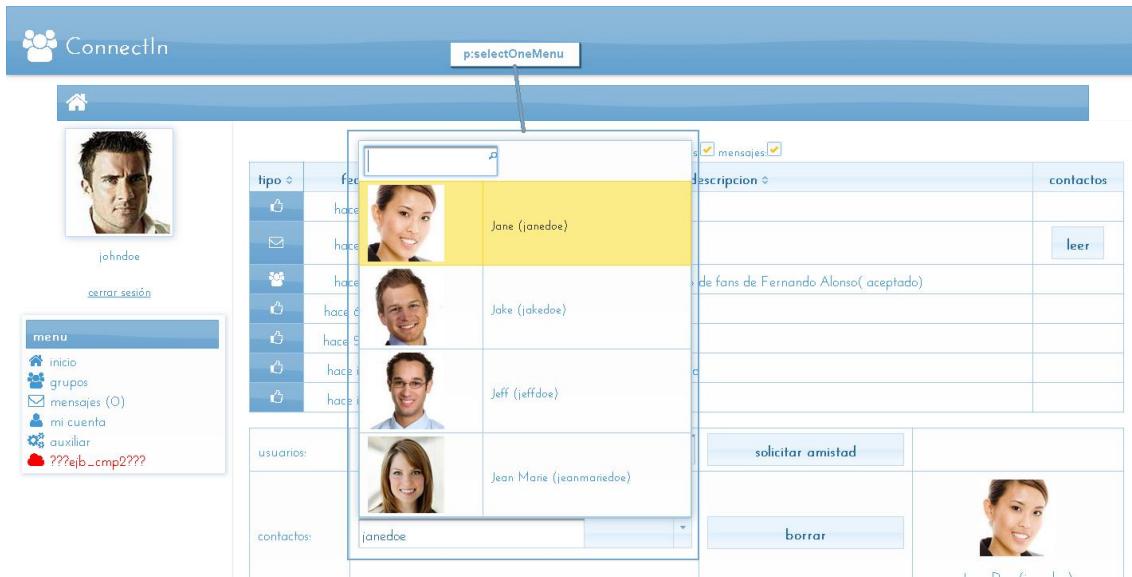
Growl: Muestra un mensaje de notificación muy familiar para los usuarios de mac.

Messages: Hace lo mismo que growl pero muestra la información de otra forma.

Panel: este componente sirve de contenedor para otros componentes y mantiene un aspecto coherente con el resto de componentes.







Capa de negocio

En la capa de negocio nos encontramos con las siguientes tecnologías: JPA, EJB3 de sesión , EJB2 CMP2 de entidad y RMI.

Data Access Object (DAO)

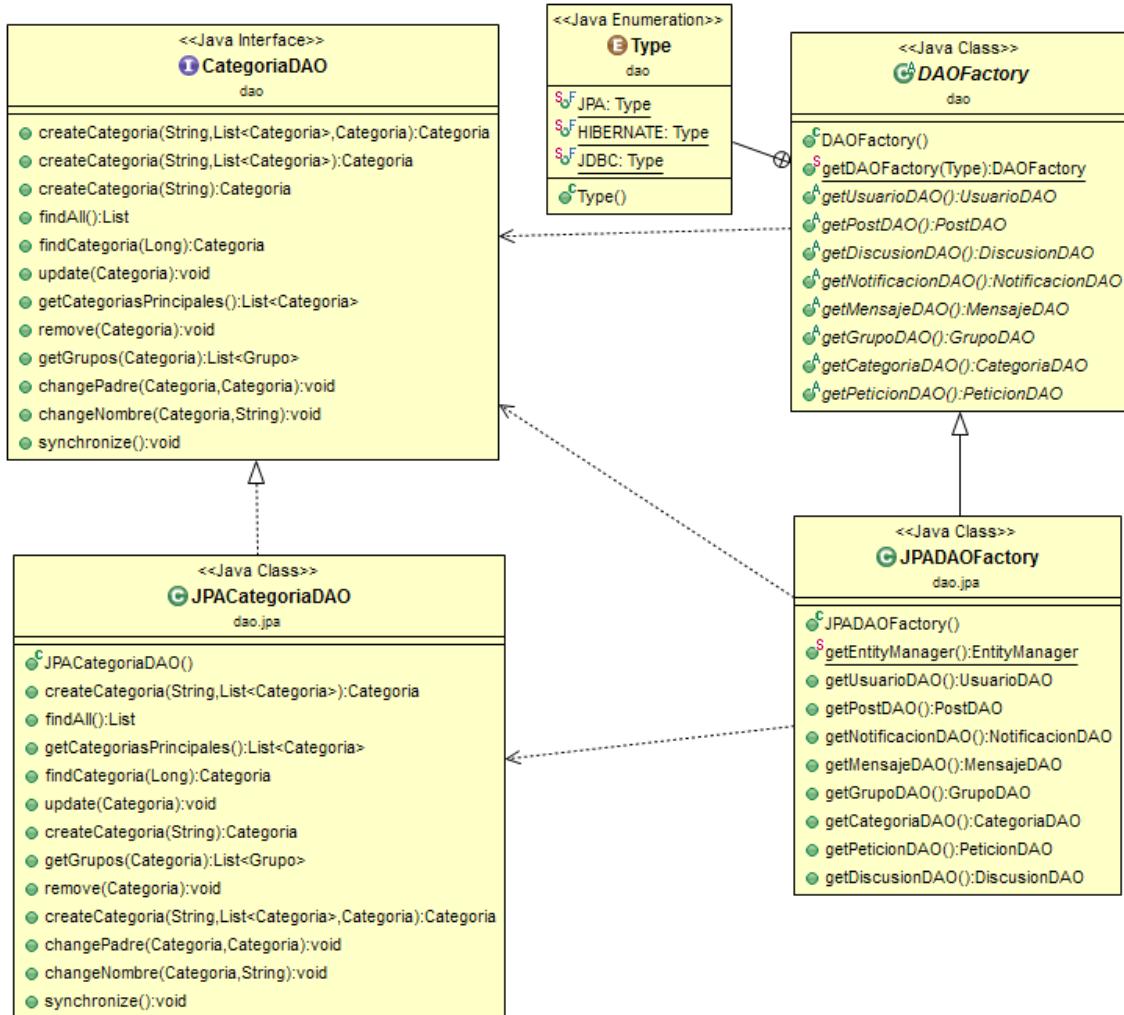
Dentro de las aplicaciones empresariales, el principal objetivo es almacenar información de alguna manera y en algún lugar, este lugar puede ser una base de datos relacional, una base de datos orientada a objetos archivos planos, o alguna otra forma que se les ocurra.

Así que para cada mecanismo de almacenamiento, existen múltiples formas de acceder a la fuente física de los datos y esto es algo que al negocio no le debería de importar, por tal motivo se debe de abstraer todo este conocimiento y envolverlo en una capa.

El patrón DAO muestra una forma de envolver ese conocimiento y evitar que el negocio se manche las manos con esta ardua tarea, lo que significa que la explotación de los datos se hará mediante objetos DAO. La información que se obtiene de las fuentes de datos es convertida o encapsulada a un objeto de tipo TransferObject, o alguna colección de estos.

Para implementar un patrón dao entran en juego una serie de patrones como Factoría Abstracta (los distintos tipos de persistencias son familias de productos y las entidades del modelo serían productos de estas familias), adaptador, método factoría (public abstract CategoriaDAO getCategoryDAO();) y singleton (no lo he implementado como un singleton).

En las siguientes imágenes vemos el diagrama de clases mostrando solo la entidad Categoría.



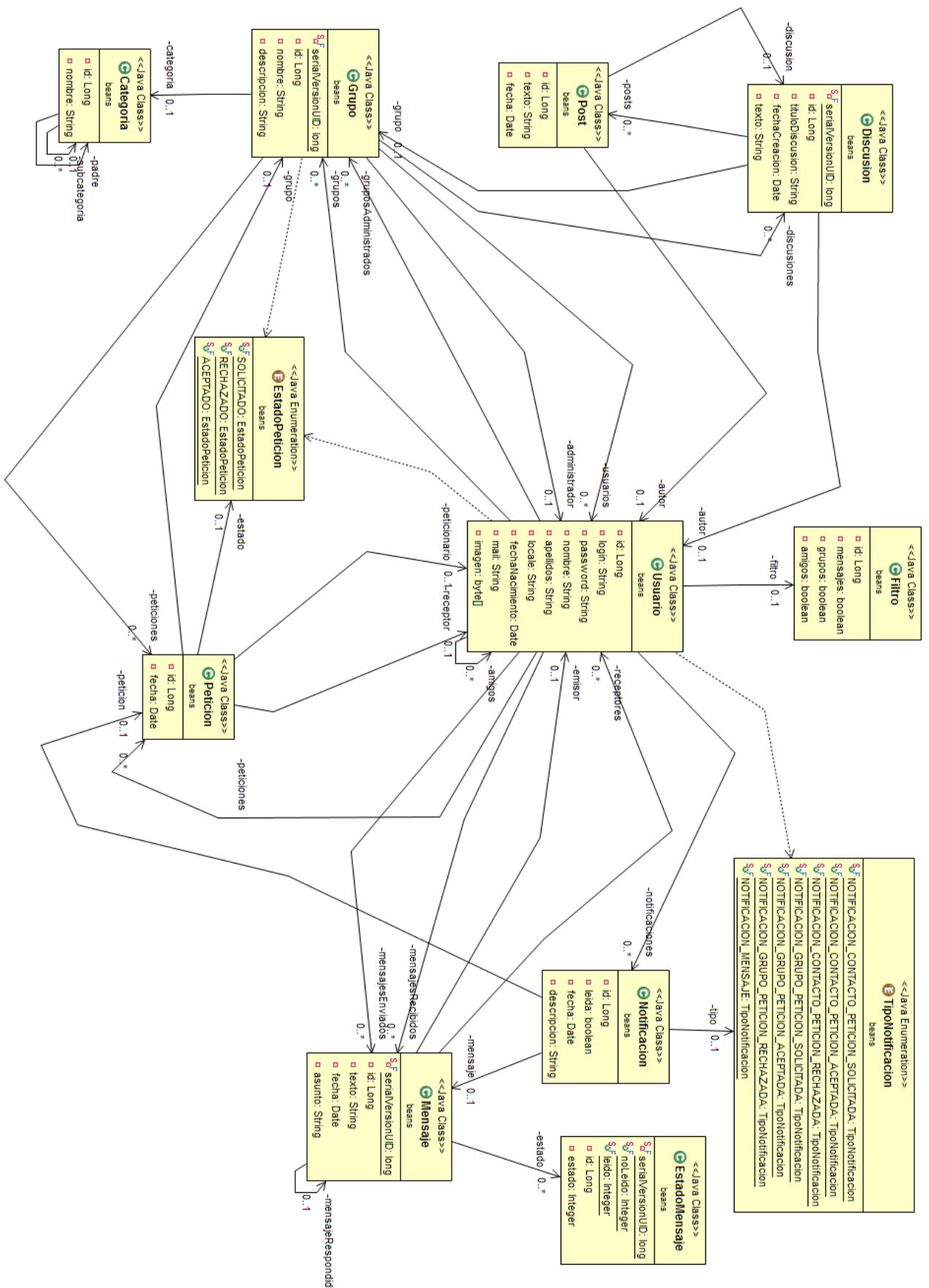
Java Persistence Api (JPA)

JPA al igual que manualmente mediante el diseño del dao nos permite abstraernos del tipo de almacenamiento, aunque este depende de los tipos de persistencia soportados por la api en sus distintas versiones.

Proporciona un modelo de persistencia basado en POJO's para mapear bases de datos relacionales en Java.

Para ello, combina ideas y conceptos de los principales frameworks de persistencia, como Hibernate, Toplink y JDO. El mapeo objeto se realiza mediante anotaciones en las propias clases de entidad.

Veamos ahora las entidades persistidas mediante JPA.



Enterprise Java Beans (EJB)

JAVA es especialmente popular en el desarrollo de grandes aplicaciones empresariales. La plataforma Java 2 Enterprise Edition (J2EE) provee grandes oportunidades para el desarrollo de sistemas distribuidos, se utiliza en la mayoría de las aplicaciones empresariales y la tecnología Enterprise JavaBeans (EJB) es una parte muy importante de la misma.

Usualmente la arquitectura de una aplicación J2EE contiene varias capas separadas. La capa de servidor típicamente contiene componentes de servidor con lógica de negocio, estos son manejados por un contenedor EJB.

El contenedor EJB es parte del servidor de aplicaciones. El servidor de aplicaciones provee el ciclo de vida de los componentes, así como servicios de seguridad y manejo de transacciones.

En el caso práctico se implementan 1 EJB3 de sesión cuya misión es controlar el idioma que está usando cada usuario en la sesión actual y 2 EJB2 CMP2 de entidad.

El objetivo principal de EJB 3.0, fue simplificar el desarrollo de aplicaciones java y estandarizar el API de persistencia. Para ello, presenta una serie de novedades con respecto a EJB2 como son:

- No precisa de contenedor. Funciona tanto en J2EE como J2SE.
- Puede usarse independientemente del resto de los servicios.
- Metadatos con Anotaciones: Se eliminan los deployment descriptors, se reduce drásticamente el número de clases a crear.
- Configuraciones por defecto: reducen la cantidad de configuración a especificar.
- No Intrusión: los objetos a persistir (Entity Beans) no necesitan implementar interfacesEJB.
- Herencia y poliformismo.
- Lenguaje de consulta (EJBQL) mejorado: inner and outer join, operaciones bulk,sql nativo.

Gracias a estas novedades, EJB 3.0, aporta las siguientes ventajas:

- Testing.
- Simplicidad: una única clase para declarar la persistencia (gracias a anotaciones).
- Facilidad de aprendizaje.
- Transparencia: las clases a persistir son simples POJOs.
- No hay restricciones con respecto a relaciones entre objetos (herencia, poliformismo).

2 entidades EJB2 CMP2

Con la persistencia manejada por el contenedor EJB es el propio contenedor EJB el que maneja todos los accesos a la base de datos requeridos por el bean de entidad. Como resultado, el código de acceso a los datos del bean, no está acoplado programáticamente a una fuente de datos específica. Esto libera al programador de tener que escribir código de acceso a los datos y permite que el bean de entidad se pueda desplegar en diferentes contenedores y/o contra diferentes fuentes de datos.

Para desplegar un EJB en el entorno de un contenedor EJB, se debe suministrar un fichero descriptor de despliegue para el contenedor EJB. Un descriptor de despliegue es un documento XML, llamado ejb-jar.xml, que especifica información sobre el bean como su tipo de persistencia y los atributos de transacción.

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar id="ejb-jar_ID" version="2.1"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
    <display-name>connectinEJB2cmp2</display-name>
<enterprise-beans>
<entity>
    <ejb-name>CategoriaCMP2</ejb-name>
    <home>interfaces.CategoríaHome</home>
    <remote>interfaces.CategoríaRemote</remote>
    <local-home>interfaces.CategoríaLocalHome</local-home>
    <local>interfaces.CategoríaLocal</local>
    <ejb-class>ejb.CategoríaCMP2</ejb-class>
    <persistence-type>Container</persistence-type>
    <prim-key-class>java.lang.String</prim-key-class>
    <reentrant>true</reentrant>
    <cmp-version>2.x</cmp-version>
    <abstract-schema-name>Categoría</abstract-schema-name>

    <cmp-field>
        <field-name>nombre</field-name>
    </cmp-field>
    <primkey-field>nombre</primkey-field>
    <query>
        <query-method>
            <method-name>findAll</method-name>
            <method-params />
        </query-method>
        <ejb-ql> SELECT OBJECT(g) FROM Categoría g </ejb-
ql>
        </query>
    </entity>

    <entity>
        <ejb-name>GrupoCMP2</ejb-name>
        <home>interfaces.GrupoHome</home>
        <remote>interfaces.GrupoRemote</remote>
        <local-home>interfaces.GrupoLocalHome</local-home>
        <local>interfaces.GrupoLocal</local>
        <ejb-class>ejb.GrupoCMP2</ejb-class>
```

```

<persistence-type>Container</persistence-type>
<prim-key-class>java.lang.String</prim-key-class>
<reentrant>true</reentrant>
<cmp-version>2.x</cmp-version>
<abstract-schema-name>Grupo</abstract-schema-name>
<cmp-field>
    <field-name>nombre</field-name>
</cmp-field>
<cmp-field>
    <field-name>descripcion</field-name>
</cmp-field>
<cmp-field>
    <field-name>categoria</field-name>
</cmp-field>
<cmp-field>
    <field-name>administrador</field-name>
</cmp-field>
<primkey-field>nombre</primkey-field>
<query>
    <query-method>
        <method-name>findAll</method-name>
        <method-params />
    </query-method>
    <ejb-ql> SELECT OBJECT(g) FROM Grupo g </ejb-ql>
</query>
</entity>
</enterprise-beans>
</ejb-jar>

```

Tenemos otro fichero de configuración jboss.xml en el que estableceremos bindings jndi para nuestros 2 objetos distribuidos el cual es dependiente del contenedor.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 3.2//EN"
"http://www.jboss.org/j2ee/dtd/jboss_3_2.dtd">
<jboss>
    <enterprise-beans>
        <entity>
            <ejb-name>CategoriaCMP2</ejb-name>
            <jndi-name>CategoriaCMP2</jndi-name>
            <local-jndi-name>local/CategoriaCMP2</local-jndi-name>
        </entity>
        <entity>
            <ejb-name>GrupoCMP2</ejb-name>
            <jndi-name>GrupoCMP2</jndi-name>
            <local-jndi-name>local/GrupoCMP2</local-jndi-name>
        </entity>
    </enterprise-beans>
</jboss>

```

Desde el cliente instanciamos estos objetos distribuidos utilizando la cadena jndi establecida en el jboss.xml y hacemos uso de los métodos que ofrece: create, findAll y findByPrimaryKey

```

...
InitialContext contexto;
CategoriaLocalHome categoriaHome;
GrupoLocalHome grupoHome;
contexto = new InitialContext();
grupoHome = (GrupoLocalHome)contexto.lookup("local/GrupoCMP2");
categoriaHome= (CategoriaLocalHome)contexto.lookup("local/CategoriaCMP2");
try {
    CategoriaLocal cat2 = categoriaHome.create("asd");
} catch (CreateException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
Collection cats = categoriaHome.findAll();
CategoriaLocal cat = categoriaHome.findByPrimaryKey("asd");
try {
    categoriaHome.remove(cat);
} catch (EJBException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (RemoveException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
...

```

Implementación de un patrón de la capa de negocio: Value Object

La labor principal de Clientes (JSP/Servlet/Programa de Terminal) para EJB's es invocar métodos presentes en el EJB, sin embargo, como se pudo observar en Interfases Locales estos Clientes no necesariamente residen en el mismo "Application Server/EJB Container" , esta residencia trajo como consecuencia el patrón de diseño Value Object.

El llamar método por método presente en distintos EJB's puede presentar una carga substancial sobre la Red y el "Application Server/EJB Container", cada método que es llamado por el Cliente requiere lo siguiente:

- *Serializar la información para ser enviada al "EJB Container".*
- *Deserializar la información para ser procesada en el "EJB Container".*
- *Revisar parámetros de Seguridad.*
- *Iniciar una Transacción.*
- *Serializar respuesta para ser procesada por el Cliente.*
- *Deserializar respuesta para utilizarse en el Cliente.*

Es normal en estos casos que este mismo proceso sea llevado acabo 5 o 6 veces consecutivamente, a través de un Objeto de Valores ("Value Object") es posible reducir esta carga a una sola llamada.

El principio de un Objeto de Valores ("Value Object") es sencillo: agrupar los distintos valores utilizados por el Cliente en un solo método.

Tenemos 2 objetos de negocio distribuidos GrupoCMP2 y CategoríaCMP2. Para implementar este patrón debemos crear una interfaz EntityValueObject que deberá ser implementada por las interfaces local y remota de los objetos de negocio. No sé si está bien hecho pero a la hora de crear la interfaz EntityValueObject necesité crear otra que era igual pero no lanzara RemoteException pues esto no tenía sentido en las interfaces locales y en tiempo de compilación me daba error así que defini 2 una que lanzaba excepción y otra que no.

Para trabajar con objetos de datos definimos el tipo base ValueObject del siguiente modo:

```
package interfaces;

import java.io.Serializable;

public interface ValueObject extends Serializable { }
```

Podemos observar que el tipo ValueObject no es más que una interface marca que define un supertipo de las clases VO y además caracteriza la propiedad serializable que permite que los objetos de estas clases sean enviados por valor utilizando RMI.

Definimos también el tipo de los objetos de negocio que devuelven un VO (EntityValueObject), que define un método para devolver el VO y otro para actualizar el objeto con los valores de un VO.

```
package interfaces;

import java.rmi.RemoteException;

public interface EntityValueObject {

    public ValueObject getVO() throws RemoteException;

    public void setVO(ValueObject vo) throws RemoteException;
}
```

Los objetos de negocio Grupo y Categoria deben devolver un VO. La clases que define este VO con toda su información serían las siguientes:

```
package interfaces;

public class GrupoVO implements ValueObject {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
```

```

private String nombre;
private String descripcion;
private String administrador;
private String categoria;

public String getCategoría() {
    return categoria;
}
public void setCategoría(String categoria) {
    this.categoría = categoria;
}
public String getNombre() { return nombre; }
public void setNombre(String nombre) { this.nombre = nombre; }

public String getDescripción() { return descripción; }
public void setDescripción(String descripción) { this.descripción =
descripción; }

public String getAdministrador() { return administrador; }
public void setAdministrador(String administrador) {
this.administrador = administrador; }

}

```

```

package interfaces;

public class CategoríaVO implements ValueObject {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private String nombre;

    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

}

```

Modificamos la interface de los objetos Grupo y Categoría (GrupoLocal y CategoríaLocal) para indicar que devuelve objetos de datos (implementa el tipo EntityValueObject).

Finalmente implementamos los métodos de la interface EntityValueObject en GrupoCMP2 y CategoríaCMP2.

```

// Implementación de Value Object

public ValueObject getVO() {
    GrupoVO grupo = new GrupoVO();
    grupo.setDescripción(getDescripción());
}

```

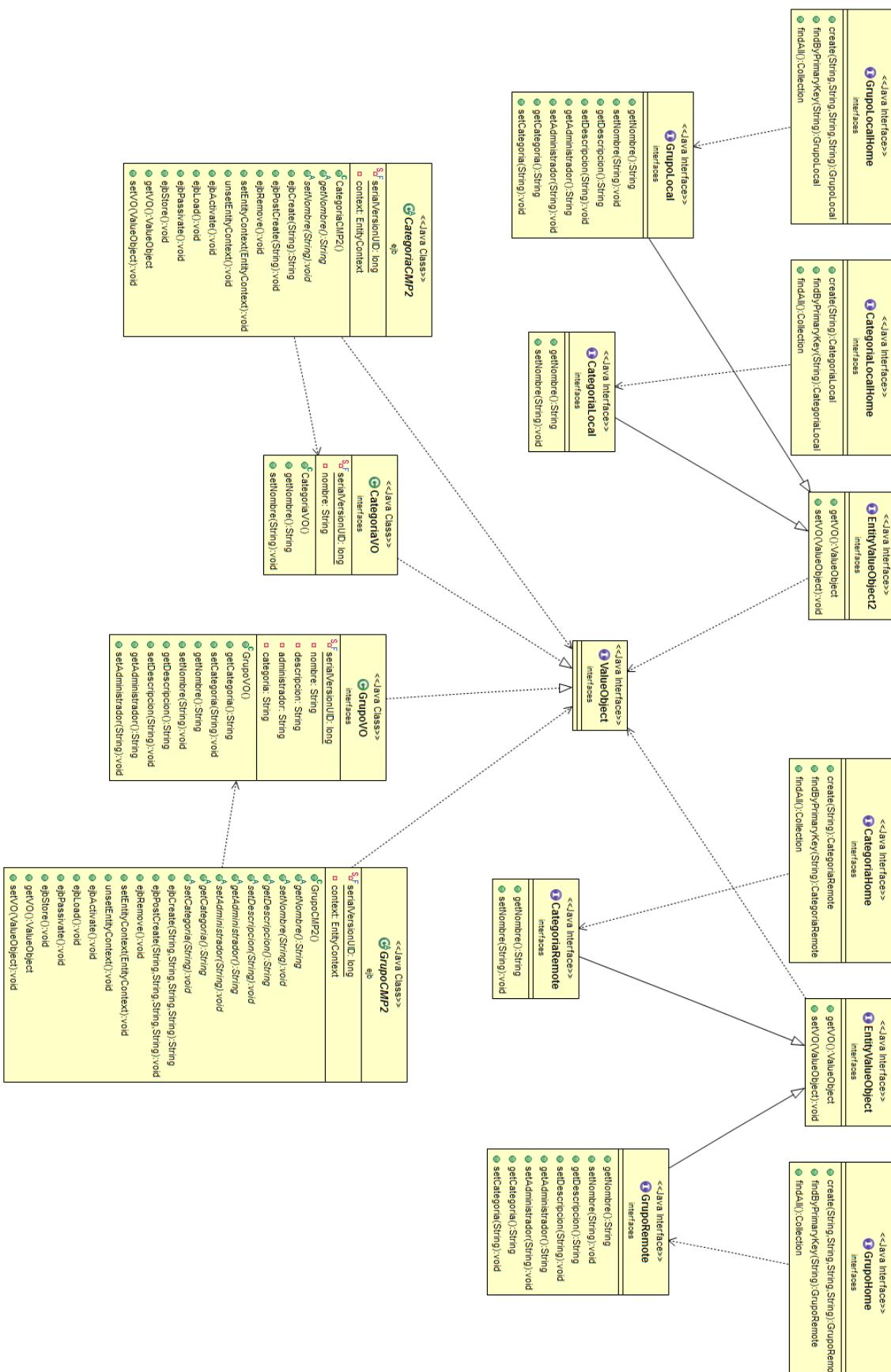
```
        grupo.setNombre(getNombre());
        grupo.setAdministrador(getAdministrador());
        grupo.setCategoria(getCategoria());
        return grupo;
    }

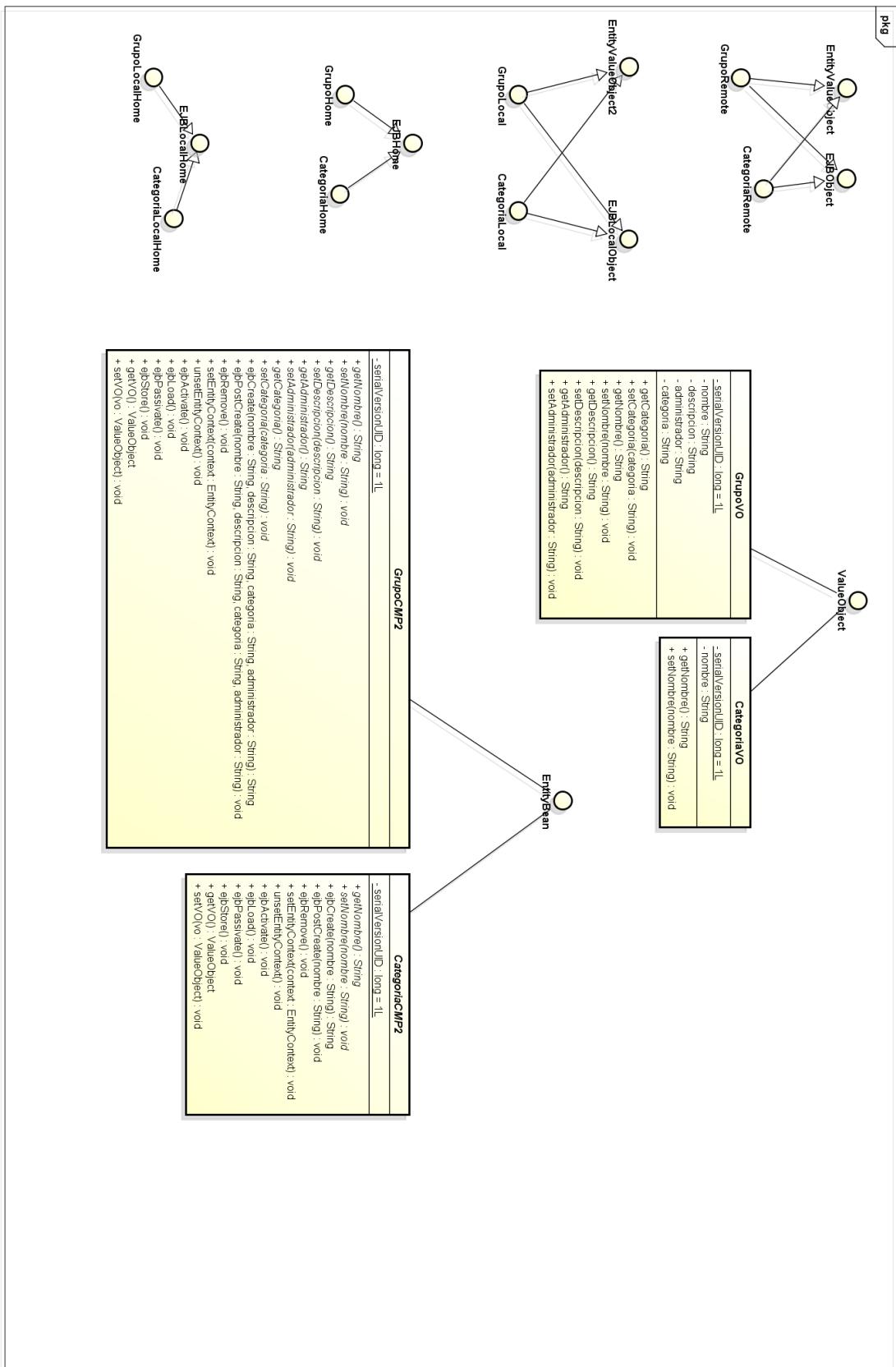
    public void setVO(ValueObject vo) {
        GrupoVO grupo = (GrupoVO)vo;
        setDescripcion(grupo.getDescripcion());
        setNombre(grupo.getNombre());
        setCategoria(grupo.getCategoría());
        setAdministrador(grupo.getAdministrador());
    }
```

```
// Implementación de Value Object
public ValueObject getVO() {
    CategoriaVO categoria = new CategoriaVO();
    categoria.setNombre(getNombre());
    return categoria;
}

public void setVO(ValueObject vo) {
    CategoriaVO categoria = (CategoriaVO )vo;
    setNombre(categoria.getNombre());
}
```

Podemos observar en las siguientes dos ilustraciones un diagrama de la solución.





Controlador de sesión para gestionar idioma (EJB3 de sesión)

Gracias a las mejoras de EJB3 con respecto a la versión anterior vemos como el desarrollo de EJB se simplifica, tal es así que puedo permitirme el lujo de mostrar todo el código necesario para implementarlo en esta memoria. Podemos observar como en esta ocasión las clases son clases normales y que no implementan ninguna interfaz de EJB.

Podemos implementar un EJB session bean como StateLess o como Stateful.

Un Stateless bean no mantiene un estado conversacional con el cliente.

Cuando un cliente invoca los métodos de un stateless bean, las variables de instancia del bean pueden contener un estado específico del cliente, pero sólo por la duración de la invocación. Cuando el método finaliza, el estado del cliente específico no debería mantenerse.

Las instancias pueden estar compartidas por los clientes. El contenedor tiene un pool de instancias, cuando el cliente invoca un método se asigna una instancia, cuando la libere es retornada al pool.

Ofrecen mejor escalabilidad para aplicaciones con gran cantidad de clientes.

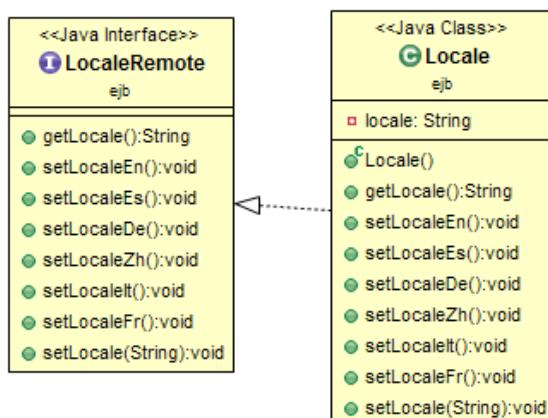
Puede implementar un web service, pero no otros tipos de enterprise beans.

En un stateful bean el estado se mantiene durante la sesión del cliente con el bean. La instancia es reservada para el cliente y cada una almacena la información del cliente. La sesión finaliza si el cliente remueve el bean o finaliza su sesión.

En este caso necesitamos un bean de tipo stateful pues debemos poder recordar durante una sesión el idioma que está utilizando para representar las cadenas internacionalizadas.

Quizás no he escogido el mejor ejemplo para implementar un bean de sesión. No mantendría el valor de idioma de internacionalización a través de un EJB de sesión en un proyecto real (antes de implementar el ejb de sesión el valor se mantenía a través de un @SessionScope (@ManagedBean), aunque en este caso lo haya utilizado de excusa para utilizarlo).

Esquema de la solución:



Para implementar este EJB de sesión simplemente realizamos los siguientes pasos en eclipse:

1. File -> New->EJB Project
2. Sobre el proyecto: click derecho -> New -> Session Bean (EJB 3.x)
3. Nos aparece un asistente para crear el Session Bean, designamos un nombre (Locale) un tipo (Stateless/**Statefull**/Singleton) y le indicamos que cree una interfaz remota y hacemos click en finalizar.



4. Finalmente solo queda definir métodos en la interfaz remota (anotada con @Remote) e implementarlos en la otra clase anotada con @Stateful y @LocalBean.

Aquí podemos ver el código de estas clases e interfaces:

```
package ejb;

import javax.ejb.Remote;

@Remote
public interface LocaleRemote {
    public String getLocale();
    public void setLocaleEn();
    public void setLocaleEs();
    public void setLocaleDe();
    public void setLocaleZh();
    public void setLocaleIt();
    public void setLocaleFr();
    public void setLocale(String localeCode);
}
```

```

package ejb;

import javax.ejb.LocalBean;
import javax.ejb.Stateful;

/**
 * Session Bean implementation class locale
 */

@Stateful
@LocalBean
public class Locale implements LocaleRemote {
    private String locale;
    public Locale() {
        locale="es";
    }

    @Override
    public String getLocale() {      return locale;      }

    @Override
    public void setLocaleEn() {locale="en";}

    @Override
    public void setLocaleEs() {locale="es";}

    @Override
    public void setLocaleDe() {locale="de";}

    @Override
    public void setLocaleZh() {locale="zh";}

    @Override
    public void setLocaleIt() {locale="it";}

    @Override
    public void setLocaleFr() {locale="fr";}

    @Override
    public void setLocale(String localeCode) {locale=localeCode;}
}

```

De lado del cliente podemos referenciar el objeto remoto y usar sus métodos de la siguiente forma:

```

InitialContext context = new InitialContext();
localeRemote =(LocaleRemote) context.lookup
("java:global/localeservice/Locale!ejb.LocaleRemote");
localeRemote.setLocaleDe();
localeRemote.getLocale(); // devolvería "de"

```

Java Remote Method Invocation (RMI)

RMI es un mecanismo ofrecido por Java para invocar un método de manera remota. Forma parte del entorno estándar de ejecución de Java y proporciona un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java.

Veamos como he diseñado la solución.

Objeto distribuido:

```
package mirmi;

public class Mail {
    private String texto;
    private String correo;
    public String getTexto() {
        return texto;
    }
    public void setTexto(String texto) {
        this.texto = texto;
    }
    public String getCorreo() {
        return correo;
    }
    public void setCorreo(String correo) {
        this.correo = correo;
    }
}
```

Interfaz remota:

```
package mirmi;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface GestorMails extends Remote {
    public boolean enviar(String texto, String correo) throws
RemoteException;
}
```

Implementación de la interfaz remota:

```
package mirmi;
import java.rmi.*;
import java.util.ArrayList;
import java.util.List;
public class GestorMailsImpl implements GestorMails {
    private List<Mail> mails;
    @Override
    public boolean enviar(String texto, String correo) throws
```

```

RemoteException {
    // TODO Auto-generated method stub

    Mail mail=new Mail();
    mail.setTexto(texto);
    mail.setCorreo(correo);
    mails.add(mail);
    if(mails.size()==3){

        for (Mail m : mails) {
            System.out.println("Enviando correo a "+
m.getCorreo() + " ...");
            System.out.println(m.getTexto());
        }
        mails.clear();
    }
    return true; // todo ha ido bien;

}
public GestorMailsImpl(){
    super();
    mails= new ArrayList<Mail>();

}
}

```

Servidor:

```

package mirmi;
import java.rmi.*;
import java.rmi.server.*;

public class Servidor {

    public static void main (String[] args) {
        System.setSecurityManager( new RMISecurityManager ());

        // Creación del Objeto Distribuido
        GestorMails gm = new GestorMailsImpl();

        // Activación del objeto. Puede lanzar una excepción Remota.
        try {

            UnicastRemoteObject.exportObject(gm);
        } catch (Exception e) {
            System.err.println("Error al activar el objeto distribuido");
            System.exit(1);
        }

        // Registro del objeto
        try {

            Naming.rebind( "gestormails", gm);
        } catch (Exception e) {
            System.out.println("Error al registrar el objeto distribuido" + e);
            System.exit(1 );
        }
    }
}

```

```

    }

    System.out.println("Objeto distribuido listo ...");

}

}

```

Cliente de prueba:

```

package mirmi;
import java.rmi.*;

public class Cliente {

    public static void main(String[] args) {

        // Instalación del Gestor de Seguridad RMI
        System.setSecurityManager( new RMISecurityManager());

        GestorMails gm = null;

        // Obtención de la referencia remota
        try {
            gm = (GestorMails ) Naming.lookup ( "gestormails" );
        } catch (Exception e) {
            System.err.println("Error al obtener la referencia remota" );
            System.exit(1 );
        }

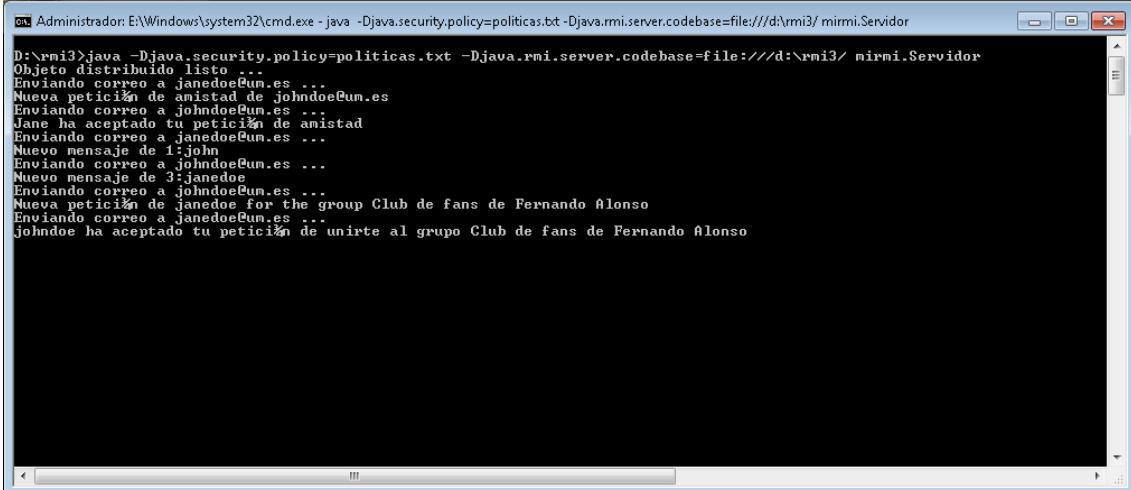
        // Uso del objeto remoto. Puede lanzar una excepción remota
        try {

            System.out.println(gm.enviar("1 Solicitud de grupo
aceptada","francisco.campillo@um.es"));
            System.out.println(gm.enviar("2 Solicitud de amistad
aceptada","pepe@um.es"));
            System.out.println(gm.enviar("3 Solicitud de amistad
aceptada","juan@um.es"));
            System.out.println(gm.enviar("4 Solicitud de amistad
aceptada","lope@um.es"));
            System.out.println(gm.enviar("5 Solicitud de grupo
aceptada","lorena@um.es"));
            System.out.println(gm.enviar("6 Solicitud de amistad
aceptada","javier@um.es"));
            System.out.println(gm.enviar("7 Solicitud de grupo
aceptada","aaron@um.es"));
        } catch (Exception e) {
            System.err.println("Error al acceder al objeto remoto");
            System.exit(1 );
        }
    }
}

```

Este código del cliente es utilizado después de cada vez que se crea una notificación en el proyecto web con los valores correspondientes a cada caso.

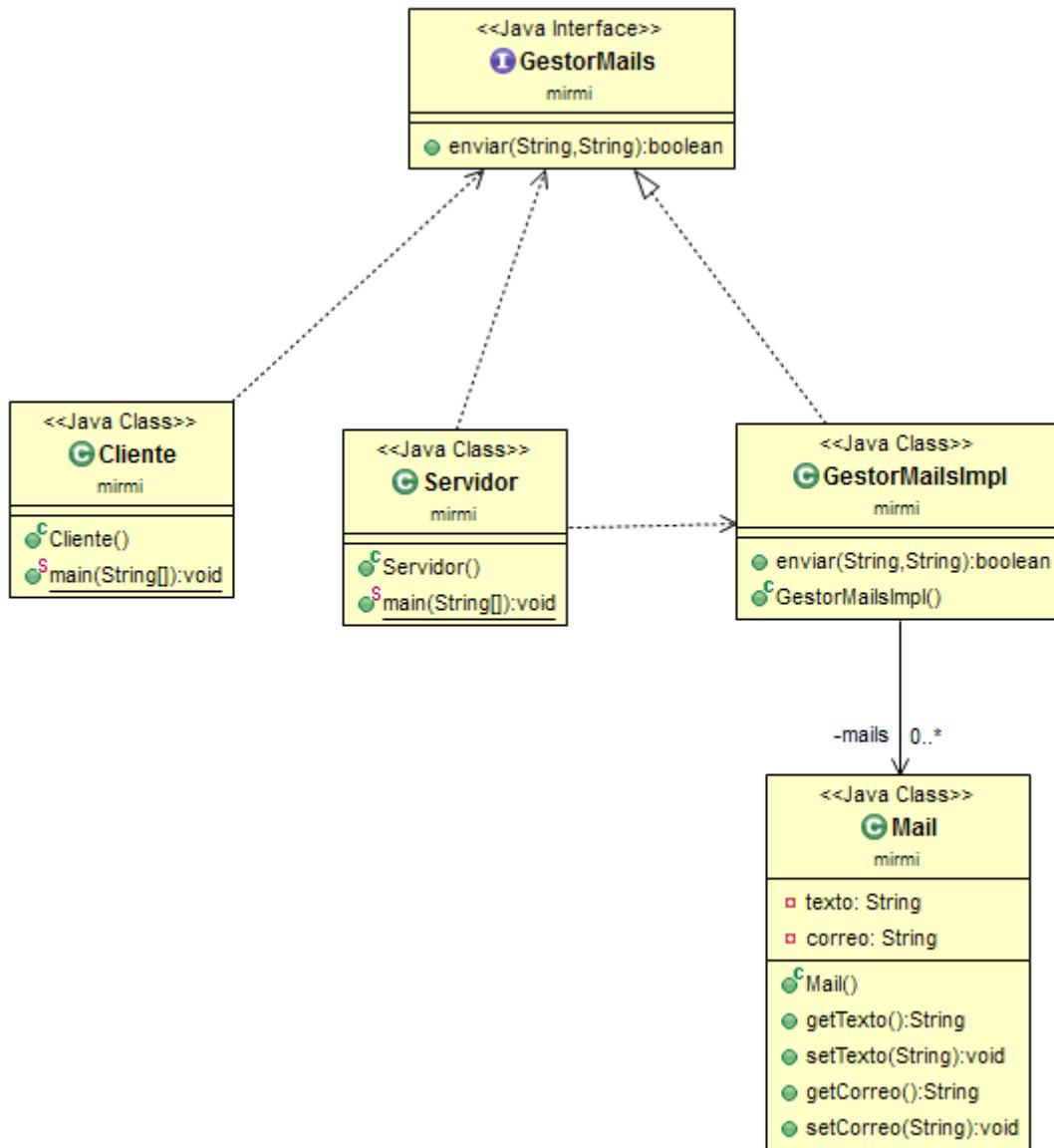
Este es un ejemplo de ejecución en el que se ve la salida por consola del servidor.



A screenshot of a Windows Command Prompt window titled "Administrador: E:\Windows\system32\cmd.exe - java -Djava.security.policy=politicas.txt -Djava.rmi.server.codebase=file:///d:/rmi3/ mirmi.Servidor". The window displays the following text output:

```
D:\rmi3>java -Djava.security.policy=politicas.txt -Djava.rmi.server.codebase=file:///d:/rmi3/ mirmi.Servidor
Objeto distribuido listo ...
Enviando correo a janedoe@um.es ...
Nueva petición de amistad de john doe@um.es
Enviando correo a john doe@um.es ...
John ha aceptado tu petición de amistad
Enviando correo a janedoe@um.es ...
Nuevo mensaje de 1:john
Enviando correo a john doe@um.es ...
Nuevo mensaje de 3:janedoe
Enviando correo a john doe@um.es ...
Nueva petición de janedoe for the group Club de fans de Fernando Alonso
Enviando correo a janedoe@um.es ...
john doe ha aceptado tu petición de unirte al grupo Club de fans de Fernando Alonso
```

Aquí podemos ver un esquema de la solución:

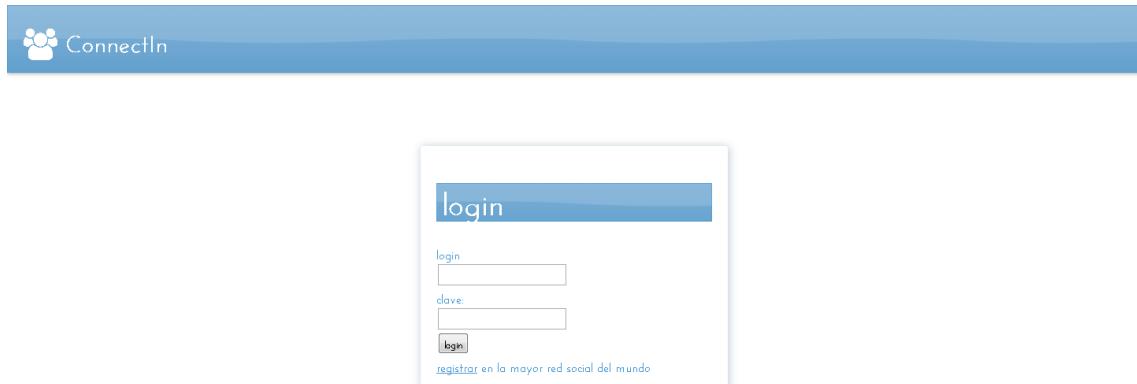


Visita guiada por la aplicación (desde el punto de vista del usuario)

Vamos a ver a través de una serie de capturas de pantalla las distintas funcionalidades que ofrece la aplicación así como los eventos que suceden ante ciertas acciones.

Empezamos con la pantalla de acceso a la aplicación es login.faces cualquier acceso a la aplicación sin tener una sesión iniciada (salvo la pantalla de registro) nos derivará a esta pantalla para que accedamos o nos registremos de la misma forma que si accedemos a la pantalla de login teniendo ya una sesión abierta en cuyo caso nos redirigiría a home.faces.

Este comportamiento lo implementaremos con Filtros de Servlets, que serán comentados en la sección nuevos aspectos con más detalle.



Después de un intento de acceso fallido nos derivará a login_error.faces

The screenshot shows the ConnectIn login page. At the top, there is a blue header bar with the ConnectIn logo and the word "ConnectIn". Below the header, the word "login" is centered in a blue bar. The main content area has a light gray background. A red rectangular box highlights a message box containing the text "login o clave incorrecta" (incorrect login or password). Below this message box are two input fields: one for "login" and one for "clave" (password), both with placeholder text. Underneath the input fields is a small button labeled "login". At the bottom of the page, there is a link "registrar en la mayor red social del mundo" (register in the world's largest social network).

Después de 3 intentos nos derivará a login_error_max_attempts.faces

The screenshot shows the ConnectIn login page. At the top, there is a blue header bar with the ConnectIn logo and the word "ConnectIn". Below the header, the word "login" is centered in a blue bar. The main content area has a light gray background. A red rectangular box highlights a message box containing the text "max attempts exceeded".

Si todavía no disponemos de un usuario debemos crear uno. En login.faces disponemos de un enlace a la página de registro register.faces .

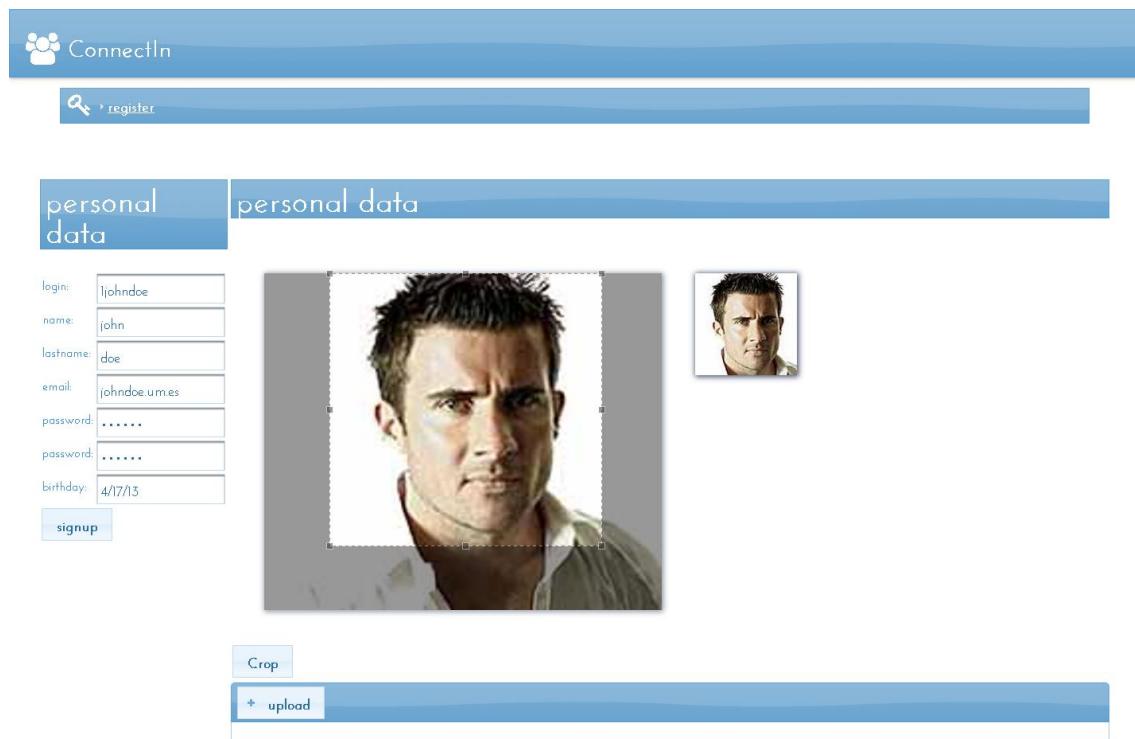
En un primer momento vamos a introducir datos incorrectos para ver cómo se comportan los validadores implementados.

En concreto los campos de nombre y apellido esperan encontrar 1 o más ocurrencias de palabras que empiecen por una letra mayúscula seguida de letras minúsculas y con una separación de un espacio entre cada palabra para registrar nombres múltiples apellidos largos etc.

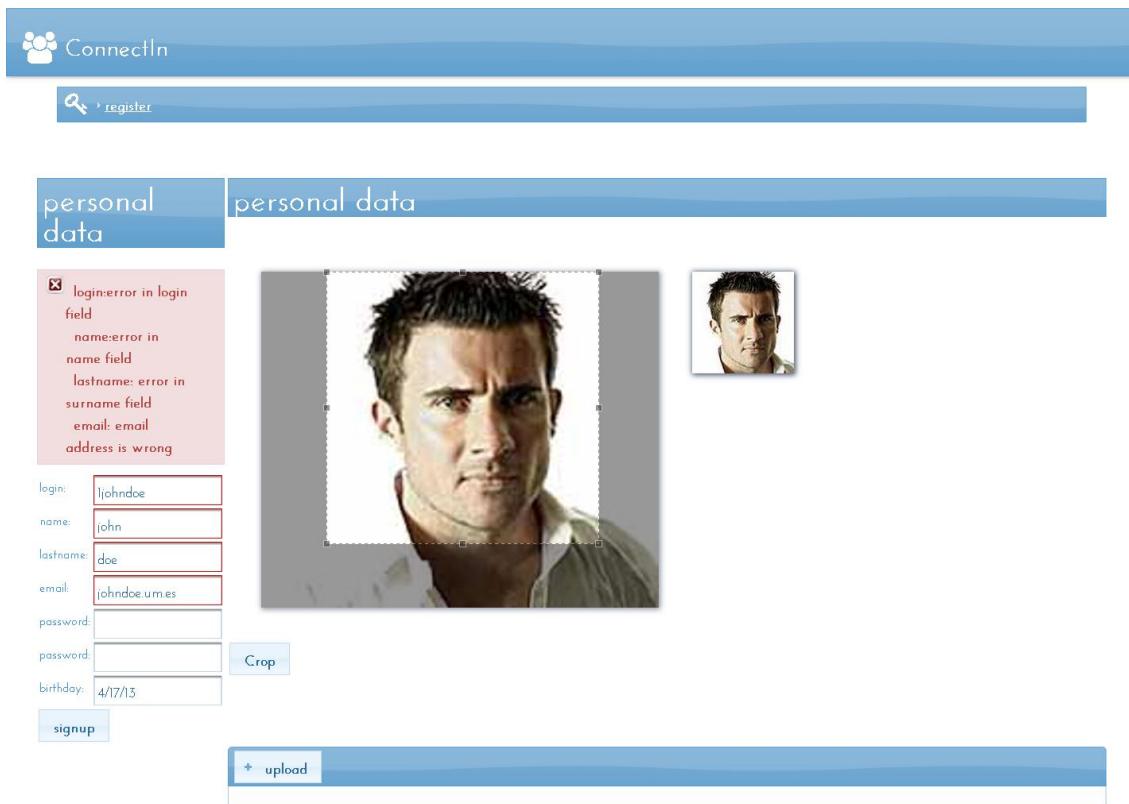
En el login esperamos una cadena que empiece por letra minúscula y le sigan letras minúsculas o números y cuya extensión sea de un mínimo de 3 caracteres y un máximo de 16.

En el password debe coincidir las 2 casillas y debe tener un tamaño mínimo de 6 caracteres.

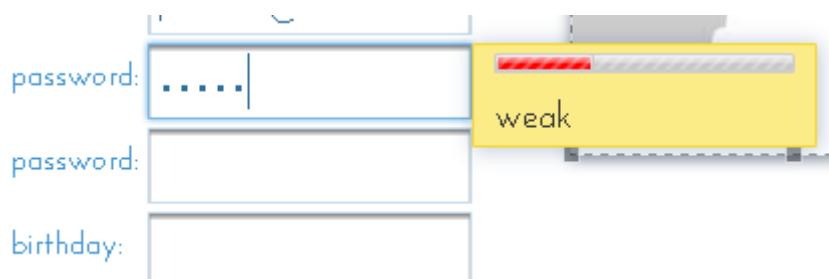
En el campo email: debe introducirse una dirección de mail válida (usuario@servidor.dominio)



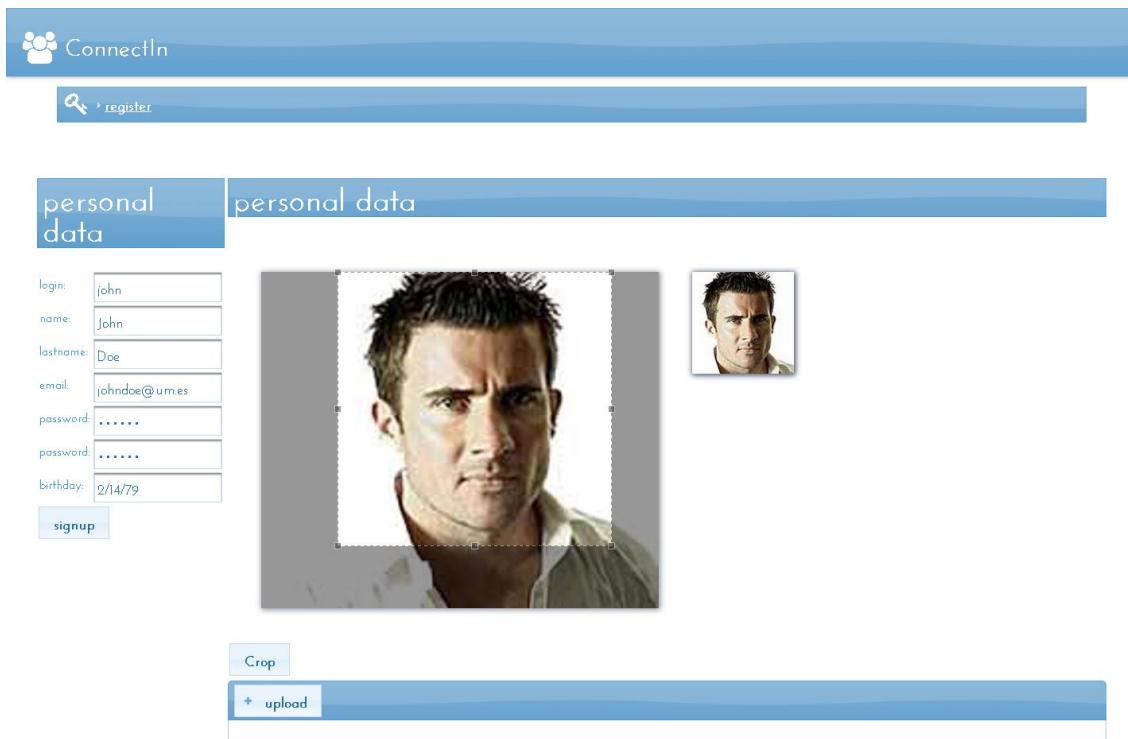
En la siguiente captura vemos como el formulario no ha completado su envío y ha sido interrumpido, y muestra una serie de mensajes de error en color rojo además de destacar en rojo los campos en los que fueron introducidos datos erróneos.



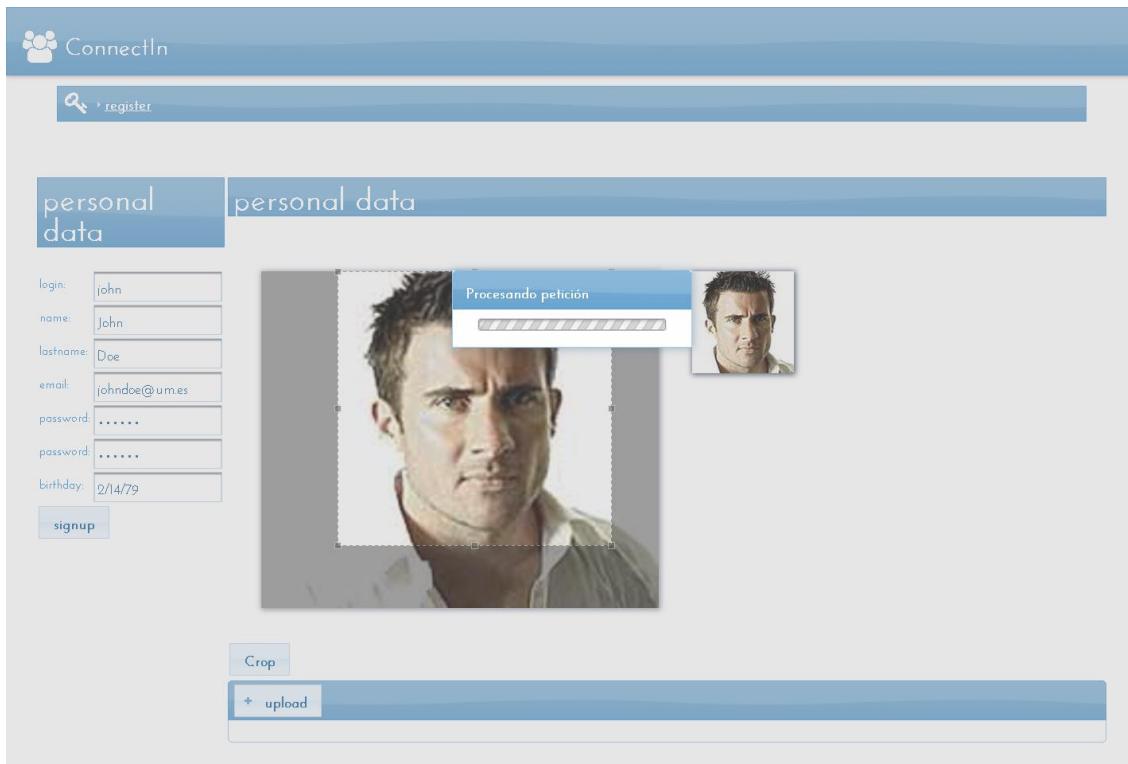
Antes de enviar vemos como el componente de password nos muestra un mensaje informativo que nos informa de la fortaleza de la contraseña elegida.



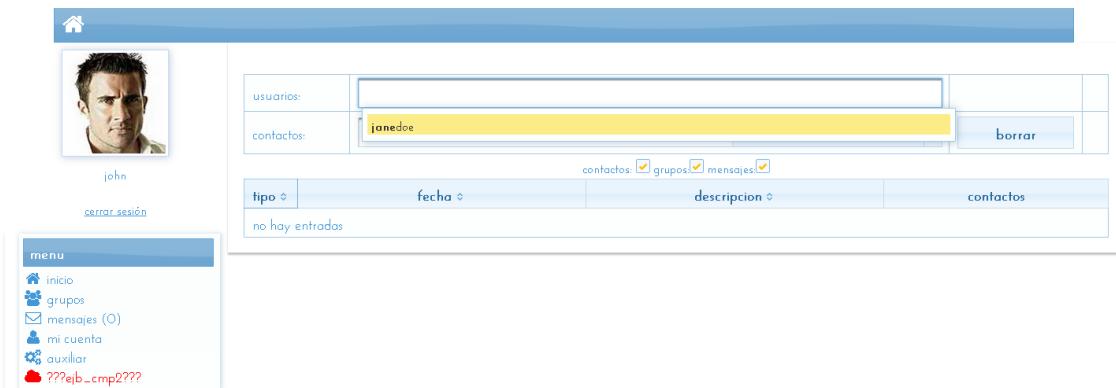
Introducimos, esta vez sí, los datos correctos en el formulario de registro. Observamos en la parte derecha de la pantalla (Sección Personal Data) que podemos subir una fotografía, reencuadrarla y recortarla para que sea nuestra foto de perfil en la aplicación.



Cada vez que se realiza alguna petición Ajax por parte de algún componente de la página veremos que la pantalla se oscurece y nos aparece un diálogo con el lema “Procesando petición” cuando la petición termina el oscurecimiento y el diálogo desaparece.



Nos volvemos a la página de login e introducimos las credenciales que indicamos en el proceso de registro. Una autenticación exitosa nos redirigirá a la página home.faces.



Podemos dividir la página en 3 secciones 2 son fijas para casi todas las pantallas pues son definidas en ficheros de plantilla y una es específica de cada página.

Tenemos en la parte superior además de la cabecera (no aparece en la siguiente imagen aunque si en otras siguientes) y el “rastro de hormigas” o breadCrumb que es un menú horizontal que nos indica en cada momento en que parte de la página nos encontramos.

Conforme vayamos adentrándonos en subniveles dentro de la página el rastro de hormigas se va alargando empezando siempre desde el home (salvo en la página de registro que el primer enlace es login.faces).

Otra sección es la barra lateral que se encuentra en casi todas las páginas de la aplicación salvo en alguna excepción como la pantalla de registro.

En esta barra lateral nos encontramos primero con la foto de perfil del usuario autenticado su nombre de usuario y un enlace para cerrar sesión y dirigirnos a login.faces.

Más abajo nos encontramos con el menú de la aplicación que nos llevará a las distintas secciones de la aplicación:

- Inicio. Esta misma página (home.faces).
- Grupos (groups.faces). Página donde se gestionan y visualizan los grupos discusiones y posts.
- Mensajes (messages.faces). Página donde se gestionan los mensajes enviados y recibidos y donde también se envían.
- Mi cuenta (myaccount.faces). Se configuran datos personales (los mismos que en el momento del registro + el idioma deseado para la interfaz).
- Auxiliar (auxiliary.faces). En esta página se gestionan las categorías.
- EJB CMP2 (ejbcmp2.faces). Página donde se prueban los dos objetos distribuidos mediante EJB2 CMP2.

Finalmente tenemos la sección de detalle que es la que varía entre páginas en este caso lo que tenemos es un formulario para buscar nuevos amigos y solicitarle amistad, otro para gestionar los amigos que ya tienes y si se desea terminar con la amistad(virtual) con lo que volverían a estar disponibles de nuevo en el primer formulario.

Más abajo tenemos un panel de notificaciones que podemos filtrar según el tipo de notificación que pueden ser relativas a grupos, mensajes o contactos

Grupos: Petición de entrar a un grupo del que eres administrador con la opción de aceptar o rechazar la petición y respuestas a peticiones tuyas de entrar a otros grupos bien aceptándola o rechazándola.

Mensajes: Nuevos mensaje recibidos con la opción de leerlos in situ.

Contactos: Nueva petición de amistad con la opción de aceptarla o rechazarla y respuestas a solicitudes de amistad enviadas anteriormente bien aceptándolas o rechazándolas

En la imagen no vemos gran parte de lo que estoy contando pues el usuario esta recién creado y no presenta actividad alguna será en posteriores capturas donde veremos el progreso de este panel de notificaciones.

Vamos a buscar usuarios de connectin para solicitarle amistad. Encontramos a Jane Doe (janedoe). Pulsamos en solicitar amistad y en ese momento jane doe recibirá en su panel de notificaciones una notificación sobre nuestra petición.

The screenshot shows the Connectin application's user interface. On the left, there is a sidebar menu with options: inicio, grupos, mensajes (0), mi cuenta, auxiliar, and a red warning message '???ejb_cmpl2???' at the bottom. The main area has a header 'ConnectIn' with a user profile picture of 'john'. Below the header, there are two forms. The top form is for sending a friend request, with fields 'usuarios:' containing 'janedoe' and a 'solicitar amistad' button. To the right is a profile picture of 'Jane Doe (janedoe)'. The bottom form is for managing notifications, with fields 'contactos:', a dropdown menu, and a 'borrar' button. At the bottom of this section, there are checkboxes for 'contactos', 'grupos', and 'mensajes', all of which are checked. Below these forms is a table with columns 'tipo', 'fecha', 'descripción', and 'contactos', with the message 'no hay entradas'.

Efectivamente Jane Doe tiene ahora la posibilidad de aceptar o rechazar mi petición.

The screenshot shows the Connectin application's user interface. At the top, there is a header bar with a logo and navigation links. Below the header, on the left, is a sidebar menu with options like 'inicio', 'grupos', 'mensajes (0)', 'mi cuenta', 'auxiliar', and a red 'ejb_cmpl?' link. The main content area displays a contact list. In the 'contactos' section, there is a row for 'janedoe' with a dropdown arrow. To the right of this row is a small profile picture of a woman and the text 'Jane Doe (janedoe)'. Below the contact list is a table showing a single entry: 'tipo' (User), 'fecha' (hace instantes), 'descripcion' (nueva petición de amistad de john), and 'contactos' (with 'aceptar' and 'rechazar' buttons). The 'aceptar' button is highlighted.

Jane decide aceptarla y a partir de ahora Jane y John están conectados son mutuamente contactos el uno del otro y aparecen en la lista de contactos como podemos ver en la siguiente imagen. Ahora podemos enviar mensajes a Jane.

This screenshot shows the same Connectin application interface after Jane accepted John's friend request. The contact list now includes both 'janedoe' and 'john'. The 'contactos' section shows 'janedoe' selected, with a profile picture of a man and the text 'John (john)' next to it. The table below shows a new entry: 'tipo' (User), 'fecha' (hace 32 minutos), 'descripcion' (Jane ha aceptado tu petición de amistad), and 'contactos' (empty). The 'aceptar' button is no longer visible.

Para enviar un mensaje a Jane debemos ir a la sección de mensajes (messages.faces). En esta sección podemos realizar distintas acciones:

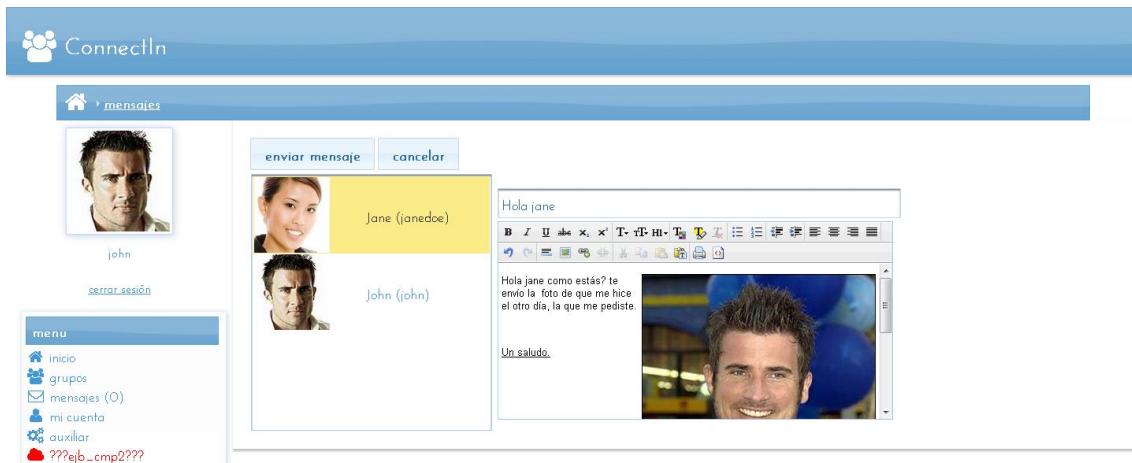
- Ver la lista de mensajes recibidos y leerlos.
- Ver la lista de mensajes enviados y leerlos.
- Escribir mensajes.
- Responder Mensajes.



Vamos a mandarle un mensaje a Jane Doe. Vemos que en la parte izquierda aparece la lista de todos los contactos que tenemos en ese momento más tu mismo por si quieres incluirte en el mensaje (como cuando te mandas un correo a ti mismo para tener una copia).

El componente peditor nos ofrece una interfaz WYSIWYG que facilita la composición de mensajes enriquecidos.

Enviamos el mensaje de la imagen a Jane Doe.



Ahora Jane Doe en su panel de notificaciones tiene una nueva notificación de mensaje. Ahora mismo tenemos pocas notificaciones pero en caso de que tuviéramos más el ícono de la izquierda nos ayuda a diferenciar rápidamente de qué tipo es cada notificación, además de poder filtrar por tipo y ordenar por cada uno de los campos.

Jane va a leer el mensaje que le ha enviado John. Aparece en forma de diálogo sin irse a la sección de mensajes. Si cerramos el mensaje seguiremos en la pantalla principal. En esta vista no he querido poner todas las funcionalidades como responder al mensaje ni que se marque como leído al leerlo desde aquí.

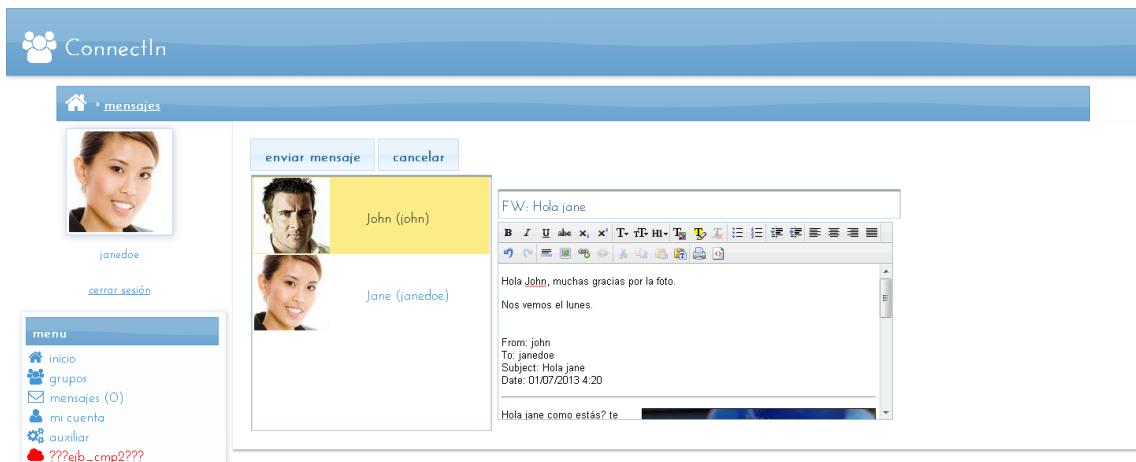
Si nos vamos a la sección de mensajes ahora en la bandeja de entrada veremos un nuevo mensaje no leído. Si lo abrimos o pulsamos el ícono de la derecha pasará a no leído.

The screenshot shows the ConnectIn application's user interface. At the top, there is a blue header bar with the 'ConnectIn' logo. Below it, a navigation bar includes a home icon, a 'mensajes' (messages) icon, and a 'nuevo mensaje' (new message) button. A sidebar on the left is titled 'menu' and contains links for 'inicio', 'grupos', 'mensajes (1)', 'mi cuenta', 'auxiliar', and a red-highlighted '???ejb_cmp2???' link. The main content area displays a list of messages. The first message is from 'John Doe (john)' with the subject 'Hola jane' and a timestamp 'hace instantes'. Below the message list, there are links for 'cerrar sesión' (log out) and 'nuevo mensaje'.

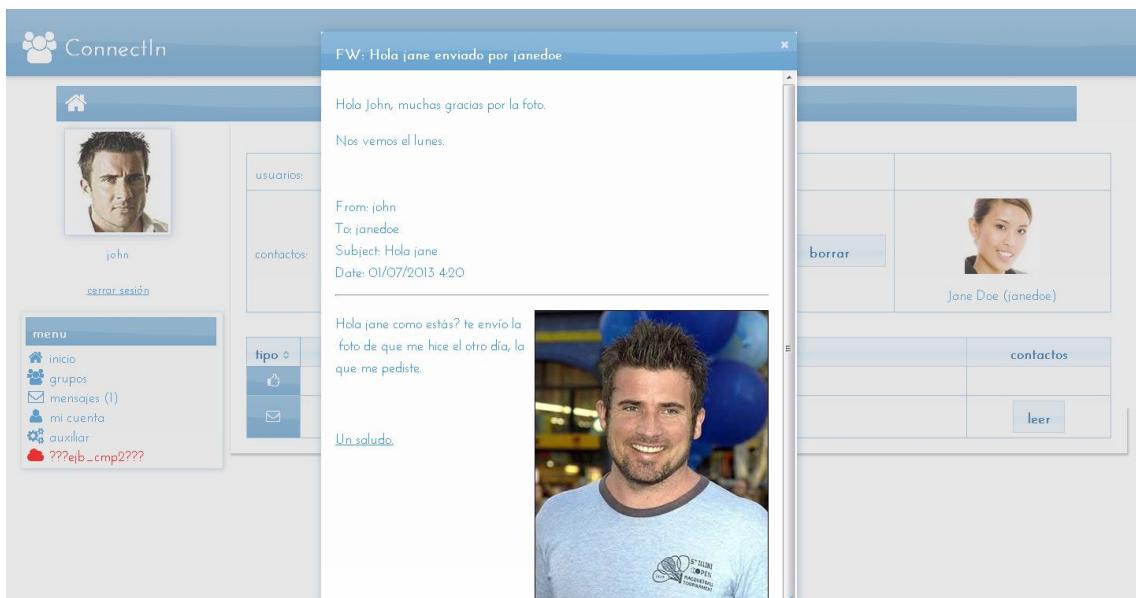
En la vista detalle del mensaje podemos o volver a la lista de mensajes o responder al mensaje.

This screenshot shows the message detail view for the message from John Doe. The top navigation bar and sidebar are identical to the previous screenshot. The main content area shows the message details: 'volver a la lista de mensajes', 'responder', and the message body 'Hola jane como estás? te envio la foto de que me hice el otro dia, lo que me pediste'. Below the message body, there is a link 'Un saludo.' and a thumbnail image of a smiling man with short brown hair, wearing a light blue t-shirt with a small logo on the chest. The sidebar on the left also includes a 'mensajes (0)' link.

Al responder al mensaje añadimos el mensaje anterior al nuevo mensaje a enviar y enlazamos ambos mensajes mediante un atributo.



Ahora John desde su panel de notificaciones visualiza la respuesta de Jane.



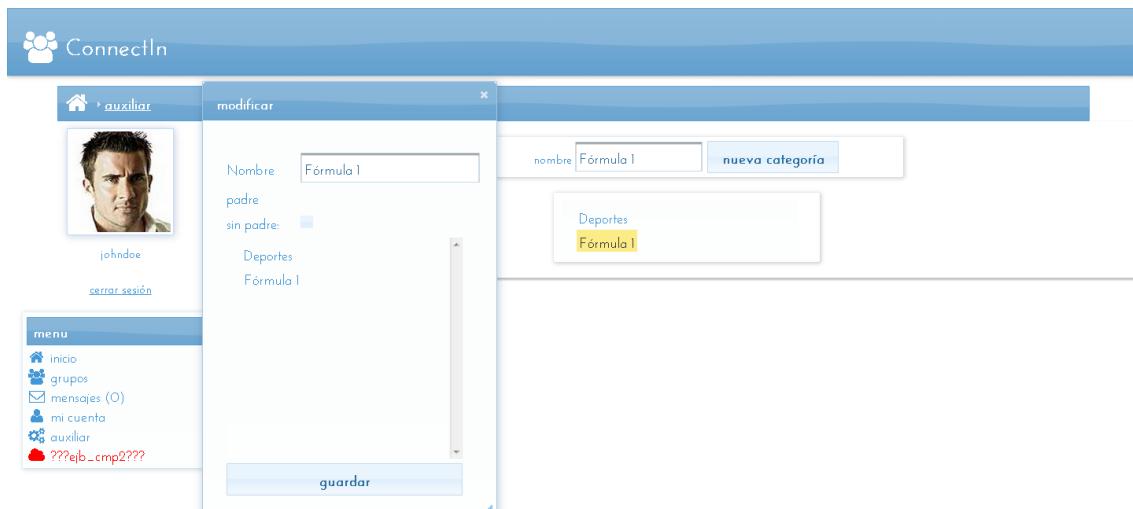
Para crear un grupo necesitamos tener una categoría donde enmarcarlo. Vamos primero a crear alguna categoría. Por ejemplo vamos a crear Deportes y dentro de esta Fórmula 1.

The screenshot shows the Connectin application's user interface. At the top, there is a blue header bar with the 'Connectin' logo. Below it, the main content area has a blue header 'auxiliar'. On the left, there is a sidebar with a profile picture of a man (labeled 'john') and a 'menu' section containing links: 'inicio', 'grupos', 'mensajes (0)', 'mi cuenta', 'auxiliar', and a red link '???ejb_cmp2???'.

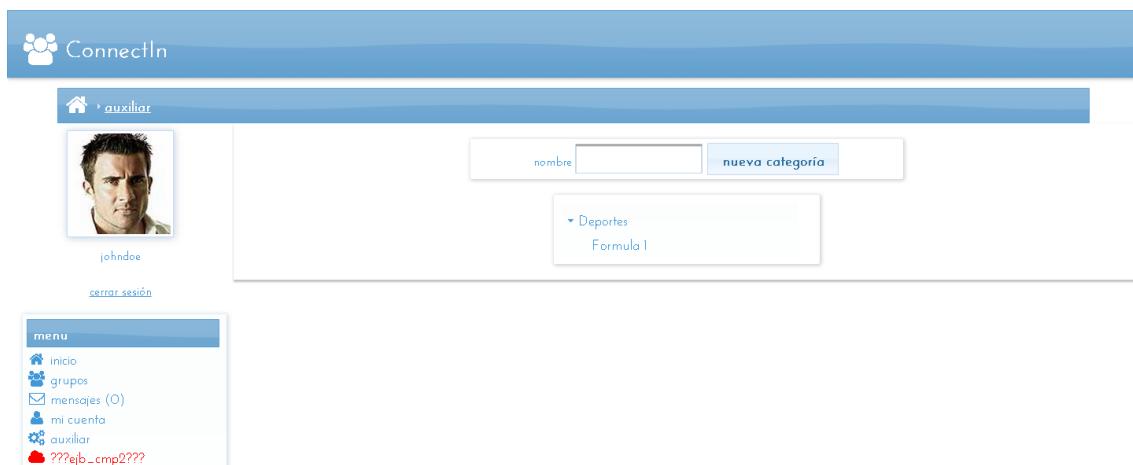
In the center, there is a form for creating a new category. It has a text input field labeled 'nombre' with the value 'Deportes' and a blue button labeled 'nueva categoría'.

This screenshot shows the same interface as the previous one, but with a different user logged in. The sidebar now shows 'johndoe' logged in. The central form has the 'nombre' field set to 'Fórmula 1' and the 'nueva categoría' button is visible. Below the form, there is a list box containing the word 'Deportes'.

Seleccionamos Formula 1 y le decimos que su padre es Deportes. De esta forma el componente p:tree lo mostrará en forma de árbol jerárquico.



Ya tenemos las categorías que queríamos para nuestro nuevo grupo.



Ahora lo siguiente será crear el grupo. Para ello indicamos un nombre, una descripción una categoría donde ubicarlo.

The screenshot shows the 'grupos' (groups) section of the Connectin application. On the left, there's a sidebar with a user profile picture of a man (johndoe), a 'cerrar sesión' (log out) link, and a menu with options: inicio, grupos, mensajes (0), mi cuenta, auxiliar, and a red '???ejb_cmp2????' link.

The main area displays a table with columns: nombre, descripción, and categoría. A message at the top says 'no hay entradas' (no entries). Below it, a note states '* tu eres el dueño del grupo' (you are the owner of the group). The form for creating a new group is filled with:

- nombre: Club de fans de Fernando Alonso
- descripción: Grupo del club de fans de Fernando Alonso. Entérate de todo las noticias relacionadas con el bicampeón mundial de fórmula uno Fernando Alonso.
- categoría: Formula 1

A blue 'crear grupo' (create group) button is at the bottom right of the form.

En la siguiente captura vemos como aparece en el listado de grupos. A los usuarios que no son creadores les aparecerá un botón para pedir ser miembro del grupo y así poder iniciar discusiones en el grupo y post dentro de estas discusiones.

Si queremos filtrar la lista para encontrar un grupo en concreto disponemos de 3 filtros uno por cada campo.

This screenshot shows the search results for the group 'Club de fans de Fernando Alonso'. The search interface includes three filter fields: nombre, descripción, and categoría. The results table shows one entry:

Club de fans de Fernando Alonso	Grupo del club de fans de Fernando Alonso...	Formula 1	petición de ingreso
---------------------------------	--	-----------	-------------------------------------

The 'petición de ingreso' (membership request) link is highlighted in blue. The rest of the interface is identical to the previous screenshot, including the sidebar and the group creation form.

Hacemos click en el botón de petición de ingreso para que cuando el administrador lo vea y lo acepte podamos crear entradas. Mientras tanto podemos ver las discusiones que hay ya creadas por otros usuarios aunque en este momento no hay ninguna.

No se ha mostrado en las imágenes pero también se ha creado una discusión dentro del grupo. Pulsamos en esta para ver los posts de la discusión. De momento no hay posts.

John accede a su cuenta y ve que tiene una petición de grupo y la acepta.

The screenshot shows the Connectin application interface. On the left, there is a sidebar with a menu containing 'inicio', 'grupos', 'mensajes (0)', 'mi cuenta', 'auxiliar', and '???ejb_cmpl???'. The main area displays a contact search bar with 'johndoe' entered, a 'borrar' (Delete) button, and a profile picture of 'Jane Doe (janedoe)'. Below this is a table of notifications:

tipo	fecha	descripcion	contactos
	hace 2 horas	Jane ha aceptado tu petición de amistad	
	hace 2 horas	nuevo mensaje de janedoe	leer
	hace 13 minutos	nueva petición de janedoe para el grupo Club de fans de Fernando Alonso	aceptar rechazar

Jane recibe la notificación de que John ha aceptado su petición. Ya puede crear contenidos dentro del grupo de John.

The screenshot shows the Connectin application interface. On the left, there is a sidebar with a menu containing 'inicio', 'grupos', 'mensajes (1)', 'mi cuenta', 'auxiliar', and '???ejb_cmpl???'. The main area displays a contact search bar with 'johndoe' entered, a 'borrar' (Delete) button, and a profile picture of 'John Doe (johndoe)'. Below this is a table of notifications:

tipo	fecha	descripcion	contactos
	hace 2 horas	nueva petición de amistad de john(aceptado)	
	hace 2 horas	nuevo mensaje de john	leer
	hace instantes	johndoe ha aceptado tu petición de pertenecer al grupo Club de fans de Fernando Alonso	

Veremos como en las mismas pantallas ahora también aparece abajo un formulario para añadir entradas, tanto discusiones como post de discusiones.

The screenshot shows a user profile for 'johndoe' with a photo and a link to 'cerrar sesión'. A sidebar menu includes 'menu', 'inicio', 'grupos', 'mensajes (1)', 'mi cuenta', 'auxiliar', and '???ejb_cmp????'. The main content area displays the title 'Club de fans de Fernando Alonso' and a brief description: 'Grupo del club de fans de Fernando Alonso. Entérate de todo las noticias relacionadas con el bicampeón mundial de fórmula uno Fernando Alonso.' Below this is a discussion titled 'Terreno propicio para Ferrari' by 'johndoe hace 10 minutos'. It has two empty input fields for 'título' and 'texto', and a blue button labeled 'nueva discusión'.

Accedemos a la discusión titulada “Terreno propicio para Ferrari”

The screenshot shows the same user profile and sidebar as the previous screenshot. The main content area now displays the discussion 'Terreno propicio para Ferrari' with the message 'enviado por johndoe hace 11 minutos' and the text 'Ferrari se juega mucho en los dos próximos grandes premios.'. Below this is a message from 'No records found.' followed by a reply from 'Seguro que gana!' with a blue 'nuevo post' button.

Escribimos un post dentro de la discusión. Vemos como aparece la información sobre quien escribe cada post y los muestra en orden de creación.

Terreno propicio para Ferrari

enviado por john Doe hace 11 minutos

Ferrari se juega mucho en los dos próximos grandes premios.

enviado por john Doe hace instantes

Seguro que ganan!

nuevo post

Vamos ahora a ver la sección mi cuenta (myaccount.faces). Básicamente podemos cambiar los mismos datos que ya guardamos en el momento de registro.

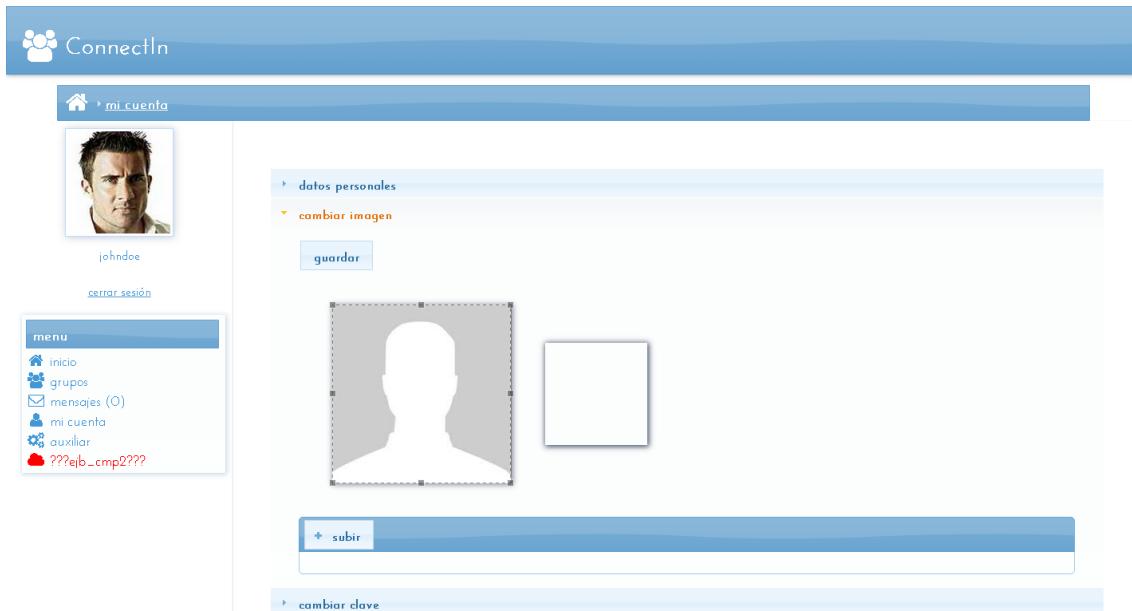
datos personales

login:	john Doe
nombre:	John
apellidos:	Doe
email:	john Doe
fecha de nacimiento:	14/02/79

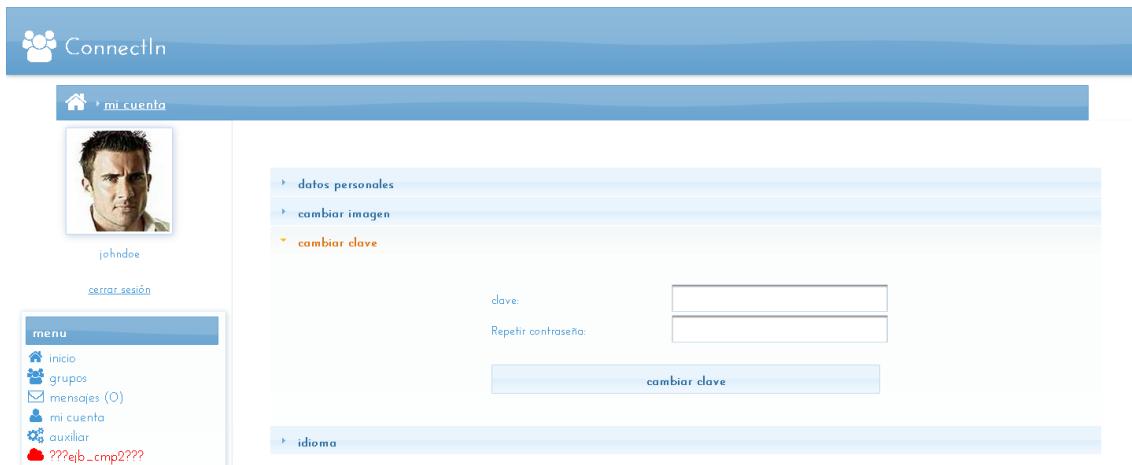
guardar restaurar

cambiar imagen
cambiar clave
idioma

Podemos cambiar la imagen.



Podemos cambiar la contraseña.



Y podemos cambiar el lenguaje de la interfaz. Esta es la única información que no se suministra en el momento de registro. Observamos como la interfaz se traduce completamente a cada uno de los idiomas.

Veamos los idiomas disponibles:

Inglés.



Español de España.



Alemán.



Chino simplificado.



Italiano.

Francés.

Finalmente vemos la sección en la que probamos el uso de 2 entidades distribuidas mediante EJB 2 CMP2. No tiene mucho sentido tener 2 entidades aisladas del resto y duplicadas con respecto a la persistencia mediante JPA pero en el marco de una práctica en el que se quieren probar todas las tecnologías tiene su cabida.

Básicamente tenemos 2 formularios y dos p:datatable a través de los formularios creamos entidades y son mostradas en las tablas.

Categories

name	create
test ejb cmp2	create

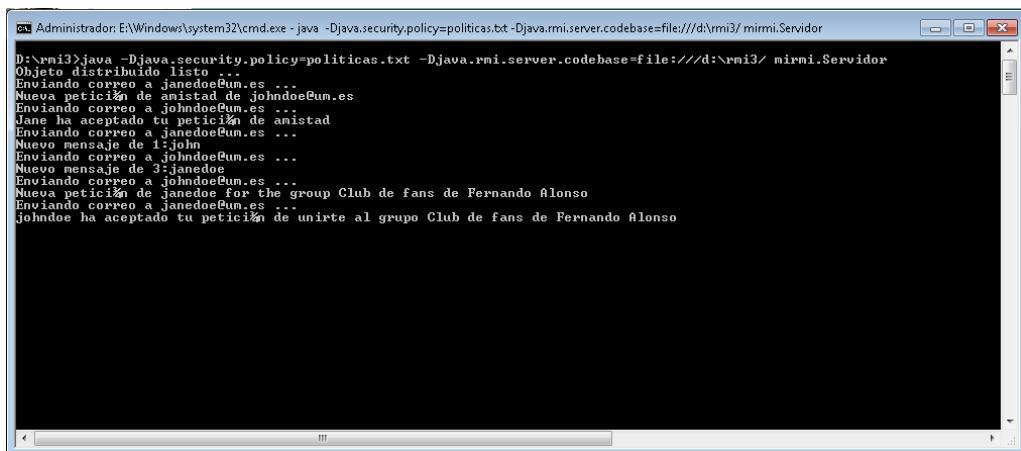
no results

groups

name	description	administrator	category	create
Series de tv	Grupo sobre series de	albert	Ocio	create

no results

Después de esta serie de interacciones entre estos dos usuarios con la aplicación el proceso servidor RMI mostró la siguiente salida. Recordar que el servidor vuelca a la pantalla los mensajes cuando acumula 3. Por lo que puede tener 2 mensajes todavía sin mostrar.



D:\rmi3>java -Djava.security.policy=políticas.txt -Djava.rmi.server.codebase=file:///d:/rmi3/ mirmi.Servidor
Objeto de servicio iniciado.
Enviando correo a janedoe@um.es ...
Nueva petición de amistad de john doe@um.es
Enviando correo a john doe@um.es ...
Jane ha aceptado tu petición de amistad
Enviando correo a janedoe@um.es ...
Nuevo mensaje de i:john
Enviando correo a john doe@um.es ...
Nuevo mensaje de i:janedoe
Enviando correo a john doe@um.es ...
Nueva petición de janedoe for the group Club de fans de Fernando Alonso
Enviando correo a janedoe@um.es ...
john doe ha aceptado tu petición de unirte al grupo Club de fans de Fernando Alonso

Nuevos aspectos

Lambaj

Lambaj emula los lenguajes funcionales, permitiendo tratar colecciones de una forma más sencilla, intentando reducir el número de líneas de código necesarias para operar sobre ellas. En java 8 se esperan novedades en este sentido por lo que no serán necesarias librerías de este tipo para este fin.

Cambiando el apellido de una lista de Personas.

```
List<Person> personInFamily = asList(new Person("Domenico"), new Person("Mario"), new Person("Irma"));
forEach(personInFamily).setLastName("Fusco");
```

Ordenando una lista de personas por su edad.

```
List<Person> sortedByAgePersons = sort(persons, on(Person.class).getAge());
```

Esta última en java sería algo así.

```
List<Person> sortedByAgePersons = new ArrayList<Person>(persons);
Collections.sort(sortedByAgePersons, new Comparator<Person>() {
    public int compare(Person p1, Person p2) {
        return Integer.valueOf(p1.getAge()).compareTo(p2.getAge());
    }
});
```

Un ejemplo en el contexto de la práctica sería el siguiente. Lo que hace el método es lo siguiente:

Primero se crea una lista con los tipos de mensajes que se desean ver.

Después, de todas las notificaciones (variable `notificaciones`), se queda con aquellas notificaciones cuyo tipo es de uno de los tipos recolectados en el paso anterior.

Finalmente ordena las notificaciones filtradas (variable `notificacionesFiltradas`) por fecha y las devuelve como retorno de la función.

```
public List<Notificacion> getNotificacionesFiltradas()
{
    List <TipoNotificacion> tipos = new ArrayList<TipoNotificacion>();
    if(filtro.isAmigos()){
        tipos.addAll(Arrays.asList(TipoNotificacion.NOTIFICACION_CONTACTO_PETICION_ACEPTADA,TipoNotificacion.NOTIFICACION_CONTACTO_PETICION_RECHAZADA,TipoNotificacion.NOTIFICACION_CONTACTO_PETICION_SOLICITADA));
    }
    if(filtro.isGrupos()){tipos.addAll(Arrays.asList(TipoNotificacion.NOTIFICACION_GRUPO_PETICION_ACEPTADA,TipoNotificacion.NOTIFICACION_GRUPO_PETICION_RECHAZADA));}
}
```

```
ADA, TipoNotificacion.NOTIFICACION_GRUPO_PETICION_SOLICITADA));}
if(filtro.isMensajes()){tipos.add(TipoNotificacion.NOTIFICACION_MENSAJE);}

//List<Integer> biggerThan3 = filter(greaterThan(3), asList(1, 2, 3, 4, 5));
List<Notificacion> notificacionesFiltradas=
    filter(
        having(
            on(Notificacion.class).getTipo(),
                isIn(tipos)
            )
        ,
            notificaciones);
notificacionesFiltradas = sort(notificacionesFiltradas,
on(Notificacion.class).getFecha());
//    System.out.println(notificacionesFiltradas.size() +
"notificaciones filtradas");
return notificacionesFiltradas;
}
```

Servlet Filters

Para el manejo del control de acceso a nuestras páginas vimos en los boletines cómo hacerlo cuando aplicábamos nosotros el patrón vista controlador, sin embargo no se comentó nada de cómo hacerlo (o por lo menos no encontré referencias al respecto en la documentación) en el caso de que se haga uso de otros planteamientos.

Conocía de dos opciones para hacer esto aparte de la implementación manual del mvc, inmiserirme en el ciclo de vida a nivel de Servlet (con Filtros) o inmiserirme en el ciclo de vida de una página JSF (usando Phase Listener). Buscando información por internet llegué a la conclusión de que la solución más adoptada e idónea era la de los Servlet Filters.

He implementado 2 Servlet Filters. Uno para controlar quien accede a las páginas de usuarios no autenticados, otro para controlar quien accede a las páginas para usuarios autenticados.

No tiene sentido que un usuario sin autenticar acceda a messages.faces, por ejemplo.

Podríamos haber añadido un atributo administrador a la entidad usuario y crear un filtro para que solo aquel usuario administrador pueda acceder a auxiliary.faces pues no tiene sentido que un usuario cualquiera elimine y cree categorías.

El funcionamiento del filtro es muy sencillo.

Creamos una clase que implemente javax.servlet.Filter y la anotamos con la anotación @WebFilter que tiene un parámetro urlPatterns en el que le indicamos el patrón que cumplen aquellas páginas que queremos filtrar. Se pueden utilizar expresiones o bien indicar la lista de páginas.

Luego dentro del método doFilter realizamos las operaciones que creamos oportunas para dirimir que hacemos.

Si queremos que prosiga el proceso hasta la página destino utilizamos el método chain.doFilter(request, response);

En otro caso podemos redirigir al usuario a otra página en este caso de ejemplo como resulta que el usuario que intenta acceder a login.faces ya está autenticado lo redirigimos a home.faces.

```
package view;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
```

```

import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import beans.Usuario;

@WebFilter(
urlPatterns={"/login.faces","/login_error_max_attempts.faces","/login_error.faces","/register.faces"})
public class LoggedUserFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws ServletException, IOException {
        System.out.println("filtro");
        System.out.println("entré logged");
        HttpServletRequest req = (HttpServletRequest) request;
        Usuario auth = (Usuario) req.getSession().getAttribute("loggedUser");

        // log(request.getServletContext(),request, response, "algo");

        if (auth == null) {
            System.out.println("entré");
            // User is logged in, so just continue request.
            chain.doFilter(request, response);
        } else {
            // User is not logged in, so redirect to index.
            System.out.println("no entré");
            HttpServletResponse res = (HttpServletResponse) response;
            res.sendRedirect(req.getContextPath() + "/home.faces");
        }
    }

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
    }

    @Override
    public void init(FilterConfig arg0) throws ServletException {
        // TODO Auto-generated method stub
    }
}

```

Gestión de foto de perfil

En la práctica se ofrece la funcionalidad de almacenar una imagen de perfil asociada a la cuenta de usuario. Para ello el usuario puede subir una foto, manipularla,(Seleccionar un cuadrante dentro de una imagen) y almacenarla en la base de datos junto con los datos de usuario.

Para ello se hace uso de 2 componentes de Primefaces:

- p:fileupload. Para subir un fichero.
- p:imageCropper. Para seleccionar un recuadro dentro de la imagen subida que será la foto de perfil.

Ambos componentes gestionados por el backing bean ImageCropperBean.

También se programa un Servlet (ServletImagen) que sirve la imagen de perfil en base al identificador del usuario pasado por parámetro.

Requisitos básicos no implementados

Todos los requisitos básicos fueron implementados, salvo JMS por haberlo aprobado recientemente en la asignatura de "Arquitectura del software".

Otros

No hay nada que quiera comentar.