# Presentación de la base de datos

Se trata de la pantalla inicial de la aplicación tras la identificación y la obtención de la conexión de la base de datos.

En esta pantalla, en la parte superior, presentamos información general de la conexión (driver, url y usuario conectado) junto con una opción para desconectarnos de la base de datos (lo que nos llevará a la pantalla de login.

Existirá un faldón lateral izquierdo donde se presentará información relativa a la conexión agrupada en opciones de menú que se agruparán en dos tipos:

* Información simple
* Información tabulada

En la parte central de la pantalla se presentan las propiedades informativas de la conexión con la base de datos. Se trata de información obtenida de la conexión basada en los métodos disponibles en un objeto connection. Es, por lo tanto, información general correspondiente a cualquier tipo de conexión JDBC realizada con el programa.

## DatabaseMetaDataAction.java

Paquete es.ubu.alu.mydatabasejc.actions

Es la clase que proporciona la información necesaria para la presentación de la pantalla *inicial.jsp*

Puesto que esta clase va a hacer un uso intensivo de la sesión del usuario, implementamos los métodos del interface *SessionAware* de struts2 que mapea en una propiedad de tipo *Map<String, Object>* la sesión web del usuario:

public class ConsultaAction extends LoginAction implements Preparable, SessionAware

...

private Map<String, Object> sesion;

…

@Override

public void setSession(Map<String, Object> map) {

this.sesion = map;

}

De esta manera disponemos de todos los datos de la sesión del usuario bajo la propiedad *sesion*.

Dispone de los siguientes métodos:

* **prepare()**: En un primer lugar, se cargará de la sesión del usuario el objeto con la conexión a la base de datos para lo que sobreescribe el método *prepare()* que será ejecutado de forma previa a cualquier otro método:

connectionImpl = (ConnectionImpl) sesion.get("conexion");

* **validate()**: Recoge las validaciones que se deben realizar. Para ello se utiliza el método *LoginAction.validarLogin(ConnectionImpl)* para lo que es necesario heredar esta clase de la clase *LoginAction*:

super.validarLogin(connectionImpl);

Un error en este método volverá a la pantalla *login.jsp (struts.xml)*

* **inicial()**: Partiendo ya de una conexión válida, se obtienen los datos generales de la conexión. Para ello, se instancia un objeto de la clase ConnectionInfo

ConnectionInfo connectionInfo = new ConnectionInfo(connectionImpl.getConnection());

y, mediante **Reflection** obtenemos todos los métodos de esta clase y los ejecutamos guardando tanto su nombre como su resultado en una lista que se visualizará en *inicial.jsp*

Method[] methods = ConnectionInfo.class.getMethods();

for (Method method : methods) {

Object o = method.invoke(connectionInfo, new Object[]{});

PropiedadValor p = new PropiedadValor(method.getName(), o);

listInfo.add(p);

}

* **getMenus()**: Carga la lista de opciones de menú. Esta lista consiste en la relación de métodos del objeto *DatabaseMetaDataImpl* que devuelven un objeto del tipo *ResultSet* o un objeto de tipo *List<Object>*. El código que carga la lista de opciones de menú es similar al anterior cambiando la condición que debe cumplir el método por la siguiente:

if (method.getReturnType()==ResultSet.class || method.getReturnType()==List.class)

Para cada método debemos obtener la lista de tipos de parámetros que posee puesto que la necesitamos posteriormente para identificar de forma única al método a invocar. En función del tipo de objeto devuelto por la función, la acción será “resultset” o “info”

y se define el objeto *Menu* que se añadirá a la lista de menú:

Menu menu = new Menu(

method.getReturnType()==ResultSet.class ? "resultset" : "info",

method.getName(),

method.getParameterTypes);

y con el objeto de no repetir cuando existe métodos con sobrecarga de parámetros:

if ((index = listMenu.indexOf(menu))==-1) {

// y lo añade el nombre del método a la lista si no existe aún

listMenu.add(menu);

}

## ConnectionInfo.java

Paquete es.ubu.alu.mydatabasejc.jdbc

Contiene una propiedad de tipo *java.sql.Connection* y un conjunto de métodos que proporcionan información relativa a dicha conexión. El constructor recibe una conexión jdbc, se la asigna a la propiedad *connection* y, por cada método, realiza la correspondiente llamada al método equivalente del objeto *Connection*

## PropiedadValor.java

Paquete es.ubu.alu.mydatabasejc

Es una clase que contiene los atributos propiedad y valor de tipo String y Object respectivamente que permitirá almacenar listas de este tipo para su presentación en pantalla.

Con el objeto de poder comparar dos objetos de esta clase, se sobreescribe el método equals() como sigue:

@Override

public boolean equals(Object obj) {

if (!(obj instanceof PropiedadValor)) return false;

if (obj==null) return false;

if (this.propiedad==null && ((PropiedadValor)obj).propiedad==null) return true;

return this.propiedad.equalsIgnoreCase(((PropiedadValor)obj).propiedad);

}

## Menu.java

Paquete es.ubu.alu.mydatabasejc

Define la estructura de cada opción de menú de la aplicación. Los atributos son:

* **action**: Es la acción que struts2 ejecutará cuando el usuario elija esta opción del menú
* **metodo**: Junto con la acción, se especifica el método que se debe ejecutar mediante *Reflection*
* **parametros**: Contiene en codificación *Base64* el array de clases que conforman la llamada al método a invocar.

A destacar el siguiente constructor:

public Menu(String action, String metodo, Class[] parametros)

Convierte el array de clases parametros en una cadena codificada en base64 mediante las siguientes líneas:

ByteArrayOutputStream bs = new ByteArrayOutputStream();

ObjectOutputStream os = new ObjectOutputStream(bs);

os.writeObject(parametros);

String params = Base64.encodeBase64URLSafeString(bs.toByteArray());

y finalmente lo asigna al atributo *parametros*

this.parametros = params;

## DatabaseMetaDataImpl.java

Paquete es.ubu.alu.mydatabasejc.jdbc

Esta clase implementa el interface *java.sql.DatabaseMetaData* por lo que sobreescribe todos los métodos de dicho interface.

Solo dispone de una propiedad de tipo *DatabaseMetaData* que es inicializada en el constructor a través de un objeto de este tipo que se le ha de pasar por parámetro y que se obtendrá de la conexión a la base de datos mediante el método *getMetaData()*.

protected DatabaseMetaData metadata;

public DatabaseMetaDataImpl(DatabaseMetaData metadata){

this.metadata = metadata;

}

Todos los métodos que sobreescriben los del interfaz implementado simplemente ejecutan el método correspondiente de la propiedad *metadata*. A modo de ejemplo vemos el siguiente:

@Override

public String getURL() throws SQLException {

return metadata.getURL();

}

Además de los métodos del interfaz implementado, se proporcionan una serie de métodos que devuelven un objeto de tipo *List<List>* y que forman una agrupación de un conjunto de métodos simples que proporcionan información sobre usuarios, conexión, datos, sql, etc. Estos métodos devuelven todos ellos información en formato *List<List>* y son:

* **getBasicInfo()**: obtiene información básica.
* **getConnInfo()**: Información de la conexión
* **getDataBaseInfo()**: Información de la base de datos
* **getDataInfo()**: Información de los datos
* **getUserInfo()**: Información del usuario de base de datos

Si quisiéramos, podríamos añadir tantas funciones adicionales para agrupar los métodos simples de *DatabaseMetaDataImpl* en función del criterio que quisiéramos. La única restricción es que el objeto devuelto debe ser del tipo *List<List>*

## inicial.jsp

En la cabecera, se presenta la información utilizada para establecer la conexión, obtenida del objeto connectionImpl:

<table>

<tr>

<td><s:property value="connectionImpl.url"/></td>

<td><s:property value="connectionImpl.driver"/></td>

</tr>

<tr>

<td><s:property value="connectionImpl.usuario"/></td>

<td>&nbsp;</td>

</tr>

</table>

En la parte central presenta la información de la conexión establecida. En la parte principal de la pantalla utiliza un iterador de struts para recorrer una estructura *List<PropiedadesValor>* y presentar tanto el nombre de la propiedad (el nombre del método pasado por la función *getText* que busca la traducción en *package.properties*) como el valor para cada propiedad

<table class="tabla">

<tr>

<th>Propiedad</th>

<th>Valor</th>

</tr>

<s:iterator value="listInfo">

<tr>

<td>

<s:property value="%{getText(propiedad)}"/>

</td>

<td><s:property value="valor"/></td>

</tr>

</s:iterator>

</table>

En el lateral izquierdo se despliega el menú de opciones para presentar distinto tipo de información. Para determinar el menu se utiliza una jsp que puede ser llamada desde cualquier jsp mediante el tag include:

<s:include value="/WEB-INF/jspf/menu.jsp"/>

## Menu.jsp

Carpeta /WEB-INF/jspf

Inicialmente se muestra el nombre de la conexión:

<s:property value="connectionImpl.connection"/>

y a continuación se muestran los métodos disponibles del objeto *DatabaseMetaDataImpl* obtenido mediante el método *getMetaData()* del objeto *connection* que devuelvan objetos de tipo *ResultSet*. La información se visualiza en forma de lista html. Cada elemento de la lista es un link que llevará a visualizar la información del *ResultSet* correspondiente en la parte central de la pantalla:

<ol>

<s:iterator value="menus">

<li>

<s:url action="%{action}" var="urlTag">

<s:param name="metodo">

<s:property value="metodo"/>

</s:param>

<s:param name="parametros">

<s:property value="parametros"/>

</s:param>

</s:url>

<s:a href="%{urlTag}">

<s:property value="%{getText(metodo)}"/>

</s:a>

</li>

</s:iterator>

</ol>

El iterador recorre *menus* (lista de objetos *Menu*) y para cada elemento compone la url destino del link acompañado como parámetros el contenido de la *propiedad* que es realmente el nombre del método de *DatabaseMetaDataImpl* que se deberá ejecutar y los tipos de parámetros que el método requiere codificados en Base64 y bajo el nombre *parametros*.

El texto visible del link es el nombre del método pasado a través de la función *getText* que, como ya vimos, obtiene una pequeña descripción del grupo de ficheros *package.properties*

## package.properties

Paquete es.ubu.alu.mydatabasejc

Como ya se comentó, en este fichero se encuentran las traducciones de mensajes y etiquetas de pantalla que se presentan por el navegador al usuario. Al pasar el mensaje o texto por la función *getText()* obligamos a struts a localizar el mensaje o texto en este fichero y presentar en su lugar el valor asignado en él. En lugar de presentar el nombre del método en pantalla, se le asigna una traducción sencilla que es la que utilizará en pantalla:

isReadOnly = Solo lectura

getSchema = Esquema

getAutoCommit = Auto commit

getCatalog = Catálogo

getClientInfo = Info cliente

getHoldability = Capacidad de retención

getNetworkTimeout = Timeout de red

getTransactionIsolation = Aislamiento de transacción

# Métodos de DatabaseMetaDataImpl

getBasicInfo = General

getConnInfo = Conexión

getDataBaseInfo = Base de datos

getDataInfo = Datos

getSQLInfo = SQL

getUserInfo = Usuario

getAttributes = Atributos

...

## help.properties

Paquete es.ubu.alu.mydatabasejc

Para dotar a la aplicación de un sistema de ayuda interactivo, es necesario asociar mensajes o etiquetas con descripciones más amplias, que expliquen con más detalle los conceptos que se manejan. En relación con la información relativa a la conexión, esta información se ubica en este fichero, siguiendo el mismo planteamiento que tiene el fichero *package.properties*. Las propiedades son los nombres de los métodos y los valores son los textos de ayuda:

isReadOnly = esta base de datos está en modo de solo lectura

getSchema = Recupera el nombre de esquema actual de esta conexión

getAutoCommit = Recupera el modo de confirmación automática actual para esta conexión

getCatalog = Recupera el nombre de catálogo actual de esta conexión

getClientInfo = Devuelve una lista que contiene el nombre y el valor actual de cada propiedad de información del cliente admitida por el controlador

getHoldability = Recupera la capacidad de retención actual de los objetos ResultSet creados con esta conexión

getNetworkTimeout = Recupera la cantidad de milisegundos que el controlador esperará a que se complete una solicitud de base de datos

getTransactionIsolation = Recupera el nivel de aislamiento de transacción actual de esta conexión

# descripción de los métodos de DatabaseMetaDataImpl

allProceduresAreCallable = el usuario actual puede llamar a todos los procedimientos devueltos por el método getProcedures

allTablesAreSelectable = el usuario actual puede usar todas las tablas devueltas por el método getTables en una declaración SELECT

autoCommitFailureClosesAllResultSets = si un momento SQLException autoCommit es true indica que todos los ResultSets abiertos están cerrados, incluso los que se pueden mantener

...

## struts.xml

Se define el package *DatabaseMetaData* para dar respuesta al namespace */DatabaseMetaData*.

<package name="DatabaseMetaData" extends="struts-default" namespace="/DatabaseMetaData">

Como norma general, cualquier error de entrada o validación en la clase action correspondiente retornará el control a *login.jsp*

<global-results>

<result name="input">/login.jsp</result>

</global-results>

La acción *inicial* se asocia a *DatabaseMetaDataAction.inicial()* y finalmente presenta la página */inicial.jsp*

<action name="inicial" class="es.ubu.alu.mydatabasejc.actions.DatabaseMetaDataAction" method="inicial">

<result>/inicial.jsp</result>

</action>