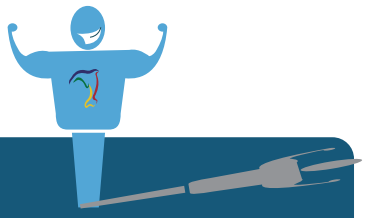
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

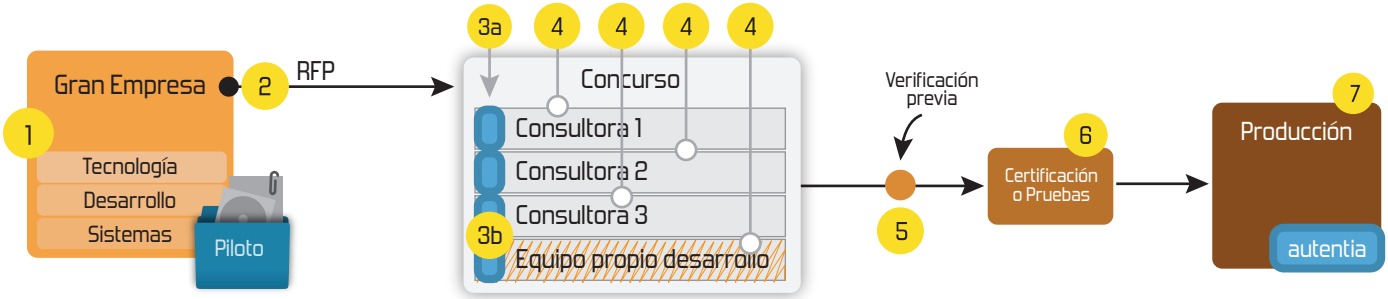
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com





Tutorial desarrollado por

Francisco Javier Martínez Páez

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia

Somos expertos en Java/J2EE

Fecha de creación del tutorial: 2008-01-23

Icefaces, JBoss, Maven2 y EJB3: Parte 5.

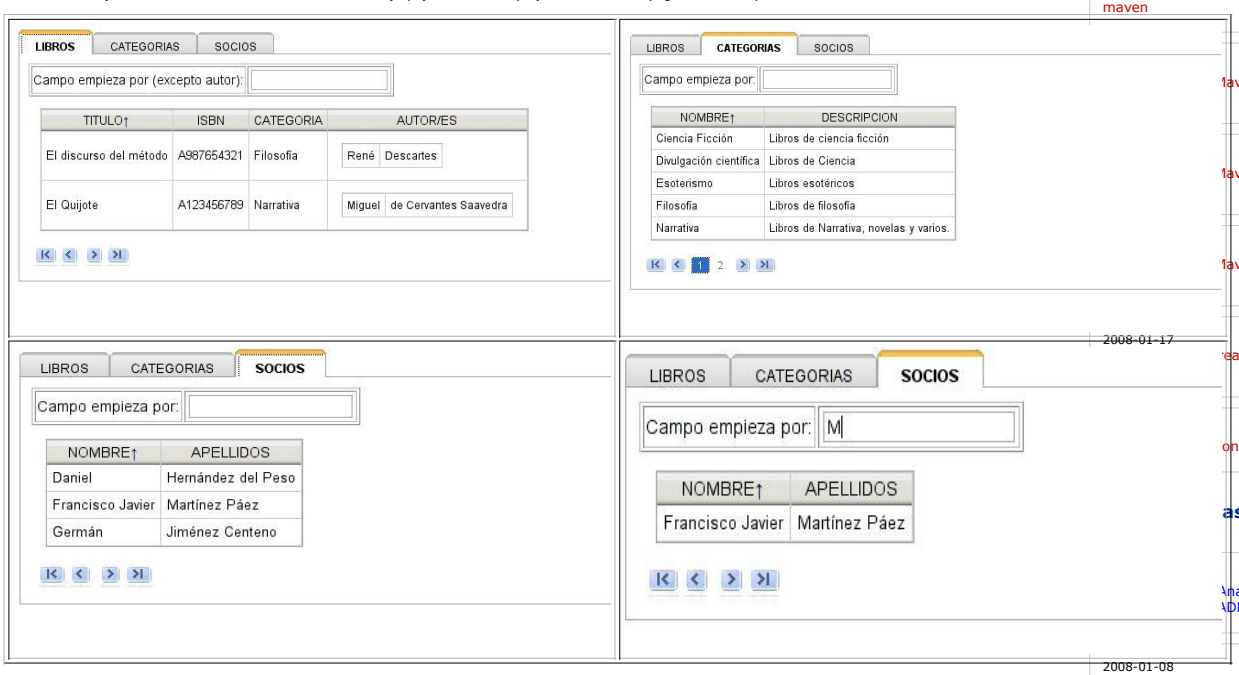
Vamos ya con la quinta entrega de este tutorial.

Antes de empezar, os dejo un enlace a los códigos fuente con lo hecho hasta ahora y con lo que voy a enseñaros en este:

- Fuentes de la parte Web: (descargar)
- Fuentes de la parte del Negocio: (descargar)
- Fichero pom.xml del Padre: (descargar)

1. **Objetivo.**

   Como objetivo en este tutorial es crear, usando icefaces y apoyándonos en el proyecto Modelo, una página con esta apariencia:



- Se mostrará una página, donde usando pestañas podremos seleccionar ver los libros, los socios o las categorías existentes en la base de datos.
- Los listados se mostrarán paginados de 5 en 5.
- Permitirá ordenar por los campos de los registros
- Existirá un campo de texto que filtrará los listados por las letras iniciales de los campos de la lista. Este filtrado se refrescará automáticamente cada vez que introduzcamos o eliminemos un carácter en el campo de búsqueda.

Bueno, pues eso es básicamente (que no es poco) lo que vamos a hacer.

2. **Cambios hechos en el Modelo.**

   Primeramente os comento una serie de cambios que he tenido que hacer en las clases del Modelo para adecuarlas a nuestros requisitos. Los cambios han sido los siguientes (si habéis seguido el tutorial completo):

- He sacado el fichero 'orm.xml' del directorio META-INF (para no tenerlo en cuenta, ya que esto se usó a modo de ejemplo), es decir, dejamos como modo de mapeo únicamente las anotaciones.

- He modificado todas las entidades donde aparecían relaciones del tipo 'MANY-TO-MANY' y 'ONE-TO-MANY' para que en lugar de usar colecciones de tipo 'Set', usen colecciones de tipo 'List'. Esto es debido a que en el listado de libros, se usa un componente 'dataTable' anidado dentro de otro componente 'dataTable' que muestra los libros, para mostrar los autores de un libro (un libro puede tener varios autores) y el atributo 'value' de este componente no convierte automáticamente colecciones de tipo Set, pero si de tipo List. Al final he decidido hacerlo en todas las relaciones de este tipo para evitar futuros errores, aunque hubiese bastado con hacerlo en la relación 'MANY-TO-MANY' de la entidad Libro.
- Ha sido necesario también, por el mismo motivo que antes, modificar la forma de obtener los autores de un libro, indicando que NO se obtengan los autores de manera 'Lazy' (por defecto) sino 'Eager', es decir siempre, porque en el 'dataTable' anidado, cuando intenta obtener los autores de un libro, sino están obtenidos previamente, intentará obtenerlos en ese momento ('Lazy'), sin embargo, la sesión del EntityManager ya está cerrada y eso provoca un error. El siguiente código muestra las dos modificaciones realizadas:

```
/**
* Modificamos la forma de obtener los autores del libro,
* ya que al mostrarlos en el listado de libros, es necesario
* obtenerlos siempre para evitar un error.
* @return
*/
@ManyToMany(fetch=FetchType.EAGER)
@JoinTable(name="Libro_Autor",
joinColumns=
@JoinColumn(name="libro",referencedColumnName="id"),
inverseJoinColumns=
@JoinColumn(name="autor",referencedColumnName="id")
)
public List<Autor> getAutores() {
return autores;
}
```

- He modificado la firma de algunos métodos del 'Dao', y por ende también la implementación 'DaoImpl'. También he añadido una clase llamada 'FilterParameter' que representa un parámetro de búsqueda en una consulta. Os muestro un extracto de algunos métodos del 'DaoImpl' modificados

```
public <T extends TransferObject> List<T> findAllAndFilterLike(
Class<T> transferObjectClass, String sortColumn, boolean ascending, boolean or, FilterParameter ... params) {
log.debug("DaoImpl:findAllAndFilterLike");
final String entityName = transferObjectClass.getSimpleName();
StringBuffer sbQuery = new StringBuffer("from ").append(entityName).append(" e");
final Query query;
FilterParameter [] params2 = null;
boolean first=true;
int i = 0;
if(params!=null && params.length>0) {

params2 = new FilterParameter[params.length];
for(FilterParameter fP:params) {
if(first) {
sbQuery.append(" where e.").append(fP.getField()).append(" like :").append("p"+i);
} else {
sbQuery.append(or?" or":" and").append(" e.").append(fP.getField()).append(" like :").append("p"+i);
}
first=false;
FilterParameter fPAux = new FilterParameter("p"+i,fP.getValue()+"%");
params2[i] = fPAux;
i++;
}
}
if (ascending)
sbQuery.append(" order by e." + sortColumn + " asc");
else
sbQuery.append(" order by e." + sortColumn + " desc");
query = createQuery(sbQuery.toString(), params2);
final List<T> resultList = query.getResultList();
if (log.isTraceEnabled()) {
log.trace(resultList.size() + " " + entityName + " recovered from database.");
}
return resultList;
}

private Query createQuery(String query, FilterParameter ... params) {
Query qQuery = em.createQuery(query);
if (params != null && params.length>0) {
for(FilterParameter fP:params) {
qQuery.setParameter(fP.getField(), fP.getValue());
}
}
return qQuery;
```

```
}
```

El primer método permite realizar la búsqueda descrita en el punto 1. de este tutorial. Recibe una entidad sobre la que buscar (en realidad la tabla), el campo sobre el que ordenar y la forma de ordenar el resultado, si la búsqueda es un 'or' o 'and' sobre los criterios de búsqueda recibidos.

2. **Creamos los ManagedBean.**

   En el proyecto Web, he creado también los ManagedBean sobre la que se apoyarán los componentes visuales para 'pintarse'. He creado tres ManagedBean, uno para cada Entidad con la que vamos a interactuar (Libros, Categorías y Socios).

   Os muestro y comento BookCtrl (los otros dos son similares, podéis refactorizar si queréis y sacar lo común a una clase abstracta o similar):

```
package com.autentia.tutoriales.icefaces.beans;


import java.util.List;
import javax.faces.component.UIData;
import javax.faces.event.ValueChangeEvent;
import javax.naming.NamingException;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.autentia.tutoriales.modelo.entidades.Libro;
import com.autentia.tutoriales.modelo.services.Dao;
import com.autentia.tutoriales.modelo.services.DaoImpl;
import com.autentia.tutoriales.modelo.services.FilterParameter;
import com.autentia.tutoriales.modelo.util.EjbLocator;


/**
* Managed Bean para los libros.
* @author fjmpaez
*
*/


public class BookCtrl {
private static final Log log = LogFactory.getLog(BookCtrl.class);
private String sortColumn = "titulo";
private boolean sortAscending = true;
private UIData tabla;
Dao dao;
private String search="";
public BookCtrl() {
log.debug("Constructor:BookCtrl");
try {
dao = EjbLocator.lookupLocalBean(DaoImpl.class);
} catch (NamingException e) {
log.error ("Error al iniciar el controlador: ", e);
}
}
public List<Libro> getAll() {
if("".equals(search)) {
return dao.findAll(Libro.class, getSortColumn(), isSortAscending());
} else {
return dao.findAllAndFilterLike(Libro.class, sortColumn, sortAscending,true,
new FilterParameter("titulo",search),
new FilterParameter("isbn",search),
new FilterParameter("categoria.nombre",search));
}
}
public String getSortColumn() {
return sortColumn;
}
public void setSortColumn(String sortColumn) {
this.sortColumn = sortColumn;
}
public boolean isSortAscending() {
return sortAscending;
}
public void setSortAscending(boolean sortAscending) {
this.sortAscending = sortAscending;
}


public UIData getTabla() {
return tabla;
}
```

```
public void setTabla(UIData tabla) {
this.tabla = tabla;
}

public String getSearch() {
return search;
}

public void setSearch(String search) {
this.search = search;
}
/**
* Este método asegura que tras una búsqueda, la tabla vuelva
* a mostrar la primera página.
* @param event
*/
public void changeSearch(ValueChangeEvent event) {
tabla.setFirst(0);
}
}
```

- En 'sortColumn' servirá para almacenar el campo por el que se ordenan los listados y 'sortAscending' indicará si la búsqueda es ascendente o descendente.
- En dao almacenaremos una referencia al 'Bean' de sesión 'DaoImpl' que inicializamos en el constructor del 'Bean' apoyándonos en la clase 'EjbLocator'
- El atributo 'search' lo usaremos para recuperar el texto introducido en el campo de búsqueda
- El atributo 'tabla' será una referencia al componente JSF que muestra los listados. Lo usamos en el método 'changeSearch(...)' para que cada vez que se produzca un cambio en el contenido del campo de búsqueda, la tabla vuelva a la página inicial. Si no hacemos esto (o algo similar), cuando estemos mostrando una página distinta a la inicial y hagamos una búsqueda, provocará que si la búsqueda no retorna los suficientes registros como para llegar a mostrar la página en la que te encuentras, en la pantalla no aparecerá nada sin saber porqué ocurre; por eso forzamos a volver a la página inicial.
- El método 'getAll()' retornará una lista con los registros encontrados. Si 'search' está vacío devuelve todos los registros de la tabla. Si no lo está, pasamos los parámetros correspondientes al método del 'dao' que filtra.

A continuación, debemos registrar los 'Managed Beans' en el fichero 'faces-config.xml' (muestro sólo el 'bookCtrl'):

```
...
<managed-bean>
<description>
Bean que maneja los libros
</description>
<managed-bean-name>bookCtrl</managed-bean-name>
<managed-bean-class>
com.autentia.tutoriales.icefaces.beans.BookCtrl
</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
...
```

2. **Creamos la parte visual.**

   Nos vamos a crear una página llamada 'listBooks.jspx' (jsp sujeto a las restricciones de XML). Cuando integramos icefaces podemos seleccionar si queremos que la página 'corra' en:

- Modo JSF: Sea procesada por `javax.faces.webapp.FacesServlet`
- Modo icefaces: Sea procesada por `com.icesoft.faces.webapp.xmlhttp.PersistentFacesServlet`

Dependiendo del 'mapping' que usermos al invocarla (podéis mirar el web.xml para comprobarlo). Si queremos usar componentes icefaces, debemos usar 'jspx' en modo icefaces (invocarlo con el mapping '*.iface)

Para conseguir nuestro objetivo, usaremos los componentes:

- Para las pestañas: '<ice:panelTabSet>' y '<ice:panelTab>'
- Para el texto de búsqueda: '<ice:inputText>' . Este componente realizará un envío parcial o 'partial submit', que es el 'truco del almendruco' de icefaces. Es decir, usando AJAX (de manera transparente gracias a Dios), icefaces se encargará de enviar la nueva información introducida en este campo al modelo de componentes en memoria, y refrescando o repintando sólo aquellos componentes que se ven afectados por esta nueva información si necesidad de refrescar la página completa (Genial ¿ no ?) Por defecto, el 'partial submit' sólo se realiza (si está activado para el componente con el atributo partialSubmit=true) cuando el campo pierde el foco ('onblur'). Yo voy a modificar esto para que se realice en el evento 'onchange' del campo de búsqueda.
- Para pintar los listados y su ordenación: '<ice:dataTable>' junto con '<ice:commandSortHeader>'. Y para la paginación: <ice:dataPaginator>
- Para los estilos, usaremos los que 'regala' icefaces.

  En definitiva, que no tenemos que hacer casi nada, sólo usar correctamente los componentes y apoyarnos el los 'managed beans' para obtener datos del modelo..

  Aquí va el código de parte de la página resaltando lo más importante:

```
<f:view xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:ice="http://www.icesoft.com/icefaces/component"
xmlns:jsp="http://java.sun.com/JSP/Page">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"></meta>
```

```
<title>Panel de control de la Biblioteca</title>
<ice:outputStyle rel='stylesheet' type='text/css'
href='./xmlhttp/css/xp/xp.css'/>
</head>
<body>
<ice:form id="formBiblioteca">
<ice:panelTabSet>
<ice:panelTab label="LIBROS">
<ice:panelGrid border="1" columns="2">
<ice:outputLabel for="searchBook">Campo empieza por (excepto autor):</ice:outputLabel>
<ice:inputText id="searchBook" partialSubmit="false" value="#{bookCtrl.search}"
onkeyup="iceSubmitPartial(form, this, event); return false;"
valueChangeListener="#{bookCtrl.changeSearch}">
</ice:inputText>
</ice:panelGrid>
<ice:panelGrid>
<ice:dataTable id="tablaLibros"
var="book"
value="#{bookCtrl.all}"
rows="5"
sortColumn="#{bookCtrl.sortColumn}"
sortAscending="#{bookCtrl.sortAscending}"
binding="#{bookCtrl.tabla}">
<ice:column id="tablaLibroscolumn1">
<f:facet name="header">
<ice:commandSortHeader
columnName="titulo"
arrow="true">
<ice:outputText value="TITULO"></ice:outputText>
</ice:commandSortHeader>
</f:facet>
<ice:outputText value="#{book.titulo}"></ice:outputText>
</ice:column>
<ice:column id="tablaLibroscolumn2">
<f:facet name="header">
<ice:commandSortHeader
columnName="isbn"
arrow="true">
<ice:outputText value="ISBN"></ice:outputText>
</ice:commandSortHeader>
</f:facet>
<ice:outputText value="#{book.isbn}"></ice:outputText>
</ice:column>
<ice:column id="tablaLibroscolumn3">
<f:facet name="header">
<ice:commandSortHeader
columnName="categoria.nombre"
arrow="true">
<ice:outputText value="CATEGORIA"></ice:outputText>
</ice:commandSortHeader>
</f:facet>
<ice:outputText value="#{book.categoria.nombre}"></ice:outputText>
</ice:column>
<ice:column id="tablaLibroscolumn4">
<f:facet name="header">
<ice:outputText value="AUTOR/ES"></ice:outputText>
</f:facet>
<ice:dataTable id="tablaLibrostablaAutores"
var="autor"
value="#{book.autores}"
rows="0">
<ice:column id="tablaLibroscolumn11">
<ice:outputText value="#{autor.nombre}"></ice:outputText>
</ice:column>
<ice:column id="tablaLibroscolumn12">
<ice:outputText value="#{autor.apellidos}"></ice:outputText>
</ice:column>
```

```
</ice:dataTable>
</ice:column>
</ice:dataTable>
<ice:dataPaginator id="dataScroll_tablaLibros"
for="tablaLibros"
paginator="true"
fastStep="3"
paginatorMaxPages="4">
<f:facet name="first">
<ice:graphicImage
url="./xmlhttp/css/xp/css-images/arrow-first.gif"
style="border:none;"
title="Primera Page"/>
</f:facet>
<f:facet name="previous">
<ice:graphicImage
url="./xmlhttp/css/xp/css-images/arrow-previous.gif"
style="border:none;"
title="Anterior Page"/>
</f:facet>
<f:facet name="next">
<ice:graphicImage
url="./xmlhttp/css/xp/css-images/arrow-next.gif"
style="border:none;"
title="Siguiente Page"/>
</f:facet>
<f:facet name="last">
<ice:graphicImage
url="./xmlhttp/css/xp/css-images/arrow-last.gif"
style="border:none;"
title="Ultima Page"/>
</f:facet>
</ice:dataPaginator>
</ice:panelGrid>
</ice:panelTab>
...
...
...
</ice:panelTabSet>
</ice:form>
</body>
</html>
</f:view>
```

- Lo primero que hemos de poner es '<f:view>' indicando los 'namespaces' correctamente (diferente a jsp)
- Luego incluimos uno de los estilos por defecto de icefaces con '<ice:outputStyle>'
- A continuación definimos las pestañas.
- Luego incluimos los listados y recogemos los valores invocando al método 'getAll' del Bean, le decimos cuales son los atributos relacionados con la ordenación, el número de elementos por página (un valor 0 no pagina)
- Incluimos el campo de búsqueda. Ponemos el 'partialSubmit' a false (aunque es el valor por defecto para darle énfasis a la explicación) porque lo que queremos es que se envíe la información después de pulsar una tecla: `onkeyup="iceSubmitPartial(form, this, event);...'` .Además registramos un 'listener' para cada vez que cambie el valor y se haga una búsqueda, la tabla muestre la página 1.
- Luego anidamos otra tabla para mostrar todos los autores de un libro, esta vez sin paginación ni ordenación.
- Por último usamos el componente de paginación indicándole que tabla está paginando.

Lo último que nos queda es modificar la página 'index.jsp' para que redireccione a esta nueva página:

```
<html>
<head>
</head>
<body>
<jsp:forward page="listBooks.iface" />
</body>
</html>
```

Usamos el mapping 'iface' para asegurarnos que el servlet que recibe la petición sea el de icefaces.

Si ahora arrancamos el servidor e invocamos la aplicación: http://localhost:8080/Web/ , el resultado obtenido debe coincidir 'sospechosamente' con lo que mostrábamos al principio del tutorial. Os recomiendo que insertéis más datos en la base de datos para probarlo mejor.

En la siguiente entrega intentaré finalizar con toda las operaciones del CRUD (Create, Read, Update y Delete) de alguna de las entidades, ya que hasta ahora sólo tenemos operaciones de lectura.

Hasta la próxima entrega.

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	pdf
EJB 3.0, un ejemplo práctico con Maven y JBoss	Este tutorial presenta un ejemplo sencillo donde se verá como desarrollar EJBs de sesión y de entidad, inyección de dependencias, llamar a los EJBs desde una aplicación Web, definición de un DataSource, y como configurarlo y hacerlo funcionar en JBoss, y	2007-08-06	3064	pdf
Comparativa entre EJB3 y Spring	En este tutorial os mostramos una comparativa entre EJB3 y Spring esperando que os ayude a decidir qué tecnología utilizar.	2007-10-17	1578	pdf
Hibernate y las anotaciones de EJB 3.0	En este tutorial Alejandro Pérez nos muestra las ventajas que nos aporta Hibernate y las anotaciones de EJB 3.0	2007-06-25	3387	pdf
Guía rápida de instalación de JBOSS Application Server 4.	En este manual veremos paso a paso la forma de instalar en tu equipo JBoss Application Server 4.	2006-11-02	3768	pdf
Icefaces, JBoss, Maven2 y EJB3: Parte 3	En esta tercera parte del tutorial vamos a desarrollar la capa de negocio basado en el modelo de entidades de una pequeña biblioteca.	2008-01-18	125	pdf
Interceptando un EJB en JBoss	En este tutorial os vamos a enseñar la arquitectura de EJBs en JBoss y a como modificarla, insertando un interceptor propio dentro de la cadena de interceptores del Proxy Cliente.	2007-03-26	3724	pdf
Integración de JSF 1.2, Facelets e ICEFaces en Tomcat 6	Integración de JSF 1.2, Facelets e ICEFaces en Tomcat 6	2007-12-10	841	pdf
Ejemplo de web con ICEfaces	Creación de una web paso a paso con ICEFaces, Tomcat 5.5 y Eclipse	2008-01-16	348	pdf
Proyecto con JSF Java Server Faces Myfaces, Maven y Eclipse: Hibernate (segunda parte)	En este artículo se va a continuar con el desarrollo de la aplicación Myfaces JSF con Maven multimódulo que comenzamos en un tutorial anterior. Además también se tratará de la integración de Hibernate con las aplicaciones.	2007-07-31	2449	pdf
EJB 3.0: Resurrection	Este turorial nos va a presentar las nuevas funcionalidades que nos aportan los EJB 3.0.	2007-05-07	3508	pdf

resolución.