APLICACIÓN BASE PARA EL DESARROLLO EN J2EE

Guia paso a paso de desarrollo de una aplicación J2EE para la JCyL

Edición: **V 2.6**

Autor: **Unidad de Informática Corporativa**
Área de Desarrollo y Mantenimiento

Fecha: **Octubre 2005**

Historia del Documento			
Versión: **V 1.0**	*Descripción*: Guía detallada de los pasos necesarios para desarrollar una aplicación J2EE siguiendo los estándares marcados por la Junta de Castilla y León (JCyL) para este tipo de desarrollos.		
	Elaborado por:	**Juan José Barrio Vizán**	*Fecha*: **Diciembre 2004**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Enero 2005**
			Fecha:
			Fecha:
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Febrero 2005**
Versión: **V 1.5**	*Descripción*: Guía detallada de los pasos necesarios para desarrollar una aplicación J2EE siguiendo los estándares marcados por la Junta de Castilla y León (JCyL) para este tipo de desarrollos.		
	Elaborado por:	**Juan Burgos Aranda**	*Fecha*: **Julio 2005**
	Revisado por:	**Marta Martín Jiménez**	*Fecha*: **Agosto 2005**
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Agosto 2005**
Versión: **V 2.6**	*Descripción*: Guía detallada de los pasos necesarios para desarrollar una aplicación J2EE siguiendo los estándares marcados por la Junta de Castilla y León (JCyL) para este tipo de desarrollos.		
	Elaborado por:	**Sonia Hernández Sánchez**	*Fecha*: **Septiembre 2005**
	Revisado por:	**Diego García Carrera**	*Fecha*: **Octubre 2005**
			Fecha:
			Fecha:
	Aprobado por:	**Juan Antonio Barras García**	*Fecha*: **Octubre 2005**

Lista de distribución del documento

Nombre	Localización

Índice

1 Introducción

En este documento se presenta una guía detallada de los pasos necesarios para desarrollar una aplicación J2EE siguiendo los estándares marcados por la Junta de Castilla y León (JCyL) para este tipo de desarrollos.

Para facilitar la comprensión de los diferentes pasos, se va a partir de un ejemplo que se va a ir desarrollando a lo largo de los diferentes capítulos de este documento.

Consideremos la siguiente situación:

Queremos construir un sistema de gestión de libros (GELI). En este sistema se van a considerar diferentes entidades. Como es lógico, tendremos los libros, que contendrán información acerca del título, breve descripción, autor, género y fecha de edición. Por otro lado estarán los autores de los libros, para los que se va a almacenar información acerca de su nombre, apellidos, lugar de nacimiento. Finalmente vamos a disponer de una serie de géneros literarios según los que se van a clasificar los libros.

Esta aplicación va a ser utilizada por dos tipos de personas

- Administradores: los usuarios de este grupo podrán realizar el mantenimiento de todas estas entidades. Además podrán realizar consultas acerca de los libros del sistema (en este caso se va a permitir buscar libros por cualquiera de sus campos), los autores, y para cada autor podrán ver todos los libros disponibles. Estos listados podrán obtenerse en formato PDF y Excel.
- Operadores: los usuarios pertenecientes a este grupo sólo podrán realizar consultas de información que se haya dado de alta en el sistema. Igual que en el caso anterior podrán obtener estos listados en formato PDF y Excel.

Este es, a grandes rasgos, el sistema que se va a construir a lo largo de este documento.

2 Generar esquema de datos

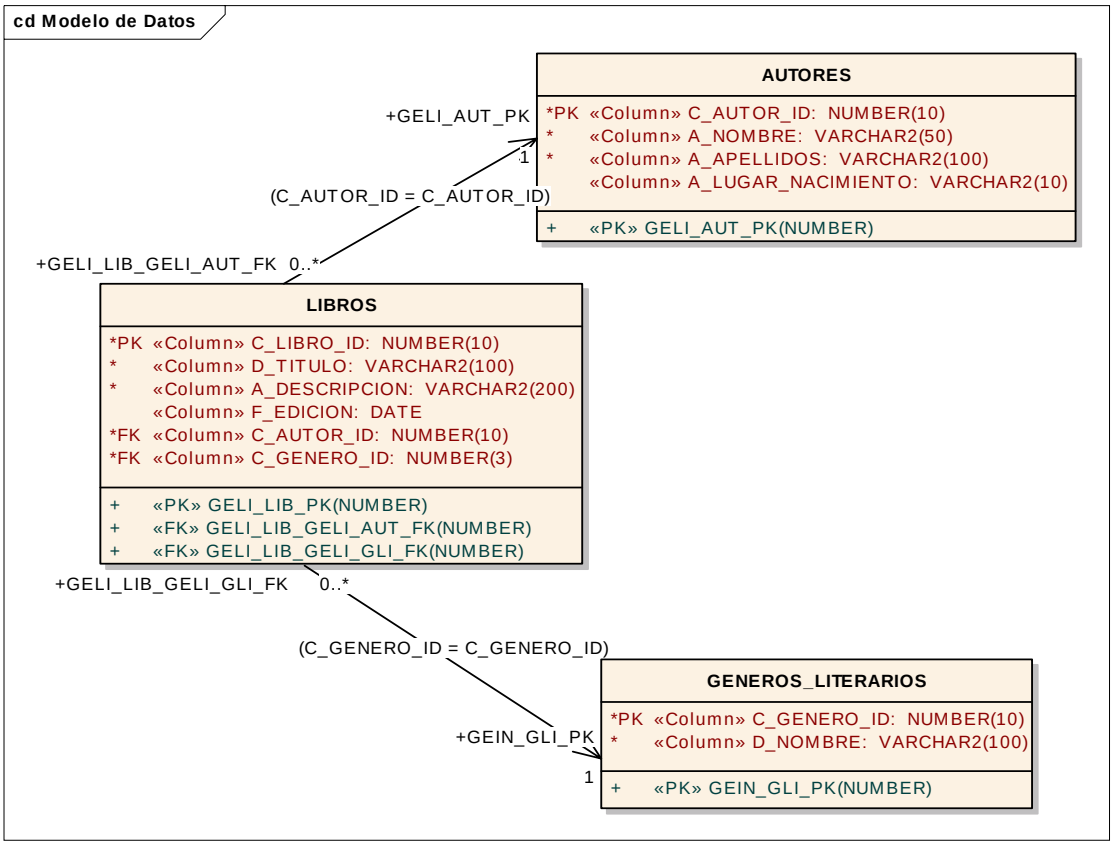
Este apartado se va a dedicar a la generación del esquema de datos necesario para la aplicación. En nuestro caso vamos a utilizar una base de datos Oracle 9i. Por regla general, ya estará creado y no será necesario hacer estos pasos.

2.1 Prerrequisitos

Para poder lanzar estos scripts, será necesario disponer de un esquema de base de datos y de un usuario con permisos para conectarse a él, crear tablas y datos. En el desarrollo de este ejemplo se va a utilizar GELI (GEstión de LIbros) y el usuario geli para acceder a dicho esquema.

2.2 Desarrollo

Dados los requisitos de la aplicación se ha planteado el siguiente modelo de datos:



Para crear este esquema de datos en una base de datos Oracle se adjuntan dos ficheros que deben ser ejecutados utilizando algún programa que permita conectarse a una BD Oracle y ejecutar sentencias SQL, por ejemplo, sqlplus o toad. El orden de ejecución debe ser el siguiente:

- estructura_bd.sql
- datos_bd.sql

2.3 Resultados esperados

En este punto se debe disponer de un esquema de BD con las tres tablas de la figura, secuencias, los datos iniciales de la aplicación (géneros literarios).

3 Generar el esqueleto base utilizando el generador de aplicaciones

Una vez creado el esquema de datos, ya podemos comenzar con el desarrollo de la aplicación ejemplo.

Este apartado se va a dedicar a mostrar la utilización del generador de aplicaciones, forma de invocarlo y resultados que se van a obtener.

3.1 Prerrequisitos

Tener instalado el Entorno de Ejecución Java (JRE) (preferiblemente versión 1.3.1 o superior). Para ver la versión instalada se puede ejecutar el siguiente comando**:** (Inicio – Ejecutar – "cmd")

```
C:\>java –version
```

obteniéndose una salida parecida a la siguiente

```
java version "1.4.2_05"
Java(TM) 2 Runtime Environment, Standard Edition
(build 1.4.2_05-b04)
Java HotSpot(TM) Client VM (build 1.4.2_05-b04, mixed mode)
```

Otro requisito fundamental, previo a la generación de la aplicación, es disponer del generador de aplicaciones. En este caso se puede descargar del repositorio de proyectos que se puede encontrar en http://gforge.jcyl.es, dentro del proyecto "UIC-Aplicacion Base Desarrollo J2EE"

3.2 Desarrollo

Para comenzar, se debe lanzar el generador de aplicaciones. Este generador viene comprimido en el fichero `APPBASE-X_X.zip`[1], que ha de ser descomprimido a un directorio temporal cualquiera. Para lanzar el generador se debe estar dentro del directorio en el que se ha descomprimido el fichero y ejecutar la siguiente línea de comando:

```
java –jar GenerarAplicaciónJ2EE.jar
```

Esto presenta la aplicación "Generador de aplicaciones" que va guiar durante la creación de la aplicación base. A continuación se muestra una pantalla de ejemplo.

[1] X_X Versión de la aplicación base 1.1 o superior

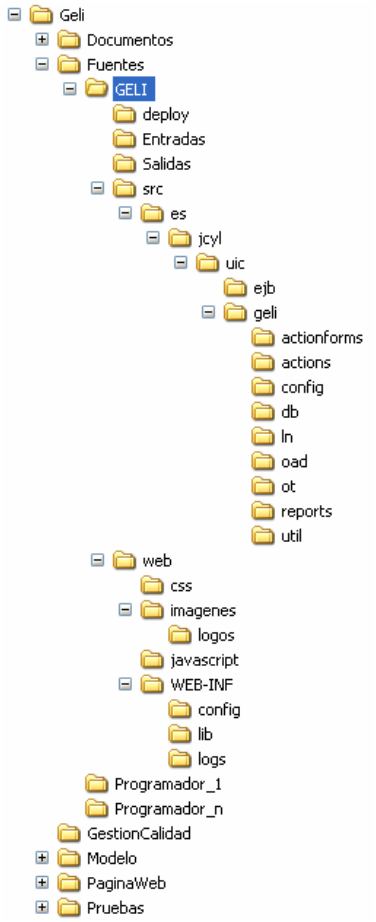
En esta misma pantalla se puede seleccionar el medio en el que se va a tener la información del componente de seguridad. Este componente se utiliza en todas las aplicaciones para comprobar la identidad del usuario que accede, así como las opciones a las que tiene acceso dicho usuario. El componente puede estar almacenado en la BD o bien en un fichero .xml. Si se posiciona el ratón en cada uno de los campos que se presentan, se mostrará información ampliada acerca del contenido de cada uno de ellos.

En el ejemplo que hemos utilizado puede verse que el nombre de la aplicación comienza con mayúscula. Esto es un requisito del generador puesto que sino dará problemas en la generación automática de las clases.

Una vez completados todos los datos, se puede generar la aplicación pulsando sobre el botón “Generar”.

3.3 Resultados esperados

En este momento se debe tener una estructura de directorios con todos los ficheros que definen la aplicación y toda la documentación de ayuda. A continuación se muestra un ejemplo de este contenido.



Como se puede ver en la imagen, ha creado un directorio con los fuentes de la aplicación base sobre la que se va a trabajar y el nombre de la aplicación que hemos decidido, Fuentes/GELI. Puede ser que alguno de estos directorios, dependiendo de la aplicación que se vaya a generar, no sea necesario, pe el directorio reports o útil.

Una vez terminado este capítulo, tendremos la estructura de directorios de una aplicación base de prueba operativa.

4 Generación de objetos OADs y OTs

Una vez generada la estructura de la aplicación base, se pueden crear los objetos necesarios para el acceso a los datos almacenados en la estructura que se ha creado en el apartado 2 de esta documentación. Estos objetos son los OADs[2] y OTs [3]

Un objeto OAD es aquel que permite el acceso a los datos de una tabla o cualquier otro repositorio, que posteriormente se van a utilizar en la aplicación. Un objeto OT, es aquel que almacena la información de cada uno de los datos recuperados por un OAD y que se mostrarán al usuario por pantalla.

4.1 Prerrequisitos

Puesto que el generador de aplicaciones crea estos objetos a partir de una BD, será necesario tener instalado un esquema en alguna BD, disponer de la cadena de conexión a dicho esquema y conocer el usuario y password con el que se puede realizar la conexión a dicho esquema.

Haber completado los apartados 2 y 3 de esta documentación.

4.2 Desarrollo

Para poder acceder a esta opción se debe pulsar sobre el botón “Generar OT y DAO” que aparece en la ventana principal del generador de aplicaciones.

Se presenta una nueva pantalla en la que se deben indicar los parámetros necesarios para la conexión con el esquema creado. Vamos a suponer un esquema GELI y un usuario GELI.

[2] OAD : Objeto de Acceso a Datos. Objeto que encapsula el código de acceso a un repositorio de datos. Se trata de la aplicación del patrón de diseño Data Access Object.

[3] OT : Objeto Transferencia. Objeto simple que almacena los datos que se intercambian entre capas. Y que puede representar un registro de una entidad. Se trata de la aplicación del patrón de diseño Value Object.

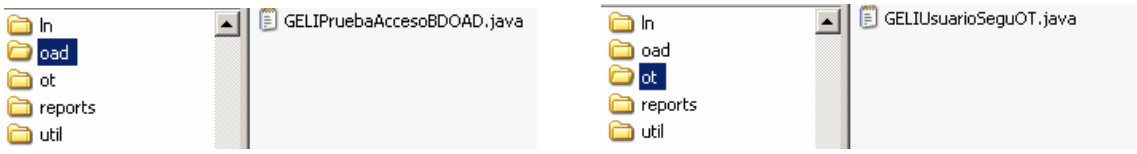
Nota 1: Los datos introducidos son de ejemplo y deberán ser sustituidos por los que corresponda en cada aplicación generada. Si se utilizan los mismos que el ejemplo, habrá que comprobar la cadena de conexión para verificar el nombre del servidor de BD en el que se ha creado el esquema.

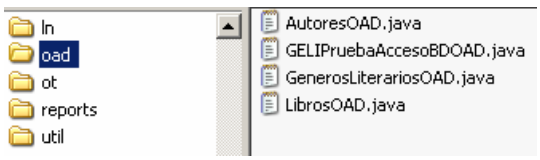
El directorio en el que se van a colocar los ficheros generados puede ser cualquiera, pero se aconseja que sea el mismo que en el que se encuentran los demás fuentes de la aplicación base. Si no se selecciona este directorio, será necesario incluirlo después en el proyecto para que se puedan utilizar en JDeveloper.

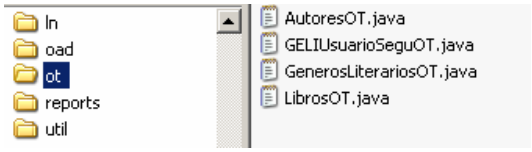
4.3 Resultados esperados

Una vez terminado este proceso, en el directorio indicado como destino, se creará una estructura de directorios equivalente al nombre del paquete indicado, sustituyendo los '.' por '/' y dentro de ese directorio, dos nuevos directorios /OAD y /OT, donde se crearán las clases que permiten el acceso y almacenamiento de los datos contenidos en las tablas seleccionadas por el usuario.

A continuación se muestra el contenido del directorio oad y ot antes y después de la generación







Nota 2: **Por simplificación el generador de OADs y OTs genera una clase por cada tabla de base de datos. Esto no tiene que ser una norma, ya que en numerosos casos será mas optimo construir objetos de acceso a datos que involucren los métodos de varias tablas y Objetos transferencia mas complejos que posean tipos compuestos como Arrays que soporten en un unico objeto los datos maestro y detalle.**

5 Implementación de los requisitos de seguridad: GestSegu

Este capítulo lo dedicaremos a la descripción de la utilización de la aplicación de seguridad para dar de alta las diferentes aplicaciones, sus usuarios, roles, opciones de menú de cada grupo de usuario, así como de las funciones a las que puede tener acceso cada usuario, y que no pertenecen al menú.

Vamos a seguir con nuestro ejemplo, y crearemos una nueva aplicación en el componente de seguridad (GELI), los distintos roles, usuarios y sus funciones.

Para poder acceder a la aplicación de gestión de seguridad, en nuestro caso, debemos conectarnos a la siguiente dirección http://aplides10g.cf.jcyl.es/GestSegu. ***Esta dirección cambia en función del servidor en el que se encuentre instalado el sistema de seguridad usado.***

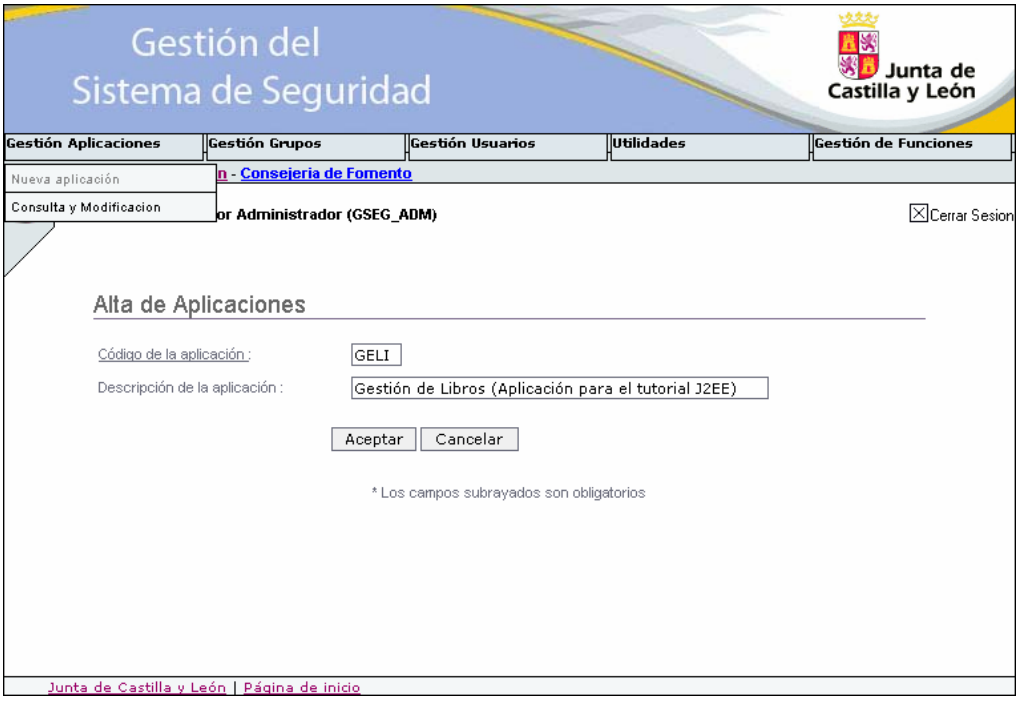
5.1 Prerrequisitos

Puesto que se trata de una aplicación Web, será necesario disponer de un navegador Web, y tener acceso al servidor de aplicaciones como por ejemplo : *aplides10g.cf.jcyl.es*.

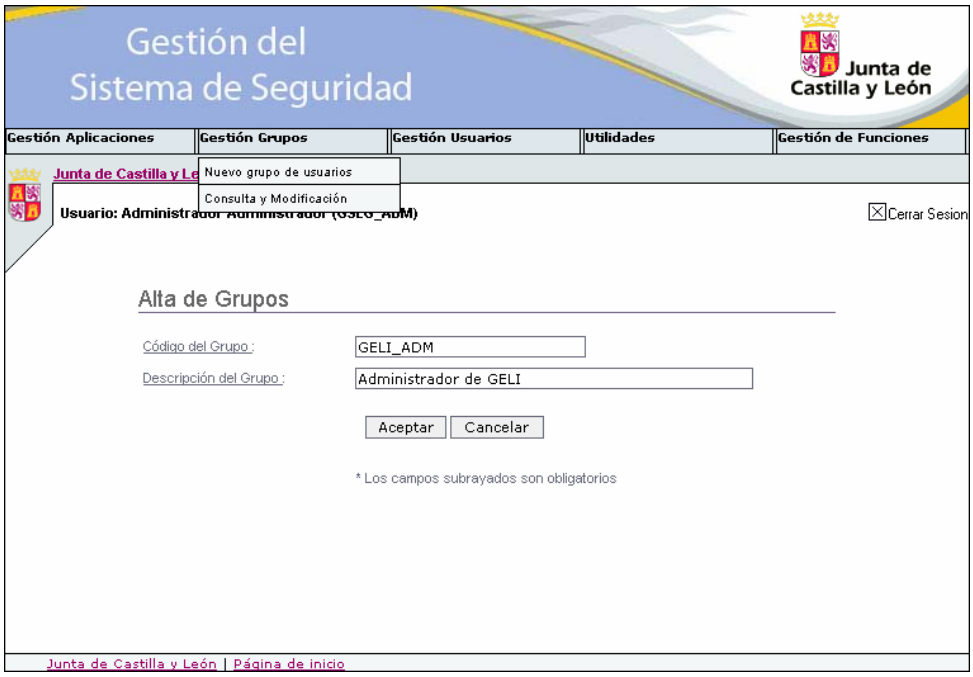
Debemos disponer de un usuario de acceso al sistema de seguridad, con suficientes privilegios para poder realizar todas estas inserciones

5.2 Desarrollo

Lo primero que debemos hacer es crear la nueva aplicación que vamos a dar de alta. En nuestro caso esta aplicación se llama GELI (este es el nombre es importante y se debe recordar puesto que posteriormente veremos que es necesario indicarlo en los ficheros de la aplicación J2EE que hemos creado). Para ello, una vez autenticados, accedemos a la opción de menú "Gestión de aplicaciones→Nueva aplicación". A continuación podemos ver los datos facilitados para la creación de la aplicación de nuestro ejemplo.



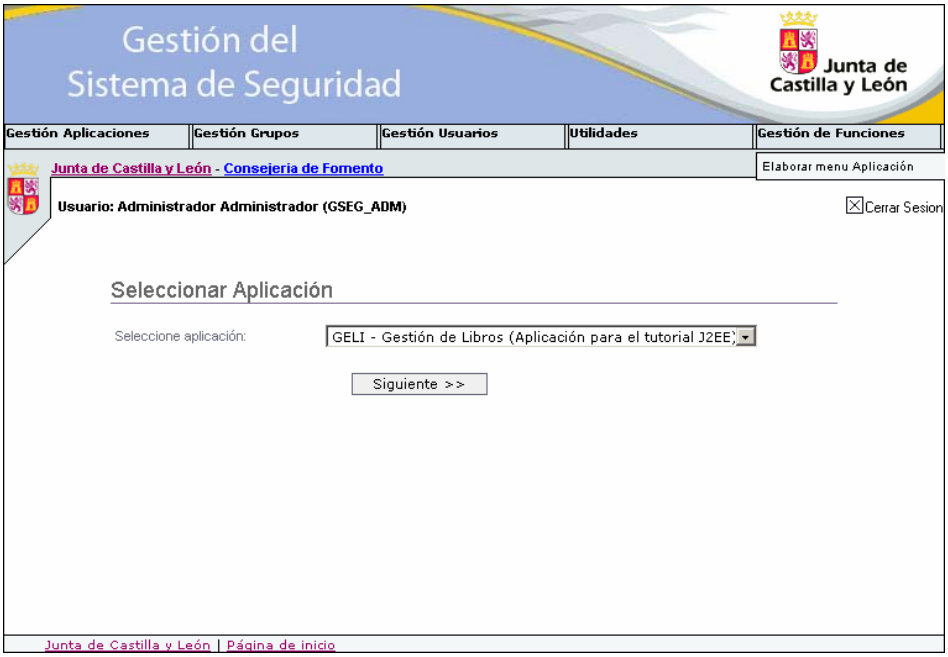
Una vez creada la aplicación, debemos crear los distintos grupos de usuarios que van a pertenecer a ella[4]. En nuestro caso, hemos identificado dos grupos, usuarios administradores y usuarios operadores. Por tanto vamos a crear dos grupos, GELI_ADM y GELI_OPE. Para nombrar estos grupos, como puede observarse, hemos utilizado letras mayúsculas y comienzan con el nombre de la aplicación para identificar de forma rápida los distintos roles. Para acceder a esta opción, debemos desplegar el menú "Gestión de grupos" y dentro de él, "Nuevo grupo de usuarios"



Finalizada la creación de grupos, podemos dar de alta los usuarios que van a tener acceso a la aplicación y seleccionar el grupo al que van a pertenecer. Dependiendo del grupo al que pertenezcan tendrán acceso a unas funciones u otras. Este paso puede hacerse a medida que se vayan conociendo los usuarios. Para ello accedemos a la opción de menú "Gestión Usuarios→Nuevo Usuario". Estos serán los datos con los que se podrá acceder posteriormente a la aplicación de ejemplo que estamos desarrollando. Cada usuario que deba tener acceso, ha de ser creado aquí e indicarle los datos con los que lo hemos creado.

[4] Los Grupos de usuarios son un simple modo de catalogar a los usuarios para asociarles unas funciones. En muchos casos se recomienda compartir los roles genéricos como ADM, CONSULTA, etc.

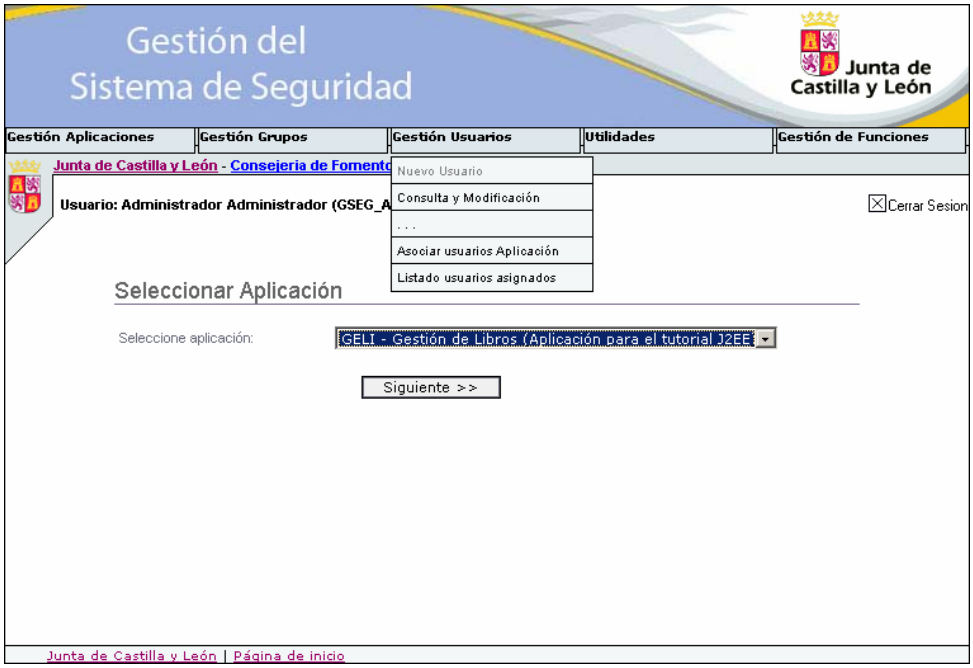
A continuación debemos crear todas las funciones a las que se va a tener acceso en la aplicación, y que posteriormente asignaremos a los distintos grupos de usuarios. Para ello seleccionamos el menú "Gestión de Funciones→Elaborar menú Aplicación". Seleccionamos la aplicación para la que vamos a crear estas funciones (en nuestro caso, la aplicación GELI) y pulsando sobre el botón siguiente pasaremos a una pantalla en la que se pueden seleccionar los roles que van a pertenecer a la aplicación.

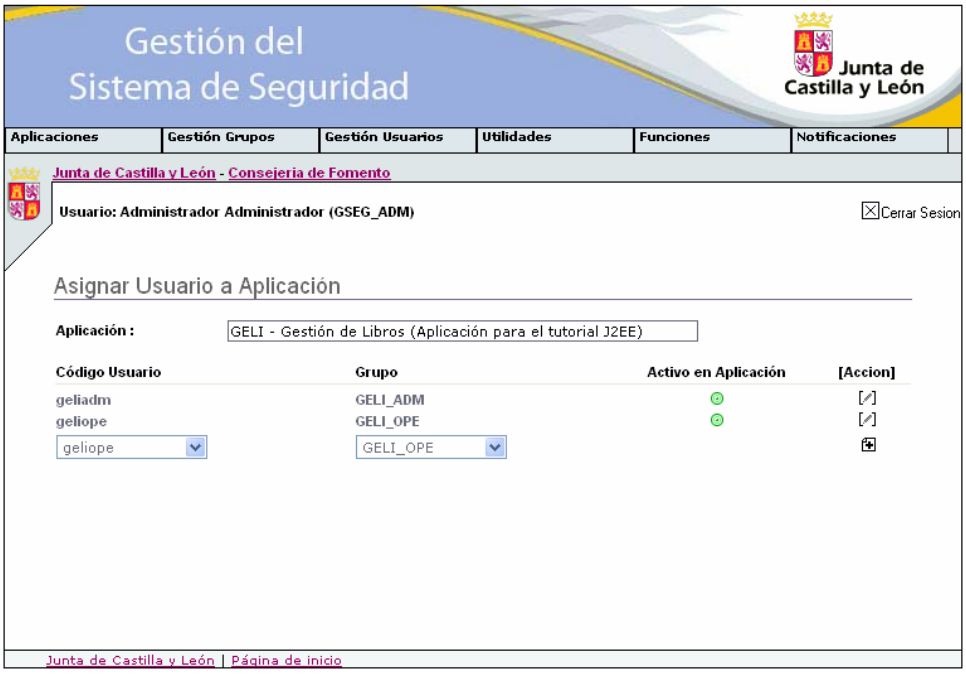


En la siguiente figura se puede ver el menú que hemos creado para el rol GELI_ADM, que previamente hemos seleccionado en la pantalla anterior y hemos pulsado sobre el botón menú. Todas las funciones que creamos desde esta pantalla aparecerán en el menú de los usuarios pertenecientes a este grupo. Es posible crear otro tipo de funciones, que no aparecerán en el menú, pero que permitirán que el usuario tenga acceso a ella (por ejemplo, al pulsar sobre un botón).

Inicialmente las acciones de los menús se pueden dejar vacías y a medida que vayamos generando la aplicación iremos rellenando con las acciones adecuadas (iremos viéndolo a lo largo del desarrollo de la aplicación de ejemplo). Este menú y las funciones creadas para un rol, pueden ser modificados en cualquier momento, y se pueden añadir nuevas funciones.

El siguiente paso, asignar a los usuarios que ya hayamos creado a nuestra aplicación. En este momento se va a decir que un usuario (geliadm) pertenece al grupo (GELI_ADM) dentro de la aplicación (GELI). En las dos siguientes imágenes se puede ver el proceso.





5.3 Resultados esperados

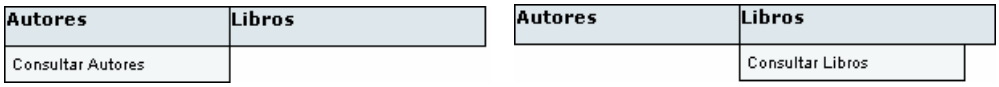
Vamos a suponer creados los siguientes roles y menús:

- GELI_ADM (usuario geliadm/geliadm)

Autores	Libros
Insertar Autor	
Consultar Autores	

Autores	Libros
	Insertar Libro
	Consultar Libros

- GELI_OPE (usuario geliope/geliope)



Nota 3: Si no se tiene acceso a esta aplicación, se puede ejecutar el script sql que se adjunta a esta documentación. Para ello, se debe disponer de conexión a la BD de seguridad, algún programa capaz de ejecutar sentencias sql contra esa BD, y permiso para insertar datos en ella. El script se llama `datos_bd_segu.sql`

6 Prueba de la aplicación generada

Una vez ejecutados los dos puntos anteriores se dispondrá de una aplicación preparada para su ejecución utilizando JDeveloper, entorno de desarrollo de aplicaciones Java establecido como estándar por la Junta de Castilla y León.

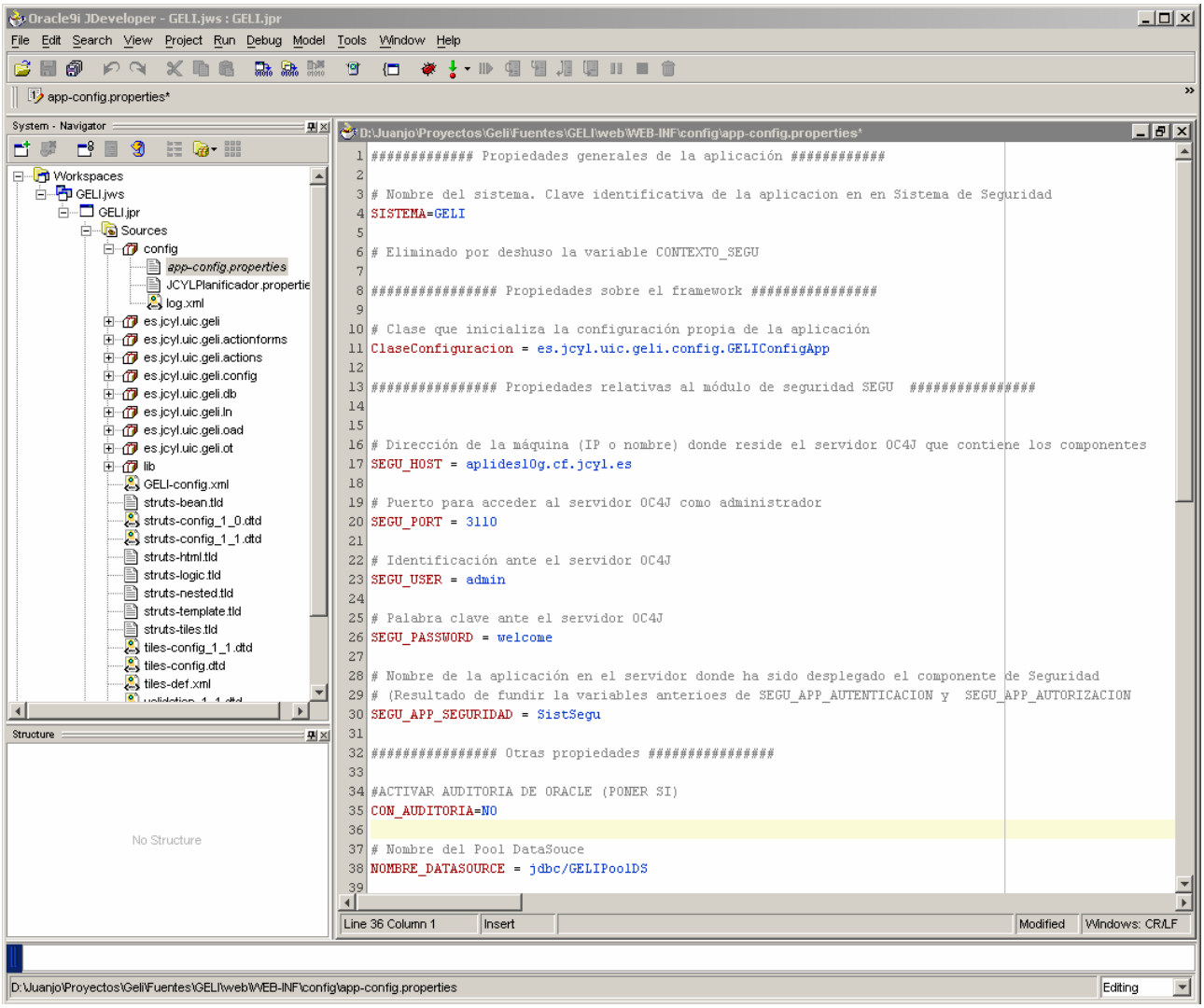
6.1 Prerrequisitos

Para poder editar y continuar con el desarrollo de la aplicación, será necesario tener instalado y configurado JDeveloper. Además será necesario introducir un elemento que nos permita conectar con la base de datos, también debemos conocer los datos para poder acceder al componente de seguridad, puesto que se deben introducir en el fichero de configuración de la aplicación.

Se debe haber ejecutado al menos el apartado 3. Si se quiere probar el acceso correcto de usuarios, es necesario haber terminado el apartado 5.

6.2 Desarrollo

Para poder ver la aplicación, se añadirá la aplicación al entorno de trabajo de JDeveloper. Para probar el correcto funcionamiento de esta aplicación será necesario realizar algunas modificaciones en los ficheros generados. Para ello se debe buscar el fichero `app-config.properties` que aparecerá en la carpeta `HTML Sources/WEB-INF/config` (navegando por la estructura de directorios mostrada en JDeveloper).



En este fichero aparecen una serie de parámetros sin valor, que será necesario completar para que la aplicación pueda funcionar. Estos valores dependerán del servidor en el que se encuentra desplegado el componente de seguridad. El sistema será el mismo que el nombre que se le haya dado a la aplicación que se haya generado, en nuestro caso GELI.

Una vez modificados todos los valores necesarios, se podrá ejecutar la aplicación pulsando sobre el botón  , que se puede ver en la barra de herramientas superior.

En esta aplicación inicial solo estará disponible una pantalla de login en la se podrá probar el correcto funcionamiento de los componentes de autenticación y autorización. Se va solicitar introducir usuario y password de acceso a la aplicación. Si los datos son correctos debe aparecerle una pantalla de bienvenida. En caso contrario, se le mostrará la misma pantalla en la que se indica que los datos introducidos no son correctos.

6.3 Resultados esperados

En el caso de haber introducido un usuario y password correcto (en nuestro caso cualquiera de los indicados en el apartado 5, debe verse una pantalla de bienvenida con el menú correspondiente al usuario con el que se ha accedido. A continuación se puede ver un ejemplo correspondiente al usuario responsable.

7 Elaboración de casos de uso de la aplicación

Una vez definido todo, se puede comenzar con el desarrollo de las diferentes opciones requeridas en nuestra aplicación de ejemplo.

7.1 Prerrequisito1: Instalación de los Asistentes de ayuda al desarrollo

Los asistentes de ayuda al desarrollo son unas extensiones que se integran dentro del entorno de desarrollo para JDeveloper con el objetivo de facilitar y acelerar el desarrollo de ciertos casos de uso comunes en las aplicaciones de la Junta de Castilla y León.

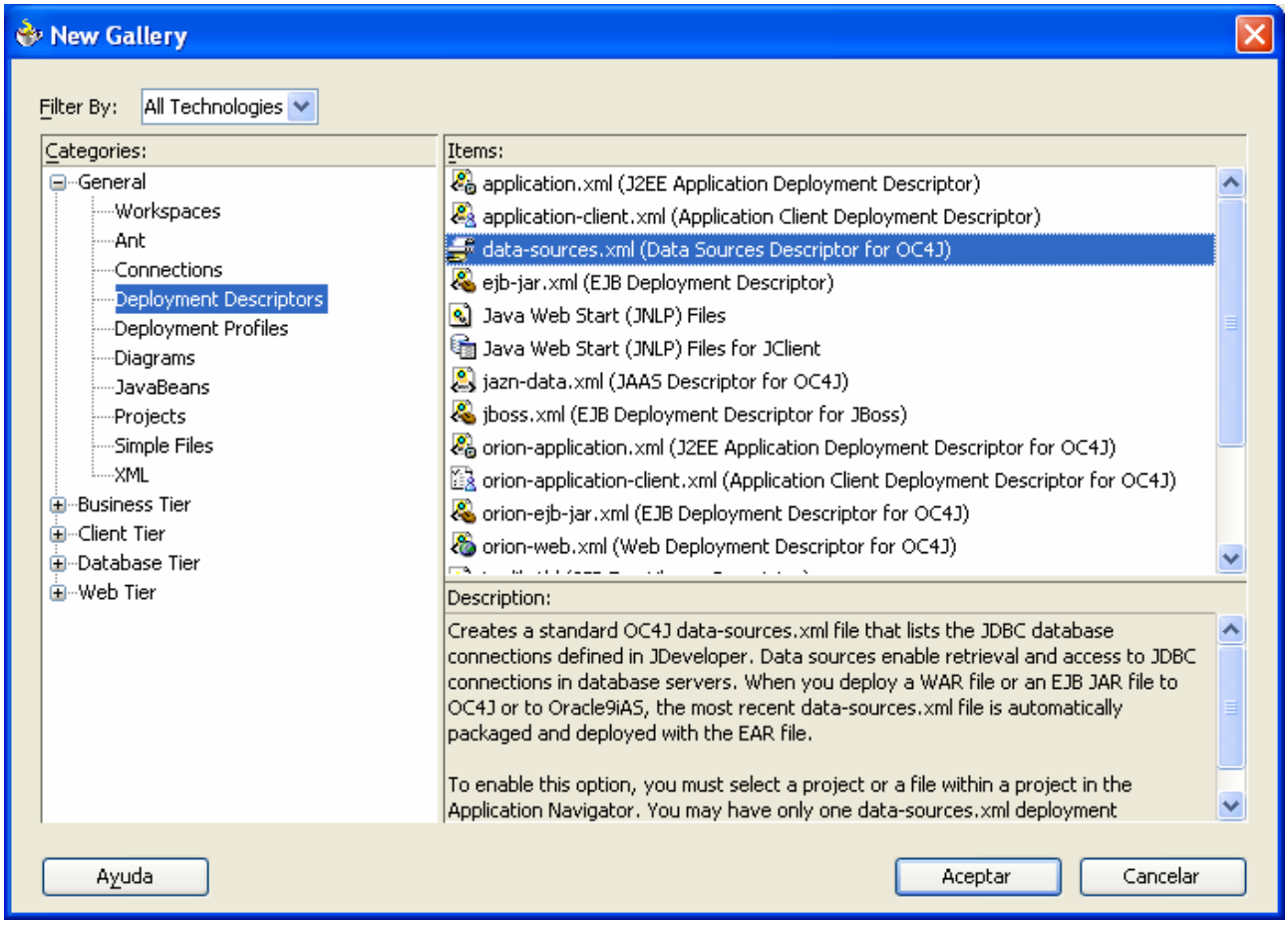
A partir de este punto del tutorial los empezaremos a utilizar por lo tanto necesitamos instalarlo en nuestra copia de JDeveloper. Para ello se debe obtener el fichero .jar que contiene los asistentes adecuado para la versión de JDeveloper que vayamos a utilizar y seguir los pasos de instalación que se pueden descargar junto con el fichero[5]. Si no se obtiene esta documentación, basta con colocar este .jar en el directorio $JDEV_HOME/jdev/lib/ext/, cerrar JDeveloper y volverlo a arrancar y con eso ya estarían accesibles.

7.2 Prerrequisito2: Conexión a bbdd a través de DataSource

Antes de comenzar con el desarrollo, y dado que los datos que va a manejar la aplicación se encuentran en una base de datos, vamos a mostrar los pasos necesarios para obtener la conexión a la BD utilizando un DataSource. Se va a definir un fichero con una estructura específica que permite obtener conexiones a la BD mediante el nombre con el que se ha creado en el fichero. Dentro de este DataSource se tendrá la cadena de conexión, usuario, password....

Para crear un nuevo DataSource, desde jDevelopper seleccionar Nuevo → Deployment Descriptors→data-sources.xml

[5] Actualmente se pueden descargar los tutoriales desde la url : http://gforge.jcyl.es/projects/appbasej2eepub y en la sección Ficheros. Existen 2 versiones, una para JDeveloper 9.0.3 y otra para JDeveloper 10g



Una vez creado el archivo debemos introducir el siguiente texto:

```
<?xml version='1.0' encoding='windows-1252'?>
<!DOCTYPE data-sources PUBLIC "Orion data-sources"
"http://xmlns.oracle.com/ias/dtds/data-sources.dtd">
<data-sources>
       <data-source
              class="com.evermind.sql.DriverManagerDataSource"
              name="GELIPoolDS"
              location="jdbc/GELIPoolDS"
              xa-location="jdbc/xa/GELIPoolXADS"
              connection-driver="oracle.jdbc.driver.OracleDriver"
              username="geli"
              password="geli"
              url="jdbc:oracle:thin:@desjcyl.cf.jcyl.es:1521:dejcyl1"
              inactivity-timeout="30"
       />
</data-sources>
```

Como puede verse en este fichero, hemos dado como nombre del DataSource `location="jdbc/GELIPoolDS"`. Esto es así porque en el fichero de configuración de la aplicación (`app-config.properties`), se ha generado una propiedad con este nombre

```
# Nombre del Pool DataSouce

NOMBRE_DATASOURCE = jdbc/GELIPoolDS,
```

y por tanto la aplicación cuando acceda a los datos almacenados en la BD buscará el un DataSource con ese nombre. Si se quiere cambiar el nombre, es necesario que en los dos ficheros figure el mismo dato puesto que sino la aplicación no podrá recuperarlo.

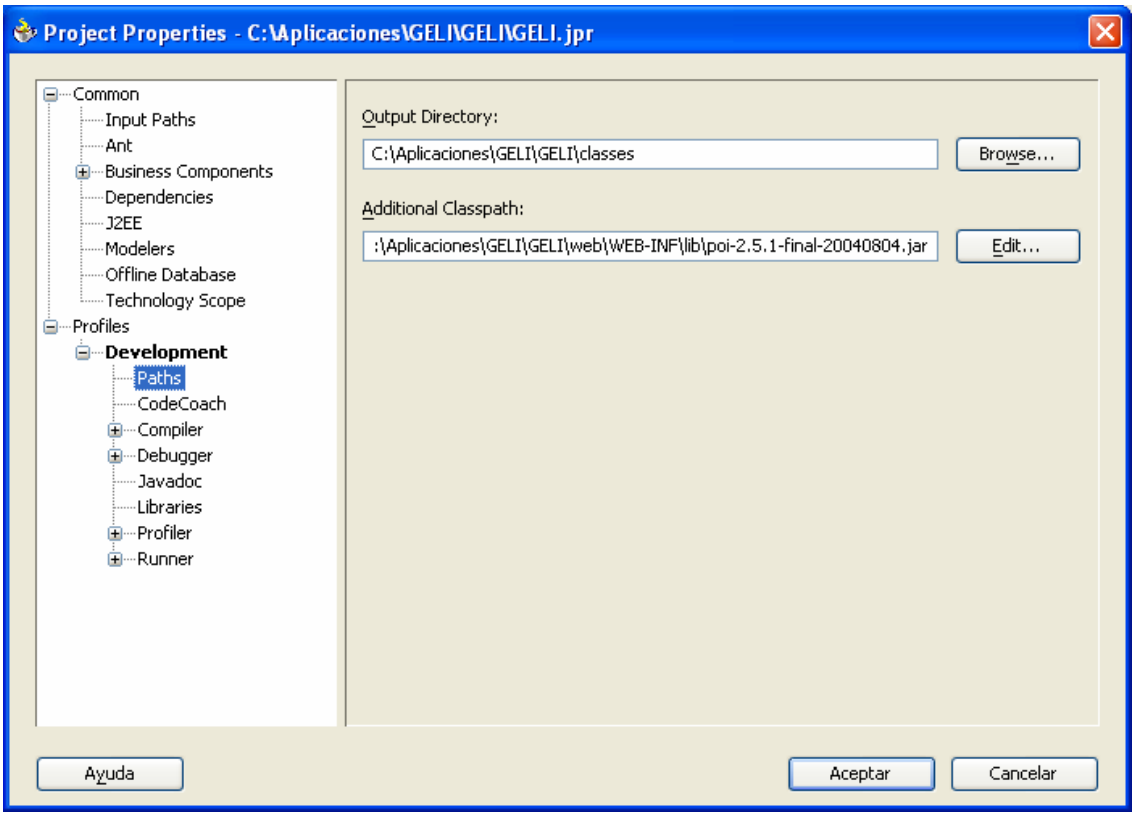
Se debe indicar que estos son los datos de conexión para acceder a nuestro sistema, será necesario cambiar estos valores para acceder a otros servidores de BD.

7.3 Prerrequisito3: Inserción de librerías externas en nuestro proyecto.

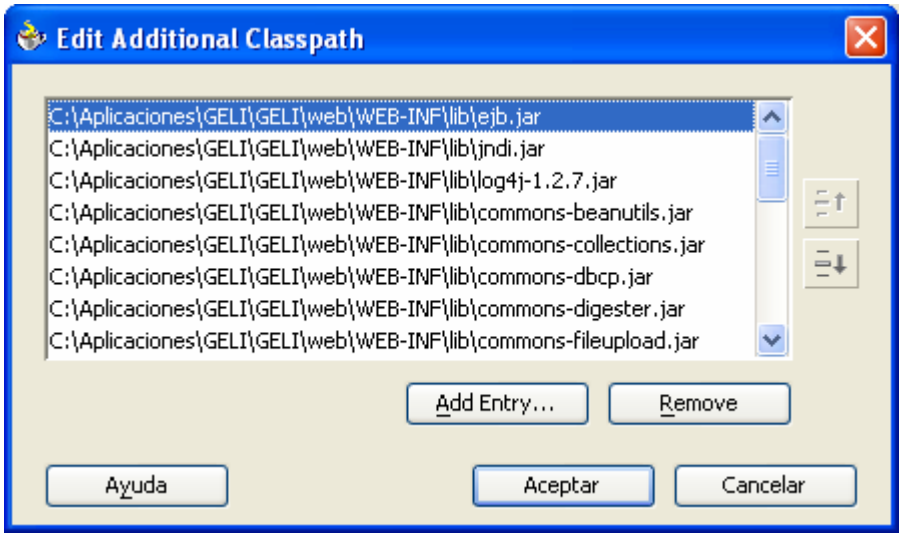
Para añadir una librería externa a nuestro proyecto debemos seguir los siguientes pasos:

1. Copiar la librería en el directorio lib
2. Añadirla al classpath del proyecto

En nuestra aplicación debemos añadir la librería *poi-2.5.1-final-2004-08-04.jar* (La podemos encontrar en la aplicación de ejemplo).
Copiamos el fichero al directorio /web/WEB-INF/lib.
Añadimos la librería al classpath del proyecto. Para ello editamos las propiedades del proyecto. Seleccionamos la opción Paths:



Pulsamos sobre el botón Edit de Additional Classpath y añadimos una nueva entrada :



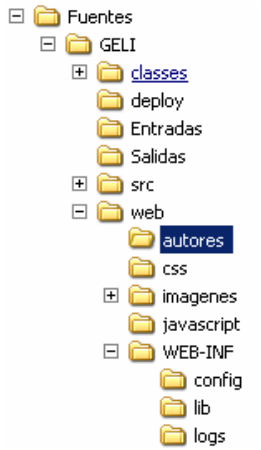
7.4 Desarrollo

Puesto que la aplicación consta de varias funcionalidades diferentes y para distintos usuarios, vamos a desarrollar el perfil que cubre más funcionalidades y una vez visto eso, para los demás perfiles sería similar. Por lo tanto, nos vamos a centrar en los administradores.

Mucho del código mostrado en este documento ha sido generado por los asistentes de ayuda al desarrollo que se van a utilizar a continuación.

Para poder comenzar a insertar libros en el sistema y realizar cualquier consulta, es necesario disponer de los autores que han escrito esos libros. Por ello, vamos a comenzar con la gestión de esos autores.

7.4.1 Inserción de nuevos autores



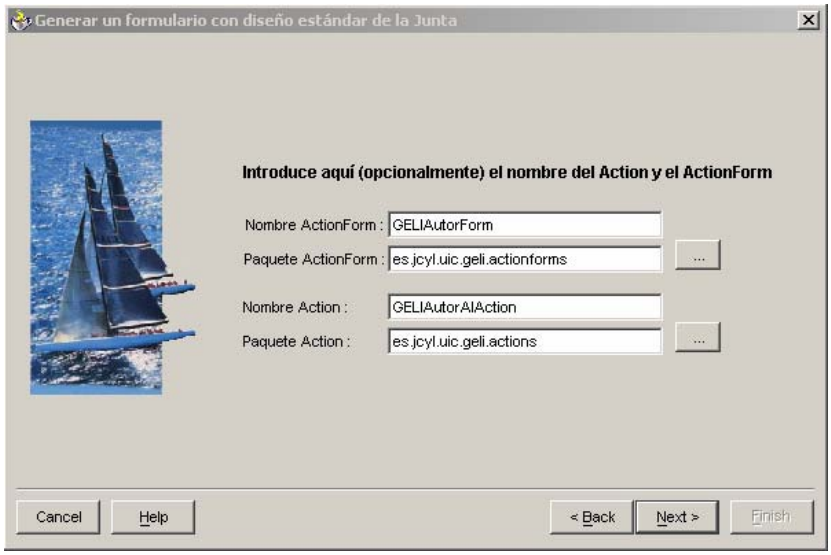
Antes de comenzar, para organizar todas las jsps de autores dentro del mismo directorio creamos un directorio dentro del directorio web llamado autores.

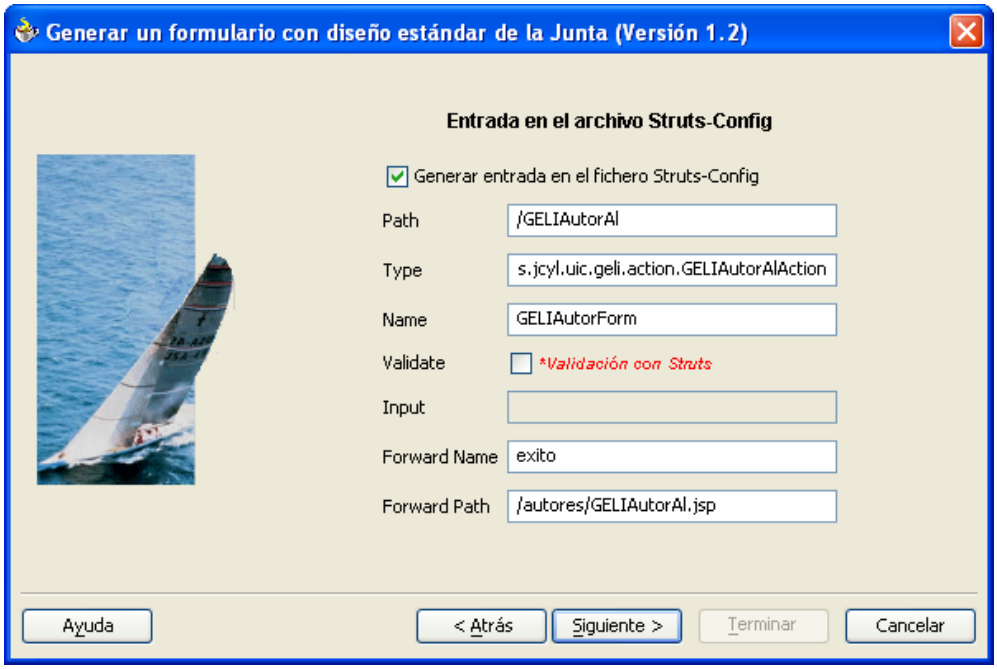
A continuación ya se va a crear la nueva pantalla. Para ello nos ayudamos de las plantillas que proporciona JCyL para JDeveloper que permiten crear páginas jsp, formularios y actions de forma sencilla, realizando la conexión necesaria entre ellas cuando sea necesario.

Para ello pulsamos sobre la opción “new” del menú “File”.En este punto aparece una pantalla con los diferentes asistentes de JDeveloper.

Si se abre el punto “Web Tier” se ve el apartado *“JCYL:* ***Asistentes de ayuda al desarrollo”*** y pulsando sobre el, se muestran los asistentes disponibles.

Para este caso, puesto que se trata de una gestión de autores (inicialmente vamos a realizar la inserción) seleccionamos la primera opción “Generar JSP, ActionForm y Action de formulario estándar”.





Una vez finalizado este proceso, se habrá creado la estructura necesaria para mostrar un formulario en el que se pueden rellenar los datos de los nuevos usuarios para insertarlos en la BD.

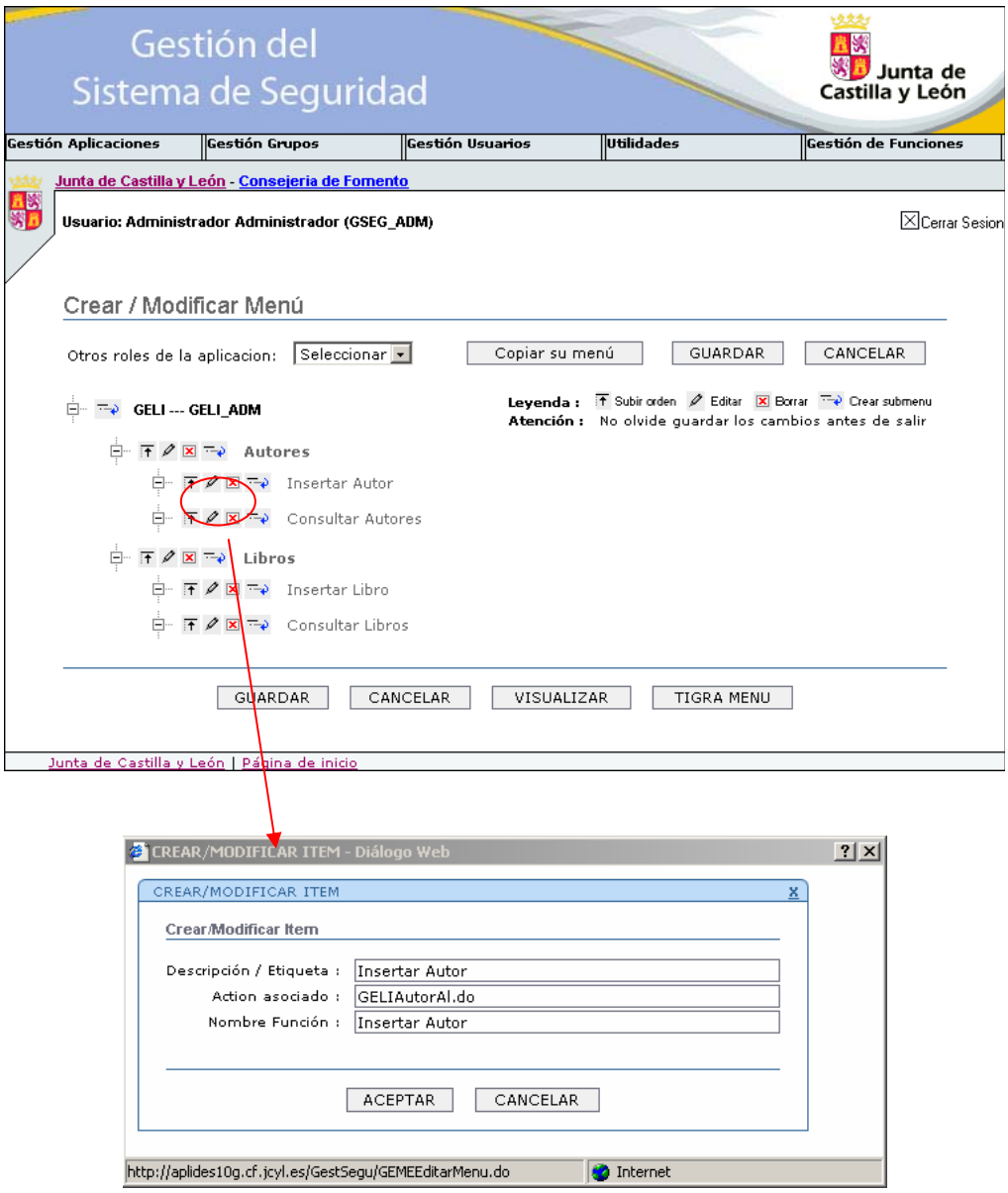
Debemos editar el fichero GELI-config.xml y comprobar que nos ha creado correctamente la entrada para el action GELIAutorAl, ya que si no hemos tenido la precaución de cerrar el fichero antes de lanzar el asistente, puede ser que no nos cree la entrada.

```
<action path="/GELIAutorAl"
        type="es.jcyl.uic.geli.actions.GELIAutorAlAction"
        name="GELIAutorForm"
        scope="request"
        validate="false">
    <forward name="exito" path="/autores/GELIAutorAl.jsp"/>
</action>
```

Este esqueleto sólo mantiene la navegación para hacer pruebas.

Para poder acceder a esta nueva opción debemos permitir que el usuario pueda ejecutar la acción de inserción.

Como dijimos en el capítulo dedicado al sistema de gestión de usuarios, los menús los creamos sin apuntar a ninguna acción. Ahora es el momento de rellenar la acción que queremos ejecutar al pulsar sobre la opción de menú “Insertar Autor”. Para ello, desde el sistema de gestión de seguridad, editando el menú del administrador, y la opción de menú “Insertar Autor”, añadimos la acción que se quiere ejecutar cuando se pulsa sobre ella.



Sin esto, no podremos invocar desde ningún punto el nuevo action que hemos creado. Este es un **error muy frecuente**, y nos aparecerá una pantalla en la que se dice que el usuario no tiene acceso al action solicitado. Una vez hecho esto, se puede probar el funcionamiento de la pantalla creada. Si ejecutamos de nuevo el proyecto, aparecerá la pantalla de login, introducimos los datos del usuario administrador (geliadm/geliadm) y seleccionamos la opción de menú “Autores->Insertar Autor”. Si todo ha ido correctamente se debe mostrar una pantalla como la siguiente.

Si pulsamos sobre el botón “Enviar” los datos se pasarán al servidor para insertarlos en el sistema (de momento no hará nada, y volverá a la misma pantalla).

Esta pantalla es muy genérica y habrá que modificarla para que aparezcan los textos adecuados a nuestra aplicación. Si editamos la pagina /web/usuarios/GELIAutorAl.jsp, podemos modificar el texto para aparezca “FORMULARIO DE ALTA DE AUTORES” por ejemplo.

```
<!-- Titulo -->
<tr><td align="center" class="textoBusquedaGrande">
  FORMULARIO DE ...
```

Lo sustituimos por

```
<!-- Titulo -->
<tr><td     align="center"     class="textoBusquedaGrande">
FORMULARIO DE ALTA DE AUTORES
```

En este formulario se puede ver el código interno del usuario, algo que no es necesario puesto que es un código interno para la BD. Así eliminamos el código encargado de mostrar este campo.

```
<!-- Propiedad "CAutorId" -->
    <tr><td class="textoTituloGris" align="left">
        CAutorId
    </td><td align="left">
        <html:text property="CAutorId"/>
</td></tr>
```

Vamos a añadir un campo oculto al formulario que indique la acción que se quiere realizar cuando se pulsa sobre el botón enviar, en nuestro caso “altaAutor”. Esto es así porque cuando pulsamos

sobre la opción de menú “Insertar Autor” se está invocando el mismo action que cuando se pulsa el botón buscar, y en el primer caso solo queremos presentar la pantalla inicial.

```
<html:hidden property="AAccion" value="altaAutor"/>
```

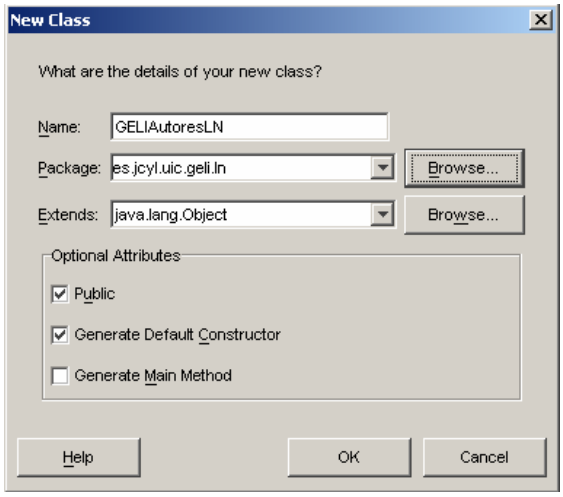
Se pueden cambiar todas las etiquetas que aparecen para que sean más fáciles de reconocer para el usuario de la aplicación.

Para poder recoger este valor en el formulario de usuarios, hemos añadido el siguiente código en la clase GELIAutorForm.

```
private String AAccion = "";

public String getAAccion() {
  return AAccion;
}

public void setAAccion(String AAccion) {
  this.AAccion = AAccion;
```

En la versión actual del asistente, tanto en los ActionForm como en las jsp que se generan, se añade un atributo genérico Class que debe ser suprimido, se trata de un error que se solventará en futuras versiones del asistente.

Ahora es necesario recoger los valores introducidos por el usuario e insertarlos en el sistema. Antes de cambiar el código del Action, es necesario añadir un Objeto de lógica de negocio que se encargue de la lógica de autores, GELIAutoresLN. A este objeto iremos añadiendo los métodos necesarios para la funcionalidad que se quiere implementar. En este caso añadimos un método `GELIAutoresLN.altaAutores(AutoresOT);`



En este caso, para crear la nueva clase, nos colocamos en el paquete es.jcyl.uic.geli.ln y abrimos el menú File->New. Aparece seleccionado por defecto la categoría “General” y seleccionamos Java Class.

Como resultado, aparecerá una nueva clase dentro del paquete indicado, con la que ya podremos trabajar. La clase inicial que nos genera, la debemos modificar para añadir el nuevo método indicado, pero antes habrá que añadir el objeto necesario para escribir en los ficheros de log.

```
public Logger logger = GELIConfigApp.logger;
```

Con esto, la nueva clase quedará como sigue

```
package es.jcyl.uic.geli.actions;
import org.apache.log4j.Logger;
import es.jcyl.uic.geli.config.GELIConfigApp;
import es.jcyl.uic.geli.ot.AutoresOT;
import es.jcyl.uic.geli.oad.AutoresOAD;

public class GELIAutoresLN {
    // objeto para escribir los logs de la aplicación
    public Logger logger = GELIConfigApp.logger;

    /**
     * Constructor genérico
     */
    public GELIAutoresLN() {
    }

    /**
     * Metodo encargado de realizar la inserción de nuevos
     * usuarios en el sistema.
     *
     * @param autorOT OT con los datos del nuevo autor.
     * @return valor entero mayor que cero si se ha insertado correctamente.
     *         0 si no se ha podido insertar.
     * @throws Exception
     */
    public int altaAutores(AutoresOT autorOT) throws Exception {
        int filas = 0;
        try {
            AutoresOAD autoresOAD = AutoresOAD.getInstance();
            filas = autoresOAD.altaAutores(autorOT);
        } catch (Exception e) {
            logger.error(e);
            throw e;
        }
        return filas;
    }
}
```

Una vez creado el método que necesitamos para invocarlo desde el action, vamos a realizar las modificaciones necesarias para crear el nuevo autor.

Abrimos GELIAutorAlAction para añadir el código necesario para construir el objeto autor que vamos a crear.

```
String sig = "exito";

GELIAutorForm formulario = (GELIAutorForm) form;

String accion = formulario.getAAccion();

if (accion != null && accion.equals("altaUsuario")) {
    // no vengo del menú, vengo de la página de alta de usuarios. Necesito
    // recoger todos los valores del formulario, crear el objeto autorOT
    // e invocar el método de la lógica de negocio que crea el autor
    GELIAutoresLN autoresLN = new GELIAutoresLN();
    AutoresOT autorOT = new AutoresOT();

    autorOT.setANombre(formulario.getANombre());
    autorOT.setAApellidos(formulario.getAApellidos());
    autorOT.setALugarNacimiento(formulario.getALugarNacimiento());
    // En este caso, por tratarse de un usuario nuevo, dejo el código del autor
    // sin asignar y el método de creación se encargará de asignarle uno
    try {
        // una vez construido el objeto OT, creo el autor
        autoresLN.altaAutores(autorOT);
    } catch (Exception e) {
        // redirigimos a error poniendo el mensaje de error
        GELIConfigApp.logger.error("Error " + e.getMessage());
        // creamos el nuevo error para que se muestre el mensaje que queremos
        ActionErrors errors = new ActionErrors();
        // la etiqueta error.creacion.autor se debe añadir al fichero
        // src/es.jcyl.uic.geli/ApplicatonResources.properties
        // error.creacion.autor=Se ha producido un error en la
        // creación del autor
        errors.add(ActionErrors.GLOBAL_ERROR,
                new ActionError("error.creacion.autor"));
        saveErrors(request, errors);
        sig= "fallo;
    }
}
```

Finalmente, debemos modificar el objeto de acceso a datos, que será quien realmente creará el objeto en el sistema. Esta modificación se necesita porque hemos eliminado la posibilidad de introducir un código de autor desde el formulario, y por tanto, será necesario generar un código único en el sistema. Para ello, vamos a añadir un nuevo método privado que se encargue de obtener este ID. Lo hemos llamado `obtenerCodigoAutor()`. Y se encuentra en la clase es.jcyl.uic.geli.oad.AutoresOAD.

El Código obtenido se asigna al OT de autores antes de invocar el método altaAutores del OAD, con lo que el método altaAutores de GELIAutoresLN nos quedaría:

```
try {
    AutoresOAD autoresOAD = AutoresOAD.getInstance();
    autorOT.setCAutorId(autoresOAD.obtenerCodigoAutor());
    filas = autoresOAD.altaAutores(autorOT);
} catch (Exception e) {
    logger.error(e);
    throw e;
}
```

```
/**
     * Método encargado de obtener un código único de autor
     *
     * @return long con el código que se debe dar al nuevo autor
     * @throws SQLException
     */
    private long obtenerCodigoAutor() throws SQLException {
        ResultSet rs = null;
        PreparedStatement st = null;
        Connection con=JCYLGestionTransacciones.getConnection();

        String sqlSecuencia = "SELECT GELI_AUT_C_AUTOR_ID_SEQ.NEXTVAL FROM DUAL";
        int cAutor = -1;
        try {
            st = con.prepareStatement(sqlSecuencia);
            rs = st.executeQuery(sqlSecuencia);

            if (rs.next()) {
                cAutor = rs.getInt(1);
            }
        } catch (SQLException e) {
            throw e;
        } finally {
            JCYLGestionTransacciones.close(con.getAutoCommit());
        } // try-catch

        return cAutor;
    }
```

Con todo este código generado, ya es posible insertar usuarios en el sistema, pero la página a la que se va a volver después de insertar el usuario es la misma que desde la que se ha insertado. Para evitar esto, vamos a cambiar la página destino por una nueva jsp en la que se muestre un texto indicando que todo ha ido correctamente. Esta página ya se ha generado con la aplicación base y se llama mensajes.jsp. Esta página debe recibir un parámetro llamado “mensaje”. Se debe modificar el action que ha realizado la inserción para añadir el siguiente código

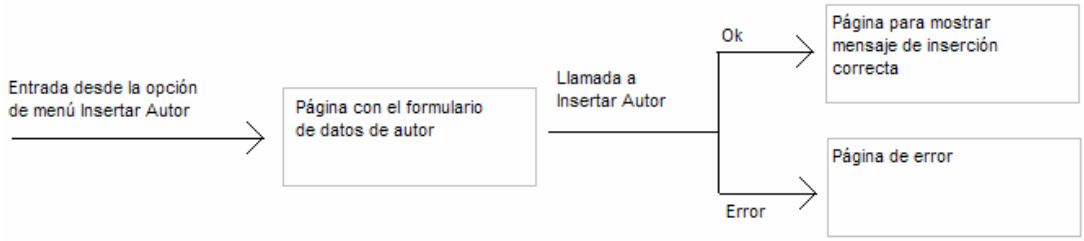
```
try {
     // una vez construido el objeto OT, creo el autor
     autoresLN.altaAutores(autorOT);
     // indicamos que  ha ido correctamente
     request.setAttribute("mensaje", "El autor ha sido creado correctamente.");
     sig = "altaAutorOk";
  } catch (Exception e) {

 ........
```

Para cambiar la redirección de página destino debemos editar el fichero GELI-config.xml y ahí encontraremos las definiciones de los action y las paginas a las que se hace la redirección.

Queremos añadir una nueva redirección para el nombre `altaAutorOk`.

Quedaría como sigue:

```
<action path="/GELIAutorAl"
      type="es.jcyl.uic.geli.actions.GELIAutorAlAction"
      name="GELIAutorForm"
      scope="request"
      validate="false">
      <forward name="exito" path="/autores/GELIAutorAl.jsp"/>
      <forward name="altaAutorOk" path="/mensajes.jsp"/>
 </action>
```



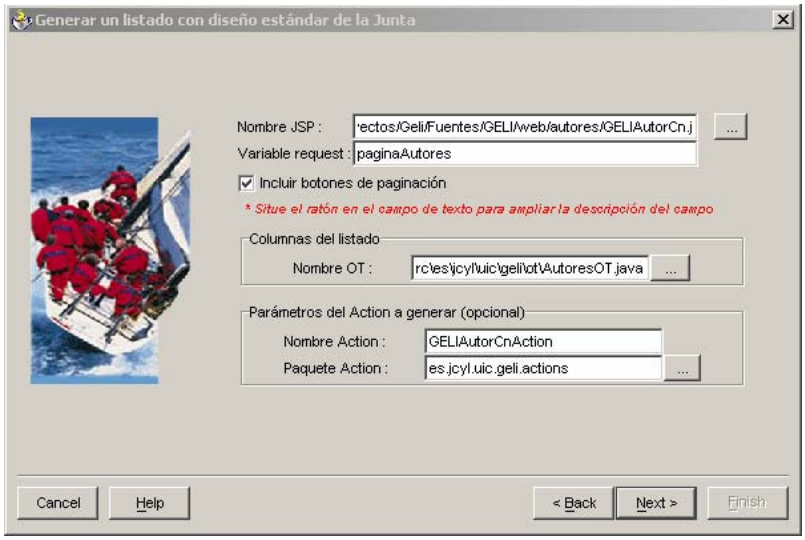
Habría que añadir las validaciones necesarias para que no se permita enviar el formulario si no se han rellenado todos los valores obligatorios, algo que puede verse en el código que se ha desarrollado.

Resumiendo, los pasos que hemos dado son los siguientes

- creación del esqueleto de la funcionalidad necesaria a través del asistente.
- Creación/modificación del menú del usuario en GestSegu
- Creación/modificación del objeto de lógica de negocio que va a implementar la funcionalidad.
- Modificación/creación de los métodos del objeto OAD para invocarlos desde el objeto de lógica de negocio.
- Modificación del action para recoger los datos desde la página y devolver el resultado correspondiente.
- Modificación de la página jsp creada.

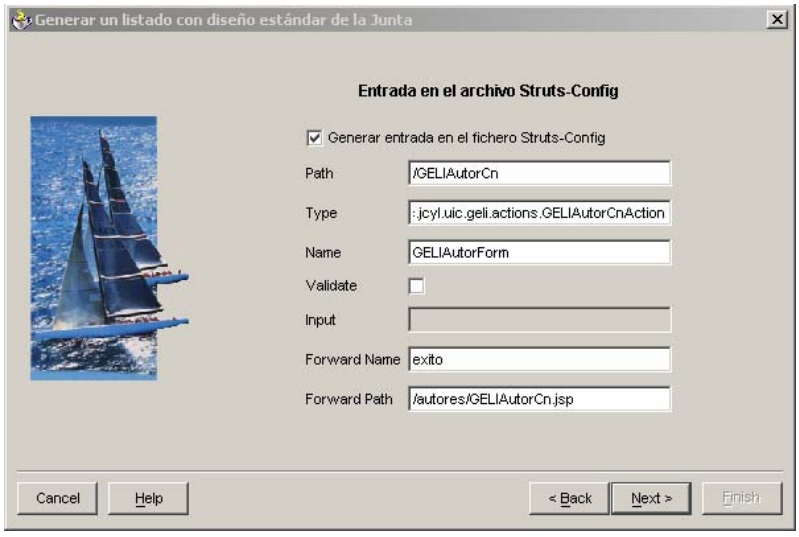
7.4.2 Consulta de autores

Una vez terminada la inserción de los autores en la BD, vamos a implementar la consulta, incluyendo paginación para navegar por los resultados.



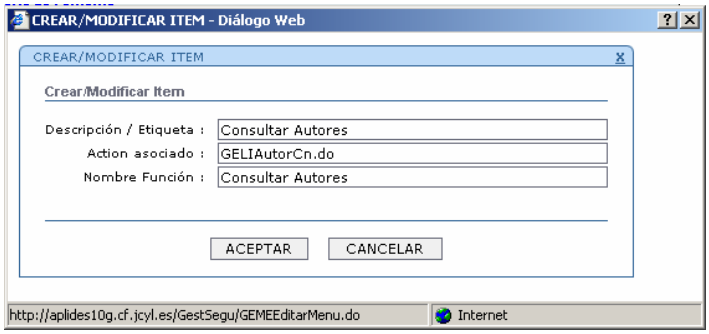
Comenzaremos con un simple listado de todos los usuarios y al final, añadiremos un formulario de búsqueda para filtrar los resultados.

Una vez más, utilizaremos los asistentes proporcionados por JCyL, en este caso seleccionamos "Generar JSP y Action de listado estándar"



Una vez finalizado el asistente, igual que ocurrió en el punto anterior, disponemos de una nueva estructura con la navegación necesaria para listar los autores que se han dado de alta en el sistema. Inicialmente no aparecerán datos porque será necesario modificar las clases para implementar la funcionalidad necesaria.

Antes de comenzar con este proceso, es necesario añadir la nueva acción al menú del usuario, para que cuando pulse sobre el menú "Consultar Autores" se muestre esta página. Desde GestSegu se puede realizar esta actualización.



El siguiente paso será probar que realmente la aplicación nos muestra esta opción de menú que acabamos de crear. Para ello se debe ejecutar el proyecto, acceder a la aplicación con el perfil adecuado y acceder a la opción de menú “Consultar Autores”.

Si todo va correctamente, deberá aparecer una página de error, puesto que en la primera entrada que se hace no hay registros y no se controla eso por defecto. Hay que editar la página jsp creada y modificar el siguiente código.

```
<!-- Recorremos los elementos -->
<logic:present name="paginaAutores" property="elementos">
    <logic:iterate id="elemento" name="paginaAutores" property="elementos">
      <tr><td>
        <bean:write name="elemento" property="CAutorId"/>
      </td><td>
        <bean:write name="elemento" property="ALugarNacimiento"/>
      </td><td>
        <bean:write name="elemento" property="AApellidos"/>
      </td><td>
        <bean:write name="elemento" property="ANombre"/>
      </td></tr>
    </logic:iterate>
</logic:present>
```

Ahora si que podremos ver correctamente la página. Igual que se comentó en el apartado anterior, se pueden modificar todos los textos de la página para que muestren los adecuados a la funcionalidad que estamos desarrollando.

Como ya se ha comentado en otros puntos de este manual, en la versión actual del asistente, tanto en los ActionForm como en las jsp que se generan, se añade un atributo genérico Class que debe ser suprimido, se trata de un error que se solventará en futuras versiones del asistente.

El resultado debe ser similar al siguiente

Puesto que la página ya está creada, vamos a añadir el código necesario para realizar la búsqueda de autores y poder mostrar datos reales en esta pantalla.

Lo primero que debemos hacer es añadir nuevos métodos en el objeto de lógica de negocio de autores, `GELIAutoresLN.listadoAutores()` y `GELIAutoresLN.listadoAutoresCuenta()`, quedando como sigue:

```
/**
     * Funcion de listado para los autores
     *
     * @param inicio  Registro a partir del cual devolver. <I>1</I> para primero.
     * @param cuantos Numero de registros a devolver. <I>-1</I> para todos.
     * @throws Exception En caso de error con la obtención del listado
     * @return Un ArrayList con OT con los autores
     */
    public ArrayList listadoAutores(int inicio, int cuantos) throws Exception {
        ArrayList listado = null;
        try {
            AutoresOAD autoresOAD = AutoresOAD.getInstance();
            autoresOAD.listadoAutores(inicio,cuantos);
        } catch (Exception e) {
            logger.error(e);
            throw e;
        }
        return listado;
    }
    /**
     * Método que devuelve el número de autores del sistema
     * @return número de autores del sistema.
     * @throws Exception si se produce algún error al recuperar
     *                   el número
     */
     public int listadoAutoresCuenta() throws Exception {
        int total = 0;
        try {
            AutoresOAD autoresOAD = AutoresOAD.getInstance();
            autoresOAD.listadoAutoresCuenta();
        } catch (Exception e) {
            logger.error(e);
            throw e;
        }
        return total;

  }
```

Este segundo método invoca a uno nuevo en el objeto de acceso a datos que previamente habremos creado y que se encarga de contar todos los autores que se encuentran en el sistema. El código de este nuevo método se puede ver a continuación

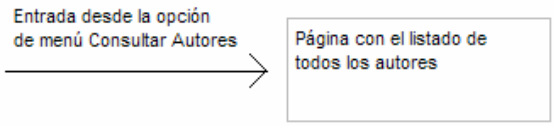
```
/**
 * Devuelve el número total de registros que hay en el sistema
 *
 * @return número total de registros.
 * @throws SQLException
 */
public int listadoAutoresCuenta() throws SQLException {
    ResultSet rs = null;
    Statement st = null;
    Connection con = JCYLGestionTransacciones.getConnection();


    String sqlSecuencia = "SELECT COUNT(*) FROM AUTORES";
    int total = -1;
    try {
        st = con.prepareStatement(sqlSecuencia);
        rs = st.executeQuery(sqlSecuencia);

        if (rs.next()) {
            total = rs.getInt(1);
        }
    } catch (SQLException e) {
        throw e;
    } finally {
        JCYLGestionTransacciones.close(con.getAutoCommit());
    } // try-catch

    return total;
}
```

Con esto ya tenemos una lógica que permite obtener todos los autores del sistema de forma paginada, sólo hace falta invocarlo desde el action que será el encargado de devolverlos al usuario. A continuación se puede ver el código necesario para realizar el siguiente cambio.

En la versión actual del asistente, este método hace uso de la clase es.jcyl.cf.utiles.Pagina que está deprecada, se trata de un error que se solventará en futuras versiones del asistente y únicamente hay que ignorar el warning que obtenemos.

```
 Collection listado = null;
try {
    GELIAutoresLN autoresLN = new GELIAutoresLN();
    listado = autoresLN.listadoAutores(primerRegistro, registrosPorPagina);
    int nListado = 0;
    nListado = autoresLN.listadoAutoresCuenta();

    // Creacion del objeto pagina
    Pagina pagina = new Pagina(mapping.getPath() + ".do", request);
    pagina.setElementos(listado);
    pagina.setNRegistros(nListado);
    pagina.setRegistroActual(primerRegistro);
    pagina.setRegistrosPorPagina(registrosPorPagina);

    // Meter en la request el resultado
    request.setAttribute("paginaAutores", pagina);

    // Redirigimos a exito
    sig = "exito";
} catch (Exception e) {
    sig = "fallo";
    GELIConfigApp.logger.error(e.getMessage());

    ActionErrors errors = new ActionErrors();
    // esta etiqueta error.consulta.usuario se debe añadir al fichero
    // src/es.jcyl.uic.geli/ApplicatonResources.properties
    // error.consulta.autor=Se ha producido un error en la consulta de autores
    errors.add(ActionErrors.GLOBAL_ERROR,
                                    new ActionError("error.consulta.autor"));
    saveErrors(request, errors);
}
return mapping.findForward(sig);
```

El flujo de pantallas obtenido debe ser el siguiente

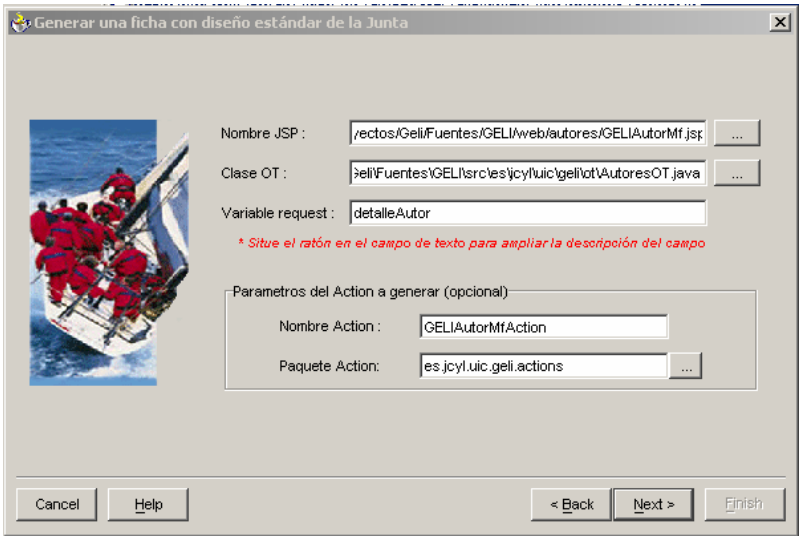


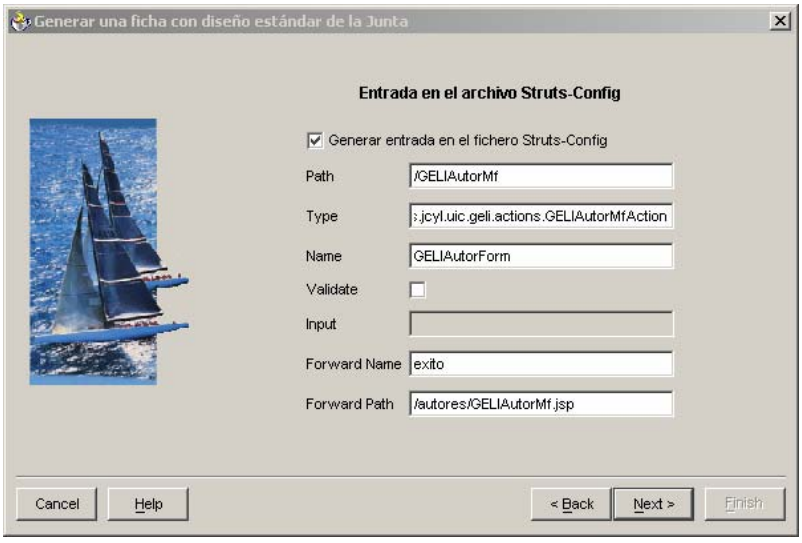
Los pasos principales que hemos dado son los siguientes:

- creación del esqueleto de la funcionalidad necesaria a través del asistente.
- Creación/modificación del menú del usuario en GestSegu
- Creación/modificación del objeto de lógica de negocio que va a implementar la funcionalidad.
- Modificación/creación de los métodos del objeto OAD para invocarlos desde el objeto de lógica de negocio.
- Modificación del action para recoger los datos desde la página y devolver el resultado correspondiente.
- Modificación de la página jsp creada

7.4.3 Edición de los datos de un autor

El siguiente paso en la gestión de los autores es la edición y modificación de los datos de los autores insertados en el sistema. Una vez más, seguiremos los mismos pasos que en puntos anteriores, apoyándonos en los asistentes proporcionados. Para esta opción utilizaremos "Generar JSP y Action de ficha"





En esta ocasión no vamos a probar directamente que lo generado funciona correctamente, puesto que antes vamos a tener que seleccionar el autor cuyos datos queremos editar. Puesto que ya tenemos el listado de autores, que hemos desarrollado en el punto anterior, vamos a modificarlo para que al pulsar sobre el campo CAutorId se invoque a esta nueva funcionalidad

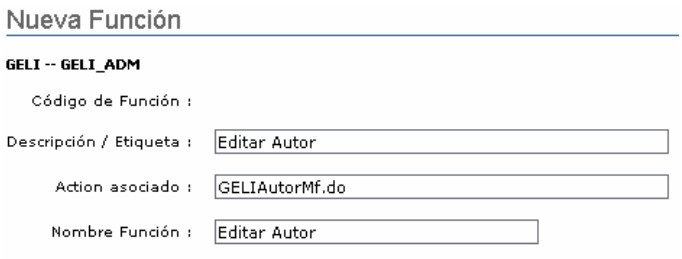
A continuación se puede ver el código modificado en la página jsp: GELIAutorCn.jsp.

```
<script language="JavaScript">
      function editar(CAutorId) {
          document.forms[0].CAutorId.value=CAutorId;
          document.forms[0].submit();
      }
    </script>
    <!--Formulario para el envío del código de usuario que queremos editar  -->
      <html:form action="/GELIAutorMf.do">
        <html:hidden property="CAutorId" value=""/>
        <html:hidden property="AAccion" value="cargar"/>
      </html:form>
```

El código para invocar esta función se debe colocar en el código del autor

```
<a href="javascript:editar('<bean:write name="elemento" property="CAutorId"/>')" >
                <bean:write name="elemento" property="CAutorId"/></a>
```

Una vez modificado el código de la página jsp, debemos permitir que el usuario acceda a este Action, para ello habrá que ir a la aplicación de gestión de seguridad y crear una nueva función (en este caso no tiene que ser de menú) asociándole el Action que queremos invocar.



Una vez realizadas todas estas modificaciones es el momento de modificar el action para que se pueda saber el código del usuario sobre el que hayamos pulsar y enviar a la página de modificación que habíamos creado con el asistente. Para ello, hemos de crear un nuevo método el en objeto de lógica de negocio de usuarios que se encargue de cargar dichos datos.

```
/**
     * Metodo encargado de cargar los datos de un autor a partir
     * de su código
     *
     * @param cAutorId código del autor que queremos cargar
     * @return
     * @throws Exception
     */
    public AutoresOT buscarAutores(long cAutorId) throws Exception {
        AutoresOT autorOT = null;
        try {
            AutoresOAD autoresOAD = AutoresOAD.getInstance();
            autorOT = autoresOAD.buscarAutores(cAutorId);
        } catch (Exception e) {
            logger.error(e);
            throw e;
        }
        return autorOT;
    }
```

El siguiente paso, igual que en funcionalidades anteriores, es modificar el Action para invocar este método pasándole el código del sobre el que el usuario haya pulsado. El código modificado es el siguiente:

```
       String sig = "exito";

       GELIAutorForm formulario = (GELIAutorForm) form;

       String accion = formulario.getAAccion();
        AutoresOT detalleAutor = null;
        GELIAutoresLN autoresLN = new GELIAutoresLN();
        if (accion != null && accion.equals("cargar")) {
             long cAutorId = Long.parseLong(formulario.getCAutorId());
             try {
               detalleAutor=autoresLN.buscarAutores(cAutorId);
             } catch (Exception e) {
                 sig = "fallo";
                 GELIConfigApp.logger.error(e.getMessage());

                 ActionErrors errors = new ActionErrors();
                 // esta etiqueta error.consulta.usuario se debe añadir al fichero
                 // src/es.jcyl.uic.geli/ApplicatonResources.properties
                 // error.consulta.autor=Se ha producido un error en
                 // la consulta de autores
                 errors.add(ActionErrors.GLOBAL_ERROR,
                         new ActionError("error.consulta.autor"));
                 saveErrors(request, errors);
             }
        }
        // Meter en la request el OT
        request.setAttribute("detalleAutor", detalleAutor);
        return mapping.findForward(sig);
```

Si a continuación ejecutamos el proyecto podremos comprobar que el funcionamiento es el que queríamos, obtenemos la ficha de detalle del autor sobre el que hemos pulsado. Igual que en casos anteriores, la jsp que se genera desde el asistente contiene textos genéricos que deberán ser actualizados para reflejar el contenido particular, y debemos eliminar el elemento adicional que nos añade el asistente.

Puesto que en la ficha generada los datos mostrados no se pueden modificar, vamos a cambiar esta página para que esto sea posible, y añadiremos un botón que permita enviar los datos al servidor para ser guardados y otro que permita volver a los valores iniciales del usuario.

Para conseguir esto se han realizado las siguientes modificaciones:

donde aparecía, por ejemplo, `<bean:write name="detalleAutor" property="ANombre"/>`

se ha cambiado por `<html:text name="detalleAutor" property="ANombre"/>`,

Además se ha añadido código javascript para verificar que se envían al menos los datos obligatorios

```
<script language="JavaScript">
    function comprobarObligatorios(){
        if(document.forms[0].ANombre.value == '' ) {
            alert('Debe rellenar el campo Nombre');
            return false;
        }
        if(document.forms[0].AApellidos.value == '' ) {
            alert('Debe rellenar el campo Apellidos');
            return false;
        }
     }
```

Este código se invoca en el submit del nuevo formulario que hemos añadido

```
<html:form action="GELIAutorMf.do" onsubmit="return comprobarObligatorios()">
    <html:hidden property="AAccion" value="actualizar"/>
```

La tabla con los datos del autor también debe formar parte del formulario, por lo tanto, el mismo debe cerrarse al final de la página. En este punto, lo único que nos queda es añadir los botones a nuestra página:

```
 <!-- Botones de envío y reset -->
 <tr><td colspan="2" class="textoTituloGris" align="left">
   <html:submit styleClass="formBoton">Actualizar</html:submit>   
   <html:reset styleClass="formBoton">Restaurarvalores</html:reset>   
   <html:buttonstyleClass="formBoton"property="Cancelar"
      onclick="document.location.href='GELIAutorCn.do'"> Volver </html:button>
 </td></tr>
 </table>
 </td></tr>
 </table>
</html:form>
<jsp:include page="/bottom.jsp"/>
```

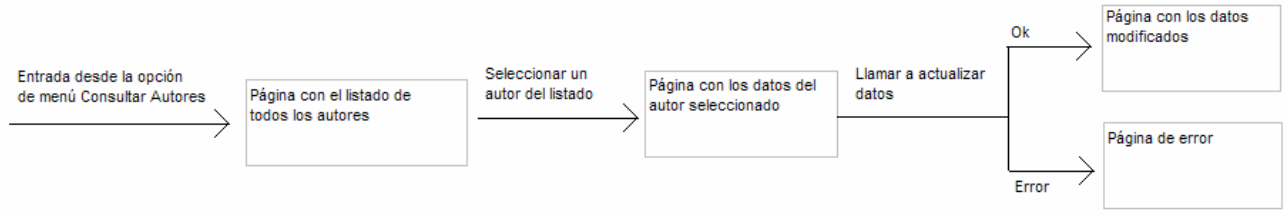
Con todo esto ya tenemos la página preparada para modificar los datos del autor. Ahora solo falta modificar la parte servidora. Igual que en todos los casos anteriores, añadiremos un nuevo método en el objeto de lógica de negocio y luego el action para recoger los nuevos valores y llamar a este método.

```
/**
     * Método encargado de realizar la modificación de los datos de un
     * usuario que recibe como entrada
     *
     * @param datos
     * @return
     * @throws Exception
     */
    public int modificacionAutores(AutoresOT datos) throws Exception {
        int filas = 0;
        try {
            AutoresOAD autoresOAD = AutoresOAD.getInstance();
            filas = autoresOAD.modificacionAutores(datos);
        } catch (Exception e) {
            logger.error(e);
            throw e;
        }
        return filas;
    }
```

A continuación se muestra la modificación del action

```
} else if (accion != null && accion.equals("actualizar")) {
      // En este caso recuperamos los datos del usuario
      // modificados y los guardamos en el sistema
      detalleAutor = new AutoresOT();
      detalleAutor.setCAutorId(Long.parseLong(formulario.getCAutorId()));
      detalleAutor.setANombre(formulario.getANombre());
      detalleAutor.setAApellidos(formulario.getAApellidos());
      detalleAutor.setALugarNacimiento(formulario.getALugarNacimiento());
      try {
          autoresLN.modificacionAutores(detalleAutor);
      } catch (Exception e) {
          sig = "fallo";
          GELIConfigApp.logger.error(e.getMessage());
          ActionErrors errors = new ActionErrors();
          // esta etiqueta error.actualizacion.autor se debe añadir al fichero
          // src/es.jcyl.uic.geli/ApplicatonResources.properties
          // error.actualizacion.autor=Se ha producido un
          // error en la actualización de los datos del autor
          errors.add(ActionErrors.GLOBAL_ERROR,
                  new ActionError("error.actualizacion.autor"));
          saveErrors(request, errors);
      }
}// Meter en la request el OT
request.setAttribute("detalleAutor", detalleAutor);
return mapping.findForward(sig);
```

El resultado obtenido debe seguir el siguiente flujo de datos



Los pasos principales que hemos dado son los siguientes:

- creación del esqueleto de la funcionalidad necesaria a través del asistente.
- Creación/modificación de las funciones del usuario en GestSegu.
- Modificación de la jsp de consulta (creada en el punto anterior) para poder invocar a la nueva funcionalidad que queremos implementar.
- Creación/modificación del objeto de lógica de negocio que va a implementar la funcionalidad.
- Modificación/creación de los métodos del objeto OAD para invocarlos desde el objeto de lógica de negocio.
- Modificación del action para recoger los datos desde la página y devolver el resultado correspondiente.
- Modificación de la página jsp creada.

Con todo esto se puede dar por terminado el mantenimiento de los autores.

7.4.4 Gestión de libros

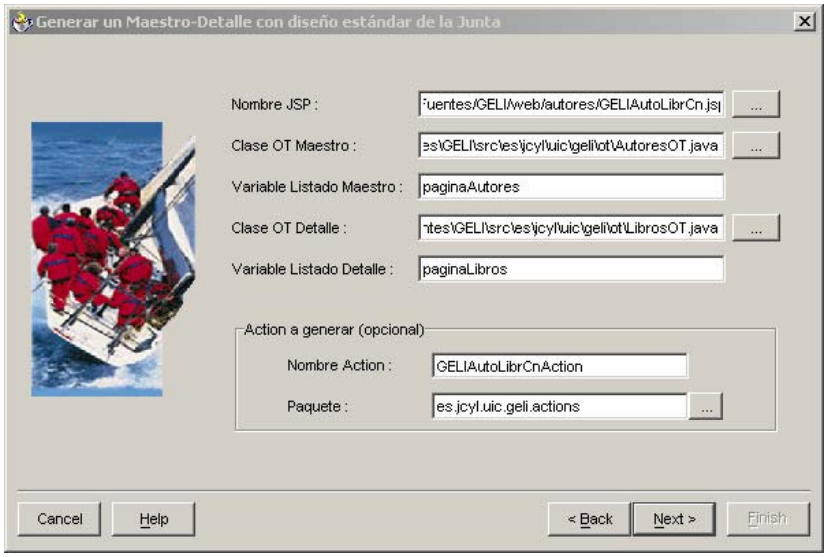
Una vez que tenemos los autores, es el momento de implementar la funcionalidad que permita dar de alta, modificar y consultar los libros de cada autor en el sistema. Puesto que las pantallas a desarrollar son las mismas que en el caso anterior, no vamos a detenernos en el desarrollo y se entregará el código generado con el resto de la aplicación. Las funcionalidades que vamos a desarrollar son

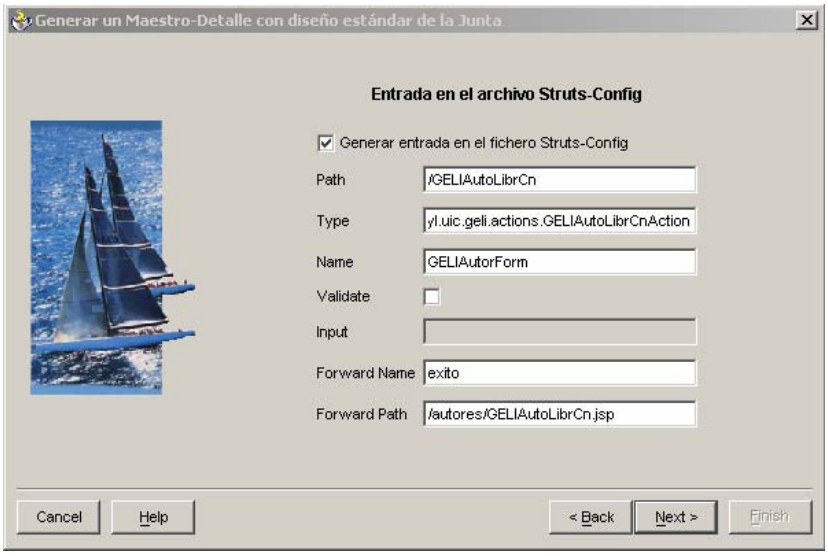
- alta de libros
- listado de libros
- modificación de libros

En este caso, y para separar las páginas generadas para esta funcionalidad de las generadas para autores, vamos a crear un nuevo directorio, a la misma altura que el de autores y que se va a llamar libros.

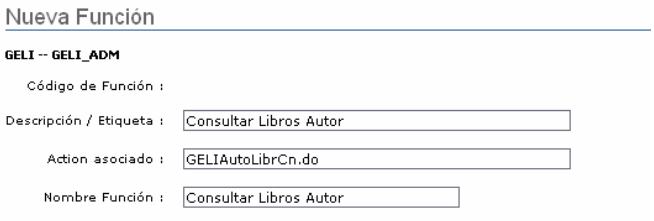
7.4.5 Consulta de libros de cada autor

Puesto que ya tenemos la funcionalidad que maneja autores y libros, vamos a añadir una nueva funcionalidad que permita seleccionar un autor, y ver para él todos los libros que ha escrito. Una vez más, utilizaremos los asistentes de generación proporcionados por la JCyL. En este caso utilizaremos la opción “Generar JSP y Action para un Maestro-Detalle”.





Una vez generado el esqueleto, debemos añadir una nueva función al perfil de administrador en el sistema de seguridad para permitir invocar esta nueva funcionalidad.



En este momento es necesario decidir el punto en el que enganchar esta nueva funcionalidad. Puesto que queremos ver los libros que ha escrito una autor, podemos modificar la página de detalle de un autor y añadir un nuevo botón que al pulsar sobre él la invoque, o en la página del listado de autores, añadamos en cada línea un botón que pulsando sobre él la invoque.

Optaremos por este segundo caso, y por tanto vamos a añadir este nuevo botón (imagen ir.gif que colocaremos en el directorio /imágenes/).

Veamos el código que es necesario añadir en la página `GELIAutorCn.jsp`.

Es necesario añadir una nueva columna (tanto en la cabecera de la tabla, como en los resultados)

```
<script language="JavaScript">
    function editar(CAutorId) {
         document.forms[0].CAutorId.value=CAutorId;
         document.forms[0].submit();
    }
    function ver(CAutorId) {
         document.forms[0].CAutorId.value=CAutorId;
         document.forms[0].action='<html:rewrite page="/GELIAutoLibrCn.do"/>';
         document.forms[0].submit();
    }
</script>
<td class="tablaFondoNegro">Ver Libros
</td>
```

y

```
<td>
   <a href="javascript:ver('<bean:write name="elemento" property="CAutorId"/>')">
      <html:img page="/imagenes/ir.gif" border="0" alt="Ver Libros Escritos" />
   </a>
</td>
```

Al pulsar sobre esta nueva imagen, pasaremos a la nueva pantalla que hemos creado con el asistente. Como ocurría con la página de listado de autores, aparecerá un error puesto que es una página genérica en la que no se ha comprobado que haya resultados. Para evitar este error, habrá que hacer esta comprobación,

```
<!-- Recorremos los elementos -->
    <logic:present name="paginaAutores" property="elementos">
        <logic:iterate id="elemento" name="paginaAutores" property="elementos">
          <tr><td>
            <bean:write name="elemento" property="CAutorId"/>
          </td><td>
            <bean:write name="elemento" property="ANombre"/>
          </td><td>
            <bean:write name="elemento" property="AApellidos"/>
          </td><td>
            <bean:write name="elemento" property="ALugarNacimiento"/>
          </td></tr>
        </logic:iterate>
     </logic:present>


<!-- Recorremos los elementos -->
    <logic:present name="paginaLibros" property="elementos">
        <logic:iterate id="elemento" name="paginaLibros" property="elementos">
          <tr><td>
            <bean:write name="elemento" property="CLibroId"/>
          </td><td>
            <bean:write name="elemento" property="DTitulo"/>
          </td><td>
            <bean:write name="elemento" property="ADescripcion"/>
          </td><td>
            <bean:write name="elemento" property="DNombreGenero"/>
          </td>td>
            <bean:write name="elemento" property="FEdicionStr"/>
          </td></tr>
        </logic:iterate>
    </logic:present>
```

En este punto tenemos la navegación deseada, pero como puede verse, en el listado de libros que aparece se pueden eliminar las columnas que hacen referencia a un código y que tienen el nombre asociado a ese código (ej, aparece CGeneroId y DNombreGenero, dejamos DNombreGenero) consiguiendo algo parecido a la siguiente pantalla

Igual que en casos anteriores, una vez conseguido que la navegación sea correcta, es necesario modificar el action para cargar los valores que se van a mostrar en la pantalla anterior. En la tabla maestro, siempre va a aparecer un único registro, por tanto, habrá que eliminar la paginación de esta tabla (ya la hemos eliminado en la imagen anterior). Hay que modificarlo en el action y en la jsp.

A continuación puede verse el código añadido al action para el correcto funcionamiento.

Obtener los datos para la tabla maestro:

```
GELIAutorForm formulario = (GELIAutorForm) form;
GELIAutoresLN autoresLN = new GELIAutoresLN();
try {
     autorOT = autoresLN.buscarAutores
                          (Integer.parseInt(formulario.getCAutorId()));
} catch (Exception e) {
      sig = "fallo";
      GELIConfigApp.logger.error(e.getMessage());
      ActionErrors errors = new ActionErrors();
      // esta etiqueta error.consulta.autor se debe añadir al fichero
      // src/es.jcyl.uic.geli/ApplicatonResources.properties
      // error.consulta.autor=Se ha producido un error en
      // la consulta de autores
      errors.add(ActionErrors.GLOBAL_ERROR,
              new ActionError("error.consulta.autor"));
      saveErrors(request, errors);
      return mapping.findForward(sig);
  }
  request.setAttribute("detalleAutor", autorOT);
```

Datos para la tabla de detalle:

```
Collection listadoDetalle = null;
int nListadoDetalle = 0;
GELILibrosLN librosLN = new GELILibrosLN();
try {
     listadoDetalle =
              librosLN.buscarLibrosAutor(primerRegistroDetalle,
              registrosPorPaginaDetalle,
              Integer.parseInt(formulario.getCAutorId()));
     nListadoDetalle =
              librosLN.buscarLibrosAutorCuenta(
                                  Integer.parseInt(formulario.getCAutorId()));
} catch (Exception e) {
     sig = "fallo";
     GELIConfigApp.logger.error(e.getMessage());
     ActionErrors errors = new ActionErrors();
     // esta etiqueta error.consulta.libro se debe añadir al fichero
     // src/es.jcyl.uic.geli/ApplicatonResources.properties
     // error.consulta.libro=Se ha producido un error
     // en la consulta de libros
     errors.add(ActionErrors.GLOBAL_ERROR,
              new ActionError("error.consulta.libro"));
     saveErrors(request, errors);

}
request.setAttribute("paginaLibros", paginaDetalle);
```

Como puede verse en este código se invoca a dos métodos del objeto de lógica de negocio de libros que previamente hemos añadido,

```
/**
     * Método para buscar todos los libros escritos por un autor
     *
     * @param inicio  Registro a partir del cual devolver. <I>1</I> para primero.
     * @param cuantos Numero de registros a devolver. <I>-1</I> para todos.
     * @param cAutorId código del autor para el que vamos a buscar sus libros.
     * @return ArrayList, lista de OT libro, con los escritos por un usuario
     * @throws Exception si se produce algún error al realizar la búsqueda.
     */
    public ArrayList buscarLibrosAutor(int inicio, int cuantos,
                                       long cAutorId) throws Exception {
        ArrayList listado = null;
        try {
            LibrosOAD librosOAD = LibrosOAD.getInstance();
            listado = librosOAD.buscarLibrosAutor(inicio, cuantos, cAutorId);
        } catch (Exception e) {
            logger.error(e);
            throw e;
        }
        return listado;
    }

    /**
     * Método para contar todos los libros escritos por un autor
     *
     * @param cAutorId código del autor para el que vamos a contar sus libros.
     * @return  int, con el número de registros encontrados
     * @throws Exception si se produce algún error al realizar la cuenta.
     */
    public int buscarLibrosAutorCuenta(long cAutorId) throws Exception {
        int total = -1;
        try {
            LibrosOAD librosOAD = LibrosOAD.getInstance();
            total = librosOAD.buscarLibrosAutorCuenta(cAutorId);
        } catch (Exception e) {
            logger.error(e);
            throw e;
        }
        return total;

    }
```

y estos invocan a dos nuevos del objeto de acceso a datos que también hemos añadido y cuyo código puede verse en los fuentes adjuntos a este documento.

Podemos modificar esta página para añadir un botón que permita volver a la página con el listado de todos los autores, o bien acceder a esta página a través de las opciones del menú.

7.4.6 Volcado de consulta de libros a fichero PDF o Excel

La funcionalidad que se ha ido desarrollando a lo largo de este documento muestra siempre los datos por pantalla, pero en alguna situación puede ser interesante disponer de esos datos en un formato listo para imprimir o para almacenarlo en disco. Para ello vamos a añadir dos nuevos botones en la pantalla de consulta de libros y en la pantalla de detalle de los libros de un determinado autor.

El primer botón permitirá obtener la información en formato PDF. Previamente habrá que desarrollar un informe utilizando Oracle Reports

El otro botón exportará la información en formato Excel, utilizando para ello un componente gratuito de desarrollo en java, Jakarta POI.

Vamos a comenzar por la pantalla de consulta de libros. El código necesario para añadir los botones se muestra a continuación.

```
<tr align="center" ><td>

<html:button styleClass="formBoton" onclick=”listadoPdf();”
   property="Descargar a PDF">Descargar a PDF
</html:button>
         
<html:button styleClass="formBoton" onclick=”listadoExcel();”
   property="Descargar a EXCEL">Descargar a EXCEL

</html:button>

</td></tr>
```

Vamos a comenzar por el listado en Excel.

Como puede verse en el código que genera el botón, se llama a una función javascript que es la encargada de enviar la petición al action. Para ello se ha generado el siguiente código

```
   function listadoExcel(){
       document.forms[0].action= '<html:rewrite page="/GELILibroCn.do" />'
       document.forms[0].AAccion.value='listadoExcel';
       document.forms[0].target='';
       document.forms[0].submit();
   }
```

Con la página preparada para enviar la nueva petición al servidor, debemos modificar el action encargado de generar el listado por pantalla, para que cuando reciba la esta petición genere el fichero Excel. A este action hay que añadir el código necesario para determinar que acción debe ejecutar (antes sólo respondía a peticiones de consulta pero ahora debe determinar si se trata de ese caso, de un listado en Excel o PDF). A continuación se puede ver

```
Collection listado = null;

GELILibroForm formulario = (GELILibroForm) form;
if (((GELILibroForm) form).getAAccion() != null &&
                ((GELILibroForm) form).getAAccion().equals("listadoExcel")) {
   // lo que se quiere es un listado en excel y tengo que obtener todos los registros
   // no solo los de una determinada página.
   // Para ello indico que registrosPorPagina = -1 y con eso aparecerán todos
   primerRegistro = 1;
   registrosPorPagina = -1;
}
try {
   GELILibrosLN librosLN = new GELILibrosLN();

   ..............................
```

A continuación se ejecutaría el mismo código que para el caso inicial puesto que se trata de obtener todos los libros. Finalmente, si se trata de uno de estos listados, la acción a ejecutar, y la página de retorno cambiará. En caso de tratarse de un listado en formato Excel, se debe generar y devolver ese fichero, sin redireccionar a ninguna página. Para construir este fichero Excel, se va a generar un nuevo método en el action, pero primeramente vamos a añadir unos imports a nuestra clase:

```
package es.jcyl.uic.geli.actions;

import es.jcyl.cf.utiles.Pagina;
import es.jcyl.uic.geli.actionforms.GELILibroForm;
import es.jcyl.uic.geli.config.GELIConfigApp;
import es.jcyl.uic.geli.ln.GELIAutoresLN;
import es.jcyl.uic.geli.ln.GELIGenerosLiterariosLN;
import es.jcyl.uic.geli.ln.GELILibrosLN;
import es.jcyl.uic.geli.ot.LibrosOT;
import org.apache.poi.hssf.usermodel.*;
import org.apache.poi.hssf.util.HSSFColor;
import org.apache.poi.hssf.util.Region;
import org.apache.struts.action.*;
```

```
    pagina.setRegistrosPorPagina(registrosPorPagina);

            // Una vez que tengo todos los registros, voy a ver si se trata de un
            // listado en excel. Si es asi construyo el fichero que voy a devolver

            if (((GELILibroForm) form).getAAccion() != null &&
                    ((GELILibroForm) form).getAAccion().equals("listadoExcel")) {
                // lo que se quiere es un listado en excell.
                // Ya tengo todos los resultados y tengo que redireccionar
                // a la nueva página que muestra el fichero excel.
                // En este caso necesito el outputStream y escribir sobre
                // él directamente el contenido generado.
                HSSFWorkbook contenido =
                        obtenerContenidoExcel(pagina.getElementos());

                OutputStream out = response.getOutputStream();

                response.setContentType("application/vnd.ms-excel");
                response.setHeader("Content-Disposition",
                        "attachment;filename="
                        + contenido.getSheetName(0) + ".xls");
                contenido.write(out);
                out.flush();
                out.close();
                // No redirecciono a ninguna página
                return mapping.findForward("ningun_lado");
            }
```

A continuación se muestra el código del nuevo método en el que se genera la estructura del fichero Excel que se va a devolver al usuario:

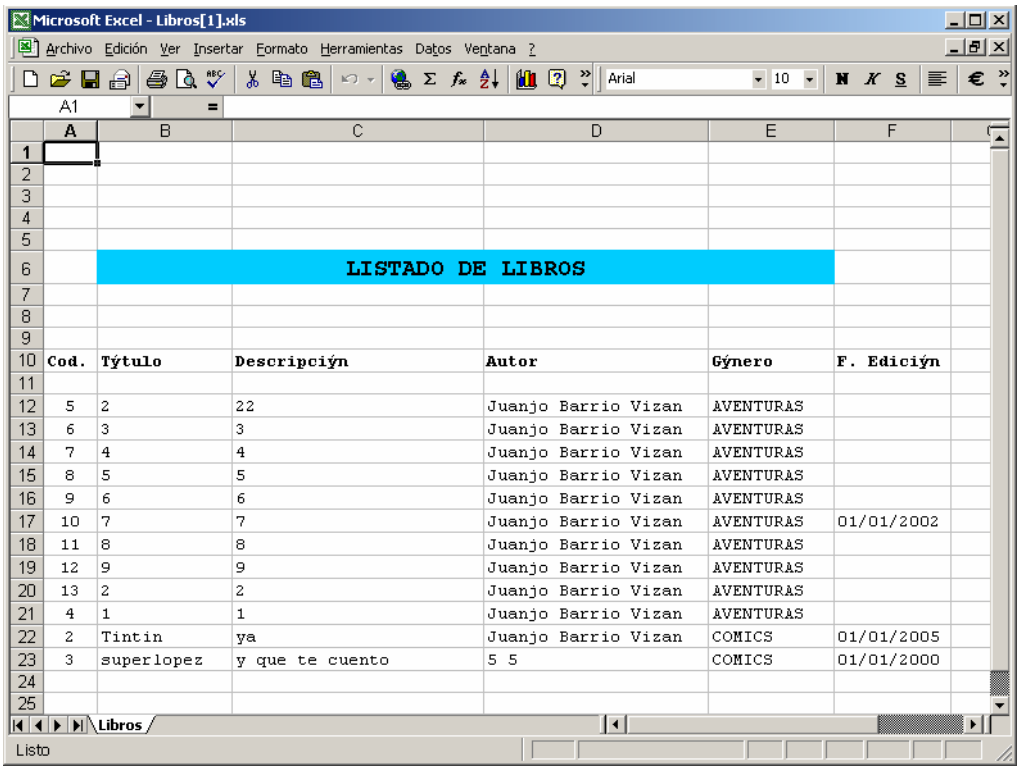
```
private HSSFWorkbook obtenerContenidoExcel(Collection elementos) {
        // creamos un nuevo documento
        HSSFWorkbook wb = new HSSFWorkbook();
        // creamos una nueva hoja
        HSSFSheet sheet = wb.createSheet("Libros");
        // declaramos un objeto fila
        HSSFRow row = null;
        // declaramos un objeto celda
        HSSFCell cell = null;
        // creamos un estilo para una celda (luego lo iremos modificando para
        // cada una de las celdas)
        HSSFCellStyle cs = wb.createCellStyle();

        // creamos un objeto fuente (que iremos modificando)
        HSSFFont font = wb.createFont();

        // Creamos el título
        row = sheet.createRow(5);

        cell = row.createCell((short) 1);

        cell.setCellValue("LISTADO DE LIBROS");
        font.setFontHeightInPoints((short) 14);
```

```
font.setFontName("Courier New");
font.setBoldweight(HSSFFont.BOLDWEIGHT_BOLD);
cs.setFont(font);
cs.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);
cs.setFillForegroundColor(HSSFColor.SKY_BLUE.index);
cs.setAlignment(HSSFCellStyle.ALIGN_CENTER);
cell.setCellStyle(cs);
sheet.addMergedRegion(new Region(5, (short) 1, 5, (short) 4));

// definimos los márgenes para que se imprima correctamente
sheet.setFitToPage(true);
sheet.setMargin(HSSFSheet.LeftMargin, 0.30);
sheet.setMargin(HSSFSheet.RightMargin, 0.30);

// creamos la cabecera de la tabla
row = sheet.createRow(9);

// volvemos a crear un nuevo estilo y fuente para la cabecera, distinto
// del utilizado para el título.
cs = wb.createCellStyle();
font = wb.createFont();

font.setFontHeightInPoints((short) 10);
font.setFontName("Courier New");
font.setBoldweight(HSSFFont.BOLDWEIGHT_BOLD);

cs.setFont(font);

cell = row.createCell((short) 0);
cell.setCellStyle(cs);
cell.setCellValue("Cod.");
sheet.setColumnWidth((short) 0, (short) (6 * 256));

cell = row.createCell((short) 1);
cell.setCellStyle(cs);
cell.setCellValue("Título");
sheet.setColumnWidth((short) 1, (short) (15 * 256));

cell = row.createCell((short) 2);
cell.setCellStyle(cs);
cell.setCellValue("Descripción");
sheet.setColumnWidth((short) 2, (short) (28 * 256));

cell = row.createCell((short) 3);
cell.setCellStyle(cs);
cell.setCellValue("Autor");
sheet.setColumnWidth((short) 3, (short) (25 * 256));

cell = row.createCell((short) 4);
cell.setCellStyle(cs);
cell.setCellValue("Género");
sheet.setColumnWidth((short) 4, (short) (14 * 256));

cell = row.createCell((short) 5);
cell.setCellStyle(cs);
cell.setCellValue("F. Edición");
sheet.setColumnWidth((short) 5, (short) (13 * 256));

// A continuación tengo que crear el contenido de la tabla
cs = wb.createCellStyle();
// nuevo estilo para centrar el contenido de la columna
// con el código de la incidencia.
HSSFCellStyle csCentrado = wb.createCellStyle();

font = wb.createFont();
```

```
        font.setFontHeightInPoints((short) 10);
        font.setFontName("Courier New");

        cs.setFont(font);
        cs.setWrapText(true);
        cs.setVerticalAlignment(HSSFCellStyle.VERTICAL_TOP);

        csCentrado.setFont(font);
        csCentrado.setWrapText(true);
        csCentrado.setVerticalAlignment(HSSFCellStyle.VERTICAL_TOP);
        csCentrado.setAlignment(HSSFCellStyle.ALIGN_CENTER);

        // recorremos la lista de elementos para ir rellenando
        // cada una de las celdas de la
        // tabla del contenido
        if (elementos.size() > 0) {
            Iterator iterator = elementos.iterator();
            int i = 0;
            while (iterator.hasNext()) {
                row = sheet.createRow(11 + i);
                LibrosOT libro = (LibrosOT) iterator.next();

                // Codigo
                cell = row.createCell((short) (0));
                cell.setCellStyle(csCentrado);
                cell.setCellValue(libro.getCLibroId());

                // Nombre
                cell = row.createCell((short) (1));
                cell.setCellStyle(cs);
                cell.setCellValue(libro.getDTitulo());

                // Descripcion
                cell = row.createCell((short) (2));
                cell.setCellStyle(cs);
                cell.setCellValue(libro.getADescripcion());

                // Autor
                cell = row.createCell((short) (3));
                cell.setCellStyle(cs);
                cell.setCellValue(libro.getDNombreAutor());

                // Género Literario
                cell = row.createCell((short) (4));
                cell.setCellStyle(cs);
                cell.setCellValue(libro.getDNombreGenero());

                // Fecha de edición
                cell = row.createCell((short) (5));
                cell.setCellStyle(cs);
                cell.setCellValue(libro.getFEdicionStr());

                i++;
            }
        }

        // devolvemos el documento que hemos creado
        return wb;
    }
```

El resultado que debe obtenerse es un mensaje de confirmación indicándonos si queremos abrir, guardar o cancelar, el fichero Excel que se ha generado. Podría ocurrir que este mensaje no

aparezca y nos muestre directamente en la pantalla este fichero. En cualquier caso, después de aceptar este fichero podremos ver un fichero similar a este



Cod.	Týtulo	Descripciýn	Autor	Gýnero	F. Ediciýn
5	2	22	Juanjo Barrio Vizan	AVENTURAS	
6	3	3	Juanjo Barrio Vizan	AVENTURAS	
7	4	4	Juanjo Barrio Vizan	AVENTURAS	
8	5	5	Juanjo Barrio Vizan	AVENTURAS	
9	6	6	Juanjo Barrio Vizan	AVENTURAS	
10	7	7	Juanjo Barrio Vizan	AVENTURAS	01/01/2002
11	8	8	Juanjo Barrio Vizan	AVENTURAS	
12	9	9	Juanjo Barrio Vizan	AVENTURAS	
13	2	2	Juanjo Barrio Vizan	AVENTURAS	
4	1	1	Juanjo Barrio Vizan	AVENTURAS	
2	Tintin	ya	Juanjo Barrio Vizan	COMICS	01/01/2005
3	superlopez	y que te cuento	5 5	COMICS	01/01/2000

Una vez finalizada la funcionalidad de listado en Excel, vamos a hacer algo parecido para conseguir la funcionalidad para exportar el listado a formato PDF. En este caso, como ya se ha comentado, es necesario crear los ficheros RDF con el formato y consultas necesarias para obtener los mismos datos que los mostrados por pantalla.

Una vez tenemos generado el fichero RDF con el diseño (en nuestro caso es un diseño sencillo), habrá que colocar ese fichero en una ruta accesible para el servidor de Reports, que será el encargado de servir las peticiones que podamos realizar a ese informe, y modificar la configuración necesaria del servidor para servir este informe. En nuestro caso, será necesario que el servidor tenga definida la cadena de conexión que va a utilizar nuestro informe, y que será referenciada como geli. En el fichero cgicmd.dat del servidor deberá aparecer una entrada del tipo

```
geli = geli/geli@dejcyl1 ...
```

Esto será tarea del administrador del servidor de reports, simplemente habrá que indicarle el usuario, password y SID de la base de datos a la que se quiere conectar.

Partiendo de la suposición que nuestro informe está correctamente en el servidor, vamos a realizar las modificaciones necesarias para hacer la llamada a este servidor. En primer lugar, igual que para el caso anterior, debemos añadir la función javascript que se ejecuta al pulsar sobre el botón y que abre una nueva ventana en la que se va a mostrar el contenido del informe, realizando la llamada al action encargado de mostrar este listado.

```
function listadoPdf(){
    window.open('','listadoPdf','width=800,height=600,
                              resizable=yes,scrollbars=yes,menubar=yes');
    document.forms[0].action=
                         '<html:rewrite page="/ GELILibroCn.do" />'
```

```
        document.forms[0].AAccion.value='listadoPdf';
        document.forms[0].target='listadoPdf';
        document.forms[0].submit();
    }
```

Cuando pulsemos sobre el botón, se llamará al action aquí indicado, y habrá que añadir el código necesario para gestionar este nuevo tipo de acción. Vamos a ver el código que se debe añadir.

```
      Collection listado = null;
       if (((GELILibroForm) form).getAAccion() != null &&
                   ((GELILibroForm) form).getAAccion().equals("listadoPdf")) {
               // lo que se quiere es un listado en pdf.
               // No necesito ejecutar nada y solo necesito enviar a la
               // página que genera el fichero pdf
               // Indico el nombre del report que quiero mostrar
               request.setAttribute("nombreReport", "GELILibroCn.rdf");
               // Para este informe no necesito ningun otro parametro que pasar
               // y no tengo que seguir ejecutando el resto del código de este
               // action
               return mapping.findForward("listadoPdf");
           }
```

Como se puede ver en este código, el resultado se envía a una pagina “listadoPdf”. En este caso no hemos generado la nueva página utilizando el asistente, y por tanto debemos modificar el fichero GELI-config.xml para añadir este nuevo mapeo dentro del action.

```
      <action path="/GELILibroCn"
           type="es.jcyl.uic.geli.actions.GELILibroCnAction"
           name="GELILibroForm"
           scope="request"
           validate="false">
           <forward name="exito" path="/libros/GELILibroCn.jsp"/>
           <forward name="listadoPdf" path="/libros/GELILibroRp.jsp"/>
       </action>
```

Además será necesario crear una nueva página jsp, `GELILibroRp.jsp` (como la hemos llamado dentro del fichero GELI-config.xml). Esta nueva jsp se encarga de recoger los atributos insertados desde el action y hacer la llamada al servidor de Reports. El contenido de esta nueva jsp es el siguiente

```
<%@ page import="es.jcyl.framework.JCYLConfiguracion"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>

<HTML>
<HEAD>
<TITLE>
LISTADO DE LIBROS
</TITLE>
<script language="JavaScript">
function enviar(){
      document.forms[0].submit();
}
</script>
</HEAD>
<body bgcolor="#FFFFFF" text="#000000" leftmargin="9" topmargin="10" marginwidth="9"
             marginheight="10" onLoad="javascript:enviar()">
    <form name="form1" method="post"
      action="<%=JCYLConfiguracion.getValor("PROTOCOLO_REPORT")%>://
              <%=JCYLConfiguracion.getValor("HOST_REPORT")%>:
              <%=JCYLConfiguracion.getValor("PORT_REPORT")%>
```

```
                <%=JCYLConfiguracion.getValor("SERVLET_REPORT")%>">
            <input type="hidden" name="cmdkey" value="geli">
            <input type="hidden" name="report" value="<bean:write name="nombreReport"/>">
    </form>
</BODY>
</HTML>
```

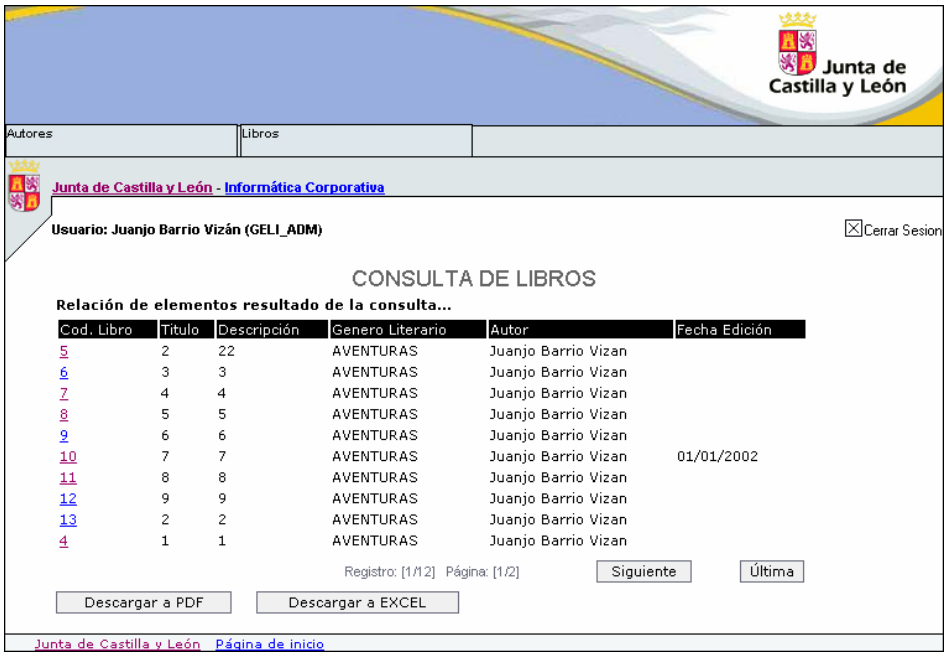
Con esta página, una vez recogidos los atributos puestos por el action, se llamará al servidor de reports. En este caso, el informe report no necesita ningún parámetro adicional, como podría ser el código de una incidencia. Si esto fuese necesario, habría que añadir al formulario el parámetro que el informe espera recibir y su valor correcto. Para hacer una llamada genérica, y fácilmente configurable, hemos obtenido los valores del servidor del fichero de configuración de la aplicación. Por tanto, será necesario añadir estos datos a este fichero. Si el servidor de reports cambia de nombre, sólo será necesario modificar este fichero y todos los informes aparecerán correctamente sin tener que modificar ninguna página jsp. El contenido del fichero de configuración que hemos añadido, se puede ver a continuación

```
################ Parámetros para la llamada al servidor de reports #############
 # Protocolo de llamada a Reports
 PROTOCOLO_REPORT=http
 # Host de Reports
 HOST_REPORT=aplides10g.cf.jcyl.es
 #PUERTO
 PORT_REPORT=80
 #Servlet
 SERVLET_REPORT=/reports/rwservlet
```

A continuación habría que hacer lo mismo con la pantalla de libros escritos por un autor, donde hemos dicho que se va a permitir obtener la información en Excel y PDF. La forma de trabajar sería la misma y no vamos a incluirla en este documento. El código desarrollado para esta opción se puede encontrar en la aplicación que se adjunta.

7.4.7 Búsqueda de libros

Finalmente, vamos a añadir una nueva funcionalidad que permita buscar cualquier libro del sistema por cualquiera de sus atributos. Para ello, partimos de la página de listado que hemos desarrollado (igual que se hizo para el caso de autores) utilizando los asistentes, y la modificaremos de forma que aparezca un formulario en la parte superior con todos los criterios por los que se puede buscar. El listado que se va a mostrar es el mismo pero limitado a los registros encontrados según los criterios que el usuario haya introducido. El aspecto inicial de la página es el siguiente



A continuación comenzamos con las modificaciones. Inicialmente vamos a añadir el formulario de inserción de datos

```
 <html:form action="/GELILibroCn.do" onsubmit="return buscarLibros();" >
<table width="80%" align="center">
    <tr>
        <td align="right" >T&iacute;tulo</td>
        <td width="20" > </td>
        <td><html:text property="DTitulo"/> </td>
    </tr>
    <tr>
        <td align="right" >Descripci&oacute;n</td>
        <td width="20" > </td>
        <td><html:text property="ADescripcion"/> </td>
    </tr>
    <tr>
        <td align="right" >Autor</td>
        <td width="20" > </td>
        <td>
          <html:select property="CAutorId">
              <html:option value="">Seleccionar</html:option>
              <html:options collection="autores" property="CAutorId"
                            labelProperty="ANombre"/>
```

```
            </html:select>
          </td>
      </tr>
      <tr>
          <td align="right" >G&eacute;nero Literario</td>
          <td width="20" > </td>
          <td>
            <html:select property="CGeneroId">
                <html:option value="">Seleccionar</html:option>
                 <html:options collection="generosLiterarios" property="CGeneroId"
                               labelProperty="DNombre"/>
            </html:select>
          </td>
      </tr>
      <tr>
          <td align="right" >Fecha de edici&oacute;n</td>
          <td width="20" > </td>
          <td><html:text property="FEdicionStr"/> (DD/MM/YYYY)</td>
      </tr>
      <tr><td colspan="3" > </td></tr>
      <tr>
         <td colspan="3" align="center"  >
            <html:submit styleClass="formBoton" value="Buscar" />
         </td>
      </tr>
      <tr><td colspan="3" > </td></tr>
  </table>
  </html:form>
```

En este formulario se tienen dos combos con los autores y géneros literarios, que previamente hemos tenido que cargar en el action al que se llama desde el menú para mostrar la página. El código que hemos añadido puede verse en el action `GELILibroCnAction`.

Una vez añadido este formulario, podemos probar la página, y veremos que sigue siendo el mismo que antes de añadirlo. Esto se debe a que los datos enviados desde el formulario, todavía no se recogen en el action y por tanto todavía no se filtran los registros. Por tanto, debemos modificar el action para tener en cuenta los valores del formulario.

El código modificado es el siguiente

```
        // modifico la llamada a listado libros para que reciba como entrada
        // un OT con los datos introducidos desde el formulario.
        // Lo mismo tengo que hacer para la cuenta
        GELILibroForm formulario = (GELILibroForm) form;
        LibrosOT libroOT = new LibrosOT();
        libroOT.setDTitulo(formulario.getDTitulo());
        libroOT.setADescripcion(formulario.getADescripcion());
        if (!formulario.getFEdicion().equals("")) {
            SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
            Date dato = formato.parse(formulario.getFEdicion());
            libroOT.setFEdicion(dato);
        }
        if( formulario.getCAutorId() != null &&
           !formulario.getCAutorId().equals("")){
            libroOT.setCAutorId(Integer.parseInt(formulario.getCAutorId()));
        }
        if( formulario.getCGeneroId() != null &&
           !formulario.getCGeneroId().equals("")){
            libroOT.setCGeneroId(Integer.parseInt(formulario.getCGeneroId()));
        }
        listado = librosLN.listadoLibros(primerRegistro,
```

```
                                        registrosPorPagina, libroOT);
        int nListado = 0;
        nListado = librosLN.listadoLibrosCuenta(libroOT);
```

Como puede verse, hemos modificado la llamada el método de lógica de negocio para pasarle el OT con los datos del formulario. Por tanto, hemos cambiado el método de lógica de negocio, y el método de acceso a datos. Este último es el que mas ha cambiado y vamos a mostrar el código necesario para tener en cuenta los datos introducidos. El código añadido al método `LibrosOAD.listadoLibros()` es el siguiente

```
"LI.C_GENERO_ID = GELI.C_GENERO_ID";
      // modificación para permitir filtrar la búsqueda por los
      // criterios introducidos en el formulario de la página de consulta.
      // Este código no se genera de forma automática. Voy a comprobar
      // que se recibe un OT distinto de null. Permito que sea null
      // para modificar lo menos posible aquellos métodos anteriores que no
      // pasaban este OT
      if (libroOT != null) {
          if (libroOT.getDTitulo() != null && !libroOT.getDTitulo().equals("")) {
              sql += " AND LI.D_TITULO LIKE '%" + libroOT.getDTitulo() + "%'";
          }
          if (  libroOT.getADescripcion() != null &&
               !libroOT.getADescripcion().equals("")) {
             sql+=" AND LI.A_DESCRIPCION LIKE '%"+libroOT.getADescripcion() + "%'";
          }
          if (libroOT.getCAutorId() > 0) {
              sql += " AND LI.C_AUTOR_ID =" + libroOT.getCAutorId();
          }
          if (libroOT.getCGeneroId() > 0) {
              sql += " AND LI.C_GENERO_ID =" + libroOT.getCGeneroId();
          }
          if (  libroOT.getFEdicionStr() != null &&
               !libroOT.getFEdicionStr().equals("")) {
              // Voy a poner que es mayor que la fecha que han dicho.
              // Si se hubiesen puesto dos campos fecha se podría
              // buscar entre una y otra.
              sql += " AND LI.F_EDICION >= " +
                     " TO_DATE('" + libroOT.getFEdicionStr() + "','dd/MM/yyyy')";
          }
      }

      st = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                               ResultSet.CONCUR_UPDATABLE);
```

Puesto que hemos añadido unos criterios de búsqueda, es necesario modificar los informes que se están generando. En el caso del informe en Excel, no es necesario puesto que se apoya en los datos que se obtienen en la consulta, y eso ya se ha modificado para mostrar los resultados en la página. Para el informe en PDF, sí es necesario hacer alguna modificación que a continuación vamos a detallar. Lo primero que habrá que hacer es modificar el fichero .rdf que hemos generado con el editor de reports, para que permita recibir como parámetros los criterios de búsqueda. Además modificaremos la query de ese informe que da lugar a los resultados para tener en cuenta los datos que reciba en los parámetros anteriores. A continuación debemos modificar el action para que al llamar al redireccionar al informe ponga en el request los valores introducidos en el formulario

```
if (((GELILibroForm) form).getAAccion() != null &&
         ((GELILibroForm) form).getAAccion().equals("listadoPdf")) {
     // lo que se quiere es un listado en pdf.
     // No necesito ejecutar nada y solo necesito enviar a la
     // página que genera el fichero pdf
     // Indico el nombre del report que quiero mostrar
     request.setAttribute("nombreReport", "GELILibrosCn.rdf");


     // Para este informe no necesito ningún otro parámetro que pasar
     // y no tengo que seguir ejecutando el resto del código de este
     // action Añado los parámetros que necesito
     // para poder llamar al servidor de reports.
     request.setAttribute("DTitulo", formulario.getDTitulo());
     request.setAttribute("ADescripcion", formulario.getADescripcion());
     request.setAttribute("CAutorId", formulario.getCAutorId());
     request.setAttribute("CGeneroId", formulario.getCGeneroId());
     request.setAttribute("FEdicion", formulario.getFEdicionStr());
     return mapping.findForward("listadoPdf");
}
```

La siguiente modificación que debemos realizar es en la página jsp que recibe estos parámetros y que se encarga de hacer la llamada correcta al servidor de reports pasándole los parámetros que éste espera.

Con esto se daría por concluido el desarrollo de la aplicación. Como ya se ha comentado, esta aplicación es muy sencilla y se podrían hacer aquellas mejoras que se estimen oportunas, para mejorar el aspecto, rendimiento, o añadir nuevas funcionalidades.

En toda la aplicación, hemos intentado reutilizar los actions para atender diferentes tipos de acciones que tenían relación, por ejemplo, obtener un listado por pantalla, en fichero Excel o PDF. En nuestro caso, hemos pasado un parámetro AAccion en el formulario, y comprobamos de que acción se trata. Esto se podría haber hecho de otras formas, pero que por ser este un documento sencillo no hemos querido incluir. **Véase DispatchAction para más información**. Como práctica podría modificarse la aplicación para adaptarla y utilizar esta forma de identificar la acción que se quiere ejecutar dentro de un action.

8 Resumen del Tutorial

Llegados a este punto hemos construido una aplicación Web sencilla cumpliendo con los estándares de desarrollo Web para la Junta de Castilla y Leon.

Esta aplicación cumple con las principales recomendaciones de buenas prácticas publicadas por las principales compañías líderes de desarrollo Web en J2EE.

Además hemos puesto en práctica las herramientas de ayuda al desarrollo como:

- Generador de Aplicaciones
- Generador de OAD's y OT's
- Sistema de Seguridad Corporativo
- Asistentes de Ayuda al Desarrollo
    - Generador de formulario base
    - Generador de listado base
    - Generador de una ficha de consulta
    - Generador de una ficha maestro - detalle
- Librerías recomendadas para la exportación a Ofimática (Jakarta POI)

8.1 Sugerencias en la organización del trabajo

La Arquitectura estándar de desarrollo para la Junta de Castilla y Leon facilita una organización del trabajo de codificación por parte de los equipos de desarrollo. Esta arquitectura divide la aplicación en varias capas que pueden ser desarrolladas por equipos de desarrolladores especializados en cada aspecto. Por ejemplo:

Los conocedores de la lógica de negocio de la aplicación pueden encargarse de:

- Clases de Lógica de negocio (Objetos LN)
- Acciones que responden a los eventos del usuario (Actions)

Los especialistas en Queries y estructura de los datos

- Clases OAD y Clases OT

Los especialistas de la interacción con el usuario y diseñadores de intertaz de pantalla:

- Capa de Presentación: JSP, Navegabilidad (struts-config)