Universidad Nacional Autónoma de México

Facultad de Ciencias

# Desarrollo rápido de sistemas Web complejos modificando y haciendo uso de portales de gestión de contenidos.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:
ARMANDO PALACIOS TREJO

DIRECTOR DE TESIS:
M. EN C. MARÍA GUADALUPE ELENA IBARGÜENGOITIA GONZÁLEZ

2012

**Datos del Jurado**

| **1. Datos del alumno** | **1. Datos del alumno** |
|---|---|
| Apellido paterno | Palacios |
| Apellido materno | Trejo |
| Nombre | Armando |
| Teléfono | 55 84 81 49 |
| Universidad Nacional Autónoma de México | Universidad Nacional Autónoma de México |
| Facultad de Ciencias | Facultad de Ciencias |
| Carrera | Ciencias de la Computación |
| Número de cuenta | 302066476 |
| 2. Datos del tutor | 2. Datos del tutor |
| Grado | M. en C. |
| Nombre(s) | María Guadalupe Elena |
| Apellido paterno | Ibargüengoitia |
| Apellido materno | González |
| 3. Datos del sinodal 1 | 3. Datos del sinodal 1 |
| Grado | M. en C. |
| Nombre(s) | Araceli Eugenia |
| Apellido paterno | Mercado |
| Apellido materno | Fernández |
| 4. Datos del sinodal 2 | 4. Datos del sinodal 2 |
| Grado | Mat. (actualmente coordinador de la carrera) |
| Nombre(s) | Salvador |
| Apellido paterno | López |
| Apellido materno | Mendoza |
| 5. Datos del sinodal 3 | 5. Datos del sinodal 3 |
| Grado | M. en C. |
| Nombre(s) | Gustavo Arturo |
| Apellido paterno | Márquez |
| Apellido materno | Flores |
| 6. Datos del sinodal 4 | 6. Datos del sinodal 4 |
| Grado | M. en A. |
| Nombre(s) | Karla |
| Apellido paterno | Ramírez |
| Apellido materno | Pulido |
| 7. Datos del trabajo escrito | 7. Datos del trabajo escrito |
| Título | Desarrollo rápido de sistemas Web complejos modificando y haciendo uso de portales de gestión de contenidos |
| Número de páginas | 160 p |
| Año | 2012 |

Dedicado a mi madre y demás familia que me ha apoyado,
dedicado a amigos y personas que de alguna forma me han hecho ser lo que soy ahora,
dedicado en sí a toda la complejidad que nos rodea.

# Índice general

# Índice de figuras

# Índice de cuadros

# Introducción

Este trabajo de tesis está orientado a la comunidad de programadores que pretenden crear un sistema complejo y seguro haciendo uso de varias tecnologías útiles para no crear todo un sistema Web desde cero.

En este trabajo de tesis se mostrará las distintas maneras alternativas que se tienen para crear aplicaciones para portales de gestión de contenidos, las cuales en conjunto podrían resultar en un sistema Web muy complejo y robusto, con la ventaja de ser de cierta forma altamente configurable.

Además se explicarán distintas maneras para crear *portlets* que no se muestran en los libros enfocados a los portales debido a que los ejemplos se hacen de forma menos complicada y se explicarán varios conceptos relacionados con tecnologías Java y temas que se deben tener en cuenta para la programación Web con *J*ava.

Se verá como ejemplo para futuros desarrollos de sistemas mas complejos, la forma de crear tres diferentes sistemas que trabajen de forma independiente dentro del portal, empezando desde un sistema básico hasta uno más avanzado. El primer sistema muestra una forma básica de hacer búsquedas de usuarios que pertenezcan al sistema. El segundo sistema hace uso de tecnologías un poco más complicadas como Hibernate para acceder a la base de datos manejando objetos y JQuery que trabaja de lado del cliente para utilizar *AJAX*, tecnologías que se explicarán en el Capítulo 5, misma que será de gran utilidad al implementar un sistema dinámico, es decir, un sistema cuyo estado evoluciona con el tiempo; este sistema mostrará a los alumnos que pertenecen a un grupo de una institución educativa al seleccionar dicho grupo y mostrando el resultado sin recargar la página, la ventaja de tener este comportamiento es que no se satura el servidor de peticiones a procesar, debido a que la mayoría de peticiones se procesan de lado del cliente haciendo la aplicación mas rápida. El tercer sistema muestra ciertas ventajas de como manejar restricciones de los usuarios que han sido registrados en el sistema, esto con la ventaja que nos permite un portal de gestión de contenidos como Liferay para manejar permisos. La finalidad de mostrar como crear estos sistemas es para hacer uso de portales de gestión de contenidos utilizando la tecnología de *portlets y* Liferay la cual se explica más adelante.

Como proyecto simularemos una empresa compuesta de tres departamentos, la cual requiere restringir acceso a los usuarios de cada departamento a las páginas del portal dependiendo de sus permisos. A la empresa le interesa un sistema que muestre gráficas con la suma de ganancias de cada mes y la suma de pérdidas respecto al mes anterior, así como las gráficas de ganancias y pérdidas anuales. Las personas de cada departamento

solamente podrán ver las gráficas que corresponden a su departamento, cabe mencionar que el administrador del sistema y ciertas personas con ciertos privilegios podrán ver las gráficas correspondientes a todos los departamentos, así como de las ganancias y pérdidas totales *(de todos los departamentos en una gráfica)*. En este sistema se podrá elegir el tipo de gráfica animada, así como el rango de búsqueda y otros tipos de datos para mostrar la gráfica como se desee. Las gráficas se actualizaran sin tener que cambiar de página. Los datos a los que se tendrá acceso dependerán del grupo a los que el usuario pertenezca. El objetivo de esto es exponer las tecnologías que nos ofrece el portal así como algunas otras utilizadas en la Web para crear sistemas que de cierta forma al utilizarlas, nos facilitan su desarrollo.

El sistema mostrará el uso de portales de gestión de contenidos, con esto se ahorra el tener que desarrollar una manera de acceso de usuarios al sistema desde cero, así como la seguridad y la administración del sistema, como por ejemplo asignar permisos a usuarios y estos a documentos que se ingresan al sistema, así como la forma para acceder a las páginas del portal.

Durante la revisión sobre este tipo de tecnologías he encontrado que la mayoría de los sistemas que usan éstas, han sido creados de una manera tal que pudo haber sido posible de forma más sencilla. Otras personas simplemente tratan de modificar aplicaciones ya creadas para los portales de gestión de contenidos, lo cual es ineficiente ya que puede llevar mucho tiempo entenderlas.

Por último cabe mencionar que este trabajo de tesis se puede citar para manejar diversas tecnologías utilizadas hoy en día en la Web, pero con la ventaja de que se utilizarán en conjunto con un portal de gestión de contenidos, el cual nos ofrece aplicaciones ya desarrollas que nos pueden ser útiles para diferentes proyectos donde se deban crear permisos a las actividades que realizarán los usuarios que accedan y las cuales no tendremos que crear ya que vienen creadas para el mismo portal, un ejemplo de esto es una aplicación para subir documentos y guardarlos en el servidor, la desventaja de utilizar los portales con tecnología de portlets es que son pesados para un servidor con poca memoria RAM ya que se ejecutan muchos procesos para restringir el acceso de usuarios al sistema, además se ejecutan diferentes aplicaciones creadas para el mismo.

# 1. Conceptos básicos

La razón principal para escoger este tema de trabajo de tesis se debe a mi experiencia como desarrollador de una ***Intranet***[1] en la cual la principal preocupación fue la seguridad del sistema en el que se administraba el acceso a los contenidos del mismo.

La *Web 2.0* está comúnmente asociada con aplicaciones Web que facilitan compartir información, interoperabilidad, diseño centrado en el usuario y la colaboración en la red. Ejemplos de la *Web 2.0* son las comunidades Web, las aplicaciones Web, los servicios de red social, los servicios de alojamiento de videos, las *wikis, blogs*, etc. [46]. Muchas de las aplicaciones que asocian la Web 2.0 las podemos encontrar en un portal de gestión de contenidos como Liferay.

Más adelante, se describirá de forma sencilla el desarrollo de ***portlets***[2] y la interacción de éstos con un ***Sistema de Gestión de Contenidos o CMS*** por sus siglas en inglés, en este caso uno de los mas populares desarrollado en Java llamado Liferay. Se hará poco uso de *marcos de trabajo ("frameworks")* para que al final los desarrolladores si gustan, puedan hacer uso del ***marco de trabajo***[3] que prefieran. También utilizaremos diversos *marcos de trabajo* que pueden interactuar con Java en la parte de visualización como lo son ***JQuery***[4] y ***Open Flash Chart (OFC)***[5] *(en este caso utilizaré una biblioteca de Java llamada* JOFC2 *la cual mostraré como modificarla para diversos usos que no trae por defecto)*. Se profundizará en la solución para corregir diversos problemas como cuando en el servidor Tomcat no se puede visualizar aplicaciones creadas en *Adobe Flash*[6] cuando se utiliza ***HTTPS***[7]sobre el navegador Web *IE 6.0*.

[1] "Una *Intranet* es una red de computadoras privadas que utiliza tecnología Internet para compartir dentro de una organización parte de sus sistemas de información y sistemas operacionales". [7]

[2] "Los *portlets* son componentes modulares de las interfaces de usuario gestionadas y visualizadas en un portal Web". [62]

[3] "*Marcos de trabajo (frameworks):* Dentro de Java en el ámbito específico de aplicaciones Web tenemos los marcos de trabajo: *Struts, "Java Server Faces", o Spring.* Estos marcos de trabajo de Java en la practica son conjuntos de bibliotecas (API's) para desarrollar aplicaciones Web". [47]

[4] "*JQuery* es una biblioteca que a su vez se maneja como marco de trabajo de *JavaScript*, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos *HTML*, manipular el árbol *DOM*, manejar eventos, desarrollar animaciones y agregar interacción con la tecnología *AJAX* a páginas Web". [46]

[5] "*OFC* es un marco de trabajo para hacer gráficas en Flash". [33]

[6] "*Adobe Flash* utiliza gráficos vectoriales y gráficos de píxeles, sonido, código de programa, flujo de vídeo y audio bidireccional *(el flujo de subida sólo está disponible si se usa conjuntamente con Macromedia Flash Communication Server)*. En sentido estricto, Flash es el entorno de desarrollo y Flash Player es el reproductor utilizado para visualizar los archivos generados con Flash. En otras palabras, *Adobe Flash* crea y edita las animaciones o archivos multimedia y Adobe Flash Player las reproduce". [14]

[7] "*Protocolo seguro de transferencia de hipertexto (Hyper Text Transfer Protocol Secure).* Es un protocolo

Se debe conocer el concepto de ***JavaScript asíncrono y XML o AJAX ("Asynchronous JavaScript And XML")*** la cual es una técnica de desarrollo Web para crear aplicaciones interactivas o ***RIA ("Rich Internet Applications")***. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas Web sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones [58].

Se mencionará la manera en que está formada la base de datos de Liferay teórica y prácticamente. Si se desea profundizar mas sobre diferentes *marcos de trabajo* o el uso de diversos componentes de Liferay como sus editores ***Lo que ves es lo que obtienes (WYSIWYG)***[8] [51] existe documentación en la red y libros.

Para este trabajo de tesis se utilizó del sistema operativo *GNU/Linux* aunque similarmente, se puede trabajar en el Sistema Operativo Windows. Las tecnologías utilizadas a las que se dará una introducción en la Tesis son: `Java 1.6.0.24` *`(JSPs y portlets)`*, `Ant 1.7.1-31.1`, `JQuery`, `Eclipse 3.6 Helios`, `PostgreSQL 9.0`, `Hibernate`, Servidores `Glassfish V3` y `Tomcat 7.0.11`, `Liferay 6.0.5`, `JQuery` y `Open Flash Chart`. ya que son mantenidas por una gran comunidad de usuarios y ofrecen mucha estabilidad para sistemas Web. [10] [60] [58] [33] .

## 1.1. Sistemas de Gestión de Contenido, Liferay y portlets

***Un Sistema de Gestión de Contenidos o (CMS en ingles)*** es una colección de procedimientos utilizados para administrar flujos de trabajo en un entorno colaborativo [27]. Los procedimientos pueden ser:

1. Permitir a un gran número de personas contribuir e intercambiar datos almacenados.

2. Controlar acceso a datos basados en roles. En el caso de Liferay se explicará como se organizan los roles, grupos y usuarios.

3. Facilitar el almacenamiento y recuperación de datos.

4. Reducir el duplicado de datos a la base de datos ya que se tiene un mejor control sobre los datos.

---

de aplicación basado en el protocolo *HTTP*, destinado a la transferencia segura de datos de Hiper Texto, es decir, es la versión segura de *HTTP*. Es utilizado principalmente por entidades bancarias, tiendas en línea, y cualquier tipo de servicio que requiera el envío de datos personales o contraseñas". [57]

[8] "*What You See Is What You Get* se aplica a los procesadores de texto y otros editores de texto con formato *(como los editores de HTML)* que permiten escribir un documento viendo directamente el resultado final, frecuentemente el resultado impreso". [51]

5. Mejorar la facilidad de hacer reportes escritos ya que se pueden utilizar editores *Lo que ves es lo que obtienes (WYSIWYG)* que vienen integrados al sistema.

6. Mejorar la comunicación entre los usuarios.

Es un sistema usado para administrar el contenido de un sitio Web. Típicamente un *CMS* consiste de dos elementos: ***La aplicación de gestión de contenido (CMA en inglés)*** y ***la aplicación de entrega de contenido o (CDA en inglés)***, el elemento *CMA* permite a una persona o autor, quien no tiene conocimiento de *Lenguaje de Marcado de Hipertexto (HTML en inglés)*, administrar la creación, modificación y remover el contenido de un sitio Web sin necesidad de ser un experto en programación Web o Webmaster. El elemento *CDA* usa y compila esa información para actualizar el sitio Web. Las características de un *CMS* varían según la madurez del *CMS*; sin embargo la mayoría incluye entre sus características publicación basada en Web, gestionador de formatos, control de versión, indexación, búsqueda y entrega de datos [50].

- ***Publicación basada en Web.*** Permite a los usuarios usar una plantilla o varios modelos de plantillas que conforman el aspecto estético del sitio Web, así como de asistentes y otras herramientas para crear o modificar su contenido[62] .

- ***Administrador de formatos.*** Permite incluir documentos electrónicos y escaneados para ser convertidos a *HTML* o *PDF (Formato de Documento Portable)* para el sitio Web [62].

- ***Control de versión.*** Señala cualquier cambio a los archivos hechos por los usuarios. Permite al contenido ser actualizado a una nueva versión o regresar a una versión previa [62].

- ***Indexación, búsqueda y recuperación.*** El CMS indexa y ordena todos los datos dentro de una organización, los usuarios pueden buscar por etiquetas o formularios datos o archivos usando palabras clave. [62].

Existen diferentes *tipos de CMS* [1], entre ellos:

- ***Gestión de portales.*** Estos CMS sirven para gestionar todo el contenido de un sitio Web ofreciendo diversas funcionalidades; entre ellos se encuentran *Liferay* y *Joomla* [62] [25].

- ***Blogs.*** Estos permiten la publicación de noticias en orden cronológico, permitiendo comentarios [36].

- ***Gestor de foros.*** Permiten la discusión en línea de usuarios, entre ellos se encuentran *phpBB, SMF, MyBB* [35].

- ***Wikis.*** Es un tipo especial de Web que permite la creación colaborativa de contenidos, entre ellos *se encuentran Wikipedia, MediaWiki y TikiWiki* [55].

- ***Gestores de comercio electrónico.*** Permiten generar sitios Web específicos para comercio electrónico, entre ellos *se encuentra osCommerce* [24].
- ***Galerías.*** Permiten generar y administrar automáticamente un portal o sitio Web que muestra contenido audiovisual, normalmente imágenes, entre ellos se encuentran *Gallery y Dragonfly CMS.*
- ***Gestores de E-Learning.*** Son los denominados *LMS (Learning Management Systems o Sistemas de Gestión de aprendizaje)*, satisfacen diferentes particularidades para el proceso de enseñanza y aprendizaje, entre ellos se encuentra *Moodle.*

El portal de Liferay, como portal de código libre, provee una interfaz unificada Web a los datos y diversas herramientas que ofrece. La interfaz del portal está compuesta de portlets. Los *portlets* son desarrollados independientemente del portal en sí mismo pero también están ligados al mismo, y siguen la filosofía de ***Arquitectura Orientada a Servicios (SOA).***[9]

El portal de Liferay también navega con ***sistemas de gestión de contenidos (CMS)*** **y** ***administración de contenido Web (Web Content Management System o WCMS)***[10], el beneficio de esto es que un usuario sin experiencia en programación puede modificar el contenido Web de la página desde un editor *WYSIWYG* que viene integrado en el portal.

El CMS de Liferay provee un básico *Sistema de Gestión de Contenidos Empresarial ("Enterprise Content Management Basic System")* [62]. El portal de Liferay tiene al menos las siguientes características:

- Trabaja en la mayoría de servidores de aplicación y contenedores servlet, base de datos y sistemas operativos. Además de 700 combinaciones de implementación.
- Usa *Java, J2EE* y tecnologías para la *Web 2.0.*
- Usa un marco de trabajo de *SOA* libre.
- ***JSR-168/JSR-286***[11] compatibles.
- Se puede hacer uso de más de 60 *portlets* con diferentes funcionalidades.

[9] "*Arquitectura Orientada a Servicios (Service-oriented Arquitecture)* permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios (comúnmente pero no exclusivamente servicios Web), lo cual facilita la interacción entre diferentes sistemas propios o de terceros". [5]

[10] "*Administración de contenido Web (Web Content Management System)* es un sistema de software que provee autoría de sitios Web, colaboración y herramientas de administración designadas para permitir a usuarios con poco conocimiento de lenguajes de programación para la Web o lenguajes de etiquetado a crear y administrar el contenido de los sitios con relativa facilidad". [26]

[11] "*Java Specification Request (JSR)*, las cuales son documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java". [23]

- Páginas personalizadas para todos los usuarios.
- Interfaz de usuario para *AJAX*-habilitado.
- Soporte a múltiples lenguajes, mas de 22 lenguajes.
- Sincronización completa de ***LDAP***[12] y soporte de seguridad ***Single Sign On (SSO)***[13].
- Autorización basada en roles de forma granular.
- ***Panel de control.*** Administración centralizada para todos los contenidos, usuarios, organizaciones, comunidades, roles, recursos de servidor; máxima personalización de los portlets y las páginas del portal.
- Configuración y arrastre dinámico de ventanas o *portlets*. Contiene un buscador de archivos basado en etiquetas. Los documentos se pueden modificar sin descargarlos a la computadora local utilizando ***WebDAV***[14].
- *CMS, WCM,* colaboración y red social.

En resumen, la manera de otorgar permisos para que ciertas personas puedan o no ver diversos *portlets* es muy sencilla. Todo depende en cómo estén organizados los permisos para los *portlets.*

## 1.2. El uso de portlets y Liferay

Los *portlets* son componentes Web como servlets, específicamente a ser agregados en el contexto de una página compuesta. Usualmente muchos portlets son invocados con una petición simple de una página de un portal. Al usar portlets nos podemos dedicar a crear la lógica del sistema sin tener que enfocarnos demasiado en la seguridad de acceso a los portlets y páginas del portal, ya que este cuenta con herramientas sencillas para manejar el acceso. Los portlets pueden hacer uso de bibliotecas y marcos de trabajo que el portal ofrece ya sea para desarrollar sistemas que por ejemplo accedan a la base de datos de

[12] "*Lightweight Directory Access Protocol (en español Protocolo Ligero de Acceso a Directorios)* que hacen referencia a un protocolo a nivel de aplicación el cual permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. LDAP también es considerado una base de datos, aunque su sistema de almacenamiento puede ser diferente, a la que pueden realizarse consultas". [61]

[13] "*Single sign-on (SSO)* es un procedimiento de autenticación que habilita al usuario para acceder a varios sistemas con una sola instancia de identificación". [62]

[14] "El objetivo de *WebDAV* es hacer de la World Wide Web un medio legible y editable, en línea con la visión original de Tim Berners-Lee. Este protocolo proporciona funcionalidades para crear, cambiar y mover documentos en un servidor remoto *(típicamente un servidor Web)*. Esto se utiliza sobre todo para permitir la edición de los documentos que sirve un servidor Web, pero puede también aplicarse a sistemas de almacenamiento generales basados en Web, que pueden ser accedidos desde cualquier lugar. La mayoría de los sistemas operativos modernos proporcionan soporte para *WebDAV*, haciendo que los archivos de un servidor *WebDAV* aparezcan como almacenados en un directorio local". [6]

Liferay o crear extensiones para el mismo portal.

El uso de Liferay presenta beneficios básicos como personalización y flujo de trabajo. ***Personalización*** significa que diferentes personas con el mismo rol pueden trabajar diferente ya que cada uno puede ser capaz de poder visitar diferentes portlets a la vez y realizar diferentes procesos permitidos para el rol que tienen en común. Diferentes roles requieren diferente información haciendo uso de la personalización. Los usuarios también necesitan acceso directo a la información y aplicaciones mediante flujos de trabajo ya que se pueden estructurar diferentes tareas para cada usuario dependiendo de los permisos que tengan para acceder a los *portlets*. Liferay tiene algo que se llama puesta en escena de flujos de trabajo, se puede activar mediante el portlet de mis comunidades, en `acciones -> administrar páginas para una comunidad` y entonces activar el flujo de trabajo. El portal de Liferay es uno de los portales mas maduros para entornos de trabajo en el mercado ya que su desarrollo es abierto y varios de sus clientes lo respaldan [39] [38].

## 1.3. Arquitectura y entorno de trabajo

***La arquitectura de Liferay*** soporta la alta capacidad de aplicaciones de misión crítica, caché altamente distribuido y soporte de replicación sobre múltiples servidores.

La Figura 1.1 presenta la arquitectura del portal de Liferay [62].

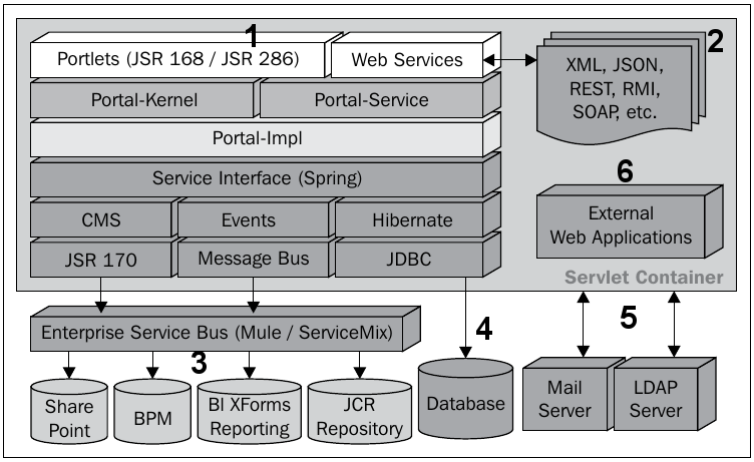


Figura 1.1.: *Arquitectura de Liferay.* Se muestran las diferentes capas de la arquitectura.

A continuación voy a explicar de forma simple las diferentes capas de la arquitectura tal y como están enumeradas en la Figura 1.1.

- Punto 1. Muestra el núcleo principal de Liferay *(Portal Kernel)*, la especificación de los *portlets (portlets (JSR))*, servicios que ofrece el *portal (Portal-Service),* ***Portal-Impl*** el cual es el conjunto de bibliotecas Java que forman el portal, eventos, y

diversos marcos de trabajo de Java como Hibernate que es una herramienta de Mapeo objeto-relacional que se conecta a la base de datos [59] y *Spring* que ofrece soluciones muy bien documentadas y fáciles de usar para las prácticas comúnes en la industria [12].

- Punto 2. Muestra los recursos para crear aplicaciones interactivas con que cuenta el portal ya que se hace uso de diversas notaciones para JavaScript como lo son *JSON* y diversos protocolos que se comunican mediante intercambio XML como lo es ***SOAP***[15] que es uno de los protocolos utilizados en los ***Servicios Web (Web service)***[16].

- Punto 3. Con Liferay se pueden integrar herramientas para procesos de negocio como lo son los ***gestionadores de procesos de negocio (BPM)*** mediante ***(BPMS)***[17], entre ellos se encuentra ***Intalio***[18].

- Punto 4. Se puede hacer uso de diferentes manejadores de bases de datos como lo son *PostgreSQL* o *MySQL*, con las cuales se puede usar ***JDBC*** [19] para efectuar operaciones sobre estas mediante Java [62].

- Punto 5. La autenticación de Liferay permite identificación de registro, conexión a un directorio *LDAP* si se dispone de un servidor *LDAP* o mediante *single Sing-On*. También se pueden añadir otras listas de servidores de correo en lugar del que está por defecto.

- Punto 6. Por ejemplo si se crea una página Web en algún otro servidor o se desea

[15] "*Simple Object Access Protocol o Protocolo de Acceso Simple de Objetos* es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML". [41]

[16] "*Servicios Web (Web service)* es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones mediante el lenguaje XML. La principal razón para usar servicios Web es que se basan en *HTTP* sobre *TCP (Transmission Control Protocol)* en el puerto 80. Dado que las organizaciones protegen sus redes mediante firewalls -que filtran y bloquean gran parte del tráfico de Internet, cierran casi todos los puertos *TCP* salvo el 80, que es, precisamente, el que usan los navegadores. Los *servicios Web* utilizan este puerto, por la simple razón de que no resultan bloqueados". [54]

[17] "*Gestión de procesos de negocio (Business Process Management o BPM en inglés)*. metodología empresarial cuyo objetivo es mejorar la eficiencia a través de la gestión sistemática de los procesos de negocio, que se deben modelar, automatizar y optimizar de forma continua. Como su nombre sugiere, *BPM* se enfoca en la administración de los procesos del negocio. Para soportar esta estrategia es necesario contar con un conjunto de herramientas que den el soporte necesario para cumplir con el ciclo de vida de *BPM*. Este conjunto de herramientas son llamadas *Business Process Management System (BPMS)*, y con ellas se construyen aplicaciones *BPM*". [53]

[18] "*Intalio* es un software Open Source basado en Java-J2EE, que implementa *BPMS*, y esta basado en un conjunto de marcos de trabajo y arquitecturas muy conocidas en la industria del software y con un madurez aceptable. Intalio utiliza la notación *BPMN* para diseñar procesos de negocio establecida por el http://www.bpmn.org/ que puede adaptarse a los requisitos de las *arquitectura orientada servicio (SOA)*". [37]

[19] "*Java Database Connectivity* es una biblioteca que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java". [10]

visualizar alguna página externa en el mismo Liferay es posible ya que se abren como cualquier otro *portlet* utilizando *marcos incorporados (IFRAMES)* [54] .

Para instalar Liferay es necesario un manejador de base de datos y, para que funcione correctamente es conveniente contar como mínimo 1GB en memoria RAM.

El flujo básico de trabajo de la arquitectura comienza en el núcleo del portal, el cual es el encargado de que las aplicaciones del portal puedan trabajar juntas. Los portlets es un ejemplo de estas aplicaciones. A su vez casi tan importante como el núcleo se encuentra `Portal-Impl` ya que contiene varias bibliotecas que ayudan a la creación de portlets *(punto 1)*.

Un portlet puede recibir datos mediante llamadas a *Servicios Web*, básicamente hace peticiones y recibe una respuesta. El portal soporta *Servicios Web* para hacer que diferentes aplicaciones se comuniquen de forma fácil. *Java, .Net* y aplicaciones propietarias pueden trabajar juntas porque los *Servicios Web* usan estándares *XML (Punto 2)*. [62]

El ***Bus de Servicio Empresarial (Enterprise Service Bus)*** es un manejador de conexión central que permite a las aplicaciones y servicios ser agregados rápidamente a una infraestructura empresarial. Cuando una aplicación necesita ser reemplazada puede ser rápidamente desconectada del bus en un solo punto *(Punto 3)*.

Los capítulos siguientes suponen que ya se ha instalado y configurado Liferay con su respectiva base de datos tal y como se explica en el Anexo.

# 2. Portlets y ambiente de desarrollo utilizando el IDE Eclipse

El objetivo de este capítulo es dar una introducción a la programación de *portlets* creando un *portlet* básico que acceda a datos de la base de datos del portal de una manera explícita. De forma similar se podrán crear portlets que puedan acceder a la base de datos del portal con el propósito que el usuario requiera dependiendo de sus necesidades.

Lo primero es mostrar una introducción a la interfaz de programación de aplicaciones o *API (del inglés Application Programming Interface)* de portlets que es el conjunto de funciones y procedimientos *(o métodos, en la programación orientada a objetos)* que ofrece cierta biblioteca para ser utilizados por los portlets como una capa de abstracción [56].

El interfaz de programación de aplicaciones ofrece los métodos necesarios para que el portlet se pueda visualizar en el portal así como diversas acciones para que cubra una cierta lógica con su interacción con el usuario lo cual se mostrará adelante . [2]

## 2.1. Introducción a la programación, API y lógica de portlets

En este apartado se verá la especificación de *portlets Java,* la especificación estandariza el *API* del que disponen los portlets, ver la Figura 2.1.

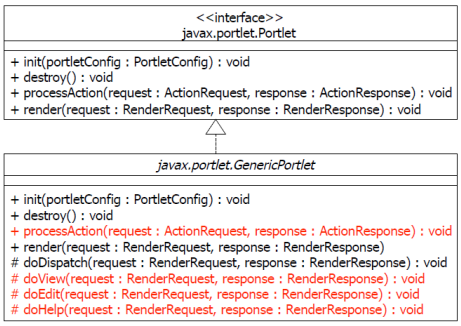


Figura 2.1.: *Interfaz de programación para los portlets (API de portlets).* Estandarización del *API* que disponen los portlets.

Todo portlet debe implementar `javax.portlet.Portlet`. Además `javax.portlet.GenericPortlet` provee una implementación por defecto para la interfaz portlet para desarrollar estos [2].

Haciendo uso de la primitiva `init` el contenedor crea una instancia del portlet, con el uso de la primitiva `destroy` el contenedor decide destruirla.

Al programar *portlets* se pueden definir tamaños de ventana y diferentes modos de portlet.

- `WindowState.NORMAL` *(normal).*
- `WindowState.MAXIMIZED` *(maximizada)*
- `WindowState.MINIMIZED` *(minimizada)*

Existen dos tipos de peticiones en los métodos de un portlet:

- *Petición de acción (action request)*: Son peticiones que modifican el estado de un portlet o causan una redirección. Los resultados varían con cada petición y se procesan redefiniendo el método *processAction,* ver la Figura 2.2.

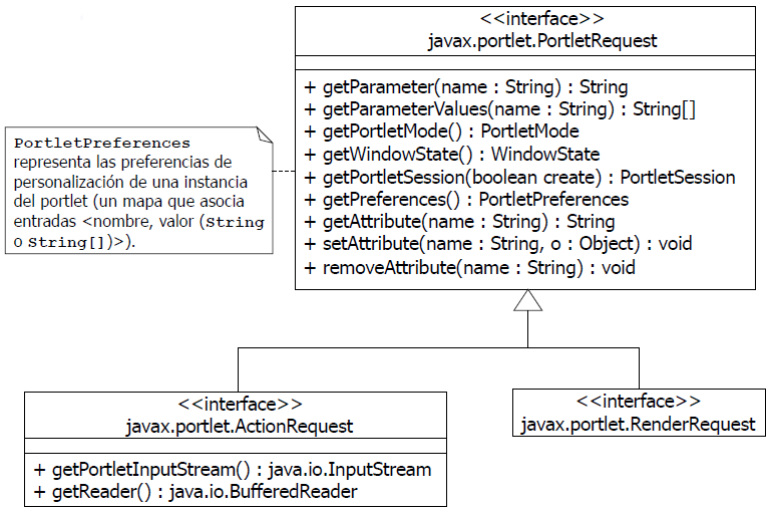


Figura 2.2.: *Petición de acción.* API para modificar el estado de un portlet.

- *Petición de redibujado (render request)*: Son peticiones para solicitar el *marcado* o datos que se mandan o solicitan para su interacción con la aplicación del portlet. La lógica de los resultados no varía con cada petición. El contenedor de los *portlets* invoca al redibujado de los datos en el portal; `GenericPortlet` lo implementa como un método plantilla, que entre otras cosas invoca a los métodos `doDispatch` *(Envía el flujo a donde sea necesario según un parámetro del request)*, `doEdit` *(Método de ayuda para servir el modo de edición)* y `doHelp` *(Método para servir el modo de ayuda)*, dependiendo del modo seleccionado. El método `doDispatch` proporcionado

por la clase `GenericPortlet` no invoca a ningún método de redibujado cuando `WindowState = minimized`, ver la Figura 2.3.

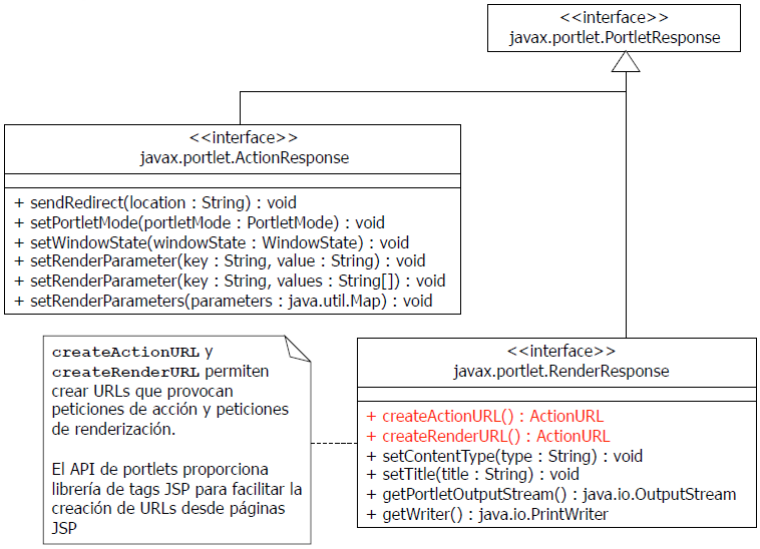


Figura 2.3.: *Petición de redibujado.* Métodos para interactuar con la aplicación del *portlet.*

- *Los métodos de redibujado* son los únicos que pueden generar marcado mediante la clase `RenderResponse`. El usuario interactúa sobre una página del portal que contiene los portlets 1,2,...,N. Se envían peticiones al portlet para manejar un redibujado del mismo, ver la Figura 2.4.

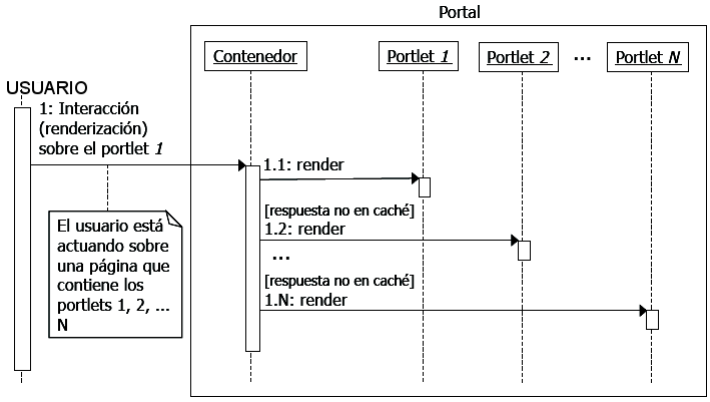


Figura 2.4.: *Procesamiento de peticiones de redibujado (renderización).*

- Una petición de acción sobre un portlet siempre va seguida de una petición de

redibujado sobre ese *portlet*, y sobre el resto de *portlets* de esa página cuyo contenido no esté cacheado para regenerar la página. Los métodos de redibujado pueden delegar la generación de marcado en una página JSP *(lo normal)* o un servlet de aplicación *portlet*. Por ejemplo el usuario puede enviar una solicitud para el redibujado de la página *(la cual ya se vuelve una petición de redibujado)* y al mismo tiempo pedir que el estado de la ventana del portlet cambie, ver la Figura 2.5.

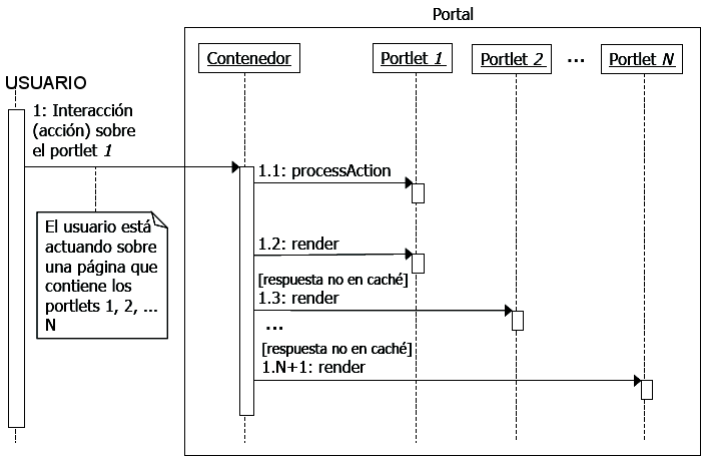


Figura 2.5.: *Procesamiento de peticiones de acción.*

Se explicará la razón por la cual el API de portlets necesita distinguir entre peticiones *(métodos)* de redibujado y acción. Asumiremos que la respuesta de los portlets no están en caché, y aunque no fuese así, tendríamos que suponerlo porque en cualquier momento puede vencer el tiempo máximo en caché.

- Supongamos todas las peticiones pueden generar marcado, es decir que todas las peticiones fuesen de redibujado.
- El usuario realiza una interacción sobre el *portlet 1* que conlleva una operación no idempotente *(añadir una cantidad de dinero a una cuenta en BD)*.
- El portal invoca el método de redibujado sobre el *portlet 1* y sobre el resto de portlets de la página *(para regenerarla)*.
- A continuación el usuario realiza una interacción sobre el *portlet 2* en la misma página.
- El portal invoca el método de redibujado sobre el *portlet 2* y sobre el resto de *portlets* de la página *(para regenerarla)*.
- La operación no idempotente sobre el *portlet 1* se vuelve a ejecutar.

Distinguiendo entre operaciones de redibujado y acción no existe este problema.

- El usuario realiza una interacción sobre el *portlet 1* que conlleva una operación no idempotente *(añadir una cantidad de dinero a una cuenta en BD).*
- El portal invoca primero el método de acción que realiza la operación no idempotente sobre el *portlet 1*, y después el método de redibujado que devuelve una respuesta visual que muestra su estado sobre el propio *portlet 1* y el resto de p*ortlets* de la misma página para regenerarla.
- A continuación, el usuario realiza una interacción sobre el *portlet 2* en la misma página.
- El portal invoca el método de acción sobre el *portlet 2* si la interacción causó una petición de acción, y en cualquier caso, invoca el método de redibujado sobre todos los portlets de la página *(para regenerarla).*
- La operación idempotente sobre el *portlet 1* no se vuelve a ejecutar *(solamente el redibujado que muestra su estado).*

*RenderResponse* permite crear URLs que causan *peticiones de acción (createActionURL) o redibujado (createRenderURL).*

Cuando el usuario realiza una petición de acción/redibujado sobre un *portlet*, el contenedor debe invocar los métodos `processAction/render` sobre ese portlet con los parámetros asociados a la petición.

En principio la petición de redibujado que sigue una petición de acción sobre un *portlet* no debe propagar los parámetros de la petición de acción. Si el *portlet* quiere establecer parámetros *(durante el procesamiento de petición)* para la petición de redibujado, tiene que hacerlo mediante los métodos. `ActionResponse.setRenderParameter`(-s).

Siempre que se invoca una petición de redibujado sobre un *portlet (suponiendo su respuesta no estaba en el Caché del servidor)* como consecuencia de una petición de acción/redibujado dirigida a otro *portlet* de la misma página, el portal debe utilizar los mismos parámetros que en la anterior invocación de redibujado, de esta manera cada vez que se realiza una interacción sobre un p*ortlet*, la página generada por el portal mostrará el resto de p*ortlets* en el estado anterior. En algunos servidores, los parámetros no se conservan.

Cada *portlet* sólo ve los parámetros de la petición dirigida hacia él.

Al igual que las aplicaciones Web, los *portlets* pueden disponer de sesión *(PortletSession)* para mantener datos mientras un usuario interactúa con el *portlet*. Métodos como `PortletRequest.getPortletSession`, `PortletSession.setAttribute` *`(name, value, scope)`* o `PortletSession.getAttribute`*`(name, scope)`*.

- `scope: PortletSession.APPLICATION_SCOPE` *(atributo disponible para cualquier portlet de la aplicación portlet en la misma sesión)* o `PortletSession.PORTLET_SCOPE` *(Atributo disponible a las peticiones dirigidas a esa instancia del portlet en la misma sesión).*

Los atributos de la sesión del *portlet* se almacenan en atributos de la sesión `javax.servlet.http.HttpSession`.

Los atributos con `scope = portletSession.APPLICATION_SCOPE` se almacenan con el mismo nombre y los atributos con `scope = PortletSession.PORTLET_SCOPE`, se almacenan con nombre codificado.

## 2.2. Creación e instalación del primer portlet

En esta sección se mostrará la creación de un *portlet* básico como ejemplo introductorio de creación de *portlets*, empezaremos con uno sencillo que después se volverá más complicado ya que interactuará con la base de datos de Liferay y será construido usando bibliotecas de *Java*.

Lo primero es ir a la carpeta de `liferay-portal-6.0.5/liferay-plugins-sdk-6.0.5/portlets/`

Al ejecutar en la consola el siguiente comando, se crea la estructura de un proyecto de portlet llamado primer-portlet para comenzar a programar sobre este, ver la Figura 2.6.

```
ant -Dportlet.name=primer -Dportlet.display.name="primer portlet" create
```

```
arpalaci@linux-5904:~/Tesis/portal/liferay-portal-6.0.5/liferay-plugins-sdk-6.0.5/portlets>
 ant -Dportlet.name=primer -Dportlet.display.name="primer portlet" create
Buildfile: build.xml

create:
    [unzip] Expanding: /home/arpalaci/Tesis/portal/liferay-portal-6.0.5/liferay-plugins-sdk
-6.0.5/portlets/portlet.zip into /home/arpalaci/Tesis/portal/liferay-portal-6.0.5/liferay-p
lugins-sdk-6.0.5/portlets/primer-portlet
    [mkdir] Created dir: /home/arpalaci/Tesis/portal/liferay-portal-6.0.5/liferay-plugins-s
dk-6.0.5/portlets/primer-portlet/docroot/WEB-INF/tld
     [copy] Copying 6 files to /home/arpalaci/Tesis/portal/liferay-portal-6.0.5/liferay-plu
gins-sdk-6.0.5/portlets/primer-portlet/docroot/WEB-INF/tld

BUILD SUCCESSFUL
Total time: 0 seconds
```

Figura 2.6.: *Creación de la estructura de un proyecto*, se crea la estructura de un proyecto para programar un portlet vista en la terminal.

Este comando creará los archivos necesarios para poder comenzar a construir nuestro primer *portlet*. A continuación tendremos que modificarlo para abrirlo como proyecto en Eclipse *(definido en el apéndice B.4)* e implementarlo al portal de Liferay.

Copiamos la carpeta llamada `primer-portlet` que se creó, en el *espacio de trabajo* de Eclipse. Dentro de la carpeta `primer-portlet` creamos dos archivos, uno llamado `.classpath` y otro llamado `.project` [1].

Ejemplo del archivo `.classpath`, ver el Cuadro 2.1 [32].

Las opciones son:
*src.* Ruta a los archivos de código fuente.
*output.* Ruta para generar los archivos binarios de los archivos de código fuente.
*con.* Ruta a la biblioteca de eclipse.

Código de archivo .classpath

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <classpath>
3  <classpathentry kind="src" path="docroot/WEB-INF/src"/>
4  <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER/
      org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-1.6"/
     >
5  <classpathentry kind="output" path="bin"/>
6 </classpath>
```

Cuadro 2.1.: *Código de archivo .classpath.* Archivo para especificar la ruta a los archivos de código fuente del proyecto.

Ejemplo del archivo `.project`, ver el Cuadro 2.2:

Las opciones son:
*name.* Nombre del proyecto.
*comment.* Comentario del proyecto.
*projects.* Los nombres de los proyectos que son referenciados por este proyecto.
Las demás opciones corresponden a bibliotecas de Eclipse.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <projectDescription>
3  <name>primer-portlet</name>
4  <comment></comment>
5  <projects>
6  </projects>
7  <buildSpec>
8   <buildCommand>
```

[1] Los archivos *classpath (donde especificamos la ruta de los archivos de código fuente) y .proyect (Archivo para abrir el proyecto como si fuera creado por eclipse)* los puede crear Eclipse al crear un nuevo proyecto Java, sin embargo se tendrán que seguir modificando. [32]

```
9       <name>org.eclipse.jdt.core.javabuilder</name>
10      <arguments>
11      </arguments>
12     </buildCommand>
13    </buildSpec>
14    <natures>
15     <nature>org.eclipse.jdt.core.javanature</nature>
16    </natures>
17  </projectDescription>
```

Cuadro 2.2.: *Código de archivo .project.* Archivo para que el proyecto se autodescriba para *Eclipse.*

Después vamos a la carpeta `primer-portlet/docroot/WEB-INF/` y creamos una carpeta llamada `src/` y un archivo llamado `web.xml` el cual evitará que Eclipse muestre un error que dice que no encuentra la dirección Web al abrir el proyecto, ver Cuadro 2.3.

```
1  <web-app>
2      <taglib>
3          <taglib-uri>http://java.sun.com/portlet_2_0</taglib-uri>
4          <taglib-location>/WEB-INF/tld/liferay-portlet.tld</taglib-location>
5      </taglib>
6  </web-app>
```

Cuadro 2.3.: *Código de archivo web.xml.* Se especifican los archivos de bibliotecas de etiquetado que usa Liferay.

Ahora vamos a la carpeta `primer-portlet/docroot/` y dentro creamos una carpeta llamada `META-INF`, dentro la carpeta creada llamada `META-INF` creamos un archivo llamado `context.xml`. Este archivo es para poder hacer un despliegue de una aplicación Web *(.war)* en el servidor de aplicaciones sin tener problemas de bloqueo de recursos o de bloqueo de paquetes de *Java*. Esto puede ser innecesario en *Glassfish* aunque en *Tomcat* es preferible crearlo, ver Cuadro 2.4.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Context antiJARLocking="true" antiResourceLocking="true"/>
```

Cuadro 2.4.: *Código de archivo context.xml.* Para desplegar una aplicación Web *(.war)* en el servidor.

Ahora podemos abrir el proyecto en Eclipse con `File - Import - General - Existing Files into Workspace`, después seleccionamos el directorio raíz, ver la Figura 2.7.

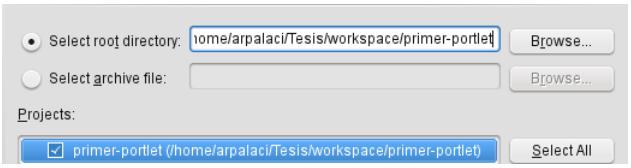


Figura 2.7.: *Abrir el proyecto,* en la imagen se muestra un pedazo de la ventana de Eclipse al abrir el proyecto.

Seleccionamos `Finalizar` y podemos observar como ya tenemos creado nuestro proyecto, ver la Figura 2.8.

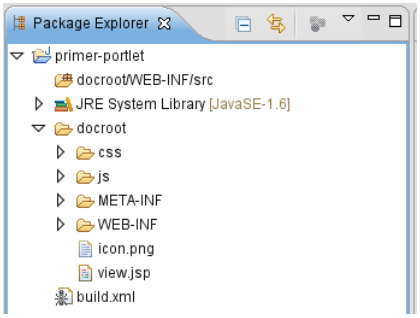


Figura 2.8.: *Estructura del proyecto abierto,* en la estructura se observa la estructura de las carpetas y archivos para comenzar a programar portlets.

Como vamos a utilizar la biblioteca ***JSTL***[2] tendremos que incluirla en nuestro proyecto, pero antes vamos a incluir la biblioteca de *PostgreSQL*. En Eclipse seleccionamos con botón derecho del ratón la primera carpeta del proyecto que es `primer-portlet`, a continuación `Build Path - Add Libraries - User Library - User Libraries - New` y llamamos a la nueva biblioteca *Referenced Library*, ver la Figura 2.9. Añadimos los archivos .jar que se encuentran en:
`liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/lib/`
al oprimir *Add JARs.* Estos archivos contienen bibliotecas que utilizaremos en nuestro proyecto.

[2] "*JavaServer Pages Standard Tag Library* especifica un conjunto de bibliotecas de etiquetas. La biblioteca Core contiene acciones para las tareas rutinarias, como incluir o excluir una parte de una página dependiendo de una condición en tiempo de ejecución, hacer un bucle sobre una colección de ítems, manipular URLs para seguimiento de sesión, y la correcta interpretación del recurso objetivo, así como acciones para importar contenido de otros recursos y redireccionar la respuesta a una URL diferente" [22].

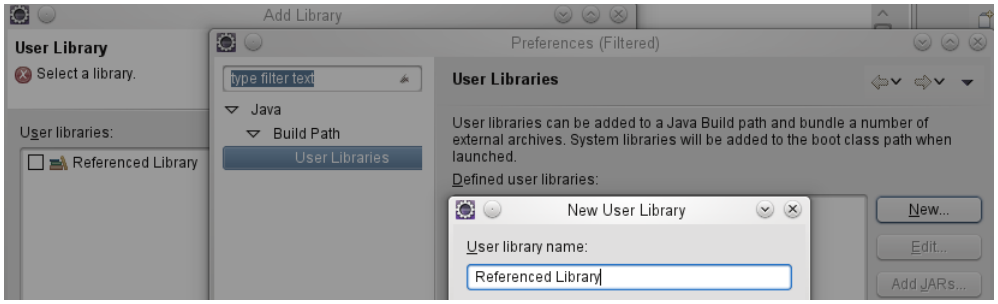


Figura 2.9.: *Creación de una nueva biblioteca donde se añaden los archivos .jar.*

A continuación vamos a agregar de forma similar las bibliotecas de Liferay en las cuales entre ellas se encuentran las bibliotecas de *JSTL*. Ahora llamamos a la nueva biblioteca Liferay y añadimos los archivos con extensión *.jar* que se encuentran en `domains/domain1/applications/liferay-portal/WEB-INF/lib/`.

Abrimos el `view.jsp` y lo modificamos para poder usar JSTL sin problemas a como se muestra en el Cuadro 2.5.

```
1 <%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
2 <%@ page isELIgnored="false" %>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4 <portlet:defineObjects />
5 This is the <b>primer portlet</b> portlet.
```

Cuadro 2.5.: *Código del archivo view.jsp.* Archivo para mostrarse como página Web ya modificado para soportar JSTL.

La expresión `page isELIgnored="true/false"` servirá para poder desactivar o activar ***Expresion Languaje (Lenguaje de Expresión)***[3].

A continuación se mostrarán dos ejemplos para mostrar la diferencia entre *JSP* sin usar *JSTL* y solo con etiquetas scriptlets, ver el Cuadro 2.6 y otro con uso de *JSTL,* ver

[3] *"Lenguaje de expresión ("Expresion Languaje EL").* Una característica de la tecnología de JSP 2.0 es que tiene soporte para EL. Una EL hace lo posible para accesar fácilmente a datos almacenados en componentes JavaBeans. Por ejemplo el EL del JSP permite a un autor de una página accesar a un bean usando simplemente una sintaxis tal como ${nombre} para una simple variable o ${nombre.foo.bar} para una propiedad anidada.

Los JavaBeans son componentes reutilizables e independientes de la plataforma. Simplemente define una clase, tratando de encapsular *(ocultar)* su implementación y mostrando al exterior *(el programador que usa el bean)* solamente los métodos y propiedades que son públicos. Sólo se muestra aquello que forma parte del servicio que el bean ofrece al exterior". [47]

el Cuadro 2.7; ambas tienen la misma lógica, obtienen una lista de direcciones, por cada dirección imprimen el nombre de un objeto del tipo *Nombres* si el campo del apellido no es vació y de longitud diferente a 0, en cualquier otro caso, imprimirá *"No hay"*. Finalmente, la página imprime la fecha actual.

```
1  <%@ page import="ejemplo.Nombres, java.util.*"%>
2  <p><h1>Customer Names</h1></p>
3  <%
4  //Obtengo una lista de direcciones de una solicitud
5  List direcciones = (List)request.getAttribute("direcciones");
6
7  //Itero sobre las direcciones.
8  Iterator direccionesIter = direcciones.iterator();
9  //Mientras haya direcciones y exista un objeto de tipo Nombres
10 //imprimo el apellido paterno, si no imprimo No hay.
11 while(direccionesIter.hasNext()) {
12  Nombres nombres = (Nombres)direccionesIter.next();
13  if((null != nombres) &&
14   (null != nombres.getLastName()) &&
15   (nombres.getLastName().length() > 0)) {
16 %>
17  <%=nombres.getLastName()%><br/>
18 <%
19  }
20  else {
21 %>
22  No hay<br/>
23 <%
24  }
25 }
26 //Se imprime la fecha actual.
27 %>
28 <p><h5>fecha actual: <%=new Date()%></h5></p>
```

Cuadro 2.6.: *Código de archivo JSP sin usar JSTL.*

```
1  <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
2  <p><h1>Customer Names</h1></p>
3  <!--Hace una iteración sobre las direcciones existentes-->
4  <c:forEach items="${direcciones}" var="address">
5   <c:choose>
6    <!--Si el objeto de tipo nombres llamado lastName no es vacio lo imprime,
         de otra manera imprime No hay-->
7    <c:when test="${not empty nombres.lastName}" >
8     <c:out value="${nombres.lastName}"/><br/>
9    </c:when>
10   <c:otherwise>
```

```
11      No hay<br/>
12    </c:otherwise>
13   </c:choose><br/>
14  </c:forEach><br/>
15  <jsp:useBean id="ahora" class="java.util.Date" />
16  <!--Imprime la hora-->
17  <p><h5>fecha actual: <c:out value="${ahora}"/></h5></p>
```

Cuadro 2.7.: *Código de archivo JSP con JSTL.*

Teniendo en ejecución el servidor de aplicaciones *Glassfish o Tomcat*, arrastramos el `build.xml` del proyecto a la ventana de *Ant* que abrimos anteriormente y lo seleccionamos oprimiendo dos veces el botón derecho del ratón. Podemos observar como se compila y despliega el proyecto correctamente en Eclipse como se muestra en la Figura 2.10. En la ventana de Eclipse llamada consola, podemos ver que se ha compilado y desplegado correctamente el proyecto para verlo como *portlet* en el portal.

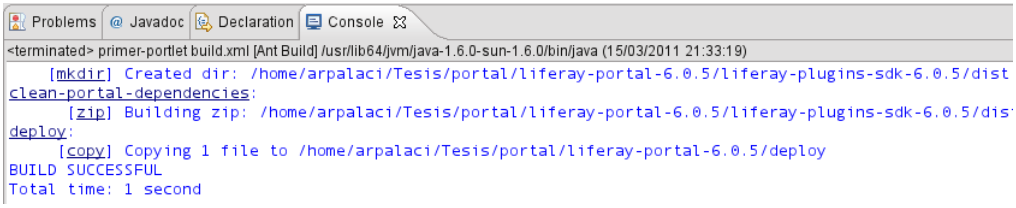


Figura 2.10.: *Compilación y despliegue del proyecto,* en la ventana de Eclipse llamada consola, se ha compilado y desplegado el proyecto.

Lo anterior implementa nuestro portlet en Liferay de una forma rápida, sencilla y cómoda ya que cada vez que hagamos cambios en nuestro proyecto tendremos que hacer lo mismo para ver reflejados los cambios en el portal, ahora podemos probar como se visualiza nuestro *portlet* al entrar a Liferay como administrador y seleccionar de la barra superior `Añadir - Más ... - Ejemplos - primer portlet` como se muestra en la Figura 2.11.


Figura 2.11.: *Despliegue del proyecto en el portal,* se coloca el primer portlet en la ventana del navegador.

A continuación vamos a crear la clase principal llamada `PrimerPortlet.java` que será la encargada de administrar el orden de ejecución de las demás clases que se utilicen en nuestro proyecto, una clase llamada `Modelo.java` que será la encargada del comportamiento del portlet dependiendo de las acciones del usuario y una clase llamada `ConexionBases.java` que será la encargada del manejo de una colección de conexiones abiertas a una base de datos de manera que puedan ser reutilizadas al realizar múltiples consultas o actualizaciones. También creamos dos nuevos archivos con extensión *.jsp* dentro de la carpeta `docroot` llamado `resultados.jsp` e `include.jsp` como se muestra en la Figura 2.12. Se observan las carpetas y archivos creados que se muestran en la ventana *Explorador de paquetes.*

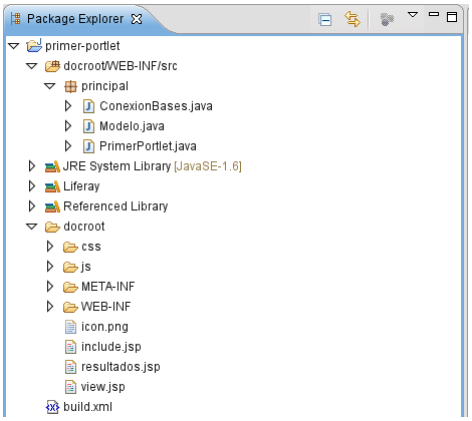


Figura 2.12.: *Esquema de los archivos creados para el proyecto,* estos se muestran en la ventana *Explorador de paquetes.*

A continuación abrimos el archivo `portlet.xml` que se encuentra en la carpeta `docroot/WEB-INF/` de nuestro proyecto. Borramos lo que se encuentra dentro de las etiquetas de `<portlet-class>` y `</portlet-class>` y en su lugar ponemos `principal.PrimerPortlet`:

`<portlet-class>principal.PrimerPortlet</portlet-class>`. Esto hace que Liferay reconozca la clase que será la principal.

### 2.2.1. Comportamiento del portlet en las búsquedas

El objetivo de nuestro primer portlet será hacer la búsqueda de los usuarios del portal mediante un filtro (*nombre y/o apellido*) en la base de datos de *Liferay* y regresar el resultado.

Antes de mostrar el código y su explicación del primer portlet en la sección 2.2.2 se mostrará el comportamiento del mismo:

- Al instalar el portlet y probarlo, primero se ejecuta el método `PrimerPortlet.init` el cual busca la página inicial señalada en el archivo `portlet.xml`.
- A continuación se ejecuta el método `PrimerPortlet.doDispatch` el cual pide el parámetro "*accion*", y como "*accion*" en este caso es *null*, `PrimerPortlet.doDispatch` llama a la superclase que redirige al método `PrimerPortlet.doView`. Que redirige al archivo `view.jsp` mostrando la página que se muestra en la Figura 2.13.

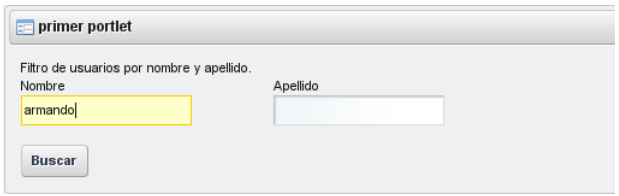


Figura 2.13.: *Ventana principal del primer portlet.*

- Al poner datos en los filtros y oprimir el botón "*Buscar*" se ejecuta primero el método `PrimerPortlet.processAction` que guarda los parámetros que se enviaron desde el formulario a la sesión. Este método a su vez hace que el resultado se vaya a mostrar en una ventana de tamaño normal y crea un parámetro de redibujado llamado "*accion*", el cual contiene el parámetro "*accion*" que se mandó desde el formulario, este parámetro de redibujado lo recibe el método `PrimerPortlet.doDispatch` que se ejecuta de nuevo para ver lo que contiene el parámetro "*accion*".
- Como ahora el parámetro *"accion" = "resultados"* ya que en este caso oprimimos el botón "*Buscar*" se ejecuta el método `PrimerPortlet.muestraResultados` que conlleva los nuevos parámetros de la sesión guardados en la petición de redibujado cuyo objeto es `RenderRequest`.
  `PrimerPortlet.muestraResultados` usando los parámetros de la sesión (*nombre y apellido*) ahora hace uso del método `Modelo.obtenResultados` para hacer la consulta de los datos enviados mediante el método `ConexionBases.consulta`. `Modelo.obtenResultados` ahora guarda en el objeto `RenderRequest` la lista con los resultados de la búsqueda.

- Teniendo ya la lista con los resultados `PrimerPortlet.muestraResultados` redirige a la página `resultados.jsp` la lista con los nombres de los usuarios buscados haciendo uso del método `PrimerPortlet.include` el cual se encarga de decirle al portlet que los resultados los lleve a la página `resultados.jsp` como se muestra en la Figura 2.14.


Figura 2.14.: *Ejemplo de resultados en la búsqueda de usuarios,* los resultados se muestran en la página `resultados.jsp` haciendo uso de las tecnologías *JSTL y EL*

- En esta página al oprimir el botón "*Regresar*" se borran los parámetros guardados en sesión (*nombre y apellido*) utilizando `PrimerPortlet.doDispatch`. Y por último se regresa a la página inicial del portlet la cual es `view.jsp`.

### 2.2.2. Código del primer portlet

Los archivos utilizados fueron modificados como sigue *(se hace uso de las clases* `logging.Log` y `logging.LogFactory`*, útiles para escribir en el archivo de registro de Glassfish los errores o diferentes mensajes para saber como se va ejecutando la aplicación, este archivo se encuentra en la ruta de Glassfish:* `domains/domain1/logs/server.log` ):

Explicación por líneas de código de la clase `PrimerPortlet.java` mostrada en el Cuadro 2.8:

*Líneas 1-20:* Se importan las clases que se utilizarán en esta clase.

*Línea 26:* Heredamos atributos y métodos de la clase abstracta `GenericPortlet` para sobrescribir los métodos que contiene haciendo uso de la etiqueta *@Override*.

*Líneas 36-43:* Inicializamos la conexión a la base de datos de Liferay utilizando la colección de conexiones de la línea 32.

*Líneas 50-59:* Atendemos peticiones tipo *actionUrl* que pueden ser mandadas desde un formulario en este caso se mandan desde el formulario del archivo `view.jsp`. Ponemos en sesión los parámetros mandados desde las entradas del formulario que son *"nombre"* y *"apellido"*. Mandamos al método `doDispatch` *(líneas 68-85)* lo que llegó del parámetro "*accion*" del formulario.

*Líneas 68-85:* Si la petición de redibujado es vacía solamente muestra la página, si dice *"resultados"* se hace uso del método `muestraResultados`. Si el parámetro es *"limpiaBusqueda"* se limpia la sesión y regreso al inicio de la página.

*Líneas 91-93:* Se regresa la fuente de datos inicializada (*Líneas 36-43*).

*Líneas 111-120:* Se redirige el portlet a un archivo con extensión *.jsp* con todo y las peticiones de redibujado que se manden.

*Líneas 125-129:* Redirige a la página principal.

*Líneas 150-158:* Se hace uso de la clase `Modelo` y del método `obtenResultados` enviando los parámetros guardados en sesión.

```
1  package principal;
2
3  import java.io.IOException;
4
5  import javax.naming.InitialContext;
6  import javax.naming.NamingException;
7  import javax.portlet.GenericPortlet;
8  import javax.portlet.PortletException;
9  import javax.portlet.PortletMode;
10 import javax.portlet.PortletRequestDispatcher;
11 import javax.portlet.PortletSession;
12 import javax.portlet.ActionRequest;
13 import javax.portlet.ActionResponse;
14 import javax.portlet.RenderRequest;
15 import javax.portlet.RenderResponse;
16 import javax.portlet.WindowState;
17 import javax.sql.DataSource;
18
19 import org.apache.commons.logging.Log;
20 import org.apache.commons.logging.LogFactory;
21
22 /**
23  * Clase principal que se encarga de administrar el orden en que se
        ejecutan las otras clases que se utilicen en nuestro proyecto.
24  * @author Palacios Trejo Armando ciencias UNAM
25  */
26 public class PrimerPortlet extends GenericPortlet{
27  /** Logger. */
28  private static Log log = LogFactory.getLog(PrimerPortlet.class);
29  /** Fuente de datos para conectar con la BD. */
30  private static DataSource fuenteDatos;
31  /** Nombre del POOL de la BD. */
32  public static final String POOL_CONEXIONES = "jdbc/LiferayPool";
33  /** Campo necesario para el mÃ©todo doView del portlet. */
34  private String viewJSP;
35
36  static {
37   try {
38    fuenteDatos =
39     (DataSource) new InitialContext().lookup(POOL_CONEXIONES);
40   } catch (NamingException e) {
41    throw new ExceptionInInitializerError(e);
42   }
43  }
44
45  /**
```

```
46      * Atiende peticiones de tipo actionUrl.
47      * @param req El objeto request.
48      * @param resp El objeto response.
49      */
50     @Override
51     public synchronized void processAction(ActionRequest req,
52       ActionResponse resp) throws IOException, PortletException {
53      PortletSession sesion = req.getPortletSession(true);
54      sesion.setAttribute("nombre", denullify(req.getParameter("nombre")));
55      sesion.setAttribute("apellido",denullify(req.getParameter("apellido")));
56      resp.setRenderParameter("accion", req.getParameter("accion"));
57                      resp.setPortletMode(PortletMode.VIEW);
58      resp.setWindowState(WindowState.NORMAL);
59     }
60
61     /**
62      * Manda el flujo a donde sea necesario seg'un un parametro del request.
63      * @param req El objeto request.
64      * @param resp El objeto response.
65      * @throws IOException Si hay error de entrada/salida.
66      * @throws PortletException Si hay error en el portlet.
67      */
68     @Override
69     public void doDispatch(RenderRequest req, RenderResponse resp)
70     throws IOException, PortletException {
71      PortletSession sesion = req.getPortletSession(true);
72      String accion = req.getParameter("accion");
73      if (accion == null) {
74       super.doDispatch(req, resp);
75       return;
76      } else if (accion.equals("resultados")) {
77       muestraResultados(req, resp);
78       return;
79      } else if (accion.equals("limpiaBusqueda")) {
80       sesion.removeAttribute("nombre");
81       sesion.removeAttribute("apellido");
82       include(viewJSP, req, resp);
83       return;
84      }
85     }
86
87     /**
88      * Regresa la fuente de datos para conexiones con la BD.
89      * @return La fuente de datos para conexiones con la BD.
90      */
91     public static DataSource getFuenteDatos() {
92      return fuenteDatos;
93     }
94
95     /**
96      * Inicializa variables necesarias para el portlet desde el portlet.xml.
97      */
98     @Override
99     public void init() throws PortletException {
```

```
100     this.viewJSP = getInitParameter("view-jsp");
101   }
102
103   /**
104    * Auxiliar para redirigir el portlet.
105    * @param ruta La ruta para redirigir el portlet.
106    * @param req El objeto request.
107    * @param resp El objeto response.
108    * @throws IOException Si hay error de entrada/salida.
109    * @throws PortletException Si hay error en el portlet.
110    */
111   private void include(String ruta, RenderRequest req, RenderResponse resp)
112   throws IOException, PortletException {
113    PortletRequestDispatcher portletRequestDispatcher = getPortletContext()
114    .getRequestDispatcher(ruta);
115    if (portletRequestDispatcher == null) {
116     log.error(ruta + " es una ruta incorrecta");
117    } else {
118     portletRequestDispatcher.include(req, resp);
119    }
120   }
121
122   /**
123    * Atiende una demanda de view del portlet.
124    */
125   @Override
126   public void doView(RenderRequest req, RenderResponse resp)
127   throws IOException, PortletException {
128    include(viewJSP, req, resp);
129   }
130
131   /**
132    * Regresa "" si la entrada es null, si no regresa el mismo String.
133    * @param string El String a revisar.
134    * @return Un String diferente de null.
135    */
136   public static String denullify(String string) {
137    if (string == null) {
138     string = "";
139    }
140    return string;
141   }
142
143   /**
144    * Muestra los resultados de la consulta.
145    * @param req El objeto request.
146    * @param resp El objeto response.
147    * @throws IOException Si hay error de entrada/salida.
148    * @throws PortletException Si hay error en el portlet.
149    */
150   private void muestraResultados(RenderRequest req, RenderResponse resp)
151   throws IOException, PortletException {
152    PortletSession sesion = req.getPortletSession(true);
153    include("/resultados.jsp", Modelo.obtenResultados(
```

```
154       req,
155       sesion.getAttribute("nombre").toString(),
156       sesion.getAttribute("apellido").toString()
157     ), resp);
158   }
159
160 }
```

Cuadro 2.8.: *Código del archivo PrimerPortlet.java*

---

Explicación por líneas de código de la clase `Modelo.java` mostrada en el Cuadro 2.9:
*Líneas 27-50:* Se hace una consulta a la base de datos de Liferay utilizando la clase `ConexionBases` con los parámetros *"nombre"* y *"apellido"* para obtener los usuarios con los nombres y apellidos especificados. Los resultados los guardamos en una lista de arreglos.

---

```
1  package principal;
2
3  import java.sql.ResultSet;
4  import java.sql.SQLException;
5  import java.util.ArrayList;
6
7  import javax.portlet.RenderRequest;
8
9  import org.apache.commons.logging.Log;
10 import org.apache.commons.logging.LogFactory;
11
12 /**
13  * Clase que se encarga del comportamiento del portlet.
14  * @author Palacios Trejo Armando ciencias UNAM
15  */
16 public class Modelo {
17  /** Logger. */
18  private static Log log = LogFactory.getLog(Modelo.class);
19
20  /**
21   * Obtiene los resultados de la consulta.
22   * @param req De los datos de cada consulta de cada pesona.
23   * @param nombre Nombre de la persona a consultar.
24   * @param apellido Apellido de la persona a consultar.
25   * @return RenderRequest Los datos de la consulta.
26   */
27  public static RenderRequest obtenResultados(RenderRequest req,
28    String nombre, String apellido) {
29   String query = "SELECT contactid, firstname, lastname from contact_ " +
30   "WHERE firstname ILIKE '%" + nombre.trim() + "%' " +
31   "AND lastname ILIKE '%" + apellido.trim() + "%'";
32   ResultSet rs = ConexionBases.consulta(query);
33   ArrayList<String[]> lista = new ArrayList<String[]>();
```

```
34    if (rs == null) { return req; }
35    try {
36     while (rs.next()) {
37      String[] columna = new String[3];
38      columna[0] = Integer.toString(rs.getInt("contactid"));
39      columna[1] = rs.getString("firstname");
40      columna[2] = rs.getString("lastname");
41      lista.add(columna);
42     }
43     req.setAttribute("datosTabla", lista);
44     rs.close();
45    } catch (SQLException e) {
46     log.error(e.getCause());
47     e.printStackTrace();
48    }
49    return req;
50   }
51
52  }
```

Cuadro 2.9.: *Código del archivo Modelo.java.*

---

Explicación por líneas de código de la clase `ConexionBases.java` la cual utiliza clases y métodos para el manejo de una colección de conexiones abiertas a una base de datos de manera que puedan ser reutilizadas al realizar multiples consultas o actualizaciones, la clase se muestra en el Cuadro 2.10:

*Líneas 28-68:* Mediante una cadena de tipo *"String"* mostrando el query se hace una consulta a la base de datos obteniendo la fuente de datos creada en la clase `PrimerPortlet`., si no hay conexión escribe un mensaje de error en el archivo de registro de *Glassfish*, y manda una *excepción* para controlar el error en tiempo de ejecución.

---

```
1   package principal;
2
3   import java.sql.Connection;
4   import java.sql.ResultSet;
5   import java.sql.SQLException;
6   import java.sql.Statement;
7
8   import org.apache.commons.logging.Log;
9   import org.apache.commons.logging.LogFactory;
10
11  import com.sun.rowset.*;
12
13  /**
14   * Clase con m'etodos 'utiles para el manejo de una colecci'on de
         conexiones
15   * abiertas a una base de datos de manera que puedan ser reutilizadas al
```

```
16  * realizar m'ultiples consultas o actualizaciones.
17  * @author Palacios Trejo Armando ciencias UNAM
18  */
19 public class ConexionBases {
20  /** Logger. */
21  private static Log log = LogFactory.getLog(ConexionBases.class);
22
23  /**
24   * Realiza una consulta en la base de datos.
25   * @param query La consulta a realizar en la BD.
26   * @return Los resultados de la consulta en forma de Resultset.
27   */
28  @SuppressWarnings("restriction")
29  public static CachedRowSetImpl consulta(String query) {
30   Connection conexion = null;
31   Statement statement = null;
32   ResultSet resultSet = null;
33   CachedRowSetImpl cachedRowSet = null;
34   try {
35    conexion = PrimerPortlet.getFuenteDatos().getConnection();
36    statement = conexion.createStatement();
37    resultSet = statement.executeQuery(query);
38    /*
39     *  Crea un CachedRowSet para poder usar los datos del ResultSet
40     *  despues de cerrar la conexion a la BD.
41     */
42    cachedRowSet = new CachedRowSetImpl();
43    cachedRowSet.populate(resultSet);
44
45   } catch (SQLException e) {
46    log.error(e.getMessage());
47    e.printStackTrace();
48   } finally {
49    try {
50     if (resultSet != null) {
51      resultSet.close();
52      resultSet = null;
53     }
54     if (statement != null) {
55      statement.close();
56      statement = null;
57     }
58     if (conexion != null) {
59      conexion.close();
60      conexion = null;
61     }
62    } catch (SQLException e) {
63     log.error(e.getMessage());
64    }
65   }
66   return cachedRowSet;
67  }
68 }
```

Cuadro 2.10.: *Código del archivo ConexionBases.java.*

Explicación por líneas de código del archivo `include.jsp` que cuyo contenido será utilizado en todos los archivos con extensión .jsp , se muestra en el Cuadro 2.11:

*Línea 1:* Incluimos un formato de etiquetado para enviar parámetros a las bibliotecas de portlets por ejemplo la etiquetas que comienzen con `<portlet,` como se puede ver el archivo `view.jsp`.

*Línea 2:* Decimos que las etiquetas de la tecnología *EL* no sea ignorada.

*Línea 3:* Usamos la tecnología *JSTL*.

*Línea 4:* Usamos la tecnología de etiquetado de portlets para definir objetos.

*Líneas 5-6:* Hacemos uso del archivo de lenguaje *JavaScript* llamado `main.js`.

*Líneas 7-8:* Hacemos uso de una hoja de estilo en cascada para la presentación de la vista de nuestro portlet.

```
1 <%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
2 <%@ page isELIgnored="false" %>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
4 <portlet:defineObjects />
5 <script type="text/javascript" src="${renderRequest.contextPath}/js/main.js
      ">
6 </script>
7 <link rel="stylesheet" type="text/css"
8 href="${renderRequest.contextPath}/css/main.css" />
```

Cuadro 2.11.: *Código del archivo include.jsp*

Explicación por líneas de código del archivo `view.jsp` que se muestra en el Cuadro 2.12:

*Línea 2*: Incluyo el archivo `include.jsp`.

*Líneas 5-26*: Se crea un formulario con las etiquetas de los portlets para enviar un parámetro de tipo `actionUrl` de nombre "*accion*" cuyo valor será "*resultados*". También este formulario enviará el nombre y apellido que se quiera encontrar al oprimir el botón "*Buscar*".

```
1 <%/**@author: Palacios Trejo Armando ciencias UNAM */%>
2 <%@ include file="include.jsp"%>
3
4 Filtro de usuarios por nombre y apellido.<br>
5 <form method="post"
6 action='<portlet:actionURL>
7 <portlet:param name="accion" value="resultados"/>
8 </portlet:actionURL>'>
```

```
 <table style="text-align: left; width: 30%;" border="0">
  <tbody>
   <tr>
    <td>Nombre</td>
    <td>Apellido</td>
   </tr>
   <tr>
    <td>
     <input value="" size="20" name="nombre" type="text">
    </td>
    <td>
     <input value="" size="20" name="apellido" type="text">
    </td>
   </tr>
  </tbody>
 </table>
 <br><input type="submit" value="Buscar">
</form>
```

Cuadro 2.12.: *Código del archivo view.jsp.*

Explicación por líneas de código del archivo `resultados.jsp` que se muestra en la Cuadro 2.13:

*Líneas 4-25*: Se crea un formulario para enviar el parámetro *"accion"* de tipo `actionUrl` cuyo valor será *"limpiaBusqueda"*, adentro del formulario se muestran los resultados de los usuarios buscados haciendo uso del lenguaje de etiquetado *EL* y *JSTL*.

```
<%/**@author: Palacios Trejo Armando ciencias UNAM */%>
<%@ include file="include.jsp"%>

<form method="post"
action='<portlet:actionURL>
<portlet:param name="accion" value="limpiaBusqueda"/>
</portlet:actionURL>'>
 <table style="text-align: left; width: 30%;" border="0">
  <tbody>
      <tr>
       <td><b>Identificador</b></td>
       <td><b>Nombre</b></td>
       <td><b>Apellido</b></td>
      </tr>
      <c:forEach var="datosTabla" items="${requestScope.datosTabla}">
          <tr>
              <td>${datosTabla[0]}</td>
              <td>${datosTabla[1]}</td>
              <td>${datosTabla[2]}</td>
          </tr>
      </c:forEach>
```

```
22      </tbody>
23     </table>
24     <br><input type="submit" value="Regresar">
25    </form>
```

Cuadro 2.13.: *Código del archivo resultados.jsp.*

### 2.2.3. Remover portlets

A veces es necesario remover portlets para que no se carguen en el portal al iniciarlo. También pueden existir portlets en Liferay los cuales nunca van a ser utilizados y es mejor que sean removidos para evitar su uso y así hacer que el portal se inicie más rápido al arrancar el servidor al no inicializarlos.

#### 2.2.3.1. Utilizando Tomcat

- Remover el portlet en la interfaz con el ícono de *cerrar (X)*. Si se sigue con el siguiente paso antes de esto, marcará errores al no encontrar el portlet ya que se mostrará como si aún estuviera activo.
- Entrar al directorio `tomcat-6.0.26/webapps/` y eliminar el archivo con extensión *.war* del portlet y posteriormente el directorio del porlet en esa misma carpeta.

#### 2.2.3.2. Utilizando Glassfish

- Remover el portlet en la interfaz con el ícono de *cerrar (X)*, de lo contrario el servidor después marcará errores al no encontrar este portlet si se sigue con los siguientes pasos.
- Entrar a la consola de administración de *Glassfish (*`http://localhost:4848/`*)* desde el navegador.
- Entrar a `Common Tasks - Applications`.
- Deshabilitar el portlet.

De manera alternativa se puede hacer de la siguiente forma.

- Remover el portlet en la interfaz con el ícono de *cerrar (X)*, de lo contrario el servidor después marcará errores al no encontrar el portlet si se sigue con los siguientes pasos.
- Ir al directorio `glassfish-3.0.1/domains/domain1/applications/` y borrar el directorio del portlet por ejemplo si queremos borrar el portlet chat, borrar la carpeta y su contenido `chat-portlet/`.

- Ir al directorio `glassfish-3.0.1/domains/domain1/autodeploy/` y borrar los archivos con extensión *.war* de los portlets por ejemplo en el caso del portlet de chat borrar *chat-portlet.war* y *chat-portlet.war_ deployed*.

# 3. Administración de Liferay

Liferay permite a los administradores del portal gestionar usuarios, grupos y roles bajo una interfaz muy intuitiva. Esto servirá para que ciertas personas puedan tener acceso a ciertos portlets o páginas del portal.

*Un permiso* en Liferay es definido como una acción actuando sobre un recurso [39].

La tabla del Cuadro 3.1 da algunos ejemplos de permisos.

| **Acción** | **Recurso** | **Explicación** |
|---|---|---|
| Visualizar *(View)* | Asuntos de mensaje / Alcance empresarial e individual. | El usuario tiene permiso para mirar cualquier asunto en el portal |
| Actualizar *(Update)* | Asuntos de mensaje / Alcance de comunidad de "Desarrolladores" | El usuario tiene permiso a solamente actualizar un asunto contenido en un mensaje en la comunidad de desarrollo |
| Borrar *(Delete)* | Asuntos de mensaje / "Asuntos de Java" Alcance individual | El usuario tiene permiso de solo borrar temas de asuntos Java, los cuales pasan a estar en un mensaje en la comunidad de desarrollo. |

Cuadro 3.1.: *Tabla de permisos en Liferay.*

En general los permisos son aditivos. Por lo tanto un usuario podría recibir los tres permisos mencionados en el Cuadro 3.1, incluso pensando que ellos están en un diferente alcance. Los permisos siempre son verificados en el siguiente orden:

- *Individual.* Para un solo individuo o usuario.
- *Comunidad.* Para todas las diferentes comunidades en el portal.
- *Empresarial.* Para la empresa en general.

Por lo tanto, tan pronto como el sistema encuentra el permiso de visualizar de un alcance individual, este termina de verificar y da al usuario el permiso para visualizar.

## 3.1. Usuarios, grupos y roles

*Roles:* un *rol* es una colección de permisos. Un rol sirve a un propósito, por ejemplo el poder actuar sobre los mensajes de la empresa. A un rol se le pueden asignar permisos para visualizar, actualizar y borrar ciertos recursos que tienen alcance empresarial donde el usuario tiene permiso para mirar asuntos en el portal. Un usuario asignado al rol "Asuntos de mensaje de Administrador" podría ser capaz de visualizar, actualizar y borrar cualquier recurso o mensaje de la empresa. Los roles pueden ser asignados a un usuario, comunidad, organización o ubicación. Si un rol es asignado a una comunidad, organización o ubicación, entonces los usuarios que son miembros de esa entidad, reciben ese rol [39].

*Usuarios:* un usuario es un individuo que realiza ciertas tareas utilizando un portal dependiendo de los permisos y roles se le hayan asignado. Antes de entrar al portal, el usuario es considerado un *invitado (guest),* los invitados tienen su propia colección de permisos para actuar en el portal, pero incluso estos permisos pueden ser modificados por los administradores del portal. Después de entrar al portal, un usuario es considerado un *usuario registrado.* Un usuario registrado puede recibir los siguientes permisos: [39]

- Un permiso es asignado al usuario al cual pertenece.
- Un permiso es asignado a una comunidad a la cual pertenece.
- Un permiso es asignado a una organización a la cual pertenece.
- Un permiso es asignado a una ubicación a la cual pertenece.
- Un permiso pertenece a un rol que es directamente asignado al usuario.
- Un permiso pertenece a un rol que es asignado a una comunidad a la cual pertenece.
- Un permiso pertenece a un rol que es asignado a una organización a la cual pertenece.
- Un permiso pertenece a un rol que es asignado a una ubicación a la cual pertenece.

*Organizaciones y ubicaciones:* las organizaciones y ubicaciones representan una jerarquía corporativa. Una organización representa una corporación madre, un ejemplo podría ser Liferay USA. Una ubicación representa una corporación hija de una organización, muchas veces se distinguen por su ubicación geográfica. Las organizaciones pueden tener más de una ubicación. Un usuario solamente puede pertenecer a una sola organización y ubicación. Por defecto las ubicaciones heredan permisos de su organización padre. Por ejemplo si el rol "*Asuntos de mensaje de Administrador*" es asignado a la organización de Liferay USA entonces todos los miembros de Liferay Chicago, Liferay San Francisco,

etc, heredarán los permisos asociados a este rol [39].

*Comunidades:* una comunidad es un grupo de usuarios asociados por intereses o habilidades en común. Por ejemplo *"Amantes de animales"* es una comunidad que consiste de usuarios que aman a los animales. Los usuarios pueden pertenecer a más de una comunidad de usuarios. Tan lejos como los permisos son colocados, las comunidades no son especificadas a cualquier organización o ubicación. Ambos, roles y permisos individuales pueden ser asignados a comunidades [39].

*Grupos de usuario:* un grupo de usuarios es un conjunto de usuarios. Al contrario que organizaciones, ubicaciones y comunidades, los grupos de usuarios no tienen un contexto asociado a ellos. Son puramente una conveniencia que ayuda a los administradores a asignar permisos y roles a un grupo de usuarios en lugar de tener solo usuarios individuales o asignar un grupo de usuarios a una comunidad. Un usuario puede pertenecer a más de un grupo de usuarios. Ambos, roles y permisos individuales pueden ser asignados a grupos de usuario; y cada usuario que pertenece a ese grupo de usuario, podrá recibir el rol o permiso [39].

La Figura 3.1 describe las formas de organización de Liferay, por ejemplo al ver las tablas `User` y `Group` lo que los liga quiere decir que cero o más de cero usuarios pueden pertenecer a un grupo. Al ver las tablas `Group` y `Role` lo que los liga quiere decir que un Grupo puede tener cero o más de cero roles. Al ver las tablas `Community` y `Group` quiere decir que las comunidades pueden ser vistas como grupos [39].

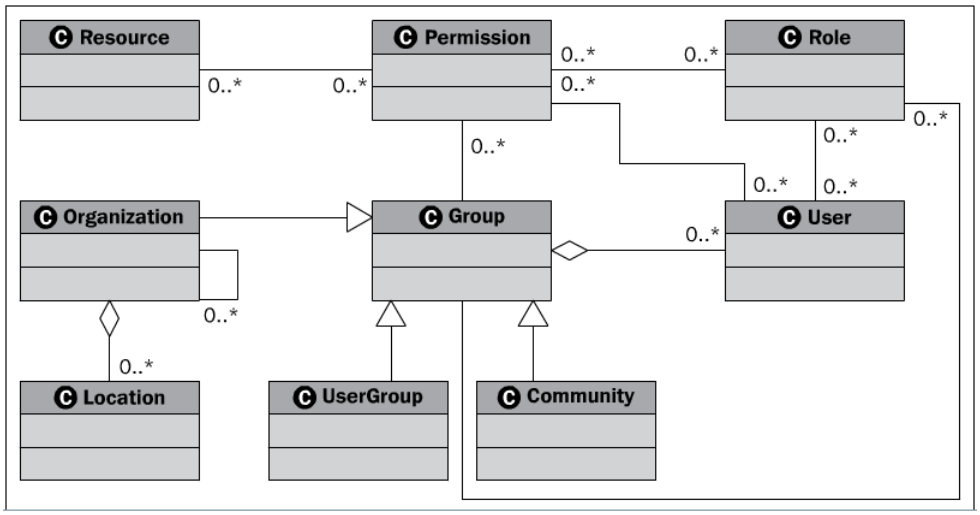


Figura 3.1.: *Esquema de las formas de organización de* Liferay, comunidades, usuarios, grupos y roles.

## 3.2. Ejemplo para la asignación de permisos para acceder a las páginas

A continuación se dará un ejemplo con usuarios, grupos de usuario y roles. Los usuarios contenidos en un grupo pueden pertenecer a otro grupo del cual este grupo puede pertenecer a un rol diferente al grupo anterior como se muestra en la Figura 3.2.

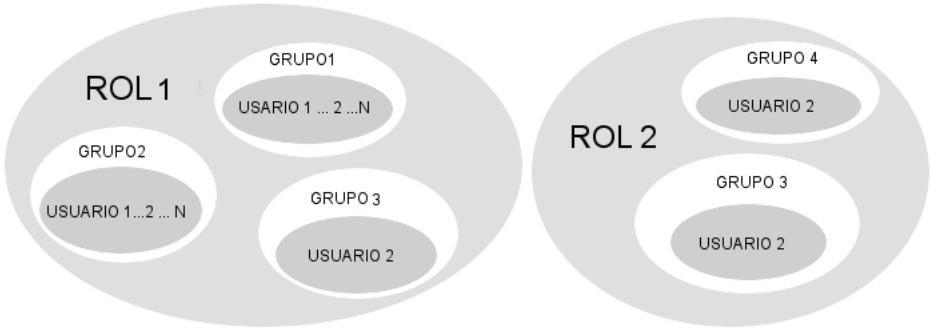


Figura 3.2.: *Asignación de roles a grupos de usuarios y usuarios a grupos.*

A continuación asignaremos permisos a ciertos usuarios, con el objetivo de que únicamente puedan acceder a ciertas páginas del portlet.

Como se puede observar el mapa de sitio como ejemplo creado en `Panel de Control - miportal.com - Páginas`, es el siguiente, ver Figura 3.3:


Figura 3.3.: *Mapa del sitio.*

Al acceder al portal como administrador podemos observar que tenemos acceso a todas las páginas, ver la Figura 3.4:

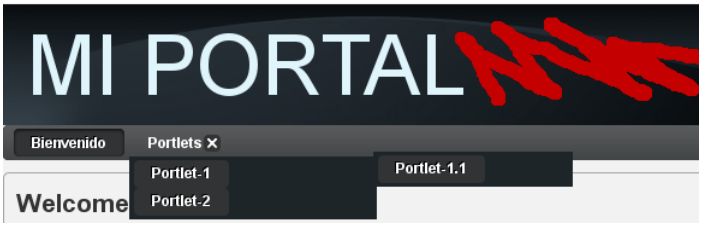


Figura 3.4.: *Se accede al portal como administrador,* se tiene acceso a todas las páginas.

Los usuarios que se han creado como ejemplo en Panel de Control - Portal - Usuarios, son los siguientes, ver la Figura 3.5:


Figura 3.5.: *Los usuarios creados como ejemplo.*

Ahora vamos a `Panel de Control - Portal - Roles` y creamos un nuevo rol llamado *"Acceso Portlet 1"*, ver la Figura 3.6.

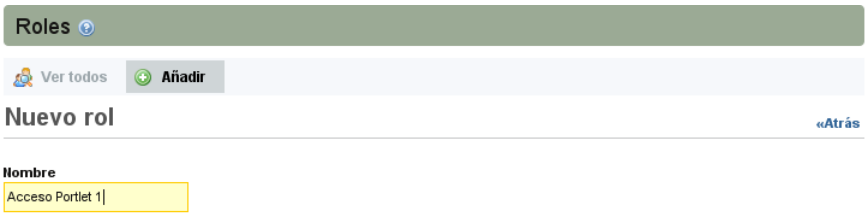


Figura 3.6.: *Creación del rol "Acceso Portlet 1".*

Ahora vamos a `Panel de Control - Portal - Grupos de usuarios` y creamos un nuevo grupo de usuarios llamado *"Grupo Acceso Portlet 1"*, ver la Figura 3.7.

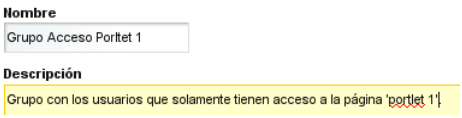


Figura 3.7.: *Creación de un nuevo grupo de usuarios llamado "Grupo Acceso Portlet 1".*

A continuación le asignamos los miembros, ver la Figura 3.8.

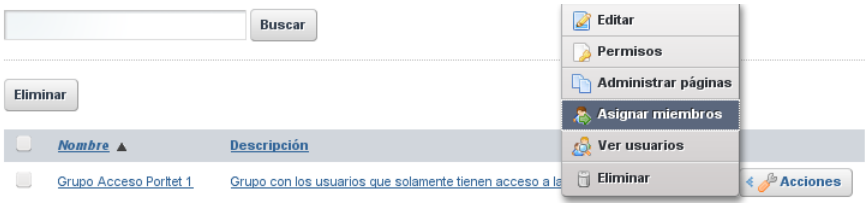


Figura 3.8.: *Asignación de miembros al grupo "Grupo Acceso Portlet 1".*

Ahora vamos a la sección disponible y seleccionamos los miembros que deseamos añadir al grupo, en este caso se han seleccionado todos los llamados *"Armando"*. Oprimimos *"Modificar asignaciones"*.

Regresamos a `Panel de Control - Portal - Roles`, seleccionamos del rol *"Acceso Portlet 1"* `Acciones - Asignar miembros`. A continuación, `Asignar miembros - Grupos de usuario - Disponible` y seleccionamos *"Grupo Acceso Portlet 1"*, por último oprimimos *"Modificar asignaciones"*, con el fin de asignar el grupo *"Grupo Acceso Portlet 1"* al rol "*Acceso Portlet 1*".

Vamos a `Panel de Control - Páginas`, seleccionamos del árbol de páginas la llamada *"Portlets"*, a continuación página, y oprimimos el botón permisos. Deseleccionamos las casillas señaladas en *Guest,* y seleccionamos la casilla *"Ver"* de *"Acceso Portlet 1"* la cual es el rol que creamos, con esto la página Portlets podrá tener el permiso de verse por los grupos y usuarios que pertenezcan al rol *"Acceso Portlet 1"*, ver la Figura 3.9.


Figura 3.9.: *Se da permiso a la página "Portlets".*

Hacemos lo mismo con la página "Portlet-1". Con las páginas "Portlet-1.1" y "Portlet-2" deseleccionamos todo lo de Acceso Portlet 1 y Guest como se muestra en la Figura 3.10.

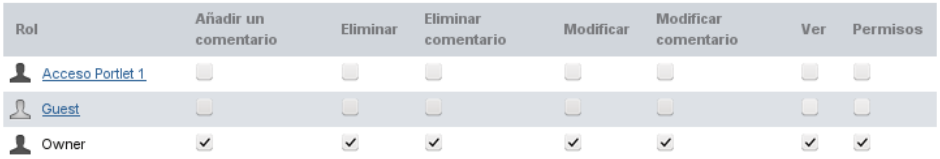


Figura 3.10.: *Se asigna los permisos a las subpáginas de la página "Portlets".*

Ahora al acceder como un usuario perteneciente al grupo *"Grupo Acceso Portlet 1"* se podrá observar que las páginas *"Portlet-1.1"* y *"Portlet-2"* se han ocultado porque se quitó el permiso de ver a estas páginas al rol "Acceso Portlet 1" cuyo grupo es *"Grupo Acceso Portlet 1"*, ver la Figura 3.11:

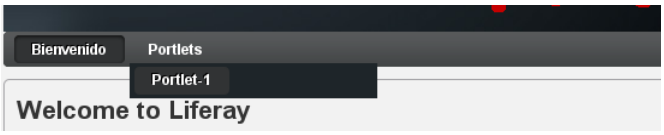


Figura 3.11.: *Los usuarios pertenecientes al grupo "Grupo Acceso Portlet 1" solo pueden visualizar la páginas "Portlets" y "Portlet-1".*

# 4. Base de datos de Liferay

Es necesario tener una base de datos para que contenga los datos que requieren guardarse en el portal para su próximo o futuro uso. En la base de datos de Liferay se guardan la relación de las páginas con los usuarios y la asignación de permisos a los grupos, usuarios, roles, etc. Además se guardan datos que necesitan ciertos portlets para su uso. La base de datos de Liferay es muy grande.

Se puede comprobar que las tablas de la base de datos no están ***referenciadas***[1]; a continuación se va a mostrar que aún así las tablas se pueden ***relacionar***[2], ya que están ***indexadas***[3] correctamente, y a su vez las ***llaves primarias, candidatas y ajenas***[4], están diseñadas de forma que es entendible la lógica de la base de datos osea tienen *Integridad Referencial* [48]. El diagrama *Entidad Relación* de las principales tablas de la base de datos de Liferay se muestran en la Figura 4.1:

[1] "*La integridad de referencia* se refiere a los valores tomados por un atributo en una relación con respecto a los valores tomados por otro atributo definido sobre el mismo dominio, llave primaria en otra relación. Sea el atributo A de R1 definido sobre el dominio primario D. Cada valor K de A en R1 debe ser un valor nulo o el valor de una llave primaria definida sobre D en una relación R2" [48].

[2] "*Una relación* se define como el subconjunto de un producto cartesiano de n conjuntos, llamados dominios: R $\underset{=}{C}$ D1*D2*D3*...*Dn con n el grado de R.

Un conjunto de relaciones semánticamente ligadas por sus dominios: R(A1:D1, A2:D2, ... An:Dn)" [48].

[3] "*El índice de una base de datos* es una estructura de datos que mejora la velocidad de las operaciones, permitiendo un rápido acceso a los registros de una tabla en una base de datos. Los índices pueden ser creados usando una o más columnas. En una base de datos relacional un índice es una copia de parte de una tabla" [48].

[4] "Ya que en una relación no hay tuplas repetidas, éstas se pueden distinguir unas de otras, es decir, se pueden identificar de modo único. La forma de identificarlas es mediante los valores de sus atributos.

La llave primaria de un relación es aquella llave candidata que se escoge para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una llave candidata y, por lo tanto, la relación siempre tiene llave primaria. En el peor caso, la llave primaria estará formada por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función. Las claves candidatas que no son escogidas como llave primaria son denominadas claves alternativas. *Una llave foránea* es un atributo o un conjunto de atributos de una relación cuyos valores coinciden con los valores de la llave primaria de alguna otra relación (puede ser la misma). Las claves foráneas representan relaciones entre datos" [48].

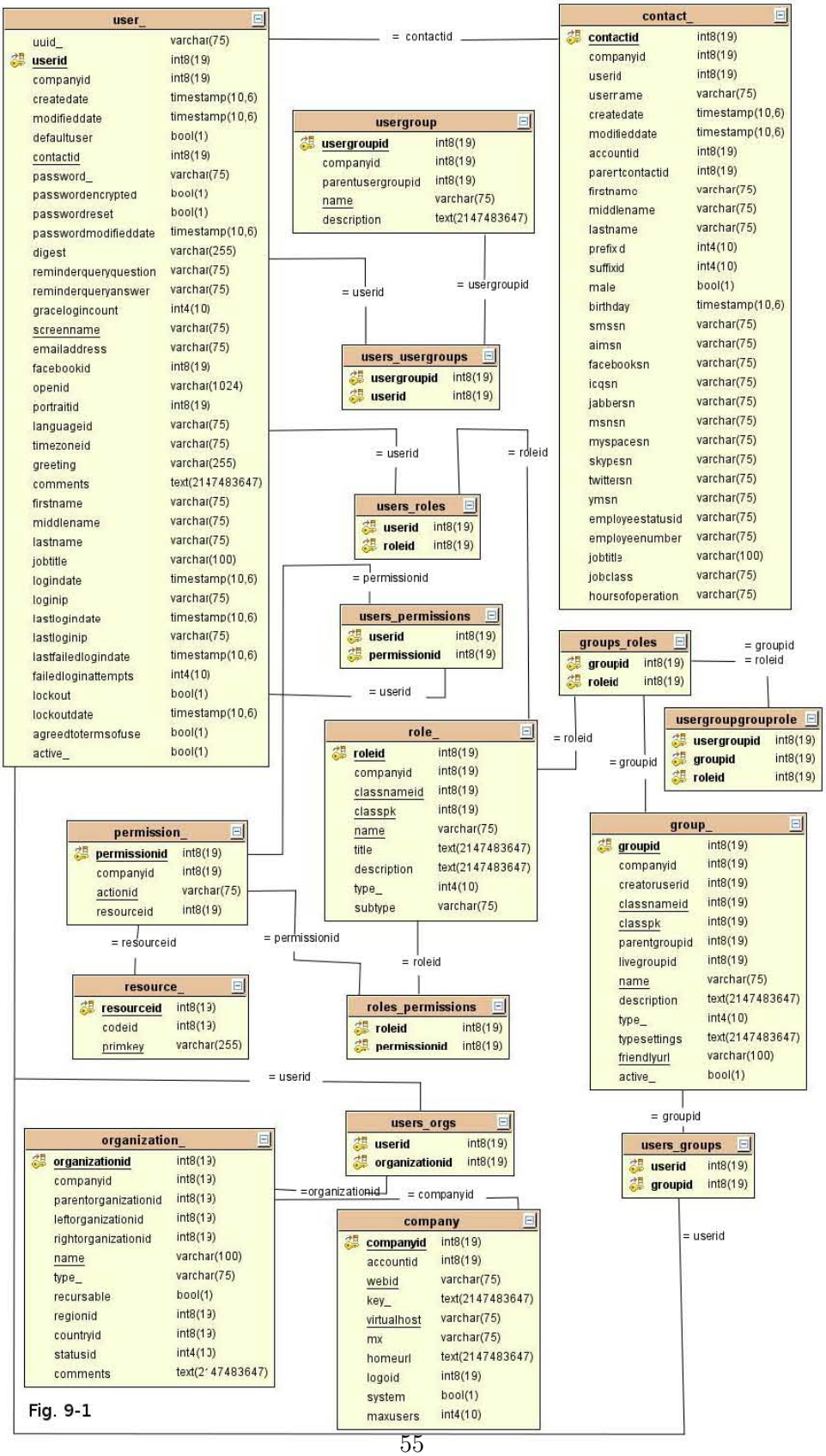




Figura 4.1.: *Diagrama Entidad Relación de las principales tablas de la base de datos de Liferay.*

El diagrama *ER (Entidad - Relación)* de la Figura 4.1 muestra las tablas que se consideran mas importantes de la base de datos de Liferay para la explicación del trabajo, como se puede observar, las tablas se pueden relacionar entre sí, al observar los índices de cada una, y así poder hacer consultas eficientes. A continuación se muestra una explicación introductoria de ellas.

- `company`*:* muestra los datos de la empresa, en este caso *`miportal`*, se almacenan las diferentes instancias del portal, por ejemplo en la tabla, la columna *"mx"* debe contener el nombre de la compañía, en este caso *miportal.com*.

- `contact_`: guarda algunos datos de los usuarios como nombre, apellido y fecha de nacimiento, etc.

- `group_`*:* es el nombre para lo que ahora se llaman comunidades, los usuarios pueden pertenecer a mas de una comunidad, y heredar permisos y roles para ellos. Cabe mencionar que en esta tabla hay una columna *classNameId* y *class*Pk, si *className y classPk* se encuentran en blanco, entonces el registro es una comunidad, si *className* es *`com.liferay.portal.model.User`* el registro representa un usuario privado de la comunidad *(solo aplica a usuarios privilegiados)*, en cambio cuando *className* es `com.liferay.portal.model.Organization`, entonces el registro representa una organización o ubicación. Si *className* es `com.liferay.portal.model.UserGroup` entonces el registro representa un grupo de usuario o *UserGroup*. La razón de ello es tener una entrada en esta tabla para cada entidad en el sistema que representa un conjunto de usuarios. Esto simplifica relaciones de otras entidades como *(permisos y roles)* con estos conjuntos de usuarios.

- `groups_roles`*:* se refiere a la relación entre `group`, `role_` *y* `usergroupgrouprole`*,* con el fin de saber los roles que pertenecen a ciertos grupos. Un grupo puede referirse a una comunidad, organización/ubicación. Los usuarios que pertenecen a estos grupos heredan los permisos de los roles correspondientes.

- `organization`*:* los usuarios pueden pertenecer a una sola organización o ubicación. Se usa la misma tabla para representar a ambos. Básicamente, si la columna *parentOrganizationId* tiene un -1, entonces es una organización, cualquier otro número es una ubicación. Los usuarios heredan permisos/roles de las organizaciones o ubicaciones a las que pertenecen.

- `permission_`*:* un permiso es una acción actuando sobre un recurso, por lo tanto esta tabla tiene una columna *actionId* y *resourceId*. Como se espera la columna *resourceId* hace referencia a la columna *resource_ id* de la tabla `resource_`. Sin embargo la columna *actionId* no tiene correspondencia a la tabla `Action_`. Todas los identificadores actionId son definidas en las clases `ActionKeys`

- `resource_`*:* un recurso es la representación de un objeto en el portal, cualquier cosa a la que se quiera dar un permiso, puede ser un portlet, una comunidad, un usuario, etc. Para esto se relaciona con la tabla `resourcecode_` *y* `permission_`.

- `role_`*:* esta es la tabla donde un rol es definido, la columna importante es name donde se guarda el nombre ya que los roles deben tener nombres únicos.
- `roles_permissions`*:* es una tabla relacional que vincula un permiso a un rol, sin estos vínculos, los roles pueden ser inútiles, ya que podrían estar vacíos.
- `user_`*:* guarda datos como contraseñas, nombre de los usuarios, identificadores de usuarios y se relaciona con tablas importantes como `contact_`.
- `user_usergroups`*:* es la tabla relacional que vincula un usuario a un grupo de usuario *(User a UserGroup)*.
- `usergroup`*:* muestra el nombre y la descripción de un grupo. Se puede relacionar con `user_` y de cierta forma con `role_` mediante otras tablas. Por ejemplo para que se almacene la descripción de un grupo y saber cual es la función de ese grupo.
- `usergroupgrouprole`*:* se relaciona con `groups_roles`, para saber los roles que tienen los grupos.
- `users_roles`*:* se refiere a la relación entre `user_` y `role_` ,para saber que roles tienen los usuarios.
- `users_groups`*:* se refiere a la relación entre `group_` *y* `user_`, para conocer los grupos y sus usuarios.
- `users_orgs`*:* se refiere a la relación entre `organization_` *y* `user_`, para conocer la organización a la que pertenece un usuario.
- `users_permissions`*:* hace la relación entre `permission_` y `user_` . para que al relacionar `permission_` y `roles_permissions` se pueda saber que permisos tiene un usuario ya que hereda los permisos que tiene el rol de este.

Ejemplo de consulta SQL a la base de datos de Liferay para obtener identificador de usuario, identificador de grupo, nombre, apellido paterno, descripción de todos los usuarios que pertenecen a un grupo: `SELECT ug.userid, ug.usergroupid, u.firstname, u.lastname, g.name, g.description FROM user_ u JOIN users_usergroups ug ON (u.userid = ug.userid) JOIN usergroup g ON (ug.usergroupid = g.usergroupid)`.

El resultado del ejemplo se muestra en el Cuadro 4.1:

```
 userid | usergroupid | firstname | lastname | name | description
-------+-------------+-----------+-----------+-----+------------
 11728 | 11716 | Armando | Palacios | Grupo Departamento1 | Grupo perteneciente
al rol Departamento1
 11737 | 11720 | Armando | Lopez | Grupo Departamento2 | Grupo perteneciente
al Rol Departamento2
 11746 | 11724 | Armando | Archundia | Grupo Departamento3 | Grupo perteneciente
al rol Departamento3
```

Cuadro 4.1.: *Ejemplo de tabla de consulta a la base de datos de Liferay,* se obtienen los usuarios y los grupos a los que pertenecen.

---

En este capítulo se ha mostrado una introducción de como se relacionan las diferentes tablas de la base de datos de Liferay para así poder crear portlets que hagan consultas a la base y obtener diferentes datos como por ejemplo los grupos a los que pertenece un usuario, así como las páginas del portal a las que tiene acceso un grupo.

# 5. Creación de un portlet avanzado utilizando Hibernate y JQuery

En este capítulo se utilizarán las tecnologías *Hibernate* y *JQuery* con *AJAX* en los portlets, herramientas que se describirán en las siguientes subsecciones con el objetivo de desarrollar un proyecto que las contenga. En general estas herramientas benefician a los desarrolladores de software ya que ayudan a recortar tiempo de desarrollo de las aplicaciones Web en este caso portlets, y benefician a los usuarios ya que ayudan a mejorar y conseguir un mejor resultado de cara al usuario final [59] .

## 5.1. Introducción a Hibernate

Hibernate es una herramienta de *mapeo objeto-relacional* más conocido como *ORM,* la cual es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional, esto posibilita el uso de las características propias de la orientación a objetos *(básicamente herencia y polimorfismo)* para la plataforma Java *(y disponible también para .Net con el nombre de NHibernate)* la cual facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos *(XML)* o anotaciones en los *beans* de las entidades que permiten establecer estas relaciones [59].

Hibernate es *software libre*, distribuido bajo los términos de la licencia ***GNU LGPL***[1], es decir, se puede cambiar el código de Hibernate o escribir una envoltura para un cierto código de Hibernate y compilarla; pero, el código fuente de esa compilación *(sus modificaciones incluyendo/las adiciones/envoltura)* se debe autorizar bajo la licencia *LGPL* o *GLP*. Cualquier otro código en la aplicación se puede autorizar como se desee. [59].

### 5.1.1. Características

Hibernate busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación, el usado en la memoria de la computadora *(orientado a objetos)* y el usado en las bases de datos *(modelo relacional)*. Para lograr esto se le permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los

[1] *"La licencia GNU LGPL* permite que los usuarios y desarrolladores puedan modificar el código y se recompile." Para mas información de esta licencia consultar la página `http://www.gnu.org/`

datos de la base de datos operando sobre los objetos, con todas las características de la *Programación Orientada a Objetos*, como por ejemplo herencia y polimorfismo. También Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por *SQL*. A su vez genera las sentencias *SQL* y libera al desarrollador del manejo de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución ya que no se hacen directamente las sentencias SQL del manejador de la base de datos con el que se trabaje [59].

Hibernate está diseñado para ser flexible en cuanto al esquema utilizado en las tablas, y así para poder adaptarse en su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible [59].

Hibernate ofrece también un lenguaje de consulta de datos llamado *HQL (Hibernate Query Language)*, al mismo tiempo que una API para construir las consultas mediante funciones *(conocida como "*criteria*")* [59].

Hibernate para Java puede ser utilizado en aplicaciones Java independientes o en aplicaciones *Java EE*, mediante el componente *Hibernate Annotations* que implementa el estándar ***JPA*** [2], que es parte de esta plataforma.

## 5.2. Introducción a JQuery

*JQuery* es un marco de trabajo para el lenguaje de programación ***JavaScript***[3]. Es un producto que sirve como base para la programación avanzada de aplicaciones, que aporta una serie de funciones o códigos para realizar tareas del lado del cliente comúnes en aplicaciones Web como revisar los datos de un formulario antes de enviarlos, de una manera más sencilla que al programarlas solamente con JavaScript.

Cuando un desarrollador tiene que utilizar JavaScript, generalmente tiene que preocuparse por hacer scripts compatibles con varios navegadores y para ello tiene que incorporar mucho código que lo único que hace es detectar el navegador del usuario, para hacer una u otra cosa dependiendo del tipo de navegador que se utilize. JQuery puede ayudar, puesto que implementa una serie de clases *(de programación orientada a objetos)* que permiten programar sin preocuparnos del navegador con el que nos está visitando el

[2] "*Java Persistence API*, más conocida por sus siglas *JPA*, es la biblioteca de persistencia desarrollada para la plataforma *Java EE*. La *Java Persistence API*, a veces referida como JPA, es un marco de trabajo del lenguaje de programación *Java* que maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones Standard *(Java SE)* y Enterprise *(Java EE)*. El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos *(siguiendo el patrón de mapeo objeto-relacional)*" [60].

[3] "*JavaScript*, se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Gracias a su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje de programación del lado del cliente más utilizado" [58].

usuario, ya que funcionan de forma exacta en todas las plataformas más habituales.

Así pues, este marco de trabajo, nos ofrece una infraestructura con la que tendremos mucha mayor facilidad para la creación de aplicaciones complejas del lado del cliente. Por ejemplo, con JQuery obtendremos ayuda en la creación de interfaces de usuario, efectos dinámicos, aplicaciones que hacen uso de AJAX, entre otras cosas. Cuando programemos JavaScript con JQuery tendremos a nuestra disposición una interfaz para programación que nos permitirá hacer funciones o efectos, los cuales serán procesados en el navegador del usuario y a su vez estaremos seguros que funcionarán para todos nuestros visitantes. Simplemente debemos conocer las bibliotecas del marco de trabajo y programar utilizando las clases, sus propiedades y métodos para la obtención de nuestros objetivos.

### 5.2.1. Ventajas de JQuery

Es importante comentar que JQuery no es el único marco de trabajo que existe en el mercado. Existen varias soluciones similares que también funcionan muy bien, las cuales básicamente nos sirven para hacer lo mismo que hace JQuery como se mencionará mas adelante. Como es normal, cada uno de los marcos de trabajo tiene sus ventajas e inconvenientes como el que algunos sean más sencillos de usar en ciertas tareas que otros, o bien que sean solamente compatibles para pocos navegadores, cabe mencionar que JQuery es un producto con una aceptación muy buena por parte de los programadores y un grado de penetración en el mercado muy amplio [58], lo que hace suponer que es una de las mejores opciones ya que lo respaldan muchas empresas [58]. Además de ser un producto serio, estable, bien documentado y con un gran equipo de desarrolladores a cargo de la mejora y actualización del mismo. Otra cosa muy interesante es la comunidad de desarrolladores de extensiones o componentes, lo que hace fácil encontrar soluciones ya creadas en JQuery para implementar asuntos como interfaces de usuario, galerías, votaciones, efectos diversos, etc.

Ahora bien, todas estas mejoras de la *Web 2.0*, que en un principio puede ser muy atractivas, también tienen un costo en tiempo de desarrollo de los proyectos. Sin un marco de trabajo como JQuery, el tiempo de creación y depuración de todos esos componentes dinámicos sería mucho mayor. Sin embargo, lo más complicado de JQuery es aprender a usarlo, igual que pasa con cualquier otro marco de trabajo de *JavaScript [58].* A continuación se presentan tablas comparando distintos marcos de trabajo de JavaScript, ver el Cuadro 5.1 y Cuadro 5.2 [44].

| | Prototype | JQuery | Yui | ExtJS | MooTools |
|---|---|---|---|---|---|
| **Soporte IE** | 6.0+ | 6.0+ | 6.0+ | 6.0+ | 6.0+ |
| **Soporte Firefox** | 1.5+ | 2.0+ | 3.0+ | 1.5+ | 2.0+ |
| **Soporte Safari** | 2.0.4+ | 3.0+ | 4.0+ | 3.0+ | 2.0+ |
| **Soporte Opera** | 9.25+ | 9.0+ | 10.0+ | 9.0+ | 9.0+ |
| **Soporte Chrome** | 1.0+ | 1.0+ | No verificado | No verificado | No verificado |

Cuadro 5.1.: *Soporte a navegadores Web*, comparación de marcos de trabajo para *JavaScript*

*El Cuadro 5.2 presenta las características principales de marcos de trabajo de JavaScript. Los campos de la tabla se describen a continuación.*

*Soporta AJAX.* Ayuda a presentar de forma asíncrona una petición HTTP al servidor Web y dibujar su respuesta sin refrescar o redibujar una nueva página Web.

*Manipulación DOM (Modelo en Objetos para la Representación de Documentos).* Ayuda a modificar parte de la página Web para reflejar los cambios o datos regresados por la respuesta HTTP.

*DOM Traversal.* Ayuda a encontrar elementos de la página Web. Si se tiene un elemento y se quiere su elemento pariente, uno de sus elementos hijos, o su anterior elemento o siguiente elemento hermano.

*Manipulación de eventos.* Ayuda a manipular eventos en tiempo real, por ejemplo hacer que un botón de la página Web cambie de color al poner el puntero del ratón encima de este.

*Json.* Ayuda a manejar funciones tipo arreglos llamados *Json* para encontrar los elementos que contiene.

*Selectores.* Ayuda a encontrar elementos *HTML* basados en su identificador.

| | Prototype | JQuery | Yui | ExtJS | MooTools |
|---|---|---|---|---|---|
| **Soporta AJAX** | Si | Si | Si | Si | Si |
| **Manipulación DOM** | Si | Si | Si | Si | Si |
| **DOM Traversal** | Si | Si | Si | Si | Si |
| **Manipulación de eventos** | Si | Si | Si | Si | Si |
| **Json** | Si | Si | Si | Si | Si |
| **Selectores** | Si | Si | Si | Si | Si |

Cuadro 5.2.: *Características principales de marcos de trabajo de JavaScript,* los campos en común se describen en este cuadro.

## 5.3. Utilizando el JQuery de Liferay

JQuery se encuentra incorporado en Liferay, esto es benéfico para los desarrolladores ya que no es necesario incorporar algún otro marco de trabajo hecho en JavaScript. Podemos usar diversos scripts de JQuery que se encuentran en `/liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/applications/liferay-portal/html/js/jquery/` al importarlas al proyecto mediante poner en nuestros archivos con extensión .jsp el código mostrado en el Cuadro 5.3 . Sin embargo la versión que viene integrada en Liferay no se encuentra actualizada, por lo que si se requiere utilizar la versión más reciente u otra versión de JQuery se debe incorporar a nuestro proyecto mediante lo que se muestra en el Cuadro 5.4 [8].

```
1 <script type="text/javascript" src="/html/js/jquery/jquery.js"></script>
```

Cuadro 5.3.: *Incorporamos el marco de trabajo* JQuery *que viene con* Liferay *a nuestro proyecto*

Incorporar diferentes versiones del marco de trabajo de JQuery, se puede hacer de dos maneras, utilizando la entrada mostrada en el *primer código* que se muestra en el Cuadro 5.4 para agregar JQuery solamente a nuestro portlet o la entrada mostrada en el *segundo código* que se muestra arriba para agregarlo a todo el portal.
*Primer código*: Se agrega la entrada mostrada en el archivo de nuestro proyecto *portlet.xml* que se encuentra en la carpeta `docroot/WEB-INF/` de nuestro proyecto.
*Segundo código*: Se agrega la entrada mostrada en el archivo *portal_ normal.vm* que se encuentra en la carpeta de nuestro tema `glassfish-3.0.1/domains/domain1/applications/liferay-portal/html/themes/classic/templates`

```
1 <header-portlet-javascript>http://code.jquery.com/jquery-(versión de JQuery
    ).js</header-portlet-javascript>
```

Cuadro 5.4.: *Incorporamos la versión del marco de trabajo de* JQuery.

## 5.4. Creación de un portlet avanzado utilizando Hibernate y JQuery (AJAX)

La creación de un portlet con las tecnologías de *Hibernate* y *JQuery* es de gran utilidad ya que como se dijo anteriormente ayudan a minimizar el tiempo de desarrollo de un sistema y mostrar un buen resultado al usuario final. A continuación se muestra un portlet como ejemplo de la implementación de estas tecnologías para así poder ocuparlas para la creación de portlets que requieran su uso. El portlet podrá acceder a otra base de datos diferente a la de Liferay que guarde datos diferentes a los del portal donde crearemos una base de datos sencilla que muestre grupos y los alumnos que están inscritos a estos grupos. Por ejemplo, una institución educativa que cuenta con 5 grupos llamados A101, A102, A103, A104 Y A105, de las cuales queremos saber que alumnos están inscritos a cada grupo de forma que solamente con seleccionar el grupo muestre sus alumnos sin recargar la página, este ejemplo se puede utilizar para cualquier institución educativa ya que se puede crear un portal para administrar los grupos que existen en la institución.

En la vista inicial se mostrarán los grupos que están almacenados en la base de datos. Al seleccionar cada grupo, se mostrará automáticamente los alumnos que están inscritos al mismo sin tener que cambiar de página, esto gracias al uso de JQuery. La Figura 5.1 muestra el estado inicial del portlet.

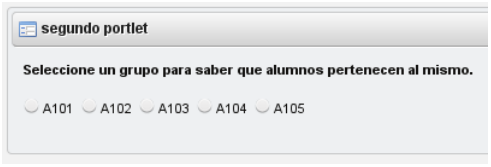


Figura 5.1.: *Portlet para saber que alumnos pertenecen a un grupo de una institución educativa.* Se muestran los 5 grupos pertenecientes a la institución educativa, al seleccionar un círculo, se mostrará los alumnos inscritos al grupo que se encuentra a lado del círculo seleccionado.

En la Figura 5.2 se muestra que al haber seleccionado el circulo perteneciente a un grupo, se muestran los nombres de las personas que están inscritas a ese grupo, cuando se selecciona un grupo diferente, se actualiza la lista mostrada, en este caso la actualización se debe a que en la base de datos cada grupo tiene una lista diferente de alumnos inscritos.


Figura 5.2.: *Selección de un grupo de la institución educativa.*
Al seleccionar el grupo A101 se muestra una lista con tres alumnos, los cuales son los únicos alumnos que están inscritos a este grupo.

### 5.4.1. Configuración inicial utilizando la arquitectura Modelo Vista Controlador (MVC)

A continuación se mostrarán las tecnologías y la configuración que se deberá tener para empezar a desarrollar el portlet, primero debemos instalar la biblioteca llamada ***JTA***[4] , para esto descargamos `jta.jar` que se encuentra en
`http://www.oracle.com/technetwork/java/javaee/jta/index.html`
y lo instalamos en
`liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/applications /liferay-portal/WEB-INF/lib/` .

A continuación se mostrará como empezar a usar una extensión muy útil para *Eclipse IDE* llamado ***Hibernate Tools,*** la cual nos permitirá crear los archivos que necesitamos para utilizar Hibernate.

Lo siguiente es preparar Eclipse para empezar a usar Hibernate. Para instalar la extensión de *Hibernate Tools* nos vamos al menú de `Eclipse Help -> Install New Software` en donde dice `Work With` escribimos:
`http://download.jboss.org/jbosstools/updates/development/` ahora, se deberá ir a donde dice `Data Services-> Hibernate Tools`, se selecciona y oprimimos el botón `Next` lo cual nos solicitará la licencia y la instalará, nos pedirá reiniciar Eclipse. Lo anterior para desarrollar con Hibernate utilizando nuestra extensión.

Para saber si se instaló correctamente la extensión de Hibernate debemos ir al menú de perspectivas de Eclipse y veremos que tenemos Hibernate, la seleccionamos, ver la Figura 5.3.

[4] *"Java Transaction API (JTA).* Es una biblioteca de *Java Enterprise Edition (Java EE)*, que provee componentes de software conocida como administradores de transacciones *(Transaction Managers o TA).* Los cuales son usados para coordinar transacciones envolviendo múltiples fuentes de datos *(las bases de datos son las mas comúnes)"*[59].


Figura 5.3.: *Perspectiva de Hibernate.*

Ya que tenemos configurado Eclipse necesitamos tener nuestra base de datos ya lista. Para este portlet se ocupará el manejador de base de datos *PostgreSQL*, el diagrama de como quedaría la base de datos llamada *"alumno"* sería la mostrada en la Figura 5.4.

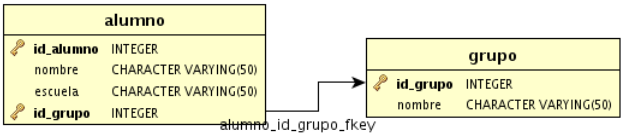


Figura 5.4.: *Diagrama Entidad-Relación de la base de datos alumno.*

El script para generarlo sería el mostrado el Cuadro 5.5. A continuación se explica el código contenido en ella.

*Líneas 1-4.* Se crea la tabla `grupo` cuyas columnas serán el *identificador del grupo* y el *nombre del grupo.*

*Líneas 6-16.* Se crea la tabla `alumno` cuyas columnas serán el identificador del *alumno*, *el nombre del alumno, la escuela del alumno* y *el identificador al grupo* al que pertenece el cual hace referencia a la tabla `grupo` que se ha creado.

*Líneas 18-22.* Se insertan 5 grupos en la columna nombre de la tabla `grupo` creándose un identificador para cada uno por defecto.

*Líneas 24-28.* Se insertan los nombres, las escuelas a las que pertenece el alumno y el identificador del grupo al que pertenece a las columnas nombre, escuela y id_grupo de la tabla `alumno`. Por ejemplo el Alumno *"Armando Palacios"* pertenece al grupo con identificador *"1"* cuyo grupo es *"A101"*.

```
1  CREATE TABLE grupo(
2  id_grupo SERIAL PRIMARY KEY,
3  nombre VARCHAR(50)
4  );
5
6  CREATE TABLE alumno
7  (
8    id_alumno SERIAL,
9    nombre VARCHAR(50) NULL,
10   escuela VARCHAR(50) NULL,
11   id_grupo INTEGER NOT NULL,
12   CONSTRAINT alumno_pkey PRIMARY KEY (id_alumno,id_grupo),
13   CONSTRAINT alumno_id_grupo_fkey FOREIGN KEY (id_grupo)
14       REFERENCES grupo (id_grupo) MATCH SIMPLE
```

```
15          ON UPDATE CASCADE ON DELETE CASCADE
16  );
17
18  INSERT INTO grupo(nombre) VALUES('A101');
19  INSERT INTO grupo(nombre) VALUES('A102');
20  INSERT INTO grupo(nombre) VALUES('A103');
21  INSERT INTO grupo(nombre) VALUES('A104');
22  INSERT INTO grupo(nombre) VALUES('A105');
23
24  INSERT INTO alumno(nombre,escuela,id_grupo) VALUES('Armando Palacios','
        escuela1',1);
25  INSERT INTO alumno(nombre,escuela,id_grupo) VALUES('Juan Perez','escuela1'
        ,1);
26  INSERT INTO alumno(nombre,escuela,id_grupo) VALUES('Maria Lopez','escuela1'
        ,1);
27  INSERT INTO alumno(nombre,escuela,id_grupo) VALUES('Maria Ana','escuela1'
        ,2);
28  INSERT INTO alumno(nombre,escuela,id_grupo) VALUES('Mario Trejo','escuela1'
        ,3);
```

Cuadro 5.5.: *Código SQL de la base de datos alumno.*

A continuación debemos crear un conjunto de conexiones llamado *Alumno* con el recurso *JDBC* llamado *jdbc/Alumno* en Glassfish similarmente a como se muestra en el anexo `B.1.1.1`, con la diferencia que en lugar de referenciar a la base de datos de Liferay llamada *lportal*, se hará referencia a la base de datos *Alumno*.

Ahora creamos un portlet básico de forma similar al que creamos en la sección 2.2, con la diferencia de que en este ejemplo la clase principal será `controlador.Controlador` para ejemplificar la arquitectura ***MVC***[5]. Es importante recalcar que es necesario que en el archivo `docroot/WEB-INF/portlet.xml` se cambie:

`<portlet-class>com.liferay.util.bridges.mvc.MVCPortlet</portlet-class>`

por `<portlet-class>controlador.Controlador</portlet-class>` para especificar que la clase `Controlador` será ahora nuestra clase principal para responder las acciones del usuario.

Se tiene que colocar la biblioteca `jta.jar` en la biblioteca Liferay de nuestro proyecto mediante la configuración del *Build Path* de *Eclipse*.

[5] *"Modelo Vista Controlador (MVC),* es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos [46].

-Modelo: Esta es la representación específica de la información con la cual el sistema opera.

-Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

-Controlador: Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y probablemente a la vista".

Por último, no se debe olvidar modificar el archivo `liferay-plugin-package.properties` de la carpeta `WEB-INF` para que no tengamos problemas en utilizar las clases que tenemos en nuestro *Build Path*, ver el Cuadro 5.6 donde se muestra el archivo para especificar las propiedades del portlet [40].

*Línea 1.* Nombre de la extensión para mostrarse a los usuarios en el portal.
*Línea 2.* Identificador del portlet.
*Línea 3.* Versión del portlet.
*Línea 4.* Etiquetas para reconocer el portlet dentro del portal.
*Línea 5.* Descripción corta del portlet.
*Línea 6.* Explicación del último cambio con respecto al anterior en el portlet.
*Línea 7.* contiene la *URL* de la página principal del portlet.
*Línea 8.* Licencia utilizada en el portlet.
*Líneas 10-20:* Archivos con extensión .jar de los que depende el portlet.

```
1  name=segundo portlet
2  module-group-id=liferay
3  module-incremental-version=1
4  tags=
5  short-description=
6  change-log=
7  page-url=http://www.liferay.com author=Liferay, Inc.
8  licenses=LGPL
9
10 portal.dependency.jars=\
11 dom4j.jar,\
12 slf4j-api.jar,\
13 slf4j-log4j12.jar,\
14 commons-collections.jar,\
15 commons-logging.jar,\
16 antlr.jar,\
17 asm.jar,\
18 cglib.jar,\
19 jta.jar,\
20 hibernate3.jar
```

Cuadro 5.6.: *Archivo liferay-plugin-package.properties.* Archivo para especificar las propiedades del portlet.

El árbol generado del proyecto tiene que quedar como se muestra en la Figura 5.5, a continuación, en el cual se puede observar que se crea un paquete llamado controlador que contiene la clase `Controlador`. Los demás archivos los genera la extensión de Liferay para la creación de portlets excepto el archivo *alumno.sql* que contiene el esquema de la base de datos alumno como ejemplo. También se muestra el archivo `liferay-plugin-package.properties` ya modificado.

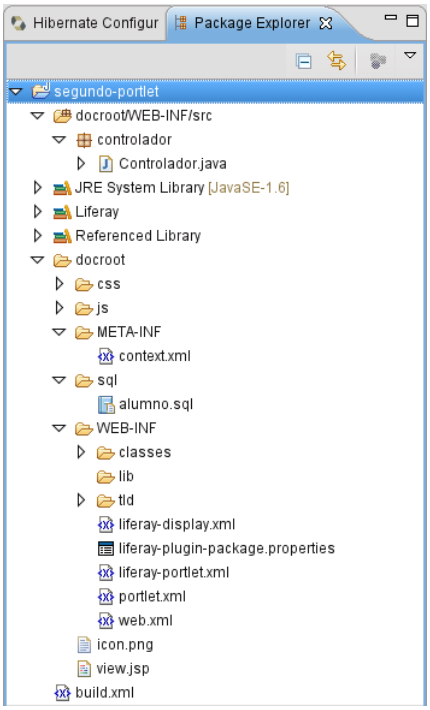


Figura 5.5.: *Esquema inicial de nuestro proyecto.*

### 5.4.2. Desarrollo del portlet avanzado

Ya que creamos el esqueleto del proyecto agregamos el controlador de *PostgreSQL* y continuamos con los pasos para desarrollar el portlet:

1. Primero seleccionamos nuestro proyecto, presionamos botón derecho y nos vamos a `Camino de construcción->Añadir bibliotecas` *("Build Path->Add Libraries")*.

2. Seleccionamos `Conectividad de la definición del controlador` *("Connectivity Driver Definition")* y presionamos `Siguiente` *("Next")* ahora nos saldrá una ventana dando las opciones disponibles para conectarnos a una fuente de datos en este caso es `Controlador JDBC PostgreSQL` *("JDBC PostgreSQL Driver")* lo seleccionamos y presionamos el pequeño botón en forma de triangulo que viene a un lado.

3. Ahora nos saldrá otra ventana con la configuración del controlador, se debe tener en cuenta que Eclipse no tiene el controlador por defecto, por eso vamos a editar la configuración para especificar el archivo con extensión *.jar* que se va a usar para la conexión, en este punto es donde nos vamos a la pestaña Lista Jar *("Jar List")* y presionaremos el boton `Añadir Jar/Zip` *("Add Jar/Zip")*.

4. Seleccionamos el controlador que se encuentra en `liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/lib/postgresql-9.0-801.jdbc4.jar`.

Una vez terminado tecleamos la opción de `Terminar` *("Finish")*, seleccionamos el proyecto y oprimimos la opción de `Refrescar` *("Refresh")*, a continuación en las bibliotecas de nuestro proyecto debe aparecer un directorio como se muestra en la Figura 5.6.

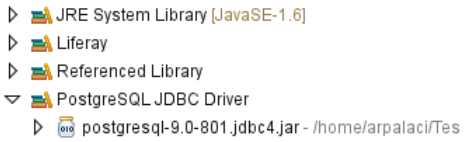


Figura 5.6.: *Se añade el controlador de PostgreSQL.*

Seleccionamos nuestro proyecto y vamos a `Nuevo->Otro->Hibernate->Configuración de Archivo Hibernate "cfg.xml"` (`"New->Other->Hibernate->` *`Hibernate Configuration File"`*).

El archivo de tipo *cfg.xml* es *XML* que contiene todo lo necesario para realizar la conexión a la base de datos .

Al continuar nos pedirá dónde crear el archivo para este ejemplo usaremos el directorio por defecto llamado *"src/"* y se llamará *hibernate.cfg.xml*, a continuación nos pedirá los datos para poder realizar nuestra conexión a la base de datos. Los campos se describen a continuación:

- *Session factory name*: Hace referencia a la conexion a PostgreSQL.
- *Database Dialect*: Hace referencia al manejador de la base de datos a utilizar para comunicarse con el controlador *JDBC*.
- *Driver Class*: Se refiere a la clase de JDBC que se va a utilizar para la conexión a la base de datos.
- *Conection URL*: Se refiere a la ruta de conexión a la base de datos.
- *Username*: Hace referencia al usuario conectado a PostgreSQL.
- *Password*: Es la contraseña utilizada para conectarse a PostgreSQL.

A continuación podemos observar la Figura 5.7 que contiene dichos campos.

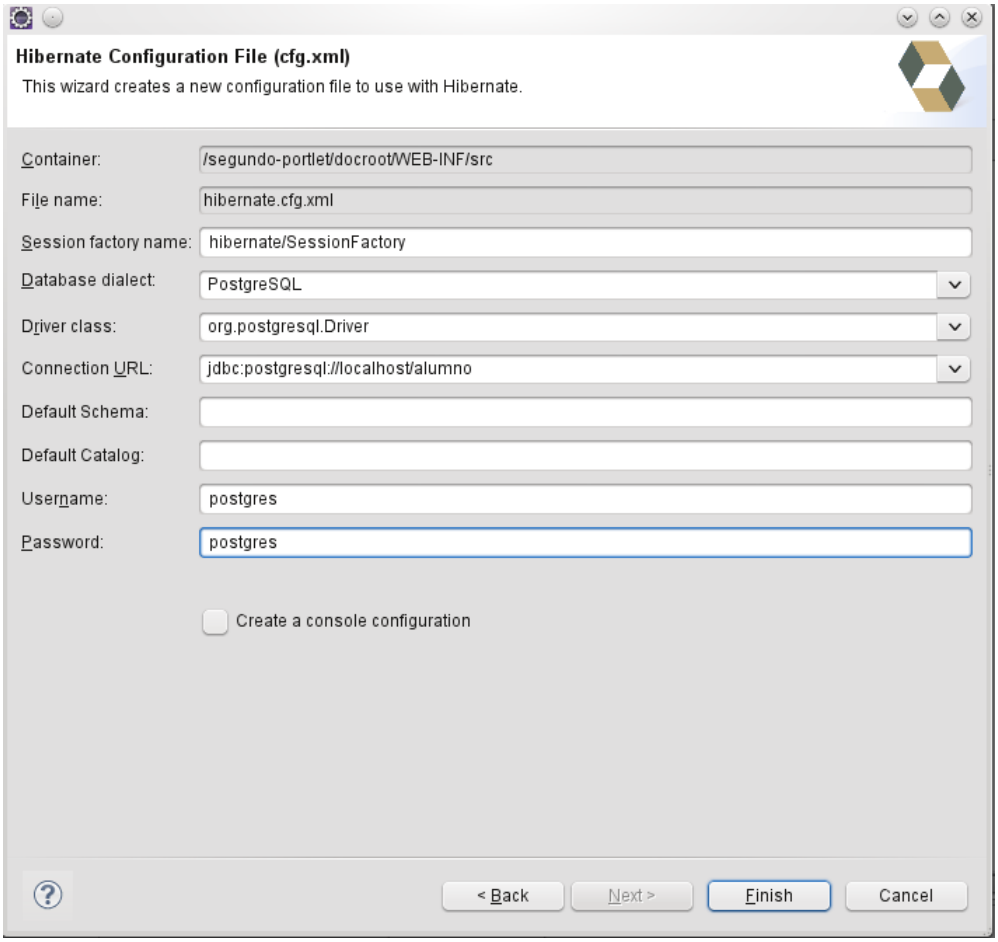


Figura 5.7.: *Archivo de configuración de Hibernate.*

A continuación se creará la consola de Hibernate mediante la *Configuración de Consola de Hibernate (“Hibernate Console Configuration”)*, la consola nos permitirá generar código y hacer consultas *HQL*. Para configurarlo vamos a `Nuevo->Otro->Hibernate ->Configuración de la Consola de Hibernate` *(“New->Other->Hibernate->*Hibernate Console Configuration*”).*

La Figura 5.8 muestra la ventana para configurar la consola de Hibernate.

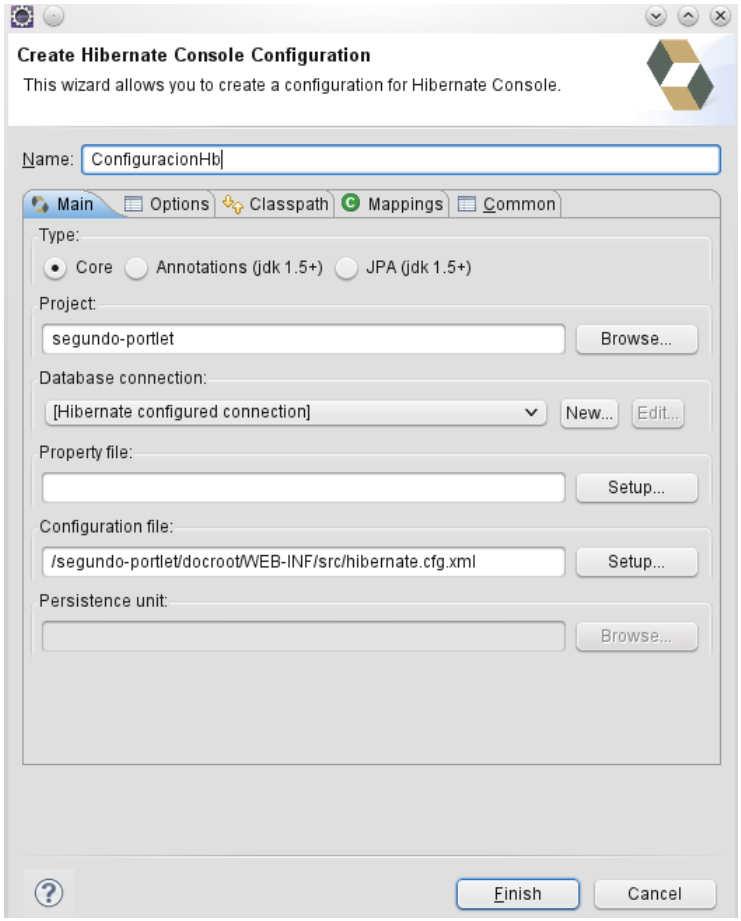


Figura 5.8.: *Configuración de la consola de Hibernate.*

En el apartado *Archivo de Configuración ("Configuration file")* viene seleccionado nuestro archivo de configuración, de otro modo, se selecciona en el campo *Nombre ("Name")* y le asignamos un nombre a nuestra configuración, en este caso llamada *"ConfiguracionHb"*.

A continuación creamos el archivo de *Ingeniería Inversa de Hibernate* de tipo *"reveng.xml"*, este archivo es el encargado de crear las clases para mapear las tablas de *PostgreSQL*. Para crear el archivo vamos a `Nuevo->Otro->Hibernate->Ingeniería Inversa de Hibernate` *("New-> Other->Hibernate->Hibernate Reverse Engineering")*. El archivo se llamará *hibernate.reveng.xml* y se debe guardar en el mismo directorio que *hibernate.cfg.xml* que está en *"src/"*, continuamos y nos aparecerá una ventana que es donde diremos que tablas queremos mapear; al refrescar se mostrará las bases de datos y sus tablas. Seleccionamos las tablas y las incluimos para mapearlas presionando el botón `Incluir` *("Include")*.
En la Figura 5.9 se puede observar la ventana para incluir las tablas a mapear de la base de datos.

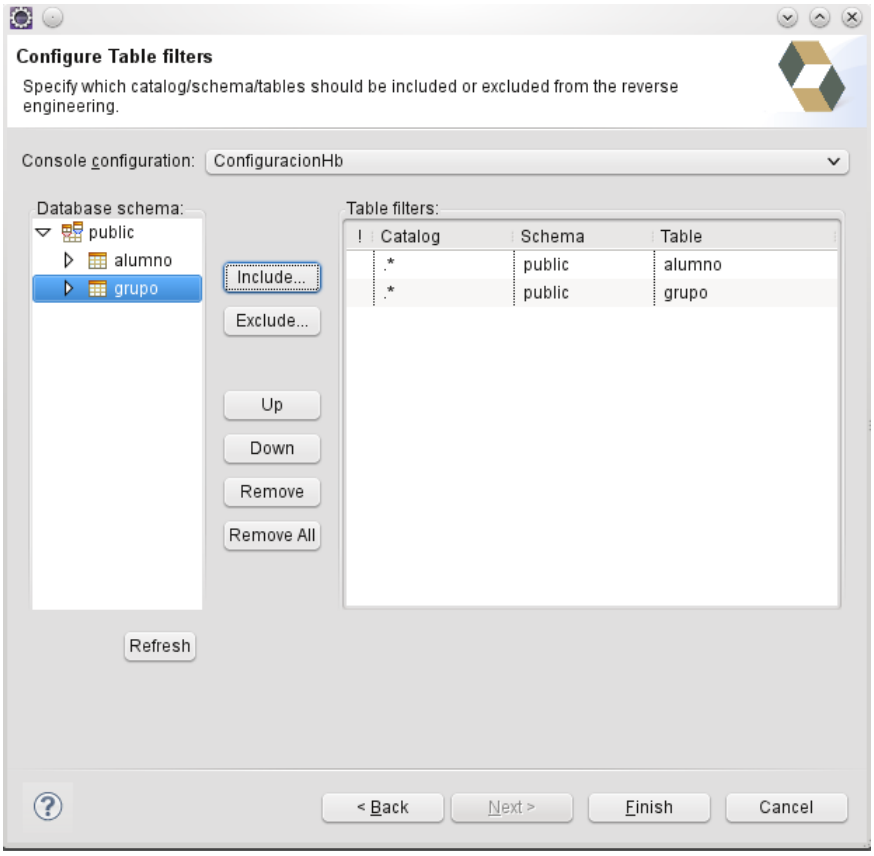


Figura 5.9.: *Ventana para incluir las tablas a mapear de la base de datos,* se pueden incluir o excluir las tablas a mapear de la base de datos, así como las columnas que queramos.

Generamos las clases de nuestra base de datos *"alumno"*, para lograr esto, vamos a `Correr->Configuraciónes de Generación de Código de Hibernate` *("Run ->Hibernate Code Generation Configurations").*

Aparecerá una ventana de configuración, cuyos campos en la pestaña principal se describen a continuación y se deben establecer como se muestra en la Figura 5.10.

*Console Configuration:* es decir la consola de Hibernate que creamos cuyo nombre es *ConfiguracionHb.*

*Output directory:* debe ser el mismo directorio de los archivos de configuración de Hibernate que es *el directorio "src/".*

*Package:* modelos es el nombre del paquete donde se guardarán las clases que mapean las tablas de la base de datos.

*revenge.xml:* se especifica la ruta de nuestro archivo *hibernate.reveng.xml* creado con anterioridad.

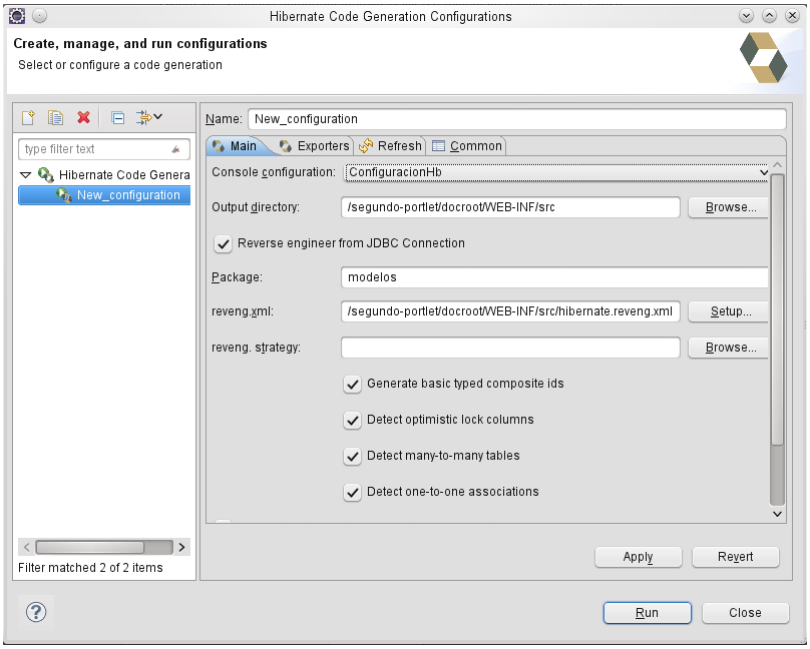


Figura 5.10.: *Configuración en la pestaña Principal ("Main").*

En la pestaña `Exportadores`, ver la Figura 5.11, indicamos los archivos que quiero que se generen.

Seleccionamos la casilla de *Código dominio ("Domain code")*, al seleccionarlo se especifican las clases que mapearán las tablas de la base de datos *"alumno"*, las cuales tendrán los nombres de estas tablas que son `grupo` y `alumno`, al generarlas se llamarán *Grupo.java* y *Alumno.java.*

Al seleccionar *Mapeos XML de Hibernate ("Hibernate XML Mappings"),* se van a generar archivos con extensión "hbm.xml" los cuales mapean las columnas de las tablas que queremos mapear.

Al seleccionar *Configuración XML de Hibernate ("Hibernate XML Configuration")*, se especificará la ruta a los mapeos de las clases, esto quiere decir a donde se encuentran los archivos con extensión *"hbm.xml"*. Esto modificará el archivo *hibernate.cfg.xml.*

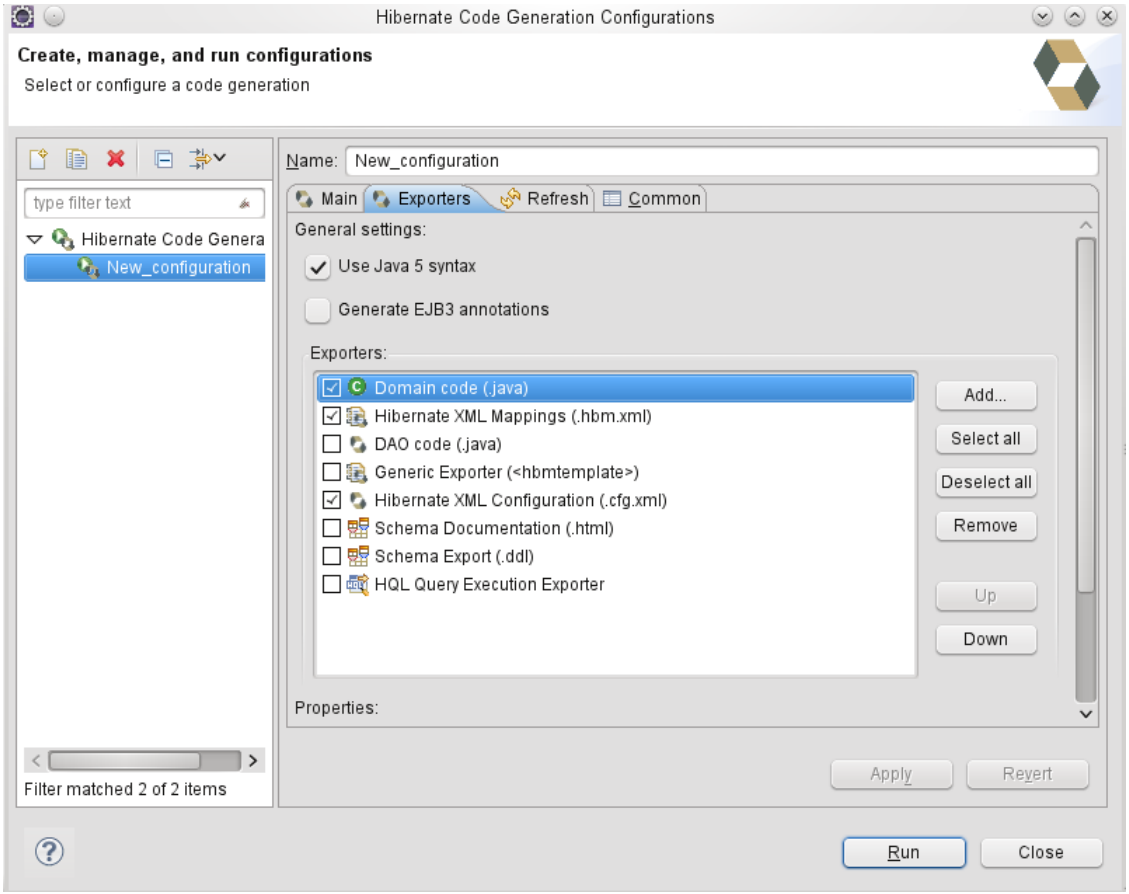


Figura 5.11.: *Configuración en la pestaña Exportadores ("Exporters").*

A continuación seleccionamos el botón de *Aplicar ("Apply")* para aplicar los cambios y *Correr ("Run")* para generar las clases de nuestras tablas y los archivos para el mapeo de la base de datos en el paquete llamado *modelo*.

La información contenida en los archivos *XML* es el mapeo de su respectiva tabla y las clases contienen las operaciones respectivas de obtención de contenido *(get)* y asignaciones *(set)* correspondientes al contenido de las columnas de las tablas de la base de datos *"alumno"*.

Para poder comprobar que lo realizado es correcto, podemos ir a la perspectiva de Hibernate y seleccionar la configuración en la pestaña `hibernate configurations` y seleccionamos `Database-> HQL Editor` que nos permitirá ejecutar sentencias *HQL*, como se puede observar en la Figura 5.12 se ejecuta el código HQL *"from Alumno"* que de resultado mostrará todos los objetos de tipo *Alumno* que representa el mapeo de los registros de la tabla `alumno` perteneciente base de datos *"alumno"* como se muestra en la Figura 5.13.

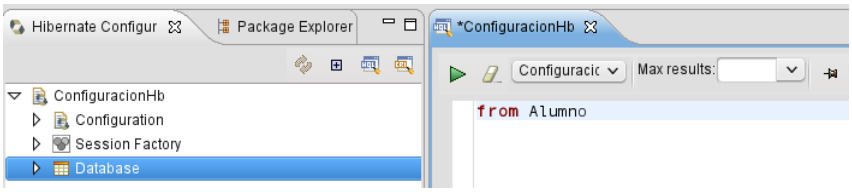


Figura 5.12.: *Lenguaje HQL.* Muestra los objetos de tipo *Alumno* con la sentencia *"from"* parecida al lenguaje *SQL.*

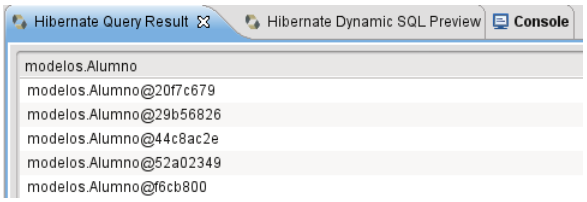


Figura 5.13.: *Resultado de la sentencia "from Alumno".* Se muestran los objetos que representan los registros de la base de datos *"alumno".*

Se ha visto la ventaja de usar la extensión de Hibernate para Eclipse ya que nos brinda la facilidad de hacer el mapeo a la base de datos sin tener que editar cada uno de los archivos con extensión *".xml".*

### 5.4.3. Comportamiento del portlet avanzado

A continuación se mostrará el comportamiento del portlet respecto a las acciones que se tomen en su ejecución antes de mostrar el código. Es importante porque se puede observar la funcionalidad de cada método del código del portlet desde que empieza a ejecutarse hasta que se termina de utilizar, con lo cual se tiene una perspectiva mas amplia de como es que actúan los portlets. También hay que notar que el procesamiento de las consultas se hace desde el servidor, lo único que se gana es la presentación de los resultados hacia el usuario ya que no se refresca la página.

- Al instalar el portlet y probarlo, primero se ejecuta el método `Controlador.init` el cual busca la página inicial del archivo `portlet.xml` que se muestra en las etiquetas `<init-param> <name>view-jsp</name> <value>/view.jsp</value> </init-param>` cuyo nombre es `view-jsp` pero representa al archivo `/view.jsp`, para continuar con el siguiente paso.

- Se ejecuta el método `Controlador.doDispatch` el cual pide el parámetro *"accion"*, como *"accion"* es *null,* ya que no se ha realizado acción alguna sobre el portlet, `Controlador.doDispatch` llama al método `Controlador.include` que redirige a `view.jsp` y le manda la lista de los grupos utilizando el método `Modelo.listGrupo`, a continuación se muestra la página con los grupos que han sido recibidos utilizando *JSTL.*

- Lo que hizo el método `Modelo.listGrupo` es que creó la consulta necesaria para enviar los grupos existentes mediante una petición de redibujado *(RenderRequest)*.
- Al mostrar la página, se carga el archivo `main.js` que utiliza la biblioteca de JQuery. Al seleccionar un grupo se mostrará la lista de alumnos que contiene, esto debido a que el archivo `main.js` contiene métodos de JQuery que hacen que cada vez que se seleccione un botón de tipo radio se haga una llamada *Ajax* que manda el identificador del grupo seleccionado a la clase Controlador, como ahora el parámetro *"accion"* no es *null* ya que se ha seleccionado un botón de tipo radio, `Controlador.doDispatch` envía el identificador del grupo al método `Modelo.muestraAlumnos`, el cual manda un *objeto* ***Json***[6] a la página, este objeto *Json* es recibido por el método *getJSON* de JQuery mediante el parámetro llamado datos de este mismo método que se encuentra en `main.js`; el objeto *Json* contiene el ***JSONArray***[7] de los alumnos que pertenecen al grupo seleccionado, los cuales se listarán.
- Cada vez que se seleccione un grupo diferente, se borrará la lista de alumnos mostrada anteriormente y se pondrá en su lugar la de alumnos del grupo seleccionado por último sin refrescar la página gracias al uso de *Ajax* que podemos manejar con JQuery.

### 5.4.4. Archivos modificados y utilizados en el desarrollo del portlet avanzado

La finalidad de este apartado es explicar y mostrar el código utilizado para la creación del portlet avanzado que se ha estado desarrollando poco a poco durante todo el Capítulo 5.

En la Figura 5.14 se muestra el árbol de carpetas, paquetas, archivos y bibliotecas necesarias para la construcción del portlet avanzado con la finalidad de mostrarla como un ejemplo para su desarrollo.

[6] *"JSON,* acrónimo de *JavaScript Object Notation*, es un formato ligero para el intercambio de datos. *JSON* es un subconjunto de la notación literal de objetos de *JavaScript* que no requiere el uso de *XML*. Un objeto *JSON* es una colección que no tiene un orden específico de pares de nombres/valores. En su forma externa es una cadena entre llaves con dos puntos entre los nombres y valores, y comas entre los valores y nombres. La forma interna es un objeto que tiene que conseguir y optar por métodos de acceso a los valores buscando por su nombre, y poner los métodos para añadir o sustituir los valores buscando por su nombre. Los valores pueden ser cualquiera de estos tipos: Boolean, JSONArray, JsonObject, Number, String, o el objeto `JSONObject.NULL`. Ejemplo: `myString = new JSONObject().put("resultado", "Hola Mundo").toString();` produce:
`{"resultado": "Hola Mundo"}`"[58].

[7] *"Un JSONArray* es una colección de valores en forma de lista. Ejemplo: ["México","Chile","Canadá"]" [58].

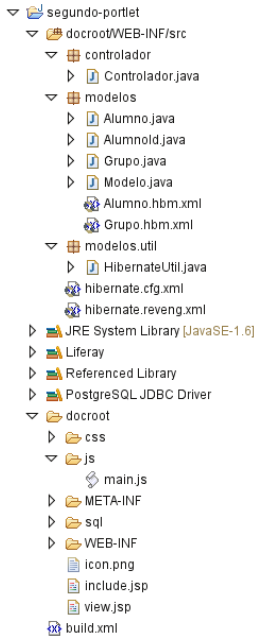


Figura 5.14.: *Árbol de carpetas, paquetes, archivos y bibliotecas del portlet avanzado.* Se muestran los archivos y bibliotecas utilizadas para el desarrollo del portlet avanzado.

El archivo `hibernate.cfg.xml` mostrado en el Cuadro 5.7 contiene la configuración para el acceso a la base de datos [59]. La explicación por líneas de este archivo es la siguiente:

*Líneas 1-37.* Se inicializa la configuración de *Hibernate* como tipo de archivo *XML*.

*Líneas 6.* Se nombra el atributo XML `session-factory` con el valor *"hibernate/SessionFactory"* que se encarga de decir al sistema donde se encuentran todos los archivos de mapeo de Hibernate y el dialecto de Hibernate a utilizar.

*Líneas 7-9.* En estas líneas se hace uso de la biblioteca *JTA* la cual ayuda a coordinar las transacciones a la base de datos con el vínculo UserTransaction para hacerlas dentro de una sesión definida dentro de un método.

*Líneas 10-12.* Usamos la clase `JTATransactionFactory` para no tratar las transacciones como una sola unidad si no como varias, aumentando el rendimiento de las mismas.

*Líneas 13-18. Hibernate* instancia cualquier clase especificada en la propiedad `hibernate.transaction.manager_lookup_class` para obtener el manejador de transacciones adecuado para el contenedor en que esta ejecutándose la aplicación. Como se indica esta es una propiedad dependiente del servidor. Para *Glassfish* es necesario `org.hibernate.transaction.SunONETransactionManagerLookup`.

*Líneas 18-21.* Se especifica que se utilizará el controlador del manejador de la base de datos PostgreSQL.

*Líneas 22-25.* Se especifica la conexión a la base de datos creada para el recurso JDBC llamado jdbc/Alumno en el servidor de Glassfish que hace referencia a la base de datos

alumno.

*Líneas 26-28.* Se especifica que cada transacción realizada en una sesión de JTA será tratada como hilos de ejecución diferentes.

*Líneas 30-33.* Se especifica que en los archivos de registro del sistema no se impriman las consultas.

*Líneas 34-35.* Se especifica la ruta a los archivos de mapeo de las clases que representan las tablas de la base de datos.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5  <hibernate-configuration>
6      <session-factory name="hibernate/SessionFactory">
7       <property name="jta.UserTransaction">
8     java:comp/UserTransaction
9       </property>
10      <property name="transaction.factory_class">
11          org.hibernate.transaction.JTATransactionFactory
12         </property>
13    <property name="hibernate.transaction.manager_lookup_class">
14     org.hibernate.transaction.SunONETransactionManagerLookup
15    </property>
16          <property name="hibernate.bytecode.use_reflection_optimizer">
17           false
18    </property>
19          <property name="hibernate.connection.driver_class">
20           org.postgresql.Driver
21    </property>
22    <property name="hibernate.connection.datasource">jdbc/Alumno</property>
23          <property name="hibernate.dialect">
24           org.hibernate.dialect.PostgreSQLDialect
25    </property>
26    <property name="hibernate.current_session_context_class">
27     thread
28    </property>
29    <property name="hibernate.jdbc.batch_size">20</property>
30    <property name="show_sql">false</property>
31          <property name="hibernate.search.autoregister_listeners">
32           false
33    </property>
34          <mapping resource="modelos/Grupo.hbm.xml" />
35          <mapping resource="modelos/Alumno.hbm.xml" />
36      </session-factory>
37 </hibernate-configuration>
```

Cuadro 5.7.: *Código del archivo hibernate.cfg.xml.*

En el Cuadro 5.8 se muestra el código del archivo para generar las clases de mapeo mediante ingeniería inversa utilizada por la extención de *Hibernate* llamada *Hibernate Tools*.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate
       Reverse Engineering DTD 3.0//EN" "http://hibernate.sourceforge.net/
       hibernate-reverse-engineering-3.0.dtd" >
3
4  <hibernate-reverse-engineering>
5    <table-filter match-schema="public" match-name="alumno"/>
6    <table-filter match-schema="public" match-name="grupo"/>
7  </hibernate-reverse-engineering>
```

Cuadro 5.8.: *Código del archivo hibernate.revenge.xml.*

En el Cuadro 5.9 se muestra el código del archivo que mapea la clase `Alumno`.

*Línea 6.* La clase `modelos.Alumno` mapea la tabla llamada `alumno`.

*Líneas 7-14.* La clase `modelos.AlumnoId` representa la composición de los dos identificadores de la tabla `alumno` cuyas columnas en la tabla de la base de datos son *"id_ alumno"* e *"id_ grupo"* que en la clase `modelos.Alumno` se representan como atributos `idAlumno` e `idGrupo` de tipo de dato entero respectivamente, esto para crear consultas donde solamente se utilizen estos dos identificadores.

*Líneas 15-17.* Se especifica que mas de un alumno puede hacer referencia a un solo grupo.

*Líneas 18-23.* Se especifican los datos de cada alumno preteneciente a un grupo de tipo de dato `String` de longitud 50.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
       3.0//EN"
3  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
4  <!-- Generated 16-abr-2011 18:31:19 by Hibernate Tools 3.4.0.CR1 -->
5  <hibernate-mapping>
6      <class name="modelos.Alumno" table="alumno" schema="public">
7          <composite-id name="id" class="modelos.AlumnoId">
8              <key-property name="idAlumno" type="int">
9                  <column name="id_alumno" />
10             </key-property>
11             <key-property name="idGrupo" type="int">
12                 <column name="id_grupo" />
13             </key-property>
14         </composite-id>
15         <many-to-one name="grupo" class="modelos.Grupo" update="false"
               insert="false" fetch="select">
```

```
16              <column name="id_grupo" not-null="true" />
17          </many-to-one>
18          <property name="nombre" type="string">
19              <column name="nombre" length="50" />
20          </property>
21          <property name="escuela" type="string">
22              <column name="escuela" length="50" />
23          </property>
24      </class>
25  </hibernate-mapping>
```

Cuadro 5.9.: *Código del archivo Alumno.hbm.xml.*

En el Cuadro 5.10 se muestra el código del archivo que mapea la clase `Grupo`.

*Línea 6.* La clase `modelos.Grupo` mapea la tabla llamada `grupo`.

*Líneas 7-10.* Se especifica el identificador de la tabla ***grupo*** que se asigna como atributo `idGrupo` a la tabla `modelos.Grupo.`

*Líneas 11-13.* Se especifica la columna de la tabla `grupo` llamada *"nombre"* como un atributo llamado nombre a la clase `modelos.Grupo`.

*Líneas 14-19.* Se especifica que el identificador de la tabla llamado *"id_grupo"* hace referencia a la tabla `alumno` de un solo grupo para muchos alumnos.

```
1   <?xml version="1.0"?>
2   <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
        3.0//EN"
3   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
4   <!-- Generated 16-abr-2011 18:31:19 by Hibernate Tools 3.4.0.CR1 -->
5   <hibernate-mapping>
6       <class name="modelos.Grupo" table="grupo" schema="public">
7           <id name="idGrupo" type="int">
8               <column name="id_grupo" />
9               <generator class="assigned" />
10          </id>
11          <property name="nombre" type="string">
12              <column name="nombre" length="50" />
13          </property>
14          <set name="alumnos" table="alumno" inverse="true" lazy="true" fetch
                ="select">
15              <key>
16                  <column name="id_grupo" not-null="true" />
17              </key>
18              <one-to-many class="modelos.Alumno" />
19          </set>
20      </class>
21  </hibernate-mapping>
```

Cuadro 5.10.: Código del archivo Grupo.hbm.xml.

En el Cuadro 5.11 se muestra el código del archivo con extensión .jsp usado como página de inicio, el cual contiene etiquetas en HTML, EL y JSTL para mostrar los grupos de la base de datos.

*Líneas 10-13.* Se hace un bucle por cada lista de grupos existentes en la base de datos la cual se obtiene utilizando *ExpresionLanguaje* y las etiquetas para realizar el bucle con un `forEach` de *JSTL.*

```
1  <%/**@author: Palacios Trejo Armando ciencias UNAM */%>
2  <%@ include file="include.jsp"%>
3
4  <b>Seleccione un grupo para saber que alumnos pertenecen al mismo.</b> <br>
       <br>
5
6  <div id="divContenedor">
7   <table width="100%">
8     <tr>
9       <td>
10          <c:forEach var="listaGrupos" items="${requestScope.listaGrupos}">
11        <input type="radio" name="grupo1" value="${listaGrupos.idGrupo}">
12          ${listaGrupos.nombre} 
13        </c:forEach>
14       </td>
15     </tr>
16  </table>
17 </div>
18 <br>
19 <div id="divTexto"></div>
```

Cuadro 5.11.: Código del archivo view.jsp.

En el Cuadro 5.12 se muestra el código del archivo con extensión .jsp usado para cargar los archivos y configuraciones a utilizar en *view.jsp.*

Líneas 5-6. Se especifica la ruta a las hojas de estilo a utilizar en el portlet.

Líneas 7-9. Se especifica la ruta a los archivos JavaScript a utilizar en el portlet.

Línea 11. En la propiedad `urlAjax = '<portlet:renderURL windowState="exclusive"/>';` se especifica el estado de la ventana en *"exclusive"* ya que de otra forma no se podrá utilizar *AJAX.* lo que recibe `urlAjax` es la dirección de la página actual.

```
1  <%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
2  <%@ page isELIgnored="false" %>
3  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

```
4  <portlet:defineObjects />
5  <link rel="stylesheet" type="text/css"
6  href="${renderRequest.contextPath}/css/main.css" />
7  <script type="text/javascript" src="/html/js/jquery/jquery.js"></script>
8  <script type="text/javascript" src="${renderRequest.contextPath}/js/main.js
       ">
9  </script>
10 <script type="text/javascript">
11     urlAjax = '<portlet:renderURL windowState="exclusive"/>';
12 </script>
```

Cuadro 5.12.: Código del archivo include.jsp.

En el Cuadro 5.13 se muestra el código del archivo de tipo JavaScript donde se utiliza la biblioteca JQuery para listar los alumnos y hacer las llamadas *AJAX* para listar los alumnos del grupo que se escoja sin recargar la página, esto usando funciones de la biblioteca de JQuery que se representan con el signo *"$"* seguido de un punto

Líneas 6-22. Función cuyo trabajo es pasar los parámetros *"accion"* e *"idGrupo"* el cual contiene el valor del botón de tipo radio de las etiquetas de tipo *HTML* del archivo *view.jsp*, después escribe la lista de los alumnos pertenecientes al grupo seleccionado dentro de las etiquetas cuyo identificador es *"divTexto"* que corresponde a las etiquetas `DIV` del código del archivo *view.jsp*.

Líneas 30-35. Función que hace que antes de ejecutar la función llamada `muestraAlumnos` para mostrar los alumnos, se ejecute un método de JQuery llamado `empty` el cual borra lo que hay dentro de una etiqueta cuyo identificador es *"divTexto"*.

```
1  /**
2   * Hace una llamada AJAX al portlet. Pasa los parametros accion e idGrupo
3   * que contiene el valor del radiobutton seleccionado. Despues escribe la
        lista
4   * de los alumnos pertenecientes al grupo seleccionado.
5   */
6  function muestraAlumnos() {
7
8      var params = {
9       'accion': 'muestraAlumnos',
10      'idGrupo': $("input[@name=grupo1]:checked").val()
11     };
12
13     $.getJSON(urlAjax, params, function(datos) {
14   $.each(datos.resultados, function(index, item) {
15    var ul = $("<ul>").appendTo($("#divTexto"));
16    ul.append(
17     $(document.createElement("li")).text(item)
18     );
19   });
20  });
```

```
21
22  };
23
24
25  // Se ejecuta cuando la pagina termina de cargar
26  $(document).ready(function() {
27
28   // La funcion unbind se utiliza para que la funcion click no se ejecute
        dos
29   // veces. Ya que pueden existir otros elementos que pueden tener el evento
        .
30   $("input[@name='grupo1']").unbind("click").click(function() {
31    $("#divTexto").empty(); // borra lo que hay dentro del div divTexto.
32    muestraAlumnos();
33      });
34
35  });
```

Cuadro 5.13.: Código del archivo main.js.

En el Cuadro 5.14 se muestra el código de la clase que se encarga se administrar las peticiones *AJAX* que se hacen con el archivo *main.js* desde el cliente. Como se puede observar, no es necesario crear constructores ya que los métodos que se heredan de la clase `GenericPortlet` se sobreescriben.

*Líneas 37-42.* Al recibir el parámetro llamado *"accion"* mandado por *AJAX* desde el cliente, se verifica si es vacío; si es vacío manda la lista de grupos mediante el método `listGrupo` de la clase `Modelo` a la página principal las cuales serán recibidas por las etiquetas de *Expresion Languaje* y con un bucle utilizando *JSTL* listarlas en la página principal mostrada por el archivo *view.jsp*.

*Líneas 44-51.* Si el parámetro llamado *"accion"* mandado por *AJAX* desde el cliente es la cadena *"muestraAlumnos"* se listan los alumnos pertenecientes al grupo seleccionado mediante el método `muestraAlumnos` de la clase `Modelo`.

*Líneas 57-60.* Se inicializa la variable `viewJSP` con el valor *"view.jsp"* la cual se asignó como página inicial desde el archivo `portlet.xml`.

*Líneas 70-79.* Método que se encarga de enviar las peticiones a una ruta. Recibe las peticiones con el parámetro `RenderRequest` y regresa una respuesta con el parámetro `RenderResponse`.

```
1  package controlador;
2
3  import java.io.IOException;
4
5  import javax.portlet.GenericPortlet;
6  import javax.portlet.PortletException;
7  import javax.portlet.PortletRequestDispatcher;
```

```
8  import javax.portlet.RenderRequest;
9  import javax.portlet.RenderResponse;
10
11 import org.apache.commons.logging.Log;
12 import org.apache.commons.logging.LogFactory;
13
14 import modelos.Modelo;
15
16 /**
17  * Clase principal que administra las demas clases.
18  * @author Palacios Trejo Armando ciencias UNAM
19  */
20 public class Controlador extends GenericPortlet{
21  /** Logger. */
22  private static Log log = LogFactory.getLog(Controlador.class);
23  /** Campo necesario para el metodo doView del portlet. */
24  private String viewJSP;
25
26  /**
27   * Manda el flujo a donde sea necesario segun un parametro del request.
28   * @param req El objeto request.
29   * @param resp El objeto response.
30   * @throws IOException Si hay error de entrada/salida.
31   * @throws PortletException Si hay error en el portlet.
32   */
33  @Override
34  public void doDispatch(RenderRequest req, RenderResponse resp)
35  throws IOException, PortletException {
36   String accion = req.getParameter("accion");
37   if (accion == null) {
38    try {
39     include(viewJSP, Modelo.listGrupo(req), resp);
40    } catch (Exception e) {
41     e.printStackTrace();
42    }
43    return;
44   } else if (accion.equals("muestraAlumnos")){
45    try {
46     Modelo.muestraAlumnos(req, resp);
47    } catch (Exception e) {
48     e.printStackTrace();
49    }
50    return;
51   }
52  }
53
54  /**
55   * Inicializa variables necesarias para el portlet desde el portlet.xml.
56   */
57  @Override
58  public void init() throws PortletException {
59   this.viewJSP = getInitParameter("view-jsp");
60  }
61
```

```
62   /**
63    * Metodo auxiliar para redirigir una peticion.
64    * @param ruta La ruta para redirigir la peticion.
65    * @param req El objeto request.
66    * @param resp El objeto response.
67    * @throws IOException Si hay error de entrada/salida.
68    * @throws PortletException Si hay error en el portlet.
69    */
70   private void include(String ruta, RenderRequest req, RenderResponse resp)
71   throws IOException, PortletException {
72    PortletRequestDispatcher portletRequestDispatcher = getPortletContext()
73    .getRequestDispatcher(ruta);
74    if (portletRequestDispatcher == null) {
75     log.error(ruta + " es una ruta incorrecta");
76    } else {
77     portletRequestDispatcher.include(req, resp);
78    }
79   }
80
81  }
```

Cuadro 5.14.: *Código del archivo Controlador.java.*

---

En el Cuadro 5.15 se muestra el código de la clase que se encarga de cargar la configuración que se especificó en `hibernate.cfg.xml` para tener acceso a los datos de la base de datos.

*Líneas 25-34.* Se inicializa la configuración para usar Hibernate que se configuró en el archivo `hibernate.cfg.xml` en el atributo sessionFactory.

*Líneas 40-49.* Método que busca el contexto ya inicializado de tipo `SessionFactory` cuyo valor es *"hibernate/SessionFactory"*.

*Líneas 55-57.* Método que abre una sesión de Hibernate para poder realizar consultas.

---

```
1  package modelos.util;
2
3  import javax.naming.Context;
4  import javax.naming.InitialContext;
5  import javax.naming.NamingException;
6  import org.apache.commons.logging.Log;
7  import org.apache.commons.logging.LogFactory;
8  import org.hibernate.Session;
9  import org.hibernate.SessionFactory;
10 import org.hibernate.cfg.Configuration;
11
12 /**
13 * Clase de ayuda, toma cuidado del inicio y da acceso al
14 * org.hibernate.SessionFactory mas conveniente.
15 * @author Palacios Trejo Armando ciencias UNAM
```

```
16 */
17 public class HibernateUtil {
18  /** Logger. */
19  private static Log log = LogFactory.getLog(HibernateUtil.class);
20  /** JNDI de HIbernate. */
21  public static final String HIBERNATE_JNDI_NAME =
22   "hibernate/SessionFactory";
23  private static SessionFactory sessionFactory;
24
25  // Crea el SessionFactory inicial de la configuracion de los archivos.
26  static {
27   try {
28    sessionFactory = new Configuration().
29    configure("hibernate.cfg.xml").buildSessionFactory();
30   } catch (Throwable ex) {
31    log.fatal("Building SessionFactory failed.", ex);
32    throw new ExceptionInInitializerError(ex);
33   }
34  }
35
36  /**
37   * Regresa las SessionFactory que contiene las instancias de la sesion.
38      * @return SessionFactory con las instancias de la sesion
39   */
40  public static SessionFactory getSessionFactory() {
41   try {
42    Context ctx = new InitialContext();
43    sessionFactory = (SessionFactory) ctx.lookup(HIBERNATE_JNDI_NAME);
44    ctx.close();
45   } catch (NamingException ex) {
46    ex.printStackTrace();
47   }
48   return sessionFactory;
49  }
50
51  /**
52   * Abre y regresa la Session del SessionFactory.
53      * @return Session con las instancias de la sesion
54   */
55  public static Session getSession() {
56   return getSessionFactory().openSession();
57  }
58 }
```

Cuadro 5.15.: *Código del archivo HibernateUtil.java*

---

En el Cuadro 5.16 se muestra el código de la clase que se encarga de mapear la tabla `alumno` en un objeto `Alumno` como un registro de la base de datos.

*Línea 13.* `serialVersionUID` se usa para el manejo de versiones en la compilación de

la clase al usar el ambiente de desarrollo de Eclipse, de otra forma al compilar marca notificaciones de cuidado.

*Línea 14*. Atributo que hace referencia al identificador de la tabla `alumno` llamado *"id"* que se maneja como un objeto de tipo `AlumnoId`.

*Línea 15*. Atributo que hace referencia al identificador de la tabla `grupo` llamado *"grupo"* y su contenido que se maneja como un objeto de tipo `Grupo`.

*Línea 16*. Atributo que hace referencia a un registro de la columna *"nombre"* de la tabla `alumno`.

*Línea 17*. Atributo que hace referencia a un registro de la columna *"escuela"* de la tabla `alumno`.

*Líneas 19-32*. Constructores de la clase.

*Líneas 34-63*. Operaciones respectivas de obtención de contenido *(get)* y asignaciones *(set)*, correspondientes al contenido de un registro de las columnas de la tabla `alumno` encapsuladas como objeto de tipo Alumno.

```
1  package modelos;
2
3  /**
4   * Las fuentes java correspondientes a los beans de la tabla Alumno.
5   * @author Palacios Trejo Armando ciencias UNAM
6   */
7  public class Alumno implements java.io.Serializable {
8
9   /**
10   * Se debe cambiar cada vez que esta clase no es compatible con versiones
11   * anteriores.
12   */
13  private static final long serialVersionUID = -5100173691100775560L;
14  private AlumnoId id;
15  private Grupo grupo;
16  private String nombre;
17  private String escuela;
18
19  public Alumno() {
20  }
21
22  public Alumno(AlumnoId id, Grupo grupo) {
23   this.id = id;
24   this.grupo = grupo;
25  }
26
27  public Alumno(AlumnoId id, Grupo grupo, String nombre, String escuela) {
28   this.id = id;
29   this.grupo = grupo;
30   this.nombre = nombre;
31   this.escuela = escuela;
32  }
33
34  public AlumnoId getId() {
35   return this.id;
```

```
36   }
37
38   public void setId(AlumnoId id) {
39    this.id = id;
40   }
41
42   public Grupo getGrupo() {
43    return this.grupo;
44   }
45
46   public void setGrupo(Grupo grupo) {
47    this.grupo = grupo;
48   }
49
50   public String getNombre() {
51    return this.nombre;
52   }
53
54   public void setNombre(String nombre) {
55    this.nombre = nombre;
56   }
57
58   public String getEscuela() {
59    return this.escuela;
60   }
61
62   public void setEscuela(String escuela) {
63    this.escuela = escuela;
64   }
65
66  }
```

Cuadro 5.16.: *Código del archivo Alumno.java*

---

En el Cuadro 5.17 se muestra el código de la clase que se encarga de mapear un identificador de la tabla `alumno` con un identificador de la tabla `grupo` en un objeto `AlumnoId`.

*Línea 13.* `serialVersionUID` se usa para el manejo de versiones en la compilación de la clase al usar el ambiente de desarrollo de *Eclipse*, de otra forma al compilar marca notificaciones de cuidado.

*Línea 14.* Atributo que hace referencia al identificador de un registro de la tabla `alumno`.

*Línea 15.* Atributo que hace referencia al identificador de un registro de la tabla `grupo`.

*Líneas 17*-23. Constructores de la clase.

*Líneas* 25-39. Operaciones respectivas de obtención de contenido *(get)* y asignaciones *(set)*, correspondientes a los identificadores de un registro de las columnas de la tabla `alumno`.

*Líneas 41-62.* Tiene que sobrescribir los métodos `equals()` y `hashCode()` si:

- Si utilizamos instancias de clases persistentes en un conjunto *o "Set en inglés" (la forma recomendada de representar asociaciones multivaluadas)* [59].
- Si utilizamos reasociación de instancias separadas [59].

Hibernate garantiza la equivalencia de identidad persistente *(fila de base de datos)* y de identidad Java sólamente dentro del ámbito de una sesión en particular. De modo tal que se mezcle instancias recuperadas en sesiones diferentes, para esto se tiene que implementar `equals()` y `hashCode()` los cuales construyen una semántica significativa para `asignaciones` [59].

La forma más obvia es implementar `equals()/hashCode()` comparando el valor identificador de ambos objetos. Si el valor es el mismo, ambos deben ser la misma fila de la base de datos ya que son iguales. Si ambos son agregados a un conjunto, sólo tendremos un elemento en este conjunto. Desafortunadamente, no puede utilizar este enfoque con identificadores generados. Hibernate sólo asignará valores identificadores a objetos que son persistentes; una instancia recién creada no tendrá ningún valor de identificador. Además, si una instancia no se encuentra guardada y está actualmente en un conjunto, al guardarla se asignará un valor identificador al objeto. Si `equals()` y `hashCode()` están basados en el valor del identificador, el código hash podría cambiar, rompiendo el contrato del conjunto. No es un problema de Hibernate, sino de la semántica normal de Java referente a la identidad de objeto e igualdad. En los métodos `equals()` y `hashCode()` se utiliza igualdad de llave empresarial *("Business key equality")*. Igualdad de llave empresarial significa que el método `equals()` sólamente compara las propiedades que forman la llave empresarial, esta es una llave que podría identificar nuestra instancia en el mundo real (una llave candidata natural) [59].

```
1  package modelos;
2
3  /**
4   * Las fuentes java correspondientes a los beans.
5   * @author Palacios Trejo Armando ciencias UNAM
6   */
7  public class AlumnoId implements java.io.Serializable {
8
9   /**
10   * Se debe cambiar cada vez que esta clase no es compatible con versiones
11   * anteriores.
12   */
13  private static final long serialVersionUID = 1892698877155247709L;
14  private int idAlumno;
15  private int idGrupo;
16
17  public AlumnoId() {
18  }
19
20  public AlumnoId(int idAlumno, int idGrupo) {
21   this.idAlumno = idAlumno;
```

```
22      this.idGrupo = idGrupo;
23    }
24
25    public int getIdAlumno() {
26      return this.idAlumno;
27    }
28
29    public void setIdAlumno(int idAlumno) {
30      this.idAlumno = idAlumno;
31    }
32
33    public int getIdGrupo() {
34      return this.idGrupo;
35    }
36
37    public void setIdGrupo(int idGrupo) {
38      this.idGrupo = idGrupo;
39    }
40
41    public boolean equals(Object other) {
42      if ((this == other))
43        return true;
44      if ((other == null))
45        return false;
46      if (!(other instanceof AlumnoId))
47        return false;
48      AlumnoId castOther = (AlumnoId) other;
49
50      return (this.getIdAlumno() == castOther.getIdAlumno())
51     && (this.getIdGrupo() == castOther.getIdGrupo());
52    }
53
54    public int hashCode() {
55      int result = 17;
56
57      result = 37 * result + this.getIdAlumno();
58      result = 37 * result + this.getIdGrupo();
59      return result;
60    }
61
62  }
```

Cuadro 5.17.: *Código del archivo AlumnoId.java.*

---

En el Cuadro 5.18 se muestra el código de la clase que se encarga de mapear la tabla `grupo` en un objeto `Grupo` como un registro de la base de datos.

*Línea 16.* `serialVersionUID` se usa para el manejo de versiones en la compilación de la clase al usar el ambiente de desarrollo de Eclipse, de otra forma al compilar marca notificaciones de cuidado.

*Línea 17*. Atributo que hace referencia al identificador de la tabla `grupo` llamado *"idGrupo"*.

*Línea 18*. Atributo que hace referencia a un registro de la columna *"nombre"* de la tabla `grupo`.

*Línea 19*. Atributo que hace referencia a un conjunto de registros de la columna de la tabla `alumno` representados como objetos de tipo `Alumno`.

*Líneas 21-32*. Constructores de la clase.

*Líneas 34-48*. Operaciones respectivas de obtención de contenido *(get)* y asignaciones *(set)*, correspondientes al contenido de un registro de las columnas de la tabla `grupo` encapsuladas como objeto de tipo `Grupo`.

*Líneas 50-56*. Operaciones respectivas de obtención de contenido *(get)* y asignaciones *(set)*, correspondientes al contenido de un conjunto de registros de la tabla `alumno` que corresponden por su identificador a un registro de la tabla `grupo`.

---

```
1  package modelos;
2
3  import java.util.HashSet;
4  import java.util.Set;
5
6  /**
7   * Las fuentes java correspondientes a los beans de la tabla Grupo.
8   * @author Palacios Trejo Armando ciencias UNAM
9   */
10 public class Grupo implements java.io.Serializable {
11
12  /**
13   * Se debe cambiar cada vez que esta clase no es compatible con versiones
14   * anteriores.
15   */
16  private static final long serialVersionUID = 7971183818721745130L;
17  private int idGrupo;
18  private String nombre;
19  private Set<Alumno> alumnos = new HashSet<Alumno>(0);
20
21  public Grupo() {
22  }
23
24  public Grupo(int idGrupo) {
25   this.idGrupo = idGrupo;
26  }
27
28  public Grupo(int idGrupo, String nombre, Set<Alumno> alumnos) {
29   this.idGrupo = idGrupo;
30   this.nombre = nombre;
31   this.alumnos = alumnos;
32  }
33
34  public int getIdGrupo() {
35   return this.idGrupo;
36  }
```

```
37
38   public void setIdGrupo(int idGrupo) {
39    this.idGrupo = idGrupo;
40   }
41
42   public String getNombre() {
43    return this.nombre;
44   }
45
46   public void setNombre(String nombre) {
47    this.nombre = nombre;
48   }
49
50   public Set<Alumno> getAlumnos() {
51    return this.alumnos;
52   }
53
54   public void setAlumnos(Set<Alumno> alumnos) {
55    this.alumnos = alumnos;
56   }
57
58 }
```

Cuadro 5.18.: *Código del archivo Grupo.java.*

---

En el Cuadro 5.19 se muestra el código de la clase que se encarga de las operaciones para listar los grupos y los alumnos que corresponden a un grupo mediante sus identificadores.

*Líneas 33-51.* Se realiza la transacción para listar todos los grupos existentes en la tabla `grupo` utilizando la clase `UserTransaction` que permite controlar el límite de las transacciones y asociar cada transacción a un hilo de ejecución. Antes de hacer encomendar una transacción *("commit")* se obtiene una sesión de Hibernate para hacer una consulta de tipo *HQL "from Grupo"* y listar los grupos de la base de datos regresandolos como una lista a la vista de la aplicación *"view.jsp"* mediante una petición de redibujado *("RenderRequest req").*

*Líneas 59-74.* Se realiza la transacción para listar todos los alumnos que pertenecen a un grupo mediante el identificador del grupo. Se obtiene una sesión de Hibernate para obtener un objeto de tipo `Grupo` que representa a un registro de la tabla `grupo` mediante su identificador representado por el entero `idGrupo`. Al haber obtenido el grupo, se obtiene el conjunto de alumnos que corresponden a este, y se mandan a la vista como un arreglo de tipo *JSON.*

*Líneas 50-56.* Operaciones respectivas de obtención de contenido *(get)* y asignaciones *(set)*, correspondientes al contenido de un conjunto de registros de la tabla `alumno` que corresponden por su identificador a un registro de la tabla `grupo`.

*Líneas 92-120.* Método que no se utiliza en el proyecto pero que está como ejemplo para insertar registros a una tabla. Para esto se tiene que obtener una sesión de Hibernate

y haber iniciado las transacciones correspondientes con la clase `UserTransaction`. A continuación se obtiene el máximo identificador de los registros de la tabla `alumno` con la sentencia *HQL "select max(id.idAlumno) from Alumno"* la cual regresa una lista con el número de identificador, se crea un nuevo objeto de tipo `Alumno` y por último para insertar el nuevo alumno a la base se utiliza el método `session.save(alumno)`, como ejemplo se muestra un molde *("cast en Java")* del resultado de la inserción el cual me regresa un objeto de tipo `AlumnoId`. Al final, se realiza la transacción cerrando la sesión de Hibernate mediante `session.flush()` y `session.close()`.

```
1  package modelos;
2
3  import java.io.Writer;
4  import java.util.List;
5  import java.util.Iterator;
6  import java.util.Set;
7
8  import javax.naming.InitialContext;
9  import javax.portlet.RenderRequest;
10 import javax.portlet.RenderResponse;
11 import javax.transaction.UserTransaction;
12
13 import org.hibernate.Session;
14
15 import com.liferay.portal.kernel.json.JSONArray;
16 import com.liferay.portal.kernel.json.JSONFactoryUtil;
17 import com.liferay.portal.kernel.json.JSONObject;
18
19 import modelos.util.HibernateUtil;
20
21 /**
22  * Clase que resuelve las consultas.
23  * @author Palacios Trejo Armando ciencias UNAM
24  */
25 public class Modelo {
26
27  /**
28   * Obtiene los resultados de la consulta que lista los grupos.
29   * @param req Para mandar la lista de los grupos.
30   * @return RenderRequest Los datos de la consulta.
31   * @throws Exception
32   */
33  public static RenderRequest listGrupo(RenderRequest req)
34  throws Exception {
35   UserTransaction transaction = null;
36   try {
37    transaction = (UserTransaction) new InitialContext()
38    .lookup("java:comp/UserTransaction");
39    transaction.setTransactionTimeout(60);
40    transaction.begin();
41    Session session = HibernateUtil.getSession();
42    List<?> grupos = session.createQuery("from Grupo").list();
```

```
43      req.setAttribute("listaGrupos", grupos);
44      transaction.commit();
45      session.close();
46     } catch (Exception e) {
47      transaction.rollback();
48      throw e;
49     }
50     return req;
51    }
52
53    /**
54     * Obtiene los alumnos de un grupo por su identificador.
55     * @param idGrupo Identificador del grupo.
56     * @return JSONArray Con los alumnos pertenecientes al grupo.
57     * @throws Exception
58     */
59    public static JSONArray jsonArrayAG(int idGrupo)
60    throws Exception{
61     JSONArray resultado = JSONFactoryUtil.createJSONArray();
62     UserTransaction transaction = null;
63     try {
64      transaction = (UserTransaction) new InitialContext()
65      .lookup("java:comp/UserTransaction");
66      transaction.setTransactionTimeout(60);
67      transaction.begin();
68      Session session = HibernateUtil.getSession();
69      Grupo grupo = (Grupo) session.get(Grupo.class, idGrupo);
70      Set<Alumno> alumnos = grupo.getAlumnos();
71      for( Iterator<Alumno> it = alumnos.iterator(); it.hasNext();) {
72       String nombre = (String)it.next().getNombre();
73       resultado.put(nombre);
74      }
75      transaction.commit();
76      session.close();
77     } catch (Exception e) {
78      transaction.rollback();
79      throw e;
80     }
81     return resultado;
82    }
83
84    /**
85     * Crea un nuevo alumno. Metodo no usado en el proyecto.
86     * @param grupoId Identificador del grupo al que serÃ¡ asignado.
87     * @param alumnoNom Nombre del alumno.
88     * @param escuela Nombre de la escuela.
89     * @return Integer con el id del alumno.
90     * @throws Exception
91     */
92    public static Integer insertaAlumno(int grupoId, String alumnoNom,
93      String escuela) throws Exception {
94     UserTransaction transaction = null;
95     Integer alumnoId = new Integer(0);
96     try {
```

```
97      transaction = (UserTransaction) new InitialContext()
98      .lookup("java:comp/UserTransaction");
99      transaction.setTransactionTimeout(60);
100     transaction.begin();
101     Session session = HibernateUtil.getSession();
102     Alumno alumno = new Alumno();
103     String hql= "select max(id.idAlumno) from Alumno";
104     List<?> list = session.createQuery(hql).list();
105     int maxId = ( (Integer)list.get(0) ).intValue();
106     AlumnoId id = new AlumnoId(maxId + 1, grupoId);
107     alumno.setId(id);
108     alumno.setNombre(alumnoNom);
109     alumno.setEscuela(escuela);
110     id = (AlumnoId) session.save(alumno);
111     alumnoId = id.getIdAlumno();
112     session.flush();
113     transaction.commit();
114     session.close();
115   } catch (Exception e) {
116     transaction.rollback();
117     throw e;
118   }
119   return alumnoId;
120 }
121
122 /**
123  * Manda el flujo de datos de los alumnos pertenecientes al grupo
124  * seleccionado en la vista.
125  * @param req El objeto request.
126  * @param resp El objeto response.
127  * @throws Exception
128  */
129 public static void muestraAlumnos(RenderRequest req, RenderResponse resp)
130 throws Exception{
131   int indice = Integer.parseInt(req.getParameter("idGrupo"));
132   JSONArray resultados = Modelo.jsonArrayAG(indice);
133   //El flujo de datos a mandar es un json.
134   resp.setContentType("application/x-json");
135   Writer out = resp.getWriter();
136   JSONObject obj = JSONFactoryUtil.createJSONObject();
137   obj.put("resultados", resultados);
138   out.write(obj.toString());
139 }
140
141 }
```

Cuadro 5.19.: *Código del archivo Modelo.java.*

### 5.4.5. Extendiendo la funcionalidad del portlet para hacer consultas en la base de datos de Liferay

Se mostrará mediante un ejemplo el uso de Hibernate para realizar consultas a la base de datos de Liferay, utilizando clases nativas del propio Liferay[8].

Ejemplo:
Creamos una clase llamada `ModeloLiferay` en el paquete `modelos`, la cual podemos se puede probar en la clase `modelos.Controlador`.

Se ejemplificará una unión *("join")* entre tablas de la base de Liferay, en este ejemplo se hará una unión entre la tabla `user_` y `contact_` referenciados por el identificador "contactId" que primero se obtendrá de la tabla `user_`, imprimirá en el registro *("log")* de *Glassfish* el nombre y apellido que proviene de la tabla `contact_` de la base de datos *"lportal"* de *Liferay*, correspondiente a los usuarios cuyo nombre sea *"Armando"* y apellido contenga una *"c"*. Esto para ejemplificar el uso de las clases nativas de Liferay para realizar consultas. Por ejemplo al tener los usuarios *"Armando Archundia", "Armando Palacios"* y *"Armando Lopez"* en el sistema, se muestra lo siguiente:

```
[#|2011-04-21T03:24:40.092+0000|INFO|glassfish3.0.1|javax.enterprise.system
.std.com.sun.enterprise.v3.services.impl|_ThreadID=37;_ThreadName=Thread-1;|
usuario: Armando-Archundia|#]
[#|2011-04-21T03:24:40.093+0000|INFO|glassfish3.0.1|javax.enterprise.system
.std.com.sun.enterprise.v3.services.impl|_ThreadID=37;_ThreadName=Thread-1;|
usuario: Armando-Palacios|#]
```

En el Cuadro 5.20 se muestra el código de la clase que muestra como ejemplo el uso de las clases nativas de Liferay para hacer consultas.

*Líneas 40-44.* Primero se carga la clase `User` que mapea la tabla `user_` de la base de datos *"lportal"*, haciendo proyecciones sobre las columnas *"contactId" y "firstName"* restringiendo la proyección de la columna *"firstName"* a que tenga el valor *"Armando"*, los identificadores y nombres del resultado se guardan en el objeto `dynamicQuery0`.

*Líneas 48-52.* Como la tabla `user_` hace referencia a la tabla `contact_` mediante el identificador *"contactId"*, se hace una unión entre los identificadores del objeto `dynamicQuery0` y la tabla `contact_` representada por la clase `Contact` que almacena los registros cuyos apellidos contengan una *"c"* al objeto `dynamicQuery1`, ordenando los registros por apellido de forma ascendente.

*Líneas 53-65.* Se obtiene la lista de objetos de tipo `Contact` que corresponden al objeto `dynamicQuery1`, que representan los registros de la tabla `contact_` de la base de datos, y por último se imprime en el registro de Glassfish mediante el método `info` de la clase

[8] Nunca utilizar las clases que contiene `portal-impl.jar` ya que es el núcleo de todo el portal. Por ejemplo, la mayoría de los programadores requieren utilizar estas clases para realizar consultas, sin embargo el sistema mostrará que no es necesario, ya que el paquete `portal-service.jar` contiene las clases necesarias para realizar consultas [62].

Log.

```
1  package modelos;
2
3  import java.util.List;
4
5  import org.apache.commons.logging.Log;
6  import org.apache.commons.logging.LogFactory;
7
8  import com.liferay.portal.kernel.dao.orm.DynamicQuery;
9  import com.liferay.portal.kernel.dao.orm.DynamicQueryFactoryUtil;
10 import com.liferay.portal.kernel.dao.orm.OrderFactoryUtil;
11 import com.liferay.portal.kernel.dao.orm.ProjectionFactoryUtil;
12 import com.liferay.portal.kernel.dao.orm.PropertyFactoryUtil;
13 import com.liferay.portal.kernel.dao.orm.RestrictionsFactoryUtil;
14 import com.liferay.portal.kernel.exception.SystemException;
15 import com.liferay.portal.kernel.util.PortalClassLoaderUtil;
16 import com.liferay.portal.model.Contact;
17 import com.liferay.portal.model.User;
18 import com.liferay.portal.service.UserLocalService;
19 import com.liferay.portal.service.UserLocalServiceUtil;
20
21 /**
22  * Clase que muestra como usar el Hibernate de Liferay para hacer consultas
        a
23  * la base de datos del mismo.
24  * @author Palacios Trejo Armando ciencias UNAM
25  */
26 public class ModeloLiferay {
27  /** Logger. */
28  private static Log log = LogFactory.getLog(ModeloLiferay.class);
29
30  /**
31   * Metodo que realiza un join entre la tabla User_ y Contact_.
32   */
33  public static void imprimeJoin() {
34   /*ClassLoader asegura que la clase de implementacion sea agregada,
35    * en este caso UserImpl y ContactImpl.*/
36   ClassLoader classLoader = PortalClassLoaderUtil.getClassLoader();
37   UserLocalService service = UserLocalServiceUtil.getService();
38   /*Devuelve solamente los valores de los identificadores de los cuales
39    * el valor firstName es 'Armando' por ejemplo [12004,12013,12022]*/
40   DynamicQuery dynamicQuery0 =
41    DynamicQueryFactoryUtil.forClass(User.class, classLoader)
42    .setProjection(ProjectionFactoryUtil.property("contactId"))
43    .add(PropertyFactoryUtil.forName("firstName")
44      .eq(new String("Armando")));
45   /*A continuacion se realiza el join con la tabla Contact con la
46    * restriccion que los resultados contengan en middleName una c
47    *  y ordena el resultado de forma ascendente. */
48   DynamicQuery dynamicQuery1 =
49    DynamicQueryFactoryUtil.forClass(Contact.class, classLoader)
50    .add(PropertyFactoryUtil.forName("contactId").in(dynamicQuery0))
```

```
51      .add(RestrictionsFactoryUtil.ilike("lastName", "%c%"))
52      .addOrder(OrderFactoryUtil.asc("lastName"));
53    try {
54     @SuppressWarnings("unchecked")
55     List<Contact> contactList =
56      (List<Contact>)service.dynamicQuery(dynamicQuery1);
57     for (int i=0; i< contactList.size(); i++)
58     {
59      log.info( "\n usuario: " + contactList.get(i).getFirstName()
60        + "-" + contactList.get(i).getLastName());
61     }
62    } catch (SystemException e) {
63     log.error(e.getMessage());
64    }
65   }
66
67  }
```

Cuadro 5.20.: *Código del archivo ModeloLiferay.java.*

---

En el Capítulo 5 se han desarrollado ejemplos haciendo uso de un portlet avanzado, con el objetivo de poder realizar un proyecto mas complejo que lleve las tecnologías antes mencionadas.

Se ha visto como crear diferentes bases de datos aparte de la que trae el portal de Liferay para que estas interactúen con los portlets, además se ha mostrado el uso de la biblioteca de JavaScript llamada JQuery la cual se puede utilizar para implementar AJAX en nuestro proyecto.

A continuación se propondrá un proyecto que lleve estas tecnologías, así como también las ventajas que ofrecen el portal y las bibliotecas de Liferay para administrar a los usuarios del sistema con respecto a los portlets de los cuales se restringe su acceso y hasta su comportamiento.

# 6. Proyecto: Desarrollo del sistema de gráficas dinámicas

El objetivo de este capítulo es desarrollar un portlet complejo como ejemplo para hacer uso de las tecnologías descritas anteriormente, este portlet es un sistema que muestra gráficas de porcentajes actualizadas y animadas dinámicamente cada vez que se actualizan los datos dentro de una base de datos así como ciertas restricciones a usuarios que acceden al sistema.

## 6.1. Objetivo

Simularemos una empresa compuesta de tres departamentos *"departamento 1"*, *"departamento 2"* y *"departamento 3"* de los cuales cada departamento genera ganancias para solventar toda la empresa que los contiene. A la empresa le interesa un sistema que muestre gráficas con la suma de ganancias de cada mes y la suma de pérdidas respecto al mes anterior, así como las gráficas de ganancias y pérdidas anuales de cada departamento por separado o en conjunto en el caso de algunas personas con ciertos privilegios, con el propósito de analizar sus ingresos para resolver los problemas de cada uno en caso que haya pérdidas. Las personas de cada departamento solamente podrán ver las gráficas que corresponden a su departamento, pero el administrador del sistema y algunas personas con ciertos privilegios podrán ver las gráficas correspondientes a todos los departamentos, así como de las ganancias y pérdidas totales (de todos los departamentos en una gráfica), con el fin de generar seguridad para que las personas de cada uno de los departamentos no puedan saber las ganancias de los otros departamentos ajenos al suyo.

Las gráficas tendrán que tener buena presentación, y se podrá escoger la manera en que se mostrarán. Cada vez que se selecciona una opción diferente, las gráficas se actualizarán al momento de seleccionar esa opción, además de que no se recargará la página.

Antes de comenzar a desarrollar el proyecto se dará una introducción a la tecnología de *"Open Flash Chart"* que se utilizará para generar las gráficas que mostrará el sistema.

## 6.2. Introducción a Open Flash Chart 2

Esta biblioteca es software libre compatible con lenguajes de programación de parte del servidor usados para desarrollar sistemas Web como Java y que nos permite crear diferentes tipos de gráficas en Flash que utilizaremos para generar gráficas en nuestro

proyecto, entre las opciones que nos ofrece tenemos gráficas de barras, de líneas, de tortas *(pies)*, etc. Es posible generar gráficas dinámicas con ***Open Flash Chart 2*** *("OFC2")* de forma profesional, simple y con muy buena vista [1].

Entre las principales características encontramos:

- Podemos ver la información en cada punto de la gráfica.
- Podemos exportar gráficas como imágenes.
- Podemos escoger diferentes tipos de gráficas: líneas, barras, barras 3D, barras en pay, de áreas y más.
- Existen varias bibliotecas disponibles para PHP, Java, Python, etc.
- Podemos cargar los datos de las gráficas en formato *JSON* definido en el apartado 5.4.3.

Para utilizar OFC2 tenemos que cargar a nuestro proyecto la biblioteca `swfobject.js` así como los componentes con extensión *.swf* que son `expressInstall.swf` y `open-flash-chart.swf`[2].

Una vez cargadas las bibliotecas, es necesario crear un documento que plasme alguna gráfica. Por ejemplo, las funciones que nos pueden ayudar a visualizar una gráfica de barras 3D podrían ser las las mostradas en el código de tipo JavaScript con extensión .js del Cuadro 6.1.

*Líneas 4-7.* Se crea un método llamado `getDatosGrafica()`, el cual regresa un objeto *JSON* que el objeto de tipo *Flash SWF* acepta como parámetro utilizando la función `swfobject.embedSWF()`.

*Líneas 12-13.* La función `swfobject.embedSWF()` se encuentra dentro de la función `ready()` de JQuery para ejecutarse al terminar de cargar la página, recibe la ruta en donde se encuentra el componente `open-flash-chart.swf` en la variable `Pelicula.pelicula` y la ruta de `expressInstall.swf` en la variable `Pelicula.expressInstall` las cuales se deben definir *(en el ejemplo no se muestra)*, *"divGrafica2"* es el * * de un archivo con extensión *.jsp* donde se plasmará la gráfica, *"500"* y *"400"* son la altura y anchura respectivamente. *9.0.0* es la versión de *Flash*, *'get-data'* recibe una función que regresa el objeto JSON en una cadena.

Se utiliza `expressInstall.swf` para tener incitar al usuario a actualizar su reproductor si la versión que ha instalado no es suficiente para mostrar la gráfica. Para no mencionar al usuario que actualice su reproductor *Flash*, se debe poner `false` en lugar de `Pelicula.expressInstall`.

---

[1] El sitio oficial de *Open Flash Chart 2* cuenta con excelente información y varios tutoriales que nos ayudarán en futúros desarrollos *http://teethgrinder.co.uk/open-flash-chart-2/*

[2] Componentes de Open Flash Chart que se deben descargar de la página del proyecto *http://teethgrinder.co.uk/open-flash-chart-2/* y de *http://code.google.com/p/swfobject/downloads/list* respectivamente

```
1  /**  * Regresa los datos para generar la gráfica inicialmente.
2  * @return {String} String que contiene el objeto JSON con la gráfica.
3  */
4  function getDatosGrafica() {
5   jsonActual = { "elements": [ { "type": "bar_3d", "values": [ 5, 6, 9, 8,
       9, 3, 6, 7, 8, { "top": 5, "colour": "#900000", "tip": "Hello #val#" }
        ], "colour": "#D54C78" } ], "title": { "text": "Mon May 02 2011" }, "
       x_axis": { "3d": 5, "colour": "#909090", "labels": [ 1, 2, 3, 4, 5, 6,
        7, 8, 9, 10 ] } };
6   return JSON.stringify(jsonActual);
7  };
8
9  // Se ejecuta cuando la página termina de cargar
10 $(document).ready(function() {
11  // Agrega Open Flash Chart a la página
12    swfobject.embedSWF(
13  Pelicula.pelicula, "divGrafica2",  "550", "400", "9.0.0", Pelicula.
       expressInstall,  {'get-data':'getDatosGrafica','loading':'Cargando...'
       },  {'wmode':'opaque'});
14 });
```

Cuadro 6.1.: *Ejemplo de código para mostrar una gráfica.* La gráfica se muestra dentro en un campo * * con identificador *"divGrafica2"* de un archivo con extensión *.jsp*.

Con esto podremos observar una gráfica como la mostrada en la Figura 6.1.

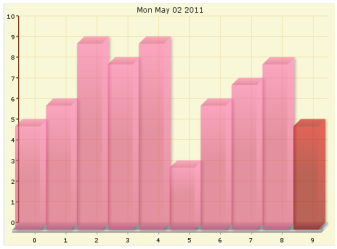


Figura 6.1.: *Gráfica generada con la tecnología de Open Flash Chart 2.*

Cabe mencionar que dado que recibe la biblioteca de *Open Flash Chart 2 "swfobject"* un objeto *JSON*, entonces si lo deseamos podremos crear nuestras propias bibliotecas en Java que creen objetos *JSON* para generar las gráficas como deseamos. Aunque se puede hacer uso de una biblioteca ya creada para generar estos objetos *JSON* llamada *JOFC2* o *Java API for Open Flash Chart 2*[3].

[3]La documentación de la biblioteca para generar objetos *JSON* para Java se encuentra en la página *http://code.google.com/p/jofc2/*. Existe una biblioteca más completa de la misma, que se encuentra

## 6.3. Arquitectura y diseño del proyecto

En este apartado se muestra la arquitectura y diseño del proyecto, ver la Figura 6.2, se muestra en la figura que la arquitectura del proyecto está conformada por un solo portlet que tomará en cuenta las características del usuario en sesión y utilizará las tecnologías Web mencionadas en los portlets ya desarrollados.

A continuación se describe la arquitectura del portlet:

- En la vista del portlet representada en archivos con extensión *.jsp* se hace uso de JQuery para que dependiendo del tipo de gráfica que se ha solicitado, generar la gráfica con *Flash/Open Flash Chart 2* con el objetivo de mostrarla sin recargar la página gracias al uso de *AJAX*. La tecnología *JSTL* se utiliza para listar los departamentos existentes en la base de datos.
- Las clases de Java generan objetos *JSON,* los cuales se envían a la vista como servicios Web utilizando la biblioteca *JOFC2*, estos objetos son recibidos por la tecnología de *Open Flash Chart* para generar la gráfica en la vista dependiendo de los valores que contenga.
- Se consulta la base de datos que llamaremos *"empresa"* la cual guarda las ganancias de cada departamento, también se consulta a la base de datos del portal *"lportal"* para saber las características del usuario que se encuentra consultando el portlet y lo que se le debe mostrar. Las consultas se hacen utilizando Hibernate.

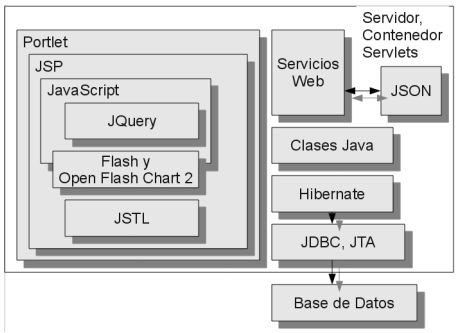


Figura 6.2.: *Arquitectura del proyecto.*

### 6.3.1. Modelo de casos de uso y vistas del proyecto

En este apartado se muestra el modelo de casos de uso del proyecto, el cual muestra las interacciones de los usuarios de cada departamento con respecto al portlet para ayudar a

en *https://github.com/MMeldrum/jofc2/blob/master/dist/jofc2-1.0-1.jar*. Para que esta biblioteca no de errores en nuestro proyecto hacemos uso de la biblioteca `xstream.jar`. Para esto debemos escribir la biblioteca en el archivo `liferay-plugin-package.properties` tal como se hizo en el Capítulo 5.

describir qué es lo que el sistema debe hacer, ver la Figura 6.3. También se mostrarán las vistas o imágenes del portlet que corresponden a cada usuario del sistema dependiendo del departamento al que pertenece.

En la Figura 6.3 se muestra 4 tipos de usuario que acceden al mismo portlet. Los que pertenecen a cada uno de los departamentos *(departamento 1, departamento 2 y departamento 3)*, los cuales podrán visualizar las gráficas que corresponden solo al departamento que pertenecen y escoger las opciones correspondientes a estas tales como rango de ganancias o perdidas mensuales o anuales y el usuario con privilegios de poder observar las gráficas correspondientes a cada uno de los departamentos así como poder visualizar la gráfica con las ganancias y pérdidas de la suma total de los departamentos.

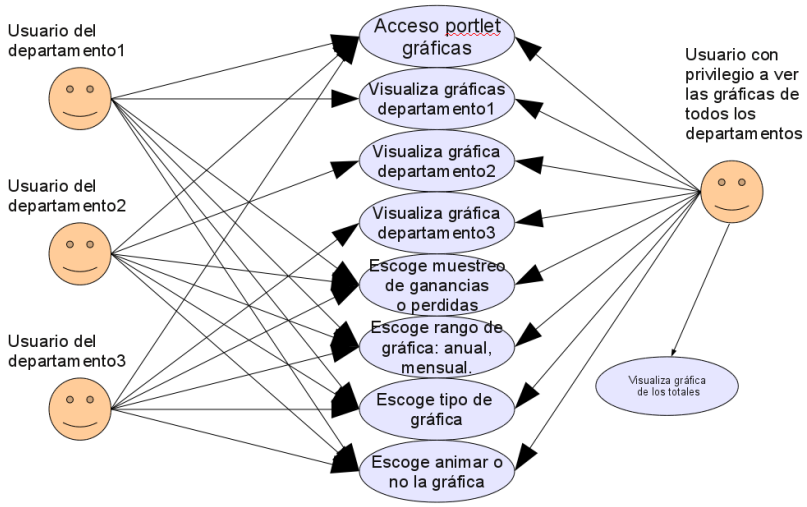


Figura 6.3.: *Modelo de casos de uso.* Se muestra 4 tipos de usuario que acceden al mismo portlet.

Supongamos que somos un usuario con todos los privilegios, de esta forma tenemos acceso a las gráficas de todos los departamentos, por lo que podremos escoger de que departamento deseamos ver sus ganancias y/o perdidas, para ello seleccionamos el botón de tipo radio cuyo nombre es el departamento de la gráfica de ganancias y pérdidas que queremos visualizar, ver la Figura 6.4. En este ejemplo hemos escogido que se muestre la gráfica de las ganancias y pérdidas de todos los departamentos en total al seleccionar el botón *"Totales"*, también hemos definido que se muestren las ganancias y pérdidas al seleccionar *"Ganancias"* y *"Pérdidas"* y el rango de ganancias a mostrar *("Anual")*, de la lista de tipos de gráficas hemos seleccionado el tipo *"Normal"* las cuales son planas, y por último hemos seleccionado que las gráficas se muestren sin animación al seleccionar el botón *"Estático"*.

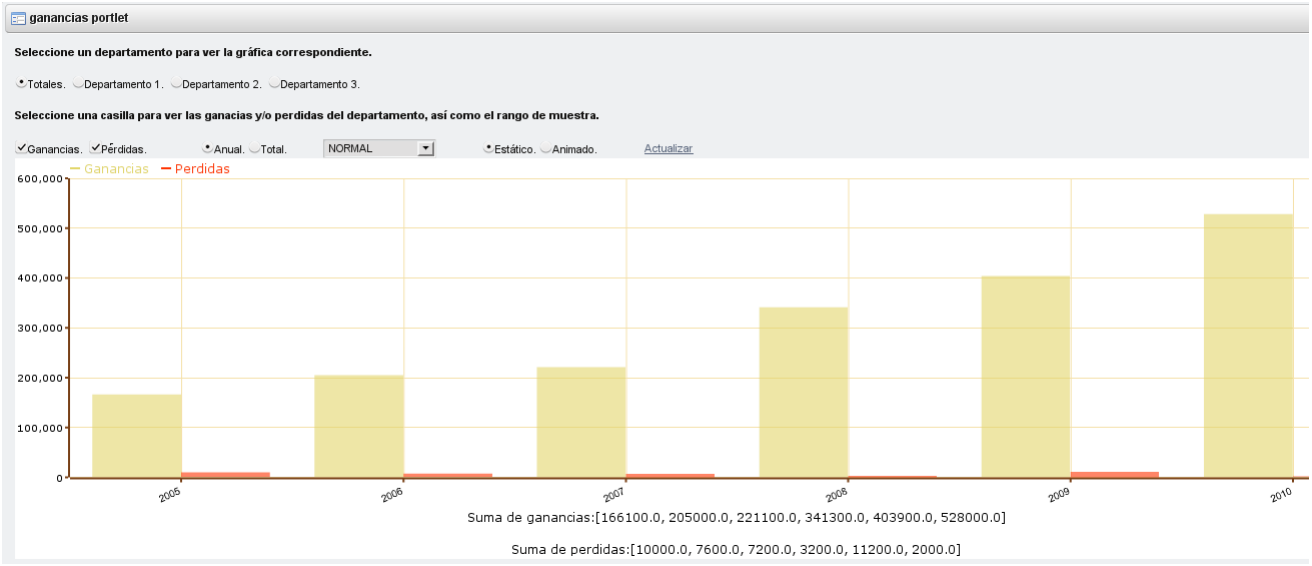


Figura 6.4.: *Gráfica mostrada a un usuario con privilegios de visualizar todas las gráficas.*

Las características a seleccionar para visualizar la gráfica son:

- Gráfica de barras correspondientes a ganancias y/o pérdidas.
- Rango de fechas: anual o total.
- Tipo de gráfica de barras, por ejemplo: NORMAL, VIDRIOSA, 3D, DOMO, etc.
- Gráficas estáticas o animadas.
- Habrá una opción de actualizar si la gráfica no se muestra correctamente.

Otro ejemplo para visualizar una gráfica al ser un usuario con privilegios se muestra en la Figura 6.5. En este ejemplo hemos escogido que se muestre la gráfica de las ganancias y pérdidas del departamento1 al seleccionar el botón *"Departamento 1"*, también hemos definido que se muestren las ganancias y pérdidas al seleccionar *"Ganancias"* y *"Pérdidas"*, a continuación hemos seleccionado el rango de ganancias a mostrar *(Anual)*, de la lista de tipos de gráficas hemos seleccionado el tipo *"Cilindro"* las cuales se visualizan en forma de cilindro, y por último hemos seleccionado que las gráficas se muestren sin animación al seleccionar el botón *"Estático"*.

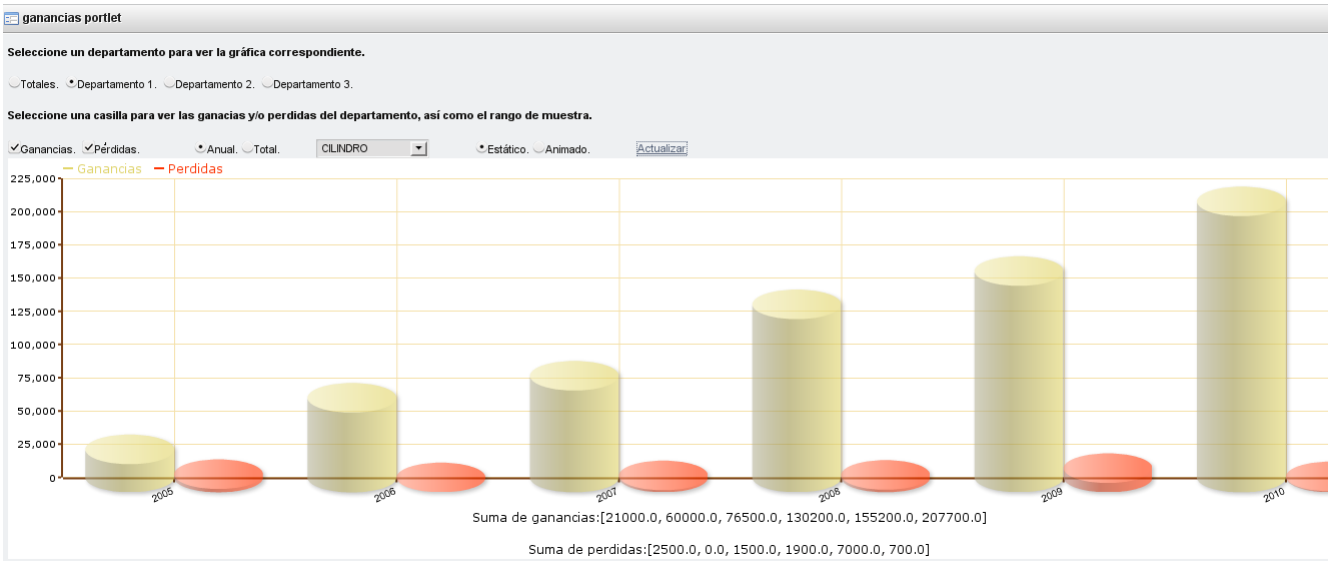


Figura 6.5.: *Gráfica mostrada a un usuario con privilegios de visualizar todas las gráficas.*

Ahora supongamos que somos un usuario del departamento 2, en este caso no podremos ver las gráficas correspondientes a los demás departamentos, por lo que visualizará una gráfica como la mostrada en la Figura 6.6, la cual muestra la gráfica de las ganancias y pérdidas correspondientes únicamente al departamento 2. En este ejemplo hemos definido que se muestren las ganancias y pérdidas al seleccionar *"Ganancias"* y *"Pérdidas"*, a continuación hemos seleccionado que el rango de ganancias a mostrar sea de cada mes al seleccionar el botón *"Total"*, de la lista de tipos de gráficas hemos seleccionado gráficas de tipo *"Domo vidrioso"* las cuales se visualizan en forma de domo, y por último hemos seleccionado que las gráficas se muestren sin animación al seleccionar el botón *"Estático"*.

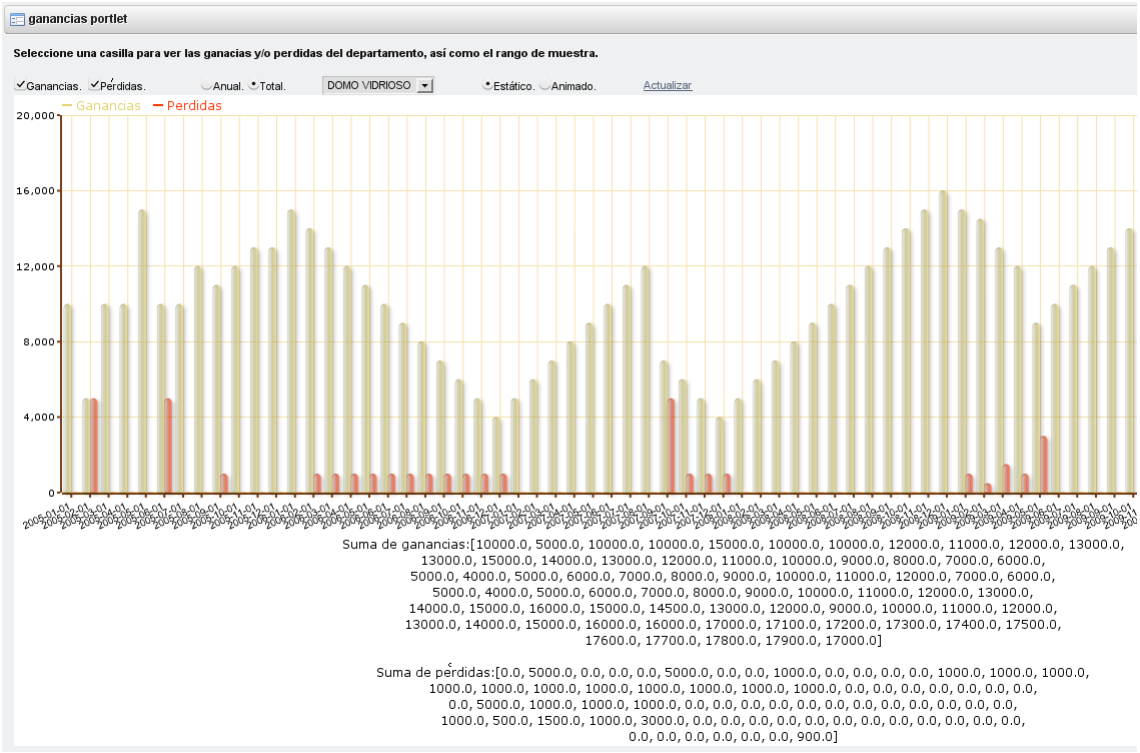


Figura 6.6.: *Gráfica mostrada a un usuario perteneciente al departamento 2.*

Ahora supongamos que somos un usuario del departamento 1, y queremos observar solamente las ganancias sin las pérdidas de nuestro departamento. Al seleccionar la segunda barra observaremos como se muestra el valor y se hace más notoria que las demás barras, ver la Figura 6.7. En este ejemplo hemos definido que se muestren solamente las ganancias al seleccionar *"Ganancias"* y *no "Pérdidas"*, a continuación hemos seleccionado que el rango de ganancias a mostrar sea anual, de la lista de tipos de gráficas hemos seleccionado gráficas de tipo *"Domo vidrioso"* las cuales se visualizan en forma de domo, y por último hemos seleccionado que las gráficas se muestren sin animación al seleccionar el botón *"Estático"*.

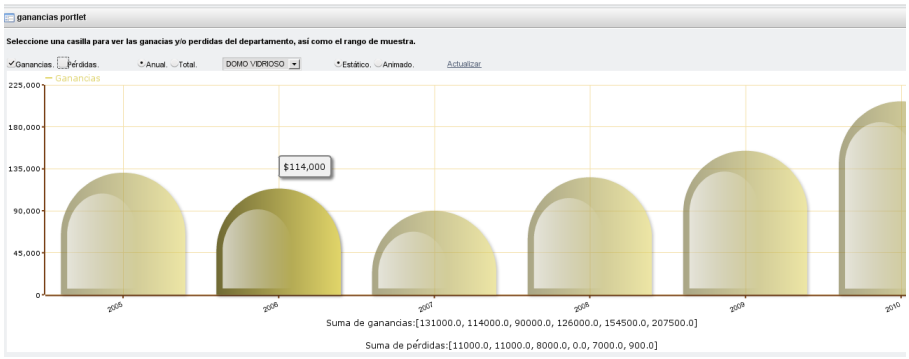


Figura 6.7.: *Gráfica mostrada a un usuario perteneciente al departamento 1.*

### 6.3.2. Modelo de base de datos del proyecto.

El modelo propuesto para este proyecto se muestra en el diagrama Entidad-Relación de la Figura 6.8, la base de datos se llama *"empresa"*. También se hará uso de la base de datos de Liferay para saber a que componentes del portlet tiene acceso cada usuario de sesión. A las tablas *"departamento1"*, *"departamento2"*, *"departamento3"* corresponden a las ganancias y pérdidas de los departamentos 1,2 y 3 respectivamente. La tabla `totales` corresponde a la suma de las ganancias y pérdidas de todos los departamentos en total. La tabla `fecha` corresponde a la fecha mensual de los cortes de de ganancias y pérdidas de la empresa.

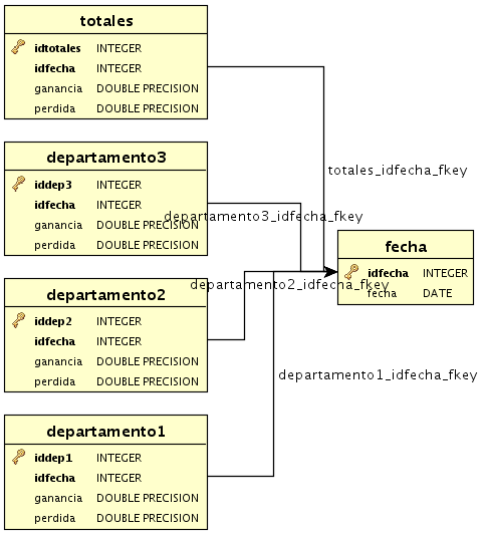


Figura 6.8.: *Diagrama ER de la base de datos "empresa".* Se muestran 4 tablas *"totales"*, *"departamento1"*, *"departamento2"*, *"departamento3"* y *"fecha"*.

## 6.4. Desarrollo simplificado del proyecto

La finalidad de este capítulo es mostrar y explicar con ejemplos el código más importante utilizado para la creación del proyecto, esto es el código para hacer consultas a la base de datos de Liferay utilizando las bibliotecas que tiene y restringir el acceso a los portlets dependiendo del tipo de usuario, además del código para mostrar las gráficas a las que tiene acceso un tipo de usuario.

En la Figura 6.9 se muestra el árbol correspondiente a los paquetes que contienen las clases Java utilizadas para realizar las consultas a la base de datos.

*Paquete modelos.* Contiene las clases que mapean las tablas correspondientes a la base de datos *"empresa"* las cuales son *"Departamento1", "Departamento2", "Departamento3", "Fecha"* y *"Totales"*. También se observan los archivos de configuración de Hibernate con extensión *.hbm.xml* que se encargan de mapear las clases a las tablas de la base de datos.

*Paquete modelos.util.* Contiene la clase *HibernateUtil* la cual interactúa con el archivo de configuración de *Hibernate* llamado `hibernate.cfg.xml` que define las opciones de configuración globales para la conexión a la base de datos, y también contiene la clase `Modelo.java` la cual se encarga de hacer la consulta a la base de datos con ayuda de las clases antes mencionadas. El archivo `hibernate.reveng.xml` es el que utiliza la herramienta *HibernateTools* para generar las clases que mapean las tablas de la base de datos y los archivos de configuración de Hibernate.

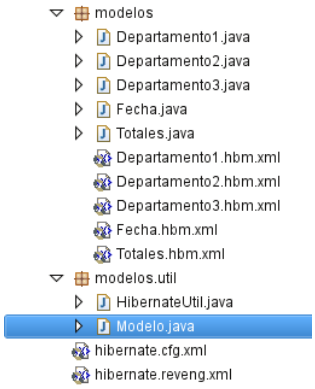


Figura 6.9.: *Árbol de paquetes que contiene las clases Java utilizadas en el proyecto, así como los archivos de configuración de Hibernate.*

El código *Java* del Cuadro 6.2, es un ejemplo para obtener todas las fechas de la base de datos *"empresa"* que es necesario cuando el usuario requiere graficar todas las fechas de ganancias y pérdidas que se han ingresado a la base de datos. Este ejemplo de código se puede colocar dentro de un método que regresa una lista de fechas.

*Líneas 4-11.* Se inicializa una transacción mediante el contexto de la configuración de Hibernate como se vió en el Capítulo 5, dando un tiempo límite de 60 segundos para realizar la transacción. Por último se obtiene una lista de fechas abriendo una sesión de

Hibernate y ejecutando una consulta a la base de datos mediante *HQL* explicado en el Capítulo 5 y se cierra la sesión realizando la transacción que devuelve las fechas.

*Línea 16.* Se regresa una lista en forma de arreglo que contiene todas las fechas.

```
1  UserTransaction transaction = null;
2  List<Date> fechas =  null;
3  try {
4   transaction = (UserTransaction) new InitialContext().lookup("java:comp/
       UserTransaction");
5   transaction.setTransactionTimeout(60);
6     transaction.begin();
7   Session session = HibernateUtil.getSession();
8    fechas = new ArrayList<Date>();
9     fechas = session.createQuery("select fecha from Fecha").list();
10    session.close();
11    transaction.commit();
12 } catch (Exception e) {
13  transaction.rollback();
14  throw e;
15 }
16 return fechas;
```

Cuadro 6.2.: *Ejemplo para obtener todas las fechas de la base de datos "empresa".*

El código Java del Cuadro 6.3, es un ejemplo para obtener todos los años y con ello las ganancias y pérdidas por año, las ganancias se meten en una lista y las pérdidas en otra, después estas listas se almacenan en una sola lista.

*Línea 11.* Se obtiene una lista de años de un determinado rango *(el método `listAnios()` no se muestra ya que se programar de forma similar al código anterior).*

*Líneas 17-18.* Se obtiene un rango de fechas por cada año para abarcar todos los días de ese año.

*Líneas 23-24.* Se hace una proyección a la suma de la columna "ganancia".

*Línea 25.* El método `critDepartamento()` encapsula al método `session.createCriteria()` cuyo parámetro departamento será la clase que representa un departamento la cual se utilizará para hace consultas a su tabla correspondiente.

*Líneas 26-29.* Se obtiene la suma las ganancias de la tabla de un departamento haciendo una unión *("join")* a la tabla `fecha`, restringiendo las fechas solamente del año al que se ha iterado.

*Líneas 33-39.* Se obtiene la suma de las pérdidas de la tabla de un departamento haciendo una unión *("join")* a la tabla `fecha`, restringiendo las fechas solamente del año al que se ha iterado.

*Líneas 43-44.* Las listas de ganancias y pérdidas se guardan en la lista llamada "valores" para mas adelante usar esta lista para graficar las ganancias y pérdidas.

```
1  UserTransaction transaction = null;
2  ArrayList<ArrayList<Number>> valores =
3      new ArrayList<ArrayList<Number>>();
4  try {
5   transaction =
6    (UserTransaction) new InitialContext().lookup("java:comp/UserTransaction"
         );
7   transaction.setTransactionTimeout(60);
8   DateFormat format = new SimpleDateFormat("yyyy-MM-dd");
9   Date d1 = new Date();
10  Date d2 = new Date();
11  ArrayList<String> anios = listAnios();
12  ArrayList<Number> ganancias = new ArrayList<Number>();
13  ArrayList<Number> perdidas = new ArrayList<Number>();
14     String anio = new String();
15  for( Iterator<String> it = anios.iterator(); it.hasNext();) {
16   anio = it.next();
17   d1 = (Date) format.parse(anio + "-01-01");
18   d2 = (Date) format.parse(anio + "-12-31");
19    Criteria crit = null;
20    transaction.begin(); //JTA crea hilos de ejecución por cada consulta.
21
22    Session session = HibernateUtil.getSession();
23    ProjectionList proList = Projections.projectionList();
24    proList.add(Projections.sum("ganancia"));
25    crit = critDepartamento(departamento, session); //Con que clase modelada
          haré la consulta. Por ejemplo puede devolverme un criterio que me
          devuelve session.createCriteria(Departamento1.class);
26    crit.setProjection(proList)// Suma los resultados de la columna ganancia.
27    .createCriteria("fecha", "idfecha")// Crea la unión (join).
28    .add(Restrictions.between("fecha", d1, d2));// Da la restriccion
29    ganancias.add((Double) crit.list().get(0));// Da la suma sin [].
30    session.close();
31
32    Session session2 = HibernateUtil.getSession();
33    proList = Projections.projectionList();
34    proList.add(Projections.sum("perdida"));
35    crit = critDepartamento(departamento, session2); //Con que clase modelada
          haré la consulta.
36    crit.setProjection(proList)// Suma los resultados de la columna perdida.
37    .createCriteria("fecha", "idfecha")// Crea el join.
38    .add(Restrictions.between("fecha", d1, d2));// Da la restriccion
39    perdidas.add((Double) crit.list().get(0));// Da la suma sin [].
40    session2.close();
41    transaction.commit();
42  }
43 valores.add(ganancias);
44 valores.add(perdidas);
45 } catch (Exception e) {
46 transaction.rollback();
47  throw e;
48 }
```

Cuadro 6.3.: Ejemplo de código para obtener todos los años y con ello las ganancias y pérdidas por año

Se utilizó la clase Criteria para hacer búsquedas con restricciones como se puede observar en las líneas 25 a 29, también se utilizó la ventaja que nos ofrece JTA ya que se realizaron dos consultas al mismo tiempo al crear hilos de ejecución por cada consulta.

El código *Java* del Cuadro 6.4, es un ejemplo que nos dará los identificadores de los grupos que pertenecen al usuario que ha accedido al sistema y se encuentra en sesión. Para esto se pueden crear dichas consultas a la base de Liferay dentro del método doView de la clase Controlador utilizando las clases de Liferay que mapean las tablas de la base de datos *"lportal"*.

*Línea 4.* Mediante el método getRemoteUser() obtenemos el identificador del usuario en sesión.

*Línea 7.* Se obtienen los datos del usuario de la tabla `User` mediante su identificador.

*Líneas 8-9.* Se obtiene el identificador del rol del usuario, al ver en la base de datos la tabla `role_` resulta que el rol administrador corresponde al identificador *"10137"*.

*Líneas 11-19.* Si el usuario en sesión no es un administrador, se obtienen todos los identificadores de los grupos a los que el usuario pertenece. Si el identificador es *"13228"* entonces significa que el usuario pertenece al departamento 1, si es *"13232"* quiere decir que el usuario pertenece al departamento 2 y si es *"13277"* quiere decir que pertenece al departamento 3, entonces se manda un número a la vista correspondiente a cada número de departamento al que el usuario pertenece, por ejemplo si el usuario pertenece al departamento 2, se envía un 2 a la vista, o un 0 en caso de ser administrador. El identificador de cada grupo se puede observar dentro de la tabla `group_` de la base de datos de Liferay *"lportal"*.

*Líneas 30-31.* Se envía el resultado a la vista del portlet, el cual es un identificador para saber si el usuario es administrador y a que grupo pertenece.

```
1  @Override
2  public void doView(RenderRequest req, RenderResponse resp)
3   throws IOException, PortletException {
4    String userId = req.getRemoteUser();
5    byte res = 0;
6    try {
7    User user = UserServiceUtil.getUserById(Long.parseLong(userId));
8    long rol = user.getRoles().iterator().next().getPrimaryKey();
9    if (rol != 10137){ //Si el rol no es administrador.
10    long grupo = 0;
11    for( Iterator<UserGroup> it = user.getUserGroups().iterator(); it.
          hasNext();) {
12      grupo = it.next().getPrimaryKey();
13     if (grupo == 13228){//Si pertenece al Grupo Departamento1.
14      res = 1;
```

```
15      } else if(grupo == 13232){//Si pertenece al Grupo Departamento2.
16       res = 2;
17      } else if(grupo == 13277){//Si pertenece al Grupo Departamento3.
18       res = 3;
19         }
20     }
21    }
22     } catch (NumberFormatException e1) {
23     e1.printStackTrace();
24     } catch (PortalException e1) {
25     e1.printStackTrace();
26     } catch (SystemException e1) {
27     e1.printStackTrace();
28     }
29    //Recibimos en el jsp un RequestScope el valor para asignar el permiso.
        Ahora, nos podemos dar una idea de como desarrollar nuestros
        algoritmos para cada caso.
30    req.setAttribute("userPermitido", res);
31    include(viewJSP, req, resp);
32  }
```

Cuadro 6.4.: Ejemplo de código que nos dará los identificadores de los grupos que pertenecen al usuario que se encuentra en sesión

---

El código Java de la Figura 6.5, es un ejemplo donde se utiliza la biblioteca *JOFC2* para crear una gráfica que contenga dos tipos de barras *("ganancias" y "pérdidas")*.

*Línea 1.* Utilizando el código del Cuadro 6.2, se asigna la lista de listas con valores de ganancias y pérdidas a la lista de listas `valGP`.

*Línea 2.* Inicializamos la gráfica de barras que por último se enviará a la vista como un objeto *JSON* que contiene todas las características para la visualización de la gráfica.

*Línea 3.* Inicializamos `elems` que es una lista que guardará los elementos a animar, es decir la barras de ganancias y las barras de pérdidas con sus valores.

*Líneas 6-17.* En la primera iteración se define un tipo de barra normal que será animada y cuyo color es el primer color hexadecimal de la línea 4 y que será representada con el texto *"Ganancias"*. En esta iteración se hace una segunda para obtener cada valor numérico de la primera lista asignada a `valGP` que es la de ganancias y asignar cada elemento a la lista de barras de a graficar que contendrá la lista de ganancias, para que antes de continuar con la siguiente iteración se asignen las barras a la lista `elems`. En la segunda iteración se define un tipo de barra normal que será animada y cuyo color es el segundo color hexadecimal de la línea 4 y que será representada con el texto *"Pérdidas"*. En esta iteración se hace una segunda para obtener cada valor numérico de la segunda lista asignada a la lista de listas `valGP` que es la de pérdidas y asignar cada elemento de pérdidas a la lista de barras a graficar, para que por último se asignen las barras de pérdidas a la lista `elems`.

*Líneas 20-27.* Se agregan al eje Y los elementos a graficar, se divide el eje Y en diez tomando en cuenta el valor máximo de las ganancias y pérdidas, por último se etiqueta cada división del eje Y con etiquetas de tamaño 10.

*Líneas 30-36.* Se agregan al eje X las fechas que corresponde a cada barra a graficar los cuales son cadenas de tipo "String", las cuales se plasmarán en el eje X de forma inclinada.

*Líneas 39-41.* Se genera el título de toda la gráfica.

```
1  ArrayList<ArrayList<Number>> valGP = (Asignamos la lista de listas con
       valores de ganancias y pérdidas);
2  Chart grafica = new Chart(); //Inicializamos gráfica
3  ArrayList<AnimatedElement> elems = new ArrayList<AnimatedElement>();
4  String [] color = {"#E2D66A","#FF3300"}; //Colores de las barras.
5  String [] tipo = {"Ganancias","Pérdidas"};
6  for (int i= 0 ; i<2; i++){ //Inicializa barras para ganancias y/o pérdidas.
7   BarChart barra = new BarChart(BarChart.Style.NORMAL)); //Tipo de gráfica
       NORMAL.
8   Barra.setOnShow(new OnShow("grow-up")); //Animamos la gráfica.
9   barra.setColour(color[i]);
10  barra.setText(tipo[i]);
11  for (Number valor : valGP.get(i)) {
12      Bar bar = new Bar(valor);
13      bar.setTooltip("$#val#"); //Al posicionar el ratón en cada barra se
            muestra su valor numérico.
14      barra.addBars(bar);
15  }
16  elems.add(barra); //Agregamos primero la lista de barras de ganancias y
       después de pérdidas.
17 }
18
19 // Se crea el eje Y.
20 YAxis ejeY = new YAxis();
21 grafica.setYAxis(ejeY);
22 grafica.addElements(elems.get(0)); Agregamos a la grafica las ganancias.
23 grafica.addElements(elems.get(1)); Agregamos a la grafica las pérdidas.
24 grafica.computeYAxisRange(10);//Crea el rango en el eje Y.
25 YAxisLabels etiquetasY = new YAxisLabels();
26 etiquetasY.setSize(10); // Etiquetas del eje Y de tamaño 10.
27 ejeY.setYAxisLabels(etiquetasY); //Hacemos la asignación.
28
29 // Se crea el eje X.
30 XAxis ejeX = new XAxis();
31 XAxisLabels etiquetasEjeX = new XAxisLabels();
32 etiquetasEjeX.addLabels((Lista con contiene los nombres de las etiquetas
       del eje X que son las fechas));
33 etiquetasEjeX.setRotation(Rotation.HALF_DIAGONAL); //Rotamos las etiquetas.
34 etiquetasEjeX.setSize(10); //Tamaño de las etiquetas.
35 ejeX.setXAxisLabels(etiquetasEjeX); //Asignamos.
36 grafica.setXAxis(ejeX); //Lo plasmamos en la gráfica.
37
38 // Genera leyenda
39 String estilo = "{font-size:12;}";
40 Text titulo = new Text("Texto debajo del eje X", estilo);
```

```
41 grafica.setXLegend(titulo);
42
43 // Características generales de la gráfica
44 grafica.setBackgroundColour("#FFFFFF");
45 //Nos devuelve un objeto JSON como una cadena.
46 return OFC.getInstance().render(grafica);
```

Cuadro 6.5.: *Ejemplo de código que utiliza la biblioteca* JOFC2 *para crear una gráfica que contenga dos tipos de barras* ("ganancias" y "pérdidas").

---

El código de tipo *JavaScript* perteneciente a la vista del proyecto mostrado en el Cuadro 6.6, es un ejemplo de función que guarda la configuración escogida por el usuario para mostrar la gráfica dependiendo de los botones que haya seleccionado.

*Línea 6.* Se utiliza el código del Cuadro 6.3 para saber que tipo de usuario está interactuando con el portlet, este valor es como una referencia para saber que tipo de gráfica mostrar.

*Línea 7.* Se guarda en `prefs.dep` el valor del departamento seleccionado utilizando la función de JQuery `$("input[@name=dep]:checked").val()`, que regresa el valor del botón de tipo radio cuyo nombre es *"dep"*.

*Línea 8.* Se guarda en `prefs.rango` el valor del rango seleccionado utilizando la función de JQuery `$("input[@name=rango]:checked").val()`, que regresa el valor del botón de tipo radio cuyo nombre es *"rango"*.

*Líneas 9-10.* Se guarda en `prefs.ganancias` y `prefs.perdidas` un valor tipo booleano *("0" o "1")*, dependiendo si se seleccionó mostrar solamente las ganancias o las pérdidas.

*Línea 11.* Se guarda en `prefs.tipoBarra` el valor de la selección del tipo de barra.

*Línea 12.* Se guarda en `prefs.animacion` un valor para saber si la gráfica debe mostrarse de forma animada o no.

---

```
1  /** * Obtiene las opciones modificadas en la interfaz.
2  * @return {Object} Objeto con las preferencias.
3  */
4  function getPreferencias() {
5   var prefs = {};
6   prefs.userPermitido = userPermitido;
7   prefs.dep = $("input[@name=dep]:checked").val();
8   prefs.rango = $("input[@name=rango]:checked").val();
9   prefs.ganancias = $('#inGanancias').attr('checked');
10  prefs.perdidas = $('#inPerdidas').attr('checked');
11  prefs.tipoBarra = $('#selTipoBar :selected').val();
12  prefs.animacion = $("input[@name=animacion]:checked").val();
13  return prefs; // Se regresa la variable que guarda los valores para
        mostrar la gráfica.
14 };
```

Cuadro 6.6.: *Ejemplo de código para guardar las preferencias de la gráfica.*

El código mostrado en el Cuadro 6.7, es un ejemplo que utiliza el marco de trabajo JQuery para recargar la gráfica cada vez que se realiza una llamada *AJAX*.

*Líneas 1-4*. los parámetros que se van a enviar al servidor son *"accion"* cuyo valor es *"actualizaGrafica"* y *"prefs"* cuyos valores se pueden obtener con el código mostrado en la **Figura 6.14**. Para enviar los valores al servidor, se deben volver de tipo *JSON* a String utilizando la función de JavaScript llamada `JSON.stringify()`.

*Líneas 5-12*. Se hace una llamada *AJAX* al servidor al poner la ruta del portlet en la variable urlPortlet, los parámetros con la variable `params` que definen como se quiere mostrar la gráfica y una función que recibirá la respuesta del servidor el cual es un objeto *JSON*, en esta función se ejecuta el método `swfobject.embedSWF()` para visualizar la gráfica con los datos enviados del servidor.

```
1  var params = {
2     'accion': 'actualizaGrafica',
3     'prefs': JSON.stringify(getPreferencias())
4    };
5  $.getJSON(urlPortlet, params, function(json){
6   jsonActual = json; // Se obtiene el objeto JSON del servidor.
7   // Agrega Open Flash Chart a la página
8   swfobject.embedSWF(Pelicula.pelicula, "divGrafica2", "100%", "100%",
9     "9.0.0", Pelicula.expressInstall,
10     {'get-data': 'JSON.stringify(jsonActual)'}, //El objeto JSON se debe
           ingresar con String.
11     {'wmode': 'opaque'});
12  });
```

Cuadro 6.7.: *Ejemplo de código para visualizar la gráfica.*

Un ejemplo para obtener los parámetros en el servidor mediante una clase Java es importando el paquete `jofc2.org.json.JSONObject` de la biblioteca *JOFC2*.

En el código siguiente, se obtiene la variable *"prefs"*, la cual contiene un objeto *JSON* con la configuración para mostrar la gráfica, dentro de este *JSON* hay un booleano llamado "ganancias".

```
JSONObject obj = new JSONObject((String) req.getParameter("prefs"));
```

el valor del booleano "ganancias" se puede obtener con el siguiente código:

```
boolean bol = obj.getBoolean("ganancias");
```

En este capítulo se ha dado como ejemplo el desarrollado de un proyecto donde se utilizan las tecnologías que se han visto en los capítulos anteriores como por ejemplo:

OFC, Hibernate, JQuery, JSTL, Liferay y EL, tecnologías utilizadas actualmente para crear sistemas Web donde los requerimientos van cambiando mediante se van desarrollando los sistemas, por ejemplo el uso del portal de Liferay nos permite centrarnos en la programación de portlets independientes a todo el sistema. La ventaja de desarrollar portlets de esta manera específica es que no tenemos que crear toda la lógica de permisos a los usuarios desde cero, simplemente concentrarnos en los requerimientos de cada portlet como una parte independiente de todo el sistema. Otra ventaja de Liferay ya mencionada, es que podemos usar sus bibliotecas para dar permisos a los usuarios del sistema, bibliotecas que interactúan con la base de datos.

El objetivo en general de este trabajo es que los desarrolladores de sistemas tomen en cuenta el uso de portlets para que se dediquen menos al desarrollo de la seguridad y la planeación de cada tipo de usuario *(grupos y roles por ejemplo)* y más al desarrollo de cada portlet que cada usuario va a utilizar en el portal.

# Conclusiones

Existen maneras para desarrollar sistemas Web, por ejemplo utilizando marcos o utilizar un lenguaje de programación sin marcos de trabajo *(en el caso de Java utilizando solamente servlets y JSP's)*, la ventaja de los portlets es que si se requiere, se puede desarrollar utilizando marcos de trabajo para Java por ejemplo **Spring**[4] además de utilizar solamente el lenguaje de programación de Java sin marcos de trabajo. La seguridad que nos ofrece el portal de *Liferay*, se disfruta al mantener un mejor control sobre los usuarios que tienen acceso a cada portlet los cuales funcionan como una parte individual de todo el portal *(aunque a la vez unida al mismo)*, lo que ayuda a mantener un mejor control sobre ellos.

En el desarrollo de esta tesis se han mostrado ejemplos de pequeños sistemas para exponer el beneficio que tiene utilizar un Portal de Gestión de Contenidos como *Liferay* al aprocechar los recursos y bibliotecas que nos ofrece, ya que podemos olvidarnos de programar la seguridad desde cero aplicada a cada usuario del sistema o a los usuarios que quieren acceder a este, y solamente dedicarnos a lo que deben hacer los portlets que van a estar dentro del portal.

Si se requiere desarrollar sistemas donde no sea necesario tener varios módulos y permisos a usuarios para cada uno, lo mejor es no usar portlets ya que la configuración de Liferay puede llevar bastante tiempo, lo mejor es desarrollar el sistema de una manera diferente, por ejemplo utilizando solamente marcos de trabajo.

De lo que se aprendió en la carrera de ciencias de la computación, se utilizó ingenieria de software para diseñar el proyecto final, administración de sistemas GNU/Linux para configurar el servidor donde correra el portal de Liferay y administrarlo, programación Web para investigar y aprender tecnologías eficientes para el desarrollo de aplicaciones Web y bases de datos para comprender la estructura de una base de datos tan compleja como la de Liferay.

A continuación se presenta un resumen de la tesis:

- Configurar un servidor para atender peticiones de usuarios y crear conexiones a las bases de datos *(Anexo A.1)*.
- Instalar, configurar el portal de Liferay de una manera simplificada solamente para tenerlo listo para comenzar a desarrollar en él *(Anexo A.2 y A.3)*.

[4] *"Spring Framework (también conocido simplemente como Spring)* es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java" [49].

- Descripción de la forma de administración de Liferay *(Capítulo 3).*
- Ejemplos de desarrollo de portlets *(Capítulos 2, 5 y 6).*
- Usar tecnologías como Hibernate, JQuery, JSON, Open Flash Chart 2 *y* JSTL interactuando con Java.
- Ahorrar mucho tiempo de programación al configurar Eclipse correctamente, así como aprovechar diversas funcionalidades que nos puede ofrecer como compilar directamente el portlet y que se despliegue directamente en el portal *(Anexo A.4).*
- Usar una extensión de Hibernate para Eclipse llamada *Hibernate Tools (Capítulo 5).*

Cabe mencionar que con Liferay se convierte en una buena opción para desarrollar un amplio abanico de aplicaciones. Algunos usos comúnes son:

- Sitios Web que requieran la presentación de diferentes páginas dependiendo del estado de un usuario que ha accedido al portal.
- Sitios Web que requieran la presentación de diferentes páginas dependiendo del rol o grupo del usuario.
- Sitios Web que requieran de la integración de múltiples aplicaciones Web existentes *(portlets)* en una sola pantalla del portal.
- Sitios Web que permitan a grupos de individuos colaborar a través de aplicaciones en contenido o documentos.

El portal de *Liferay* en costos de desarrollo no es barato debido a la gran comunidad de programadores que tiene detrás. La ventaja de que se encuentre implementado en Java es que no importa que CPU o sistema operativo se quiera utilizar debido a que es un lenguaje multiplataforma que hace que el portal pueda migrarse por ejemplo de Windows a GNU/Linux. Cabe mencionar que en *Liferay* no se tiene que pagar por los derechos de licencia de usuario ya que su licencia LGPL lo concede, además permite que los usuarios y desarrolladores modifiquen el código y lo recompilen [38].

Aparte de *Liferay*, existen otros *Sistemas Manejadores de Contenido o CMS* como por ejemplo ***Plone***[5] si se requiere programar en *Python*. Si se desea programar en Java, *Liferay* es una buena opción, ya que ha obtenido varios premios y reconocimientos como por ejemplo ser lider en portales Java en el año 2011, entro otros como se muestra en la página `http://www.liferay.com/about-us/awards`.

[5]"Plone es un sistema de gestión de contenidos o CMS por sus siglas en inglés *(Content Management System)*, basado en Zope (que tiene miles de desarrolladores en todo el mundo) y programado en Python. Es un desarrollo basado en código abierto. Plone puede utilizarse para construir portales, sitios Webs corporativos, sitios de noticias, servidor de extranet o intranet, como sistema de publicación, repositorio de documentos, etc" `http://plone.org/`.

Un ejemplo de sistemas utilizando las tecnologías mencionadas en esta tesis, es uno que he desarrollado para una red interna de computadoras privadas que compartían sistemas de información y sistemas operacionales con la mayor seguridad posible. Este sistema llamado *SIP ("Sistema de Información Policial")* consta de varios módulos, entre ellos un módulo llamado *COMPSTAT* acrónimo de estadísticas comparativas *( en inglés: "COMPuter STATistics o COMParative STATistics")*, el cual es un sistema para graficar la eficiencia de la policía en el Distrito Federal mediante el número de delitos que hay en la ciudad y las ***reincidencias*** *("Reincidencia es la reiteración de una misma culpa o defecto." )* que hace la policía, otro módulo es Reincidencias el cual hace la búsqueda de todas las personas que están o que alguna vez estuvieron en la cárcel. Los usuarios de *SIP* están restringidos a solo acceder al módulo al que tienen permiso. En el sistema *SIP* se comparten documentos, pero cada documento tiene permisos para ciertos usuarios. Como se puede apreciar, *SIP* es un sistema que se puede hacer complejo mientras los requerimientos y módulos cambien tomando en cuenta que siempre puede haber diferentes usuarios para cada módulo o módulos para el sistema.

Por ahora se puede tener en cuenta el utilizar Liferay para desarrollar redes internas o Intranets, redes externas o Extranets y fantásticos sitios públicos donde se requiera tener diferentes tipos de usuarios, como por ejemplo el sistema ya mencionado llamado *SIP*, el cual es una Intranet, y lo que se ganó al utilizar Liferay para desarrollar este sistema fue la facilidad para ingresar usuarios a diferentes grupos y restringir su acceso solamente a los portlets a los que se asignaba su trabajo.

Un trabajo a futuro es implementar un sitio Web para una institución bancaria, donde además de ofrecer páginas a usuarios anónimos y autenticados, podría disponer de ciertas páginas accesibles solo por determinados tipos de usuarios. De este modo un cliente normal vería páginas y servicios básicos, mientras que un cliente empresarial podría acceder a páginas específicas para él.

# A. Anexo Información complementaria

## A.1. Instalación inicial de Liferay para que utilice el manejador de base de datos PostgreSQL

A continuación se muestra la forma recomendada para ejecutar el portal en *Glassfish* y *Tomcat*. Es posible usar Liferay con *Glassfish V2* ya que tiene soporte para conglomerado de computadores *(a veces en español como clúster)*, algo que en futuras versiones de Liferay con *Glassfish* será posible. GlassFish es un completo servidor de aplicaciones Java, que implementa las API como JPA, EJB y más, mientras que Apache Tomcat no es un servidor completo de aplicaciones Java, sino un contenedor de servlets para aplicaciones Web [57] [21].

También se verá un ejemplo de configuración con la base de datos de *Microsoft SQL server 2005*.

*PostgreSQL 9.0* incluye importantes mejoras que tienen que ver con aspectos de diseño y rendimiento de las aplicaciones que dependen de una base de datos. Esto incluye soporte para *Windows 64-bi*t, desencadenantes condicionales a nivel columna que son funciones dependientes de una columna que se invocan cuando se actualiza alguno de los registros de la columna utilizando alguno de los comandos de actualización como UPDATE *(dicho de otro modo es crear triggers a nivel de columna)*. Podemos realizar una actualización a partir de las versiones 8.3 y 8.4. La lista completa de nuevas características está disponible en la información sobre la versión en la Web oficial de la aplicación. [10]

Se decidió utilizar la base de datos *PostgreSQL* ya que es un manejador de base de datos muy estable y en la versión 9.0 trae avances en materia de seguridad, soporte de aplicaciones, supervisión, mejor rendimiento y almacenamiento de datos especializados. Además, el interés de los usuarios es grande por las nuevas posibilidades, que acelerarán su adopción en un alojamiento en la nube, lo cual significa dentro de un servidor al cual se accede mediante Internet y aplicaciones extensibles.

"*La replicación interna de PostgreSQL 9.0 y la posibilidad de consultar un servidor de recuperación son las características más solicitadas durante muchos años*", dijo Simon Riggs, director de tecnología y líder del proyecto 2ndQuadrant *"conmutación por error" ("hot standby" en VO). "Una réplica eficaz y contínua, y baja latencia ofrece una mejor protección de sus datos, mientras que la conmutación por error reduce el costo total de propiedad (TCO o "Total Cost of Ownership" en VO). Dentro de los productos de esta*

*naturaleza, esta posibilidad, opcional, cuesta varios miles de dólares.*" [11]

### A.1.1. Pasos para Glassfish

Descargamos liferay-portal-glassfish-6.0.5.zip de la página de Liferay o algún repositorio de la comunidad, y lo descomprimimos en una carpeta, vamos al directorio `/bin` y ejecutamos `sh asadmin start-domain` , asadmin es el script de arranque de Glassfish, se ejecuta como se muestra en la Figura A.1.

```
linux-5904:/home/arpalaci/Tesis/portal/liferay-portal-6.0.5/glassfish-3.0.1/bin # sh asadmin start-domain
Waiting for DAS to start .....................................................................................
.....................
Started domain: domain1
Domain location: /home/arpalaci/Tesis/portal/liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1
Log file: /home/arpalaci/Tesis/portal/liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/logs/server.log
Admin port for the domain: 4848
Command start-domain executed successfully.
linux-5904:/home/arpalaci/Tesis/portal/liferay-portal-6.0.5/glassfish-3.0.1/bin # cd ..
linux-5904:/home/arpalaci/Tesis/portal/liferay-portal-6.0.5/glassfish-3.0.1 # tail -f domains/domain1/logs/server.log

[#|2011-03-03T03:05:33.384+0000|INFO|glassfish3.0.1|javax.enterprise.system.container.web.com.sun.enterprise.web|_Thr
eadID=28;_ThreadName=Thread-1;|Loading application __admingui at /|#]
```

Figura A.1.: *Ejecución de Liferay.* Vamos a la carpeta /bin y se ejecuta el comando `sh asadmin start-domain` para ejecutar Liferay.

El comando `tail -f` es muy útil para poder observar lo que ocurre en el servidor cuando esta en ejecución.

Al haber hecho esto es posible visualizar la página de Liferay al escribir en la barra de un navegador Web *http://localhost:8080/*, como se puede observar en la Figura A.2.


Figura A.2.: *Inicio de la página de Liferay.* Se puede observar la página de Liferay al acceder a *http://localhost:8080/.*

A continuación podremos modificar las características del servidor Glassfish al poner en la barra de direcciones: *http://localhost:4848/.*

Podemos verificar si las opciones para un mayor rendimiento del servidor son correctas modificando los valores para del *montículo o heap* de la máquina virtual de *Java* ***-Xms*** *(en este caso* XX:MaxPermSize*)* y ***-Xmx***. El ***heap*** es el espacio de memoria en tiempo de ejecución donde las instancias de clases *(objetos)* y arreglos son asignados. El *heap* es creado al inicio de la máquina virtual y el encargado de separar este espacio es el sistema de administración de almacenamiento automático, o mejor conocido como *recolector de basura (Garbage Collector*); los objetos nunca son explícitamente desasignados. El *heap*

puede ser de un tamaño modificado *(por parámetros de la* Máquina Virtual de Java o JVM*)*, puede ser expandible automáticamente por demanda o disminuida si un *heap* muy grande se torna innecesario. Los parámetros *Xms* y *Xmx* establecen *el tamaño inicial y máximo de la cantidad de memoria* que la JVM utilizará para ejecutar programas Java pero, esto no implica que nuestro proceso Java ocupe en memoria un tamaño máximo como el establecido en el parámetro *Xmx*. Como se ha especificado antes, la *JVM* es un proceso en si mismo y, como tal, necesita memoria para poder cargar su código y datos. Por tanto, un proceso Java puede llegar a ocupar mas espacio en memoria que el determinado en *Xmx*. La JVM divide el espacio en "generaciones" o particiones en los que poder aplicar sofisticados algoritmos de recolección. Cuando alguno de estos espacios se llena, se fuerza una recolección de objetos y, la eficiencia de este sistema se basa en que la mayoría de los objetos tienen una vida útil muy corta. El espacio en memoria, se divide generalmente en dos particiones: la nueva y la vieja generación. Cuando el tamaño de la nueva generación se llena, se invoca una recolección de objetos muy rápida *(GC)* que acaba con los objetos que no vayan a ser utilizados y que traspasa a la vieja generación los objetos con vida. Cuando la vieja generación se llena, se invoca una recolección completa *(Full GC)* que implica a toda la memoria y que es mucho más lenta que la recolección rápida [52].

- Un mayor *heap* puede almacenar más objetos pero puede incluso aumentar los tiempos del *Recolector de Basura* [52].
- Se recomienda poner los valores basados en la memoria física. Por ejemplo si se tiene 2GB de memoria se puede utilizar -Xms (-XX:MaxPermSize) = 192m y -Xmx a 1024m como se muestra en la Figura A.3.
- Se recomienda poner los mismos valores para *-Xms (-XX:MaxPermSize)* y *-Xmx* para evitar el aumento del *heap* durante las pruebas de rendimiento.

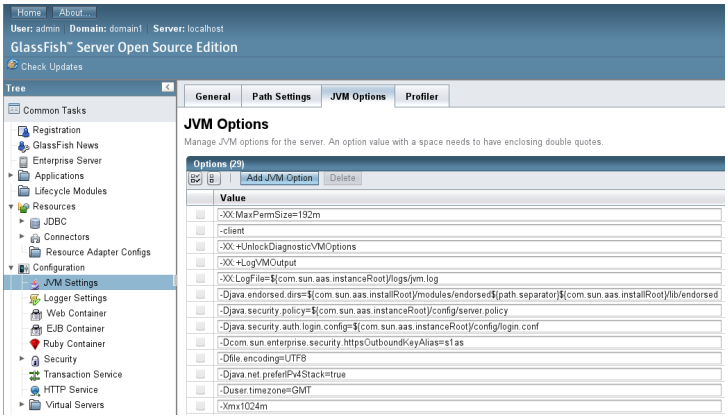


Figura A.3.: *Asignación de memoria a la JVM.* Se asigna el tamaño inicial a 192m y el máximo de la cantidad de memoria a 1024m.

Nos vamos al directorio `/bin` de Glassfish y paramos el servidor estando en la carpeta `glassfish-3.0.1/bin` con los comandos:

```
sh asadmin stop-domain
```

#### A.1.1.1. Configuración de la base de datos con el manejador de base de datos PostgreSQL

Para continuar es necesario ya haber instalado el manejador de base de datos PostgreSQL, (suponiendo que ya se encuentra instalado), creamos la base de datos como se muestra en la Figura A.4 [21].

```
postgres@linux-5904:~> psql
Contraseña:
psql (8.4.5)
Digite «help» para obtener ayuda.

postgres=# CREATE DATABASE lportal ENCODING='UNICODE';
CREATE DATABASE
postgres=#
```

Figura A.4.: Creación de la base de datos. Se crea la base de datos *lportal* desde consola.

Para llenar la base de datos hay que modificar el archivo `portal.properties` de `liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/applications/liferay-portal/WEB-INF/lib/portal-impl.jar`

Para poder modificarlo vía consola se puede hacer uso del comando `jar`, por ejemplo; `[jar xf </path/to/file.jar>]` para extraer el contenido y `[jar cf </location/of/new/archive/file>]` para reempacar el archivo. Aunque ahora si se desea se puede hacer uso de diversas herramientas como *xarchiver* en linux o *winrar* en windows para poder modificar los archivos que se encuentran dentro del archivo con extensión *jar* [62].

En las líneas 880-890 comentar la configuración de Hypersonic como se muestra en la Figura A.5 para que se deje de utilizar la basa de datos que se usa por defecto llamada *Hypersonic* la cual es escrita en *Java* y es ideal para pruebas y prototipos.

```
881
882     #
883     # Hypersonic
884     #
885     #jdbc.default.driverClassName=org.hsqldb.jdbcDriver
886     #jdbc.default.url=jdbc:hsqldb:${liferay.home}/data/hsql/lportal
887     #jdbc.default.username=sa
888     #jdbc.default.password=
889
```

Figura A.5.: *Desactivar el uso de la base de datos Hypersonic.* Solamente comentando las líneas que se muestran.

Configurar las líneas 920-930 para utilizar la base de datos de *PostgreSQL* ya creada anteriormente como se muestra en la Figura A.6 [62].

```
922     #
923     # PostgreSQL
924     #
925     jdbc.default.driverClassName=org.postgresql.Driver
926     jdbc.default.url=jdbc:postgresql://localhost:5432/lportal
927     jdbc.default.username=postgres
928     jdbc.default.password=postgres
929
```

Figura A.6.: *Activar el uso de la base lportal de PostgreSQL.*
Se debe descomentar (quitar los #) en las líneas 920-930.

Reiniciar el servidor con los comandos `asadmin start-domain` como se hizo anteriormente.

Se puede observar en el archivo de registro de la base de datos que la base lportal ha sido poblada. Para esto basta con escribir en la consola `psql lportal` estando como usuario postgres para entrar a la base lportal Figura A.7. Una vez ya dentro de la base lportal escribir el comando \d para mostrar todas las tablas que se han generado Figura A.8.

```
postgres@linux-5904:~> psql lportal
Contraseña:
psql (8.4.5)
Digite «help» para obtener ayuda.

lportal=# \d
```

Figura A.7.: *Entrando a la base lportal desde el usuario postgres.*

```
                 Listado de relaciones
 Esquema |             Nombre              | Tipo  |  Dueño
---------+---------------------------------+-------+----------
 public  | account_                        | tabla | postgres
 public  | address                         | tabla | postgres
 public  | announcementsdelivery           | tabla | postgres
 public  | announcementsentry              | tabla | postgres
 public  | announcementsflag               | tabla | postgres
 public  | assetcategory                   | tabla | postgres
 public  | assetcategoryproperty           | tabla | postgres
 public  | assetentries_assetcategories    | tabla | postgres
 public  | assetentries_assettags          | tabla | postgres
 public  | assetentry                      | tabla | postgres
 public  | assetlink                       | tabla | postgres
 public  | assettag                        | tabla | postgres
 public  | assettagproperty                | tabla | postgres
 public  | assettagstats                   | tabla | postgres
 public  | assetvocabulary                 | tabla | postgres
lines 1-18
```

Figura A.8.: *Listado de tablas en la base lportal* mostrado en consola.

Ingresar a `http://localhost:4848/` a través de un navegador Web y registrarse como usuario *test@liferay.com* y password *test* para entrar como administrador [39].

Ir a `Resources - JDBC - Connection Pools - new`.

A continuación se crea una conexión en *Glassfish* con la base de datos de Liferay. Como se puede observar en la Figura A.9 se define el nombre de la conexión, el tipo de recurso el cual es una biblioteca que permite la ejecución de operaciones sobre la base de datos desde el lenguaje de programación *Java* y se define el nombre del manejador de la base de datos [21].

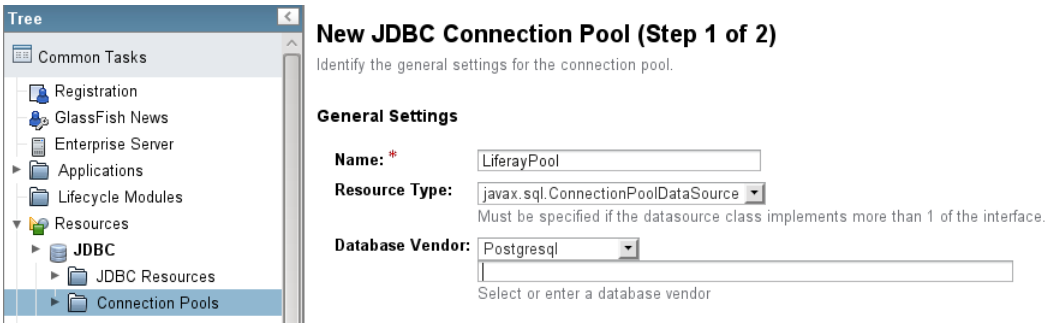


Figura A.9.: *Se define una nueva conexión a la base de datos para Glassfish.* El nombre de la conexión definida es LiferayPool la cual puede tener cualquier nombre, tipo de recurso `javax.sql.ConectionPoolDataSource` y el nombre del manejador de la base de datos *Postgresql*.

La Figura A.10, se basa en mi experiencia para configurar la conexión a la base de datos, aunque si se desea, se puede dejar a como está por defecto ya que lo unico que cambia es el tiempo de espera de respuesta cuando se hace una petición a la base de datos [21].

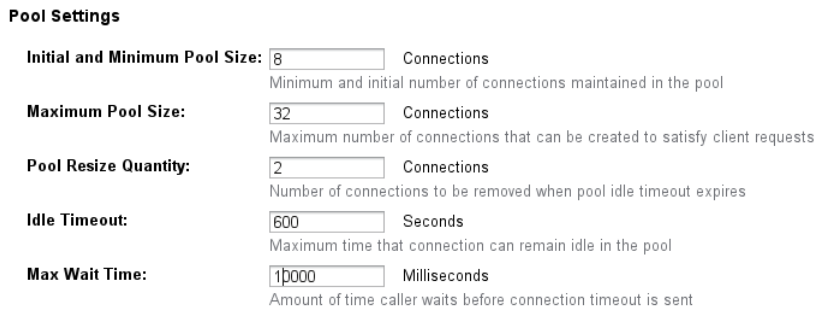


Figura A.10.: *Configuración de la conexión.* Debajo de las casillas se especifica para que se utiliza cada configuración.

Continuamos modificando propiedades adicionales. En la Figura A.11 asignamos `JDBC30DataSource` a `true` en dado caso que se generen avisos de error al utilizar *PostgreSQL* para hacer consultas.

El parámetro de configuración `DefaultAutoCommit` indica que al lanzar una consulta ésta se ejecutará al momento de ser llamada al poner el parámetro `true` o en caso contrario al poner el parámetro `false`, si está en `false` las conexiones se mantienen abiertas hasta que se dispara el "*Commit (acción de hacer una tranzacción)*" la cual finaliza la transacción de la base de datos, cabe mencionar que cuando se usan procedimientos JDBC de lado de la base de datos el `AutoCommit` se apaga.

Por defecto la habilidad de *commit* de *JDBC* está encendida lo que significa que cada declaración SQL se va a ejecutar para hacer una transacción con cada una . Si mas de una declaración SQL es ejecutada por su programa, entonces un pequeño aumento de rendimiento puede alcanzarse al apagar `DefaultAutoCommit` y hacer una sola transacción con todas las consultas declaradas [4] [19].

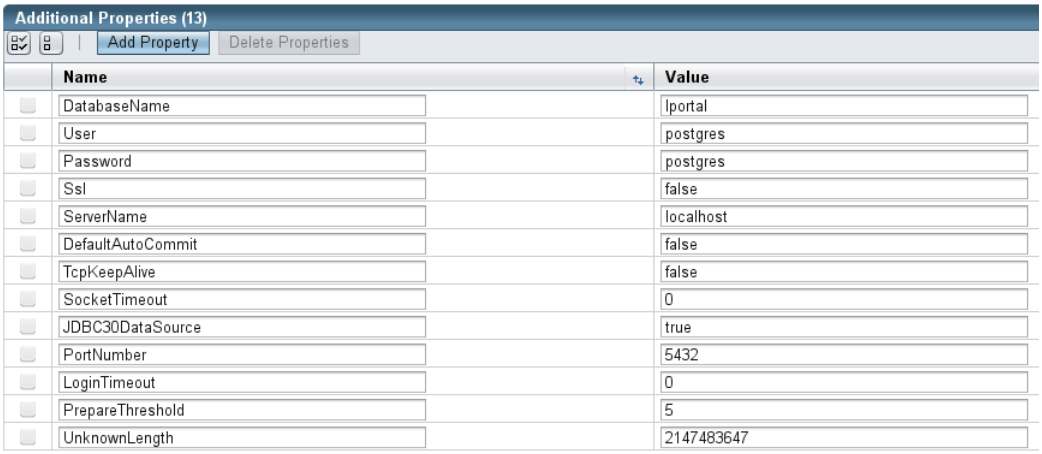


Figura A.11.: *Propiedades adicionales.*

Al haber concluido lo anterior podemos probar que se realizó una configuración correcta para que Glassfish se conecte a la base de datos lportal, para esto se oprime el botón Ping y si se tuvo éxito mostrará *Ping Succeeded* como se puede observar en la Figura A.12.

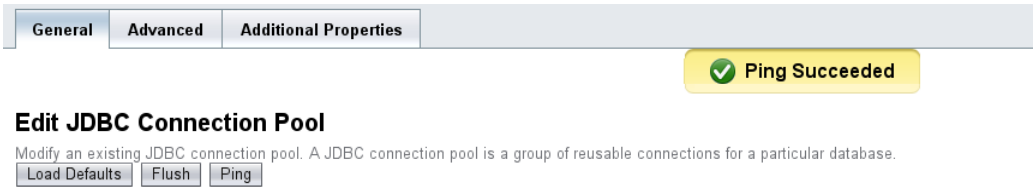


Figura A.12.: *Probando la conexión a la base de datos lportal.* Se puede observar que es exitosa.

Ir ahora a `JDBC Resources - new` y modificar los parámetros como se muestra en la Figura A.13. Se especifica un nombre ***JNDI***[1] que identifica el recurso *JDBC* que se ha creado y que será utilizado en la programación para consultar la base de datos *"lportal"*.

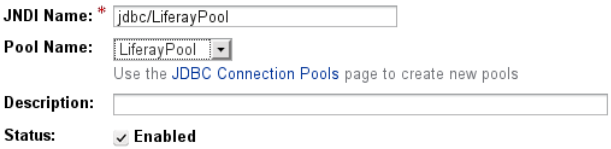


Figura A.13.: *Se identifica el recurso JDBC utilizado.* La cual se logra específicamente dando un nombre a la Interfaz de Nombrado y Directorio Java (JNDI).

### A.1.1.2. Creando otra conexión, ejemplo con el manejador de base de datos Microsoft SQL server

El propósito de este capítulo es mostrar un ejemplo de la configuración de una base de datos para el manejador de Microsoft SQL server. Solamente se da como ejemplo por si el lector está interesado en utilizar este manejador de base de datos.

Ingresar a `Resources - JDBC - Connection Pools - new` y configurar la conexión a la base de datos de forma similar a como se realizó para la conexión a la base de datos *lportal*.

```
Name: SQLPool Resource
Type: javax.sql.ConnectionPoolDataSource
Database Vendor: Microsoft SQL Server
Initial and Minimum Pool Size: 8
Connections Maximum Pool Size: 32
Connections Pool Resize Quantity: 2
Connections Idle Timeout: 600 Seconds
```

[1] "*La Interfaz de Nombrado y Directorio Java (Java Naming and Directory Interface)* es una Interfaz de Programación de Aplicaciones (API) para servicios de directorio". [20]

```
Max Wait Time: 10000 Milliseconds

user nomusuario
password xxxx
networkProtocol tcp portNumber 1433 (O el puerto que tenga)
serverName xx.xx.xx.xx
databaseName SQLPool
```

Ahora en `JDBC Resources - new` y definir el nuevo recurso *JDBC*.

```
JNDI Name: jdbc/SQLPool
PoolName:SQLPool
Status habilitado
```

#### A.1.1.3. Configurar seguridad en el servidor con certificados .pfx en Glassfish

Una empresa que solicite seguridad requerirá de conexiones seguras a su sistema, para ello se puede configurar el servidor Glassfish con soporte para conexiones seguras entre el cliente y el servidor mediante el protocolo ***HTTPS*** [2].
La ventaja de utilizar este protocolo es que la información generada con el sistema desde el cliente al servidor, estará cifrada, un ejemplo sería que al ingresar a un sistema con este protocolo, los datos de nombre de usuario y contraseña se enviarán cifrados al servidor *Glassfish*. Una desventaja se da cuando se usa ***Nombres Basados en Anfitriones (Hosts) Virtuales*** ya que es un método popular, el cual identifica diferentes anfitriones virtuales permitiendo usar la misma dirección IP y el mismo puerto para diferentes sitios. Cuando las personas usan *SSL (cifrado que utiliza HTTPS)*, parece natural asumir que el mismo método puede ser usado para tener diferentes anfitriones virtuales con *SSL* en el mismo servidor, sin embargo es imposible. La razón es que el protocolo *SSL* es una capa separada que encapsula el protocolo *HTTP*, por lo que la sesión *SSL* es una transacción separada que toma lugar antes que la sesión *HTTP* haya comenzado. El servidor recibe una petición *SSL* en la dirección IP X y puerto Y *(usualmente 443)*. Como la petición *SSL* no contiene ningún campo de anfitrión, el servidor no tiene manera de decidir que *anfitrión virtual con SSL* utilizar, usualmente usa el primero que encuentra, que coincide con el puerto y la dirección IP especificada. [30]

Se puede utilizar por supuesto el *Nombre Basado en Anfitrión Virtual* para identificar muchos *anfitriones virtuales sin SSL (todos en el puerto 80 por ejemplo)* y entonces

[2] *"Hyper Text Transfer Protocol Secure (en español: Protocolo Seguro de Transferencia de Hipertexto).* El sistema *HTTPS* utiliza un cifrado basado en *SSL/TLS* para crear un canal cifrado (cuyo nivel de cifrado depende del servidor remoto y del navegador utilizado por el cliente) más apropiado para el tráfico de información sensible que el protocolo *HTTP*. De este modo se consigue que la información sensible (usuario y claves de paso normalmente) no pueda ser usada por un atacante que haya conseguido interceptar la transferencia de datos de la conexión, ya que lo único que obtendrá será un flujo de datos cifrados que le resultará imposible de descifrar". [34]

tener un solo *anfitrión virtual con SSL (en el puerto 443)*. Si se hace esto, debemos asegurarnos de especificar el número de puerto sin *SSL* en la directiva del *Nombre de Anfitrión Virtual*, por ejemplo [30]:

```
NameVirtualHost 192.168.1.1:80
```

Otra solución sería usar diferentes direcciones IP para diferentes anfitriones *SSL*. Utilizando diferentes números de puerto para diferentes anfitriones *SSL* [30].

Para habilitar la seguridad, lo primero a realizar es cambiar el password del servidor de aplicaciones *Glassfish* como se muestra en la Figura A.14 [21].

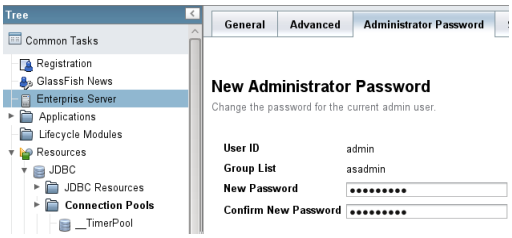


Figura A.14.: *Cambiando el password de Glassfish.*

Vamos a `Network Listeners` y accedemos a `http-listener1` para configurarlo, tal y como se muestra en la Figura A.15. De los valores mostrados por defecto seleccionamos la casilla de *Security (Seguridad)* [21].

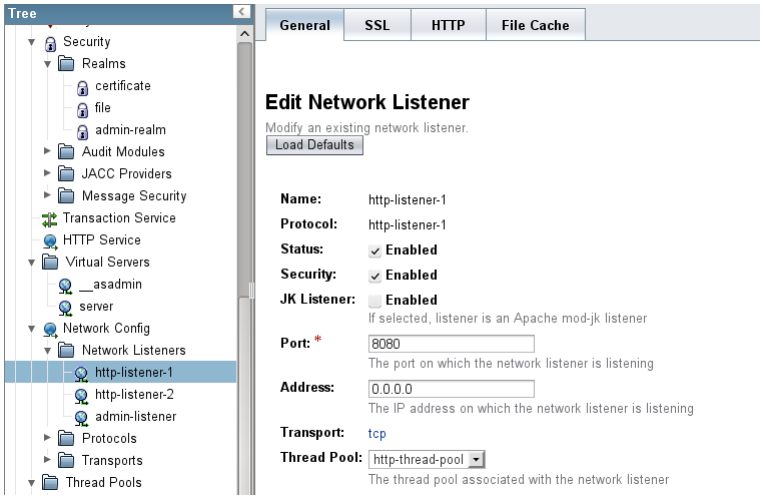


Figura A.15.: *Habilitamos la seguridad.* Al seleccionar la casilla *Security* y en el puerto dejar por defecto 8080, se habilita la seguridad en el puerto 8080 *(el cual se especifica en Port*)* [17].

Al reiniciar el servidor podremos acceder a la dirección: `https://localhost:8080` y confirmamos la excepción de seguridad para entrar.

Volvemos a `http://localhost:4848/`, a continuación vemos que el servidor nos pide contraseña.

A continuación se muestra como utilizar un certificado *.pfx* los cuales son formatos de archivos ***PKCS#12*** *(Se refiere a un grupo de estándares de criptografía de llave pública concebidos y publicados por los laboratorios de RSA en California)* comúnmente usados para almacenar claves privadas con su certificado de llave pública protegido mediante ***llave simétrica*** [3] [42] [63].

Para lo siguiente utilizamos una biblioteca llamada `pk12import.jar`

Copiamos `pk12import.jar` y `pk12import.sh` y el certificado *.pfx* que se puede crear con un generador *pfx (un ejemplo en el sistema operativo GNU/Linux es usar el comando openssl para generarlo)*, en
`liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/config/`

Vamos a `liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/config/` en consola escribimos el siguiente comando para borrar el auto-asignado certificado [63]
`keytool -delete -alias s1as -keystore keystore.jks`

Ejecutamos los siguientes comandos en consola para crear de nuevo el alias *s1as* y asignarlo al archivo que guarda el certificado del servidor llamado *keystore.jks*. La contraseña que usaremos es *changeit* tal y como se muestra a continuación para cambiar la contraseña a la clave.
`sh pkcs12import.sh -file certificado_intranet.pfx -alias s1as -keystore`
`keystore.jks -storepass changeit -pass intranet` *(Donde intranet es el password del certificado pfx aquí llamado certificado_ intranet.pfx)*
`key-password: changeit`

Ejecutamos los siguientes comandos para almacenar el certificado en el archivo *s1as.cer:*

`keytool -export -file s1as.cer -keystore keystore.jks -alias s1as -storepass changeit`

Ejecutamos los siguientes comandos para imprimir el certificado en la consola:
`keytool -printcert -file s1as.cer`

[3] *"La criptografía simétrica* es un método criptográfico en el cual se usa una misma llave para cifrar y descifrar mensajes. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la llave a usar. Una vez ambas tienen acceso a esta clave, el remitente cifra un mensaje usándola, lo envía al destinatario, y éste lo descifra con la misma". [3]

A continuación se almacenamos los certificados confiables en el archivo *cacerts.jks* de *Glassfish* el cual incluye todos los certificados confiables, contando los certificados de los clientes con los siguientes comandos: [22]

`keytool -import -alias s1as -keystore cacerts.jks -file s1as.cer Escriba la contraseña del almacén de claves:`

`error de keytool: java.lang.Exception: Certificado no importado, el alias <s1as> ya existe` *(No importa aun así ponerlo)*

Iniciamos de nuevo servidor para entonces tener el *HTTPS* configurado.

#### A.1.1.4. Actualización a la versión de PostgreSQL *(JDBC)* más reciente

Cuando se requiere utilizar una versión del manejador de base de datos *PostgreSQL* con nuestros programas *Java* lo que se debe hacer es lo siguiente:

Reemplazar el archivo
`liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/lib/postgresql.jar`
por la versión más reciente del manejador de la base de datos *PostgreSQL*.

Después de haber realizado lo anterior, reiniciamos de nuevo el servidor para poder utilizar el manejador mas actual de *PostgreSQL*.

#### A.1.1.5. Funcionalidad de Liferay para subir una cantidad de archivos significante

El *CMS* de Liferay provee de un lugar central para almacenar y administrar todos los contenidos. Cada comunidad en el portal de Liferay tiene su propia biblioteca de documentos separada, la cual esta equipada con carpetas personalizables y actúa como un manejador para compartir documentos basado en Web para todos los miembros del grupo, sin importar donde se encuentren, ya que el contenido es accesible solamente para aquellos autorizados por el administrador, cada archivo individual es tan abierto a los demás o seguro como quieras que sea. [9]

El portlet de biblioteca de documentos provee de un administrador de documentación que puede ser respaldado por diferentes sistemas de persistencia. Esto incluye registro de ingreso y salida de documentos, metadatos y características de versión. Cada archivo puede contener muchas versiones y la historia de cada versión está disponible como una lista en Liferay. [9]

Para lo anterior es mejor tener un sistema de persistencia avanzado para subir una cantidad de archivos significante y más rápido que el que el sistema de persistencia que viene por defecto, para ello hacemos lo siguiente:

Vamos a `glassfish-3.0.1/domains/domain1/applications/`

`liferay-portal/WEB-INF/lib/portal-impl.jar`.

En `portal.properties` comentar
`dl.hook.impl=com.liferay.documentlibrary.util.FileSystemHook`
y descomentar la siguiente línea
`dl.hook.impl=com.liferay.documentlibrary.util.AdvancedFileSystemHook`
en el renglón que tiene el apuntador, tal y como se muestra en la Figura A.16.

```
5814 ##
5815 ## Document Library Portlet
5816 ##
5817
5818     #
5819     # Set the name of a class that implements
5820     # com.liferay.documentlibrary.util.Hook. The
5821     # document library server will use this persist documents.
5822     #
5823     dl.hook.impl=com.liferay.documentlibrary.util.AdvancedFileSystemHook
5824     #dl.hook.impl=com.liferay.documentlibrary.util.CMISHook
5825     #dl.hook.impl=com.liferay.documentlibrary.util.FileSystemHook
5826     #dl.hook.impl=com.liferay.documentlibrary.util.JCRHook
5827     #dl.hook.impl=com.liferay.documentlibrary.util.S3Hook
```

Figura A.16.: *Utilizar un sistema avanzado de persistencia para administrar los documentos.* `AdvancedFileSystemHook` [9].

Después de esta sección se le puede asignar el tamaño máximo de los archivos a subir por si es necesario, la característica de dónde encontrarlo se escribe en comentarios de la forma *"##"*.

### A.1.2. Pasos para Tomcat

La instalación del portal con Tomcat es muy similar a Glassfish. La razón por la cual explico la instalación con Tomcat es que muchas empresas hacen uso de este servidor el cual es muy estable, y de la cual para obtener mas rendimiento, es necesario hacer ciertas cosas para usar sus bibliotecas nativas tal y como se verá en la sección A.1.2.3.

#### A.1.2.1. Configuración de la base de datos

Crear la base de datos vacía *"lportal"* en Postgres *(en formato UTF-8 y el usuario postgres como el usuario propietario).*

Para llenar la base de datos hay que modificar el archivo `portal.properties` que está dentro de `liferay-portal-6.0.5/tomcat-6.0.26/webapps/ROOT/WEB-INF/lib/portal-impl.jar`, como se muestra en la Figura A.17.

Comentar la configuración de *Hypersonic* como se hizo en *Glassfish* y **des-comentar** la de *PostgreSQL*:

```
922     #
923     # PostgreSQL
924     #
925     jdbc.default.driverClassName=org.postgresql.Driver
926     jdbc.default.url=jdbc:postgresql://localhost:5432/lportal
927     jdbc.default.username=postgres
928     jdbc.default.password=postgres
```

Figura A.17.: *Utilizando el manejador de la base de datos PostgreSQL*, para modificar el archivo `portal.properties`.

Después se tendrá que ejecutar Liferay y automáticamente se crearán las tablas en la base de datos *"lportal"*.

#### A.1.2.2. Configuración para poder ver películas Flash en Internet Explorer sobre conexiones seguras

Este problema solo sucede cuando se quiere poner en algún portlet películas *Flash* y el dominio es *HTTPS (Se hace uso de conexiones seguras)*. La solución se encontró directamente acudiendo a los foros de `www.liferay.com` con la búsqueda *"File downloads over SSL don't work with cache control in IE browser" (Subida de archivos sobre SSL no funciona sin control de caché en IE)*. Este problema es específico de Intertet Explorer 6.X ya que la descarga de archivos sobre conexiones seguras no trabajan con control de la caché.

Dentro de la etiqueta `<Context>` de `/liferay-portal-6.0.5/tomcat-6.0.26/conf /context.xml` poner:

```
<Valve className="org.apache.catalina.authenticator.NonLoginAuthenticator"
disableProxyCaching="true" securePagesWithPragma="false" />
```

Este código muestra una tecnología de Tomcat llamada *Valve* la cual permite asociar una instancia de una clase Java con un contenedor particular del contenedor de servlets de Tomcat llamado Catalina. En `disableProxyCaching` deshabilita la caché en el proxy y `securePagesWithPragma` determina si deshabilitamos el caché en el proxy con cabeceras incompatibles con Internet Explorer como lo dice la página
`https://issues.apache.org/bugzilla/show_bug.cgi?id=27122.`

#### A.1.2.3. Actualizar Liferay con Tomcat por defecto al versión más actual, para usar APR con SSL *(seguridad y rendimiento!!!)*

*APR (Apache Portable Runtime)* es una biblioteca de código nativo escrita en *C/C++* dependiente de la plataforma. No es estrictamente un conector, pero cuando se activa el conector estándar delega la mayoría de sus operaciones en él. Al utilizar código nativo incrementa la eficiencia y escalabilidad en la respuesta del servidor hacia el cliente [57].

Es mas eficiente y escalable ya que:

- Utiliza una llamada `sendFile()` en modo núcleo para enviar grandes archivos estáticos directamente desde el sistema de archivos nativo.
- Usa un solo consultor de persistencia de código nativo para implementar conexiones persistentes para un gran número de conexiones.
- Utiliza código nativo de OpenSSL, el cual tiene capacidad de acelerar la implementación del controlador SSL *(mediante hardware)* [57].

Pasos para utilizar Tomcat con APR basado en la página
`http://www.liferay.com/es/community/wiki/-/wiki/Main/Tomcat+Native+Library`

- Descargamos la versión de Tomcat más actual *(en este caso apache-tomcat-7.0.10.zip)* y la descomprimimos.
- Nos aseguramos que las bibliotecas libssl-dev y libapr1-dev están instalados en el sistema *(Cada sistema tiene diferentes comandos para saber si estas bibliotecas están instaladas).*
- Vamos al directorio de Tomcat bin: `$ cd ${TOMCAT_HOME}/bin`.
- Extraemos el archivo comprimido o tarball *(archivo tarred y gzipped)* de la biblioteca de Tomcat nativo con el comando `tar -xvzf tomcat-native.tar.gz`
- Vamos al directorio fuente `tomcat-native-1.1.x/jni/native.`
- Configuramos la instalación de bibliotecas necesarias para su instalación ingresando los comandos de `configure, make y install`:
  `./configure --with-apr=/usr && make && sudo make install.`
- Cambiamos de directorio de sistema con el comando `cd /usr/lib`
- Hacemos un enlace conveniente a nuestra nueva biblioteca con el comando para que se use la biblioteca de *APR* en el sistema:
  `sudo ln -s /usr/local/apr/lib/libtcnative-1.so libtcnative-1.so`
- Editamos `${TOMCAT_HOME}/bin/catalina.sh` donde `${TOMCAT_HOME}` es la ruta al directorio del servidor *TOMCAT* añadiendo las siguientes líneas al final del archivo antes de que el programa en Java sea ejecutado:
  `LD_LIBRARY_PATH=/usr/lib:$LD_LIBRARY_PATH`
  `export LD_LIBRARY_PATH`

Con esto ya tenemos *APR* funcionando[4].

Abrimos dos ventanas, una con el directorio donde se encuentra instalado el Tomcat existente ***(TOMCAT_EXISTENTE)*** y otra donde tenemos el Tomcat que está dentro del

[4] *Hay que tener cuidado cuando se sobreescriban los archivos, sobre todo cuando el Tomcat cuenta con muchas aplicaciones instaladas. En este caso comprobar antes cuales son las diferencias y actuar en consecuencia.*

paquete de Liferay *y* Tomcat que se descarga en la página de Liferay (*LIFERAY _TOMCAT*) y realizamos los siguientes pasos para tener Liferay con el Tomcat que configuramos con *APR*:

- Copiamos el archivo `LIFERAY_TOMCAT/conf/Catalina/localhost/ROOT.xml` a `TOMCAT_EXISTENTE/conf/Catalina/localhost/.`
- Copiamos el archivo `LIFERAY_TOMCAT/conf/catalina.properties` a `TOMCAT_EXISTENTE/conf/`, sobreescribiendo el contenido.
- Copiamos el archivo `LIFERAY_TOMCAT/conf/jaas.config` a `TOMCAT_EXISTENTE/conf/`
- Copiamos el archivo `LIFERAY_TOMCAT/conf/server.xml` a `TOMCAT_EXISTENTE/conf/`, sobreescribiendo el contenido.
- Copiamos el archivo `LIFERAY_TOMCAT/bin/setenv.sh` a `TOMCAT_EXISTENTE/bin/`. *(Este archivo es el que se encarga de aumentarle la memoria a la máquina virtual).*
- Copiamos la carpeta `LIFERAY_TOMCAT/lib/ext` a `TOMCAT_EXISTENTE/lib/`
- Borramos el contenido de la carpeta `TOMCAT_EXISTENTE/webapps/ROOT` y copiamos dentro todo el contenido de la carpeta `LIFERAY_TOMCAT/webapps/ROOT`

  Para realizar los pasos anteriores se puede utilizar el comando *rsync* para copiar archivos de un directorio a otro del mismo sistema.

#### A.1.2.4. Configuración de la base de datos con el manejador de base de datos PostgreSQL para Tomcat

En este apartado se mostrará ejemplos de configuración de bases de datos como se muestra en la documentación de Liferay y también como se recomienda para usar conexiones seguras.

En `apache-tomcat-7.0.10/conf/Catalina/localhost/ROOT.xml` escribir las conexiones a las bases de datos como se muestra en la Figura A.18.

*Líneas 3-10.* Se define un recurso para manejar el correo electrónico en el portal de Liferay.

*Líneas 13-28.* Se define una conexión a la base de datos de Liferay llamado *"lportal"* con propiedades descritas en la configuración para *Glassfish.*

```
1  <Context path="" crossContext="true">
2
3      <Resource
4          name="mail/MailSession"
5          auth="Container"
6          type="javax.mail.Session"
7          mail.transport.protocol="smtp"
8          mail.smtp.host="localhost"
9          mail.store.protocol="imap"
10         mail.imap.host="localhost"
11     />
12
13     <Resource
14         name="jdbc/LiferayPool"
15         auth="Container"
16         type="javax.sql.DataSource"
17         driverClassName="org.postgresql.Driver"
18         url="jdbc:postgresql://lportal"
19         username="postgres"
20         password="postgres"
21         testOnBorrow="true"
22         validationQuery="SELECT 1"
23         testWhileIdle="true"
24         timeBetweenEvictionRunsMillis="5000"
25         maxActive="100"
26         maxIdle="30"
27         maxWait="10000"
28     />
29
30 </Context>
```

Figura A.18.: *Se crean dos recursos de conexión a la base de datos.* Uno para poder enviar correos electrónicos *(línea 3-11)* y otro para conectarse con la base de datos *"lportal"*.

A continuación configuramos el archivo `apache-tomcat-7.0.10/conf/server.xml` como se muestra en el Cuadro A.1 para utilizar la biblioteca *APR* con conexiones seguras.

*Líneas 35-36*: Se crea un conjunto de conexiones para que se compartan en diferentes hilos de ejecución.

*Líneas 39-46*: Se crea un conector para que utilice los anteriores hilos de ejecución compartidos.

*Líneas 48-65*: Se hace que el conector creado defina conexiones seguras usando APR en el puerto 8443 cuya ruta del archivo del certificado se especifica en el parámetro *SSLCertificateFile*, esto para que cuando el cliente acceda a la página Web del sistema con conexión SSL se le pueda mostrar un certificado que valida la comunicación con el servidor mediante un canal cifrado.

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <Server port="8005" shutdown="SHUTDOWN">
3
4    <!--APR library loader. Documentation at /docs/apr.html -->
5    <Listener className="org.apache.catalina.core.AprLifecycleListener"
         SSLEngine="on" />
6    <!--Initialize Jasper prior to webapps are loaded. Documentation at /docs
         /jasper-howto.html -->
7    <Listener className="org.apache.catalina.core.JasperListener" />
8    <!-- JMX Support for the Tomcat server. Documentation at /docs/non-
         existent.html -->
9    <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener"
         />
```

```
10    <Listener className="org.apache.catalina.mbeans.
          GlobalResourcesLifecycleListener" />
11
12    <!-- Global JNDI resources
13         Documentation at /docs/jndi-resources-howto.html
14    -->
15    <GlobalNamingResources>
16      <!-- Editable user database that can also be used by
17           UserDatabaseRealm to authenticate users
18      -->
19      <Resource name="UserDatabase" auth="Container"
20                type="org.apache.catalina.UserDatabase"
21                description="User database that can be updated and saved"
22                factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
23                pathname="conf/tomcat-users.xml" />
24    </GlobalNamingResources>
25
26    <!-- A "Service" is a collection of one or more "Connectors" that share
27         a single "Container" Note: A "Service" is not itself a "Container",
28         so you may not define subcomponents such as "Valves" at this level.
29         Documentation at /docs/config/service.html
30     -->
31    <Service name="Catalina">
32
33      <!--The connectors can use a shared executor, you can define one or
            more named thread pools-->
34
35      <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
36          maxThreads="300" minSpareThreads="50"/>
37
38      <!-- A "Connector" using the shared thread pool-->
39      <Connector
40                executor="tomcatThreadPool"
41                port="8080"
42                protocol="org.apache.coyote.http11.Http11AprProtocol"
43                connectionTimeout="20000"
44                redirectPort="8443"
45                acceptCount="100"
46                maxKeepAliveRequests="15"/>
47
48      <!-- Define a SSL HTTP/1.1 Connector on port 8443
49           This connector uses the JSSE configuration, when using APR, the
50           connector should be using the OpenSSL style configuration
51           described in the APR documentation -->
52      <Connector
53                executor="tomcatThreadPool"
54                port="8443"
55                protocol="org.apache.coyote.http11.Http11AprProtocol"
56                connectionTimeout="20000"
57                redirectPort="8443"
58                acceptCount="100"
59                maxKeepAliveRequests="15"
60                SSLCertificateFile="${catalina.base}/conf/certificado.crt"
61                SSLPassword="[Password que tenga el certificado]"
```

```
62                SSLEnabled="true"
63           SSLProtocol="TLSv1+SSLv3"
64                scheme="https"
65                secure="true"/>
66
67
68      <!-- Define an AJP 1.3 Connector on port 8009 -->
69      <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
70
71      <!-- You should set jvmRoute to support load-balancing via AJP ie :
72      <Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
73      -->
74      <Engine name="Catalina" defaultHost="localhost">
75        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
76               resourceName="UserDatabase"/>
77
78        <!-- Define the default virtual host
79             Note: XML Schema validation will not work with Xerces 2.2.
80         -->
81        <Host name="localhost"  appBase="webapps"
82              unpackWARs="true" autoDeploy="true"
83              xmlValidation="false" xmlNamespaceAware="false">
84
85        </Host>
86      </Engine>
87    </Service>
88  </Server>
```

Cuadro A.1.: *Código del archivo server.xml.*

Se pueden crear mas conjuntos de conexiones a bases de datos modificando el archivo apache-tomcat-7.0.10/conf/context.xml como se muestra en la Figura A.19 . Cada etiqueta <Resource ... /> es un conjunto de conexiones a una base de datos diferente.

```
<Context>

    <!-- Default set of monitored resources -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>

    <Resource
        name="jdbc/OtraBase"
        auth="Container"
        type="javax.sql.DataSource"
        driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://localhost/otrabase"
        username="postgres"
        password="postgres"
        testOnBorrow="true"
        validationQuery="SELECT 1"
        testWhileIdle="true"
        timeBetweenEvictionRunsMillis="5000"
        removeAbandoned="true"
        maxActive="50"
    />

    <Resource
        name="jdbc/BaseMicrosoftSQLServer"
        type="javax.sql.DataSource"
        driverClassName="net.sourceforge.jtds.jdbc.Driver"
        url="jdbc:jtds:sqlserver://10.13.xx.xx:1433/BaseMicrosoftSQLServer"
        username="usuario"
        password="contrasena"
        testOnBorrow="true"
        validationQuery="SELECT 1"
        testWhileIdle="true"
        timeBetweenEvictionRunsMillis="5000"
        removeAbandoned="true"
        maxActive = "50"
   />
```

Figura A.19.: *Creación de más conexiones a bases de datos.* Se modifica el archivo context.xml

### A.1.2.5. Utilizando archivo con extensión .pfx para certificado en Tomcat

Convertir ***pfx***[5] a ***PEM*** [6]es correcto al ejecutar *Internet Information Services o IIS (conformado por una serie de servicios para las computadoras que funcionan con el sistema operativo Windows*) con Tomcat ya que este no soporta el certificado *pfx*.

Para convertir un certificado *.pfx* de Microsoft a *PEM* hacemos lo siguiente:

[5]"Un archivo *PFX (Personal Information Exchange Format)*. Puede contener todas las claves privadas, las claves públicas y los certificados. Se almacena en formato binario. Este formato habilita la transferencia de certificados y sus claves privadas correspondientes de una computadora a otra o de una computadora a un medio externo *(removable media)*. El PKCS#12 *(o Public Key Cryptography Standard #12)* es un formato que es apropiado para llevar de un sitio a otro o almacenar y restaurar dicho certificado y sus claves privadas asociadas. Esto se puede hacer entre productos del mismo proveedor o incluso de proveedores distintos." [43]

[6]"Los certificados de firmas digitales se basan en los algoritmos de firma DSA y en el algoritmo RSA para criptografía de llave pública según los algoritmos PKCS. El formato del certificado depende de la aplicación, ya que no hay unanimidad en los estándares de los formatos. Se suele disponer de claves privadas cuando nos referimos a los formatos *PEM* y DER. El formato por defecto para la aplicación OpenSSL es *PEM*. Para aplicaciones Java el formato DER suele encajar mejor con las necesidades de la aplicación en cuanto a la importación de claves privadas y certificados. El archivo *PEM* puede contener todas las claves privadas *(RSA y DSA)*, las claves públicas *(RSA y DSA)* y los certificados *(x509)*. Este es el formato por defecto de OpenSSL. Almacena los datos en formato DER codificado Base64, entre cabeceras ASCII, de modo que es apropiado para transferencias en modo texto entre sistemas. Su extensión suele ser .pem o .crt." [43]

- Ponemos el certificado .pfx en `apache-tomcat-7.0.10/conf/` y en esta carpeta ejecutamos en consola el comando:
  `openssl pkcs12 -in certificado_intranet.pfx -out certificado_intranet.crt -nodes`

  Otra forma seria convertirlo directamente en páginas Web como `https://www.sslshopper.com/ssl-converter.html` que convierten de un `.pfx` a `PEM`.

## A.2. Configuración inicial del portal

En esta sección se mostrará la configuración de ciertos aspectos del portal como lo es la autenticación de usuarios, cambiar el nombre del sitio Web y corregir diversos problemas de Liferay cuando no se configura correctamente como ver todos los niveles y subniveles de las páginas que se crean en el portal.

### A.2.1. Idioma

Entramos al sistema como *test@liferay.com* y contraseña *test* que es la que está por defecto en el administrador del sistema, posteriormente nos dirigimos al panel de control que se encuentra en la barra superior del portal y hacemos lo que se muestra en la Figura A.20 para modificar las preferencias de presentación en nuestra cuenta:

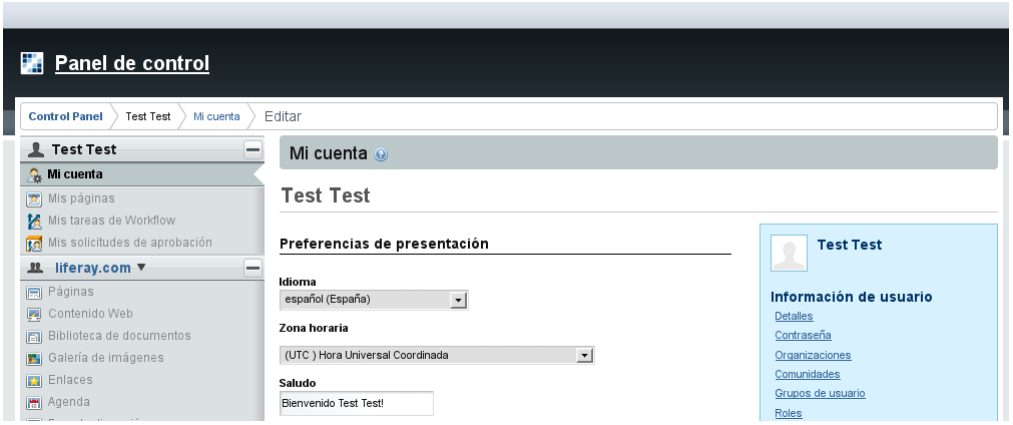


Figura A.20.: *Modificación de las preferencias del idioma en nuestra cuenta.*

Y continuamos modificando el idioma para el portal en general como se muestra en la Figura A.21:


Figura A.21.: *Modificación del idioma en el sistema en general.*

### A.2.2. Opciones de autenticación *("login")*.

La *"autenticación"* es un modo de asegurar que los usuarios son quién ellos dicen que ellos son, que el usuario que intenta realizar funciones en un sistema es de hecho el usuario que tiene la autorización para hacer eso. Los métodos de autenticación están en función de lo que se utiliza para la verificación de los usuarios para que puedan acceder al sistema por ejemplo, por nombre de usuario en el cual debemos ingresar *usuario* y *contraseña* para acceder al sistema. [39].

Podemos modificar el método de autenticación que va a utilizar el portal. La Figura A.22 muestra las opciones de autenticación que se pueden escoger señalando las casillas.


Figura A.22.: *Opciones de autenticación de usuarios.*

Después cambiamos a la pestaña *OpenID* y deshabilitamos la casilla de habilitado *(Para en lugar de ingresar con el número de identificador de usuario, utilizar un nombre de usuario si se desea).*

### A.2.3. Cambiar el nombre de la compañía y correo electrónico

Vamos a `Panel de Control - Portal - Configuración` y modificamos los datos de nombre de compañía y correo electrónico de esta con el fin de poder utilizar Liferay como servidor de correo con el nombre de la compañía, asignado como se muestra en la página: `http://www.liferay.com/es/community/wiki/-/wiki/Main/Mail+Integration`.

### A.2.4. Deshabilitar caché del servidor *(para ambientes de desarrollo como Netbeans o Eclipse)*

Deshabilitar el caché del servidor es útil a la hora de desarrollar, ya que cada cambio que hacemos al sistema se muestra a continuación y así observamos como van quedando las modificacones que hacemos en el portal y los portlets. Para esto se requiere estar en la siguiente ruta:

```
liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/applications/
liferay-portal/WEB-INF/lib/portal-impl.jar.
```

Donde deshabilitaremos el caché de documentos estáticos, para lo cual habrá que agregar a `portal.properties` la propiedad `CacheFilter` mostrada en la Figura A.23. Asignamos el valor a falso *(false)* para dehabilitar el caché, de lo contrario no se verán los cambios realizados en archivos de *JavaScript* o *CSS* aunque se limpie el caché del navegador.

```
5193    #
5194    # The cache filter will cache content. See ehcache.xml to modify the cache
5195    # expiration time to live.
5196    #
5197    com.liferay.portal.servlet.filters.cache.CacheFilter=false
```

Figura A.23.: *Deshabilitando el caché para probar nuestras aplicaciones.*

Esta opción sólo se debe habilitar en ambientes de desarrollo y no en producción, ya que hará que el usuario del sistema en producción pueda realizar acciones sin tener que volver a cargar la misma configuración del sistema ha como la ha visto por última vez en su último ingreso. [62].

En producción se debe anular la implementación de un portlet *(undeploy)* antes de volver a desplegar la implementación del portlet *(deploy)*, de lo contrario el portal lanzará errores a la hora de volver a subir los cambios que se han hecho al portlet.[62].

Otra forma de hacer esto aunque menos segura es crear un archivo llamado portal-ext.properties en:
`liferay-portal-6.0.5/glassfish-3.0.1/domains/domain1/applications/liferay-portal/WEB-INF/classes/` y poner los valores que se requieren modificar [62].

### A.2.5. Corregir error de Liferay que no permite ver los subniveles completos del menú

Este es un error que aún no se ha corregido en la versión 6.0.5 de *Liferay*, sin embargo en este trabajo se corrigió. Cabe mencionar que durante la realización de esta tesis se llegó a corregir al ver que no se podía acceder a un subnivel del menú de páginas de Liferay.

Se puede observar que al crear una subpágina de la página Portlet-1 llamada Portlet-1.1, esta ya no aparece. Al corregir este error, aparece la página Portlet-1.1 tal como se muestra en la Figura A.24 más adelante.

Apagamos el servidor y abrimos el documento siguiente para modificar la plantilla que muestra los subniveles de las páginas de Liferay:
`glassfish-3.0.1/domains/domain1/applications/liferay-portal/html/themes/classic/templates/navigation.vm.`

Reemplazamos su contenido por el siguiente código mostrado en el Cuadro A.1, el cual crea el tercer submenú utilizando el marco de trabajo llamado *Velocity* de Java que usaron los creadores de Liferay para este fin y cuya extensión de archivos es .vm, la guía de usuario se puede encontrar en la página siguiente para cualquier aclaración:
`http://velocity.apache.org/engine/devel/translations/user-guide_es.html`

El código del Cuadro A.1 se basa en el creado anteriormente por los creadores de Liferay, sin embargo se hizo modificacion para generar el tercer submenú de un nivel de páginas de Liferay.

*Líneas 1-8.* Se cambia la clase del CSS por cada página del menú seleccionada.

*Líneas 10-11.* Al seleccionar una página, esta nos puede enviar a su URL.

*Líneas 13-20.* Al seleccionar una página del menú principal, se muestran los niveles que contiene.

*Líneas 20-25.* Al seleccionar un nivel que contiene el menú, se muestran los subniveles de este menú.

*Líneas 27-31.* Se cambia la clase del CSS por cada página del menú del subnivel seleccionado.

*Líneas 33-35.* Al seleccionar una página de un subnivel, esta nos puede enviar a su URL.

```
1 <div id="navigation" class="sort-pages modify-pages">
2 <ul>
```

```
3  #foreach ($nav_item in $nav_items)
4  #if ($nav_item.isSelected())
5  #set ($nav_item_class = "selected")
6  #else
7  #set ($nav_item_class = "")
8  #end
9
10 <li class="$nav_item_class">
11 <a style="font-weight:bold;font-size:12px;color:white" href="$nav_item.
       getURL()" $nav_item.getTarget()><span>$nav_item.getName()</span></a>
12
13 #if ($nav_item.hasChildren())
14 <ul class="child-menu" >
15 #foreach ($nav_child in $nav_item.getChildren())
16 #if ($nav_child.isSelected())
17 #set ($nav_child_class = "selected")
18 #else
19 #set ($nav_child_class = "")
20 #end
21
22 <li class="$nav_child_class" >
23 <a href="$nav_child.getURL()" onmouseover="if(this.nextSibling){ var n=this
       .nextSibling; if(n.nodeType != 1) n = n.nextSibling; if(n) if(n.style.
       display=='block') n.style.display='none'; else n.style.display='block';
        }" $nav_child.getTarget()>$nav_child.getName()</a>
24 #if ($nav_child.hasChildren())
25 <ul class="child-menu" style="display:none;margin-left:100%;margin-top:-30
       px;width:80%;z-index:9999;" >
26 #foreach ($nav_childjr in $nav_child.getChildren())
27 #if ($nav_childjr.isSelected())
28 #set ($nav_childjr_class = "selected")
29 #else
30 #set ($nav_childjr_class = "")
31 #end
32
33 <li class="$nav_childjr_class">
34 <a href="$nav_childjr.getURL()" $nav_childjr.getTarget()>$nav_childjr.
       getName()</a>
35 </li>
36 #end
37 </ul>
38 #end
39 </li>
40 #end
41 </ul>
42 #end
43 </li>
44 #end
45 </ul>
46 </div>
```

Cuadro A.2.: *Código del archivo navigation.vm*

El resultado se puede apreciar en la Figura A.24.


Figura A.24.: *Se muestran todos los subniveles de los menús.* En este ejemplo las páginas del subnivel de la página Portlet-1 es Portlet-1.1.

## A.3. Modificación del tema de Liferay

La modificación es útil cuando se requiere cambiar la presentación del portal y no utilizar el que viene por omisión en Liferay.

La modificación del tema se puede realizar mediante la descarga de otro tema en el mismo Liferay, o de una manera mas rápida y personalizada como se mostrará en las siguientes subsecciones.

### A.3.1. Modificación del esquema del sitio

Al tener en funcionamiento el servidor se abre la página principal del sitio en cuestión, con ello podemos observar como van quedando las modificaciones que hacemos.

Al comentar la línea de `@import url(custom.css);` que se encuentra en `glassfish-3.0.1/domains/domain1/applications/liferay-portal/html/themes/classic/css/main.css` se pueden observar los cambios de manera automática aunque el servidor se encuentre en ejecución en ese momento.

Al haber terminado de realizar los cambios esa misma línea, se debe descomentar y repetir la misma acción por cada cambio requerido.

A continuación el usuario se podrá referir a la siguiente carpeta donde se guardan las hojas de estilo en cascada o CSS para modificar la apariencia del portal.
`glassfish-3.0.1/domains/domain1/applications/liferay-portal/html/themes/classic/css/`

El primer archivo que explicaremos será *custom.css* ya que es el que establece la apariencia a todo el sitio.

En los bloques de código que dicen `#banner {...}` y `#banner h1.logo {...}` se establece la posición y características que tendrá la cabecera que se verá en el sitio.

Por ejemplo al modificar el código del *CSS* en la propiedad *#banner* se puede observar que la cabecera ha cambiado de tamaño, fondo y altura.

```
1- #banner {
2- /*background: none;
3- height: auto; */
4- background: transparent url(../images/common/banner_bg.jpg) repeat 0 0;
5- height: 100px;
6- }
```

En la línea 3 se quita la altura automática de la cabecera para ajustarla en la línea 5 al 100 %. La línea 4 muestra la imagen cuyo nombre es `banner_bg.jpg` en la cabecera del sitio.

En los bloques de código siguientes, modificamos el color del fondo del sitio de azul claro a gris claro:

```
1- body {
2- /*-- comentamos background: #EEF0F2; **/
3- background: #F2F2F2;
4- font-size: 11px;
5- }
```

En la línea 2 se comenta el fondo de azul claro y en la línea 3 se hace el fondo del sitio a gris claro. La línea 4 solamente cambia el tamaño de la letra del sitio a 11 píxeles.

En los bloques que dicen *#navigation* establecemos el color de la barra de navegación, es decir del menú para ir a las diferentes páginas del sitio.

De forma similar con conocimiento en diseño para presentación de páginas Web se puede modificar este archivo .css para crear una diferente presentación del sitio.

Cerramos el archivo *custom.css*.

Vamos a `glassfish-3.0.1/domains/domain1/applications/liferay-portal/html/themes/classic/images/common` y sobreescribimos o reemplazamos `banner_bg.jpg` a nuestro gusto para cambiar la imagen de la cabecera de nuestro sitio.

### A.3.2. Cambio del logo del sitio.

Esta sección menciona la forma para cambiar en el sitio el logo de Liferay por el de nuestra compañía o el que se requiera.

Vamos a `Panel de Control - Portal - Configuración` y seleccionamos `Preferencias de presentación`.

Esto nos mostrará otra ventana donde podremos cambiar el logo solo seleccionando la opción *"cambiar"*.

Aplicamos los cambios oprimiendo el botón de guardar dentro del panel de configuración. Actualizamos el sitio y se verán los cambios inmediatamente al regresar a la página principal del portal tal como se muestra en la Figura A.25.

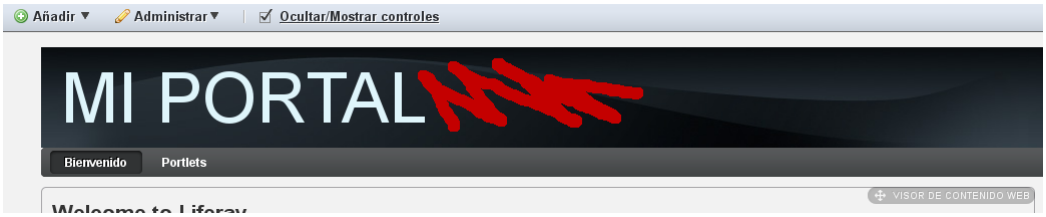


Figura A.25.: *Cambio del logo de nuestro sitio.*

## A.4. Configuración del Entorno de Desarrollo Integrado de Eclipse para desarrollar portlets

Para desarrollar cualquier software o programa siempre es cómodo utilizar un entorno de desarrollo el cual nos facilite la programación. En el caso de los portlets podemos utilizar **Eclipse**[7].

### A.4.1. Configuración de los archivos

Descargamos de la página de Liferay la versión más actual del complemento *sdk* que es el entorno de desarrollo proporcionado para la creación de nuevos plugins: portlets, temas y plantillas exclusivos para el gestor de portales de Liferay y haremos uso de este para crear nuestro portlet; en este caso es `liferay-plugins-sdk-6.0.5.zip` y seguimos los siguientes pasos [62] .

1. Descomprimimos la carpeta dentro de `liferay-portal-6.0.5/`
2. Cambiamos el nombre del archivo `liferay-plugins-sdk-6.0.5/build.properties` por `liferay-plugins-sdk-6.0.5/build.`*[nombre de tu usuario en linux]*`.properties`.

[7] *"Eclipse* es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados *(del inglés IDE)*, como el IDE de Java llamado Java Development Toolkit *(JDT)* y el compilador *(ECJ)* que se entrega como parte de Eclipse *(y que son usados también para desarrollar el mismo Eclipse)*". [32]

3. Abrimos el archivo con un editor de texto.

4. Comentamos la sección que dice *# Specify the paths to an unzipped Tomcat bundle* que en español quiere decir *"especifica las rutas a un archivo descomprimido de* Liferay *con Tomcat equipado"*, esta sección se deja descomentada si se trabaja con Tomcat y se puede pasar al punto 6.

5. Descomentamos y modificamos la parte de *# Specify the paths to an unzipped Glassfish bundle* que en español quiere decir *"especifica las rutas a un archivo descomprimido de Liferay con Glassfish equipado*" como se muestra en la Figura A.26 para que cuando compilemos el portlet se instale directamente a Liferay.

```
26    #
27    # Specify the paths to an unzipped Glassfish bundle.
28    #
29    app.server.type=glassfish
30    app.server.dir=${project.dir}/../glassfish-3.0.1
31    app.server.deploy.dir=${app.server.dir}/domains/domain1/autodeploy
32    app.server.lib.global.dir=${app.server.dir}/domains/domain1/lib
33    app.server.portal.dir=${app.server.dir}/domains/domain1/applications/liferay-portal
```

Figura A.26.: *Especificamos que usamos Glassfish.* Especificamos la ruta respectiva a los directorios de Liferay de forma similar a como se muestra en el ejemplo.

6. Copiamos los archivos `build-common.xml`, `build-common-plugin.xml` de `liferay-plugins-sdk-6.0.5/` y `build-common-portlet.xml` de `liferay-plugins-sdk-6.0.5/portlets` al *workspace* o *espacio de trabajo* que utilizaremos para poner las aplicaciones que desarrollemos con Eclipse, esto es donde se ubicarán los proyectos *(código de los portlets)*. Este espacio de trabajo se crea al correr el IDE de *Eclipse*, este pregunta al inicio donde se encuentra localizado nuestro *espacio de trabajo*.

7. Modificamos el `build-common-portlet.xml` que se encuentra en el *espacio de trabajo* que se creó al correr Eclipse de forma similar a la Figura A.27 para que al compilar utilice bibliotecas que vienen con los complementos del *sdk* de Liferay.

```
1  <?xml version="1.0"?>
2
3  <project name="build-common-portlet">
4      <property name="project.dir" value="/home/arpalaci/Tesis/portal/liferay-portal-6.0.5/liferay-plugins-sdk-6.0.5" />
5
6      <import file="build-common-plugin.xml" />
7  </project>
```

Figura A.27.: *Especificamos la ruta a los complementos sdk de Liferay.* [62]

### A.4.2. Modificación del entorno de trabajo para utilizar Eclipse IDE

Tenemos que tener instalado Eclipse y ***Ant***[8]para continuar con este apartado.

[8] *"Apache Ant* es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción *(build)*. Es, por tanto, un

Al importar los proyectos desde el disco con el menú de `Import - General - Existing Projects from workspace`, se verá que cada uno contiene un `build.xml`, ese `build.xml` pasarlo a la ventana de *Ant* desde el menú `window-Show View-Ant`.

En el menú que se muestra en la ventana de *Ant* únicamente dar click sobre `deploy` *(desplegar)*, esto automáticamente hará que el proyecto se compile y se instale en el portal de Liferay como un portlet.

En si lo que hace esto es crear un archivo *.war*[9] y enviarlo a la carpeta `autodeploy/` la cual lo instala en Liferay para que a continuación la podamos como un *portlet*.

Al abrir Eclipse lo primero que vamos a hacer es mostrar la ventana de *Ant* para al seleccionar el archivo build.xml de esta ventana, automáticamente se instale el portlet en el portal, esto al seleccionar `window - Show view - Ant`.

Para evitar problemas con ciertas bibliotecas de Java como `CachedRowSetImpl` en Eclips. Existen dos formas, vamos a `Windows - Preferences - Java - Compiler - Errors/Warnings - Deprecated and restricted API - Forbidden reference (access rules)` y cambiar la variable para ignorar avisos, la cual se debe cambiar a `warning`. Y la otra forma es que al crear el proyecto como lo vamos a ver más adelante, borrar la biblioteca JRE System Library e importarla de nuevo.

### A.4.3. Consejos para un buen desarrollo utilizando el IDE de Eclipse

Si se requiere del sistema de gestión de versiones llamado ***Subversion***[10], se recomienda ***Subclipse*** [11] para comparar las diferentes versiones de nuestro código utilizando nuestro IDE y tener un mejor control sobre los cambios de este.

Como vamos a programar en *Java*, lo mejor es utilizar una herramienta para encontrar errores en nuestro código como *Findbug*:

`http://findbugs.sourceforge.net/downloads.html`

Abrimos Eclipse y vamos a la pestaña de `help - install new software`, añadimos la fuente de Findbug, ver la Figura A.28 y oprimimos instalar, de forma similar podemos instalar Subclipse u otras extensiones para *Eclipse*. Así los utilizaremos en nuestro *IDE de Eclipse*.

---

software para procesos de automatización de compilación, similar a *Make* para uso en el lenguaje C y desarrollado en lenguaje Java." [28]

[9] *"Archivo War* o *(Web application archive o archivo de aplicación Web)* es un archivo para distribuir una colección de aplicaciones hechas con el lenguaje de programación Java que constituyen una aplicación Web". [18]

[10] *"Subversion* es un sistema de control de versiones. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras". [29]

[11] *"Subclise* es el componente más utilizado para incluir Subversion a Eclipse". [13]

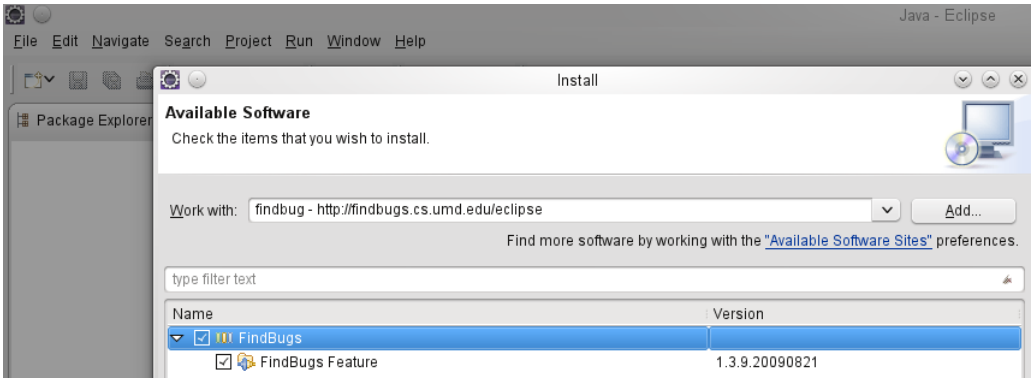


Figura A.28.: *Instalando extenciones para el IDE Eclipse.*

Otro software recomendable es ***Aptana*** [12] el cual genera referencia a clases automaticamente al utilizar un lenguaje de programación y así facilitar la programación.

Una vez instalado el componente de *Aptana* en Eclipse tenemos que configurarlo, para ello vamos a `Window - Preference - Editors - Java Script - Code Assist` y seleccionamos la casilla de `JQuery` o las bibliotecas de *Java Script* que vamos a utilizar como se muestra en la Figura A.29.

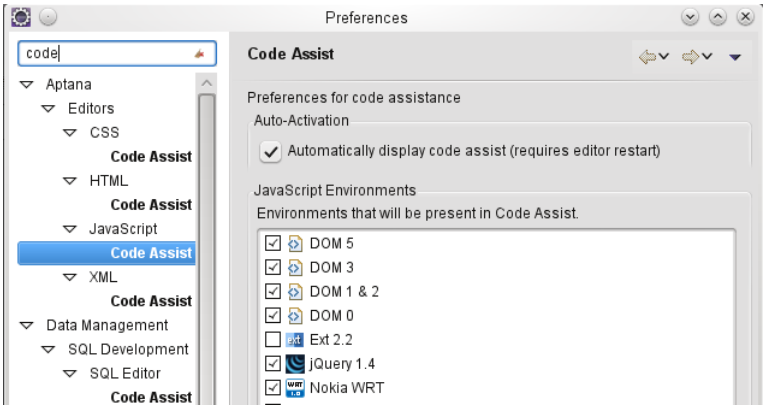


Figura A.29.: Utilizando componentes para asistencia de código para hacer aplicaciones Web al instalar *Aptana*.

A continuación seleccionamos `Editors - File Associations - *.js`. Seleccionamos `Aptana JS Editor` para tener como editor por omisión *Aptana* para todos los archivos *JavaScript* que tenga el proyecto Figura A.30.

[12] "*Aptana* es un software para desarrollar aplicaciones Ajax y que funciona como un plugin de Eclipse, provee soporte para lenguajes como: *Php, Python, Ruby, CSS, Ajax, HTML y Adobe AIR*". [31]

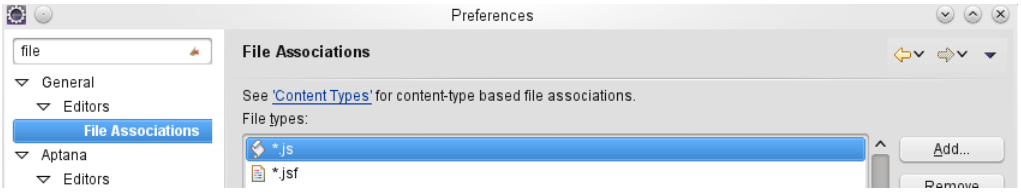


Figura A.30.: *Uso de Aptana para todos los archivos de JavaScript que utilizan la extensión js.*

Solo resta reiniciar Eclipse y cada vez que editemos un archivo *.js, html o css* podremos utilizar Aptana como se muestra en la Figura A.31.


Figura A.31.: *Aptana en funcionamiento dentro de Eclipse.*

Siempre es bueno al programar hacer un código sencillo de comprender y que respete las convenciones de programación o estándares de código para el lenguaje que se utilizará al desarrollar las aplicaciones y poder hacer mas entendible su propósito, por ello al programar portlets es bueno seguir las convenciones de Java en su desarrollo para poder hacer cambios al mismo sin problemas como su difícil entendimiento [16].

Una convención en general para la mayoría de lenguajes de programación es respetar un delimitador de números de línea. Para configurarlo en Eclipse solo basta con ir a `Window - Preferences - General - Editors - Text Editor` y activar la casilla que dice `Show print margin`. Ahí mismo en `Text Editor` deshabilitamos la corrección ortográfica ya que los lenguajes traen por defecto el lenguaje inglés para su sintaxis, por lo que es preferible ir a `Spelling` y deshabilitamos la casilla de `Enable spell checking`.

## A.5. Scripts útiles para administrar el portal

Cuando trabajamos con servidores en producción, algunas veces es necesario darles mantenimiento, por ejemplo para cambiar la versión del servidor o del portal. Lo que se verá a continuación es la forma de detener todos los procesos que genera el portal en su ejecución, además de respaldar los datos que contiene el portal para su uso posteriormente.

Para crear scripts para administrar el servidor, debemos instalar *mawk* que es un interprete para el lenguaje de programación ***AWK***[13], el cual se usará en conjunto con el

[13] "*AWK* es un lenguaje de programación diseñado para procesar datos basados en texto, ya sean archivos o flujos de datos. AWK es ejemplo de un lenguaje de programación que usa ampliamente el tipo de datos de listas asociativas *(es decir, listas indexadas por cadenas clave)*, y expresiones regulares. Este

lenguaje para interpretar comandos en terminal , por ejemplo ***bash***[14].

Los siguientes códigos se deben ejecutar como scripts en *bash*, ya sea como administrador o tiendo un usuario con permisos de ejecución de scripts [15], para esto, el script debe tener permisos de ejecución [45], estos permisos se pueden cambiar utilizando el comando de terminal llamado ***chmod*** [15]. El primer script inhabilita los procesos del servidor *Glassfish* para que no hagan uso ineficiente de la memoria de la máquina donde se encuentra, ver el Cuadro A.2, el segundo script reinicia el servidor de *Glassfish,* ver el *Cuadro A.3,* y el tercer script respalda la base de datos del portal para prevenir pérdida de datos en algún futuro, ver el Cuadro A.4.

A continuación se explica el código del script de bash para matar los procesos de Glassfish mostrado en el Cuadro A.2.

*Línea 1.* Señalamos que el script usará el lenguaje para interpretar órdenes llamado *bash.*

*Línea 2.* Lo que hace es que escanea los procesos que se están ejecutando con el comando `ps`, después filtra los que digan *"glassfish*" con ayuda del comando `grep` y con el lenguaje *AWK* me da el segundo campo de los filtros anteriores, los cuales son los identificadores de los procesos del servidor *Glassfish* que se guardan en la variable "`a`".

*Línea 3.* Se imprime lo que contiene la variable *"`a`"* con el comando `echo` solamente para ver que procesos tiene en ejecución *Glassfish* aunque no es necesario.

*Línea 4.* Destruye los procesos con el comando `kill` que ya estaban guardados en la variable *"`a`"*. Se puede reemplazar el nombre de *"glassfish"* por *"mysql"* o *"postgresql"*, dependiendo del proceso que se desee eliminar.

```
1 #!/bin/sh
2 a=‘ps aux | grep glassfish | mawk ’{print $2}’‘
3 echo $a
4 ‘kill -9 $a‘
```

Cuadro A.3.: *Script de bash para matar los procesos de Glassfish.* Ayuda a que los procesos no queden en la memoria de la máquina en que se encuentra el servidor *Glassfish.*

es un programa "Hola mundo" muy simple escrito en `AWK: BEGIN { print "Hola mundo!"; exit }`’’ [15].

[14] *"Bash* es un interprete de comandos. Está basado en la terminal de Unix y es compatible con POSIX. Fue escrito para el proyecto GNU y es el intérprete de comandos por defecto en la mayoría de las distribuciones de Linux. Su nombre es un acrónimo de Bourne-Again Shell *(otra terminal bourne)* — haciendo un juego de palabras *(born-again que significa renacimiento)* sobre el Bourne shell *(sh)*, que fue uno de los primeros intérpretes más importantes de Unix" [15] [45].

[15] "El nombre del comando c*hmod* es e la abreviatura de *Change Mode (cambio de modo)*, una orden que permite otorgar o quitar permisos a un usuario concreto o a un grupo de usuarios del sistema y cuya estructura, desde la línea de comando, sigue el siguiente patrón: chmod [who] [opcion][permiso] [nombre_archivo]" [15].

A continuación se explica el código del script de bash para reiniciar el servidor, así como la base de datos mostrado en el Cuadro A.3.

*Líneas 1-2.* Detiene el servidor *Glassfish* y el manejador de bases de datos *PostgreSQL*, que en este caso contiene la base de datos del portal Liferay.

*Línea 3.* Se ejecuta el script del Cuadro A.1.

*Líneas 4-5.* Se inicia el manejador de bases de datos *PostgreSQL* y el servidor de *Glassfish*.

```
1 sh /...ruta al archivo que inicia Glassfish.../asadmin stop-domain
2 /etc/init.d/postgresql stop
3 sh /...ruta al script que mata los procesos...
4 /etc/init.d/postgresql start
5 sh /...ruta al archivo que inicia Glassfish.../asadmin start-domain
```

Cuadro A.4.: *Script de bash para reiniciar el servidor, así como la base de datos.* Detiene el servidor *Glassfish* y mata sus procesos de forma segura para que no gasten memoria de la máquina donde se encuentra, por último lo inicia.

A continuación se explica el código del script para respaldar la base de datos de Liferay y prevenir pérdida de datos en algún futuro mostrado en el Cuadro A.4.

Línea 2. Reemplaza el respaldo por el anterior, borrando todos los archivos de respaldo con extensión *.sql* si fue modificado hace 8*24 , osea hace 192 horas utilizando el subcomando `-mtime` del comando `find` y para borrar los archivos con el comando `rm`, si no lo deja y pone el nuevo respaldo con extensión *.sql*.

Línea 3. Respalda todas las bases de datos de que usan *PostgreSQL* nombrándolos primero Respaldosem-lportal- y a continuación por fecha de respaldo usando el comando `date` y dándoles la extensión *.sql*.

```
1 #!/bin/sh
2 find /...ruta a los respaldos.../respaldos -name \*.sql -mtime +8 -exec rm
      {} \;
3 /...ruta a los archivos de instalación de PostgreSQL.../postgresql-9.0/src/
      bin/pg_dump/pg_dump -U postgres -d [contraseña de postgres] >
      Respaldosem-lportal-$(date +%Y%m%d)-$(date +%H%M%S).sql
```

Cuadro A.5.: *Script para respaldar la base de datos de Liferay y prevenir pérdida de datos en algún futuro.*

Para ejecutar scripts el momento que se guste, hacer un "cron"[16], ya sea para hacer los respaldos o para reiniciar el servidor durante ciertas horas cuando nadie hace uso de éste.

[16]"*Un cron* es un administrador regular de procesos en segundo plano *(demonio)* que ejecuta procesos o guiones a intervalos regulares *(por ejemplo, cada minuto, día, semana o mes)*. Los procesos que deben ejecutarse y la hora en la que deben hacerlo se especifican en el archivo crontab. El nombre cron viene del griego chronos que significa "tiempo".

Cron se podría definir como el 'equivalente' a Tareas Programadas de Windows" [45].

# Bibliografía


[1] González Alonso, Jorge. Art. distintos tipos de CMS. *Instituto Superior Pedagógico de la Habana y posgrados en el Centro Nacional de Investigaciones Científicas de Cuba*, 2009.

[2] Portlet API. Portlet API. Última consulta 7/15/2012, `http://www.bluesunrise.com/portlet-api/`.

[3] Ashley, Mike. Guía de "Gnu Privacy Guard". Última consulta 7/15/2012, `http://www.gnupg.org/gph/es/manual.html`.

[4] Bales, Donald. *Java Programming with Oracle JDBC.* Editorial O'Reilly, 1er ed, 450, 2002.

[5] Bell, Michael. Art. introduction to Service Oriented Modeling. *Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley and Sons*, 2008.

[6] Berners-Lee, Tim. Art. realising the full potential of the web. *Based on a talk presented at the W3C meeting, London, w3c.org*, 1997.

[7] Callaghan, James. *Inside Intranets and Extranets: Knowledge Management and the Struggle for Power.* Editorial Palgrave Macmillan, 1er ed, 259, 2002.

[8] Cavanaugh, Nate. Using Jquery (or any Javascript library) in Liferay. Última consulta 7/15/2012, `http://www.liferay.com/web/nathan.cavanaugh/blog/-/blogs/using-jquery-or-any-javascript-library-in-liferay-6-0`.

[9] Liferay community. Document Library Portlet. Última consulta 7/15/2012, `http://www.liferay.com/es/community/wiki/-/wiki/Main/Document+Library+Portlet`.

[10] PostgreSQL community. Postgresql RDBMS. Última consulta 7/15/2012, `http://www.postgresql.org/`.

[11] PostgreSQL community. Postgresql RDBMS. Última consulta 7/15/2012, `http://www.2ndquadrant.com/`.

[12] Springsource community. What is Spring. Última consulta 7/15/2012, `http://www.springsource.org/`.

[13] Tigris community. Subclipse. Última consulta 7/15/2012, `http://subclipse.tigris.org/`.

[14] Adobe company. Adobe Software. Última consulta 7/15/2012, `http://www.adobe.com`.

[15] Cooper, Mendel. *Advanced Bash-Scripting Guide*. Editorial tldp.org, 1er ed, 696, 2011.

[16] Oracle corp. Code Conventions for the Java Programming Language. Última consulta 7/15/2012, `http://www.oracle.com/technetwork/java/codeconv-138413.html`.

[17] Oracle corp. Glassfish Server Open Source Edition 3.1 Installation Guide. Última consulta 7/15/2012, `http://glassfish.java.net/docs/3.1.1/installation-guide.pdf`.

[18] Oracle corp. Java Servlet Technology. Última consulta 7/15/2012, `http://www.oracle.com/us/technologies/java/servlet-138661.html`.

[19] Oracle corp. Oracle JDBC. Última consulta 7/15/2012, `http://docs.oracle.com/javase/6/docs/technotes/guides/jdbc/`.

[20] Oracle corp. Oracle RDBMS. Última consulta 7/15/2012, `http://www.oracle.com`.

[21] Oracle corp. Sun Glassfish Enterprise Server v3 Administration Guide. Última consulta 7/15/2012, `http://docs.oracle.com/cd/E19226-01/820-7692/index.html`.

[22] Oracle corp. The Java EE 5 Tutorial, Securing Java EE Applications. Última consulta 7/15/2012, `http://docs.oracle.com/javaee/5/tutorial/doc/bnbyk.html`.

[23] Oracle corp. JSR 289: SIP Servlet v1.1. Última consulta 7/15/2012, `http://www.oracle.com/technetwork/java/jsr/jsr-289-089896.html`.

[24] Ponce de Leon, Harald. Comercio electrónico y administración online. Última consulta 7/15/2012, `http://www.oscommerce.com/`.

[25] Empson, Rip. Joomla Quietly Crosses 23 Million Downloads, Now Powering Over 2,600 Government Sites. Última consulta 7/15/2012, `http://techcrunch.com/2011/06/11/`.

[26] Faithfull, Woric. Art. woric faithfull; using xslt to make websites. *woric.net*, 2007.

[27] Association for Information and Image Management (AIIM). Art. what is Enterprise Content Management (ECM). *AIIM*, 2011.

[28] Apache foundation. Ant. Última consulta 7/15/2012, `http://ant.apache.org`.

[29] Apache foundation. Subversion. Última consulta 7/15/2012, `http://subversion.apache.org`.

[30] Apache foundation. Why is it not possible to use Name-Based Virtual Hosting to identify different SSL virtual hosts? Última consulta 7/15/2012, `http://httpd.apache.org/docs/2.0/ssl/ssl_faq.html`.

[31] Aptana foundation. Aptana. Última consulta 7/15/2012, `http://aptana.com/`.

[32] Eclipse foundation. Eclipse ide. Última consulta 7/15/2012, `http://www.eclipse.org`.

[33] OFC foundation. Open Flash Chart. Última consulta 7/15/2012, `http://teethgrinder.co.uk/open-flash-chart-2/`.

[34] O. Freier, Alan. Reporte. the Secure Sockets Layer (SSL) Protocol Version 3.0. Technical Report ISSN: 2070-1721, Netscape Communications, 2011.

[35] Frérejean, Erik. Seo urls. Technical report, phpBB .ªrsia"3.2 Development, Area51 phpBB, 2011.

[36] García, Andy. El mejor CMS para blogs. Última consulta 7/15/2012, `http://blog.andy21.com/2011/el-mejor-cms-para-blogs/`.

[37] Intalio inc. BPMS products. Última consulta 7/15/2012, `http://www.intalio.com/`.

[38] Liferay inc. Casos de éxito. Última consulta 7/15/2012, `http://www.liferay.com/es/products/liferay-portal/stories`.

[39] Liferay inc. Liferay Portal 6.0 - Administration Guide. Última consulta 7/15/2012, `http://www.liferay.com/es/documentation/liferay-portal/6.0/administration`.

[40] Liferay inc. DTD for the Liferay Plugins XML. Última consulta 7/15/2012, `http://docs.liferay.com/portal/6.0/definitions/liferay-plugin-package_6_0_0.dtd.html`.

[41] Joseph, Joshy. Art. handling attachments in soap. *IBM*, February 2002.

[42] RSA Laboratoies. PKCS 12 : Personal Information Exchange Syntax Standard. Última consulta 7/15/2012, `http://www.rsa.com/rsalabs/node.asp?id=2138`.

[43] RSA laboratories. What are certificates? Última consulta 7/15/2012, `http://www.rsa.com/rsalabs/node.asp?id=2277`.

[44] Lennon, Joe. Compare Javascript frameworks. Última consulta 7/15/2012, `http://www.ibm.com/developerworks/java/library/wa-jsframeworks/index.html?ca=drs-`.

[45] Newham, Cameron. *Learning the bash Shell: Unix Shell Programming (In a Nutshell (O'Reilly))*. Editorial O'Reilly, 3ra ed, 360, 2005.

[46] O'Reilly, Tim. Art. qué es web 2.0. patrones del diseño y modelos del negocio para la siguiente generación del software. *Artículos de la Sociedad de la Información*, 2006.

[47] Shan, Tony. Reporte. taxonomy of java web application frameworks. Technical Report ISBN:0-7695-2645-4, Proceedings of 2006 IEEE International Conference on e-Business Engineering (ICEBE 2006), IEEE Computer Society Washington, DC, USA, 2006.

[48] Silberschatz. *Fundamentos de bases de datos.* Editorial McGraw-Hill, 5a ed, 962, 2006.

[49] Spring source community. Spring Framework. Última consulta 7/15/2012, `http://www.springsource.org/`.

[50] Svarre, Klaus. Art. content Management System (CMS). *TechTarget*, December 2000.

[51] Tsai, Michael. Art. the personal computing paradigm. *ATPM*, 2011.

[52] Venners, Bill. Java's garbage-collected heap. An introduction to the garbage-collected heap of the Java Virtual Machine. Última consulta 7/15/2012, `http://www.javaworld.com/javaworld/jw-08-1996/jw-08-gc.html`.

[53] Brocke, Jan vom. *Handbook on Business Process Management, Volumen 1.* Editorial Springer, 1er ed, 612, 2010.

[54] The World Wide Web Consortium (W3C). Web standards. Última consulta 7/15/2012, `http://www.w3.org/`.

[55] Cordoba Torrecilla, Josune y Cuesta Morales, Pedro. Art. adaptando un sistema de wikis para uso educativo. *Departamento de Informática*, 2009.

[56] Cunningham y Cunningham. API. Última consulta 7/15/2012, `http://c2.com/cgi/wiki?ApiVsProtocol`.

[57] Brittain, Jason y Darwin, Ian F. *Tomcat: The Definitive Guide (Spanish Edition).* Editorial O'Reilly, 2da ed, 496, 2008.

[58] Bibeault, Bear y Katz, Yehuda. *JQuery in Action.* Editorial Manning, 2da ed, 376, 2010.

[59] Bauer, Christian y King, Gavin. *Hibernate in Action (In Action series).* Editorial Manning, 1er ed, 400, 2005.

[60] Bernard, Emmanuel, Ebersole, Steve y King, Gavin. Hibernate Entitymanager. Última consulta 7/15/2012, `http://docs.jboss.org/hibernate/entitymanager/3.5/reference/en/html_single/#architecture-javase`.

[61] Vakali, Athena y Koutsonikola, Vassiliki. Art. ldap: Framework, practices, and trends. *IEEE Internet Computing*, 8, September 2004.

[62] X. Yuan, Jonas. *Liferay Portal Systems Development*. Editorial Packt Publishing,1er ed, 546, 2012.

[63] Zamani, Nabi. Security configuration in Glassfish. Última consulta 7/15/2012, `http://www.nabisoft.com/tutorials/glassfish/installing-glassfish-301-on-ubuntu`.