

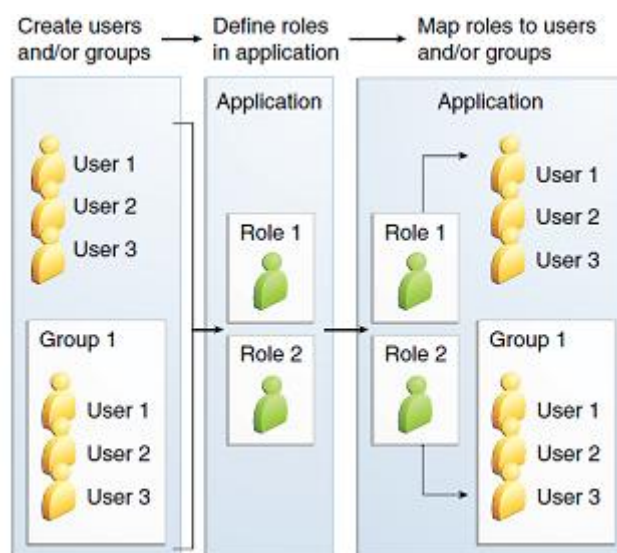
JAAS: Servicio de autenticación y autorización de Java.

Se usa con dos propósitos:

- ✓ Determinar de forma segura quien está usando el código Java de nuestra aplicación.
- ✓ Asegurarse de que los usuarios pueden llevar a cabo las acciones para las que están autorizados y no otras.

Para llevar a cabo esta tarea tenemos que implantar en nuestro servidor web un dominio de seguridad para nuestra aplicación.

Un dominio de seguridad lo forman un conjunto de usuarios y grupos identificados como válidos para una o más aplicaciones y controlados por la misma política de autenticación.



Como muestra la figura, en el proceso de creación del dominio de seguridad tenemos que informar a nuestro servidor de aplicaciones de los usuarios de nuestra aplicación y a qué grupo pertenecen. Podemos definir un grupo como un conjunto de usuarios autenticados que tienen rasgos comunes. Clasificar a los usuarios en grupos, hace que sea más fácil el control de acceso cuando el número de usuarios es grande.

En la aplicación definiremos zonas de acceso protegido y especificaremos qué roles tienen acceso a ese recurso protegido. Podemos ver un rol, por tanto, como un permiso que deben tener los usuarios para acceder a un determinado recurso.

El grupo en el servidor Glassfish, tiene un ámbito distinto a un rol. Un grupo tiene validez en todo el servidor Glassfish, mientras que un rol está asociado únicamente con una aplicación específica. De hecho estos roles se definen en el archivo `web.xml` de cada aplicación a asegurar.

Para facilitar la asignación de los usuarios que tienen acceso a un determinado recurso e una aplicación (pertenecen a un determinado rol) estableceremos una correspondencia

entre los roles y los grupos que pertenecen a dichos roles. Podemos asociar varios grupos a un determinado role en una aplicación. Esta correspondencia la definiremos en el archivo glassfish-web.xml

Creación de un dominio de seguridad en el servidor Glassfish del tipo jdbc

En el servidor Glassfish disponemos de varias opciones a la hora de implantar un dominio de seguridad, o lo que es lo mismo, hay varios tipos de dominios de seguridad. Los distintos tipos tienen distintas formas de almacenar los usuarios y grupos autenticados. Existen los siguientes tipos en Glassfish:

- ✓ **File Realm:** Glassfish almacena las credenciales de los usuarios en un fichero de claves. Es el dominio por defecto para este servidor de aplicaciones.
- ✓ **Certificate realm:** El servidor Glassfish almacena las credenciales de usuario en una base de datos certificada. En este caso el servidor usa certificados con el protocolo HTTPS para autenticar los clientes web.
- ✓ **LDAP realm:** El servidor obtiene las credenciales de usuario desde un servidor LDAP.
- ✓ **JDBC realm:** El servidor obtiene las credenciales del usuario de una base de datos.

En nuestra aplicación crearemos un dominio de seguridad de tipo JDBC. Esto quiere decir que los usuarios y grupos autenticados de nuestro servidor Glassfish estarán almacenados en una base de datos a la que el servidor debe tener acceso.

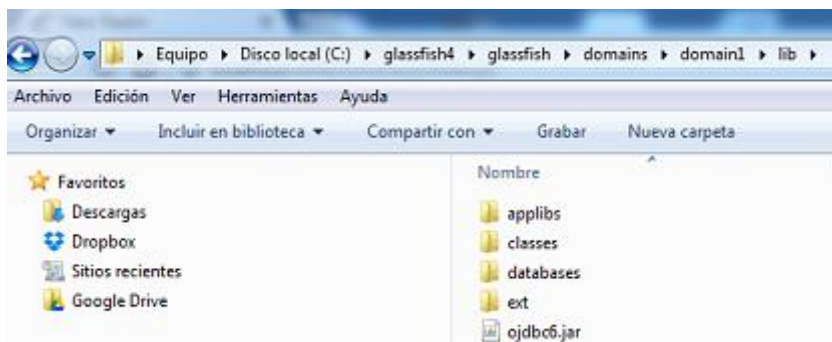
Concretamente debemos tener una tabla que almacene los usuarios y otra en la que almacenamos los grupos a los que pertenecen los usuarios, en algún esquema de nuestra base de datos

En nuestro caso la base de datos a utilizar es Oracle y la tabla usuarios y grupos del esquema biblioteca permitirán almacenar los usuarios autenticados de nuestra aplicación.

Pasos a realizar para definir nuestro dominio de seguridad JDBC.

1. Situar el driver en la librería de nuestro dominio de GlassFish.

Dado que el servidor debe tener acceso al esquema de base de datos que contiene las tablas que permiten almacenar los usuarios y grupos, nuestro servidor debe disponer de la implementación del driver JDBC para conectar a una base de datos Oracle. El driver se debe situar en el directorio que muestra la figura:



2. Definir las tablas que alojarán los usuarios y grupos.

Para nuestra aplicación Biblioteca, hemos situado estas dos tablas en el esquema Biblioteca. (Archivo: seguridadadjaas.sql enviado junto a la práctica).

```
CREATE TABLE USUARIOS(
  USUARIO VARCHAR2(20),
  CLAVE VARCHAR2(32),
  CONSTRAINT PK_USUARIOS
  PRIMARY KEY(USUARIO)
);
CREATE TABLE GRUPOS(
  IDGRUPO VARCHAR2(30),
  IDUSUARIO VARCHAR2(20),
  CONSTRAINT PK_GRUPOS
  PRIMARY KEY(IDGRUPO,IDUSUARIO),
  CONSTRAINT FK_GRUPOS_USUARIOS
  FOREIGN KEY(IDUSUARIO)REFERENCES USUARIOS(USUARIO)
)
;
```

La clave en la tabla usuarios se almacenará encriptada con la función MD5. Disponemos de una función almacenada en nuestro esquema Biblioteca implementada en PLSQL para encriptar una cadena con el algoritmo MD5.

El código es el siguiente y lo tenéis en el script que se mandó junto con el enunciado de la práctica (Archivo: hashmd5.sql)

```
CREATE OR REPLACE FUNCTION md5hash (str IN VARCHAR2)

  RETURN VARCHAR2
  IS v_checksum VARCHAR2(32);

  BEGIN
    v_checksum := LOWER(RAWTOHEX( UTL_RAW.CAST_TO_RAW(
sys.dbms_obfuscation_toolkit.md5(input_string => str) )) );
    RETURN v_checksum;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      NULL;
    WHEN OTHERS THEN
      -- Consider logging the error and then re-raise
      RAISE;
  END md5hash;
/
```

La implementación en Java la podemos ver en el código de la clase Hash package util;

```
public class Hash {
    /**
     *
     * @param txt
     *      , text in plain format
     * @param hashType
     *      MD5 OR SHA1
     * @return hash in hashType
     */
    public static String getHash(String txt, String hashType) {
        try {
            java.security.MessageDigest md = java.security.MessageDigest
                .getInstance(hashType);
            byte[] array = md.digest(txt.getBytes());
            StringBuffer sb = new StringBuffer();
            for (int i = 0; i < array.length; ++i) {
                sb.append(Integer.toHexString((array[i] & 0xFF) | 0x100)
                    .substring(1, 3));
            }
            return sb.toString();
        } catch (java.security.NoSuchAlgorithmException e) {
            // error action
        }
        return null;
    }

    public static String md5(String txt) {
        return Hash.getHash(txt, "MD5");
    }

    public static String sha1(String txt) {
        return Hash.getHash(txt, "SHA1");
    }
}
```

Ejemplo de uso:

```
package util;

public class TestHash {

    public TestHash() {
        // TODO Auto-generated constructor stub
    }

    public static void main(String[] args) {
        System.out.println(Hash.md5("ADMIN"));
    }

}
```

3. Creamos los usuarios y grupos.

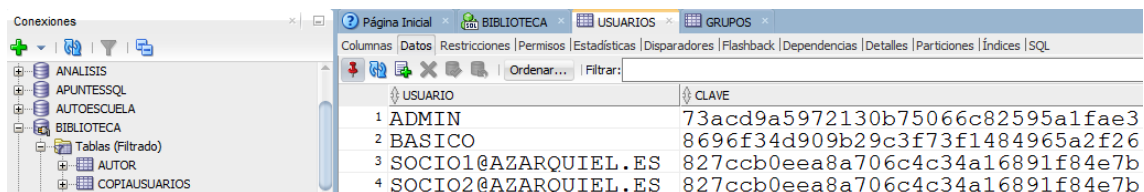
Con cualquier cliente para acceder a la base de datos, crearíamos los usuarios y los grupos. Se muestra un extracto de las tablas usuarios y grupos.

Las siguientes órdenes darían de alta al usuario ADMIN con clave encriptada 'ADMIN' y perteneciente al grupo 'adminbiblioteca'.

Insert into usuarios values('ADMIN',md5hash('ADMIN'));

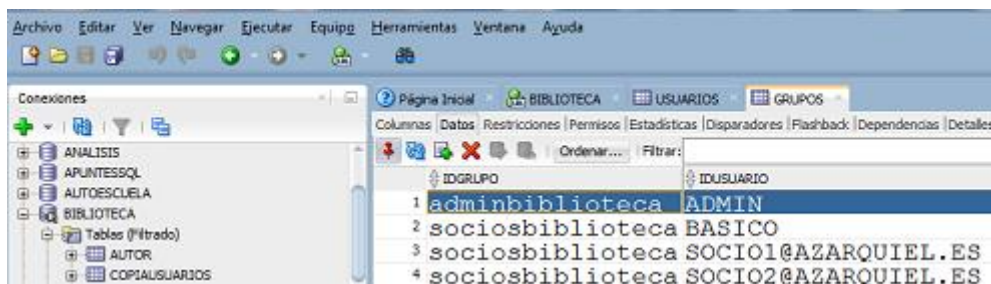
Insert into grupos values('adminbiblioteca','ADMIN');

Extracto de la tabla usuarios



	USUARIO	CLAVE
1	ADMIN	73acd9a5972130b75066c82595a1fae3
2	BASICO	8696f34d909b29c3f73f1484965a2f26
3	SOCIO1@AZARQUIEL.ES	827ccb0eea8a706c4c34a16891f84e7b
4	SOCIO2@AZARQUIEL.ES	827ccb0eea8a706c4c34a16891f84e7b

Extracto de la tabla grupos



	IDGRUPO	IDUSUARIO
1	adminbiblioteca	ADMIN
2	sociosbiblioteca	BASICO
3	sociosbiblioteca	SOCIO1@AZARQUIEL.ES
4	sociosbiblioteca	SOCIO2@AZARQUIEL.ES

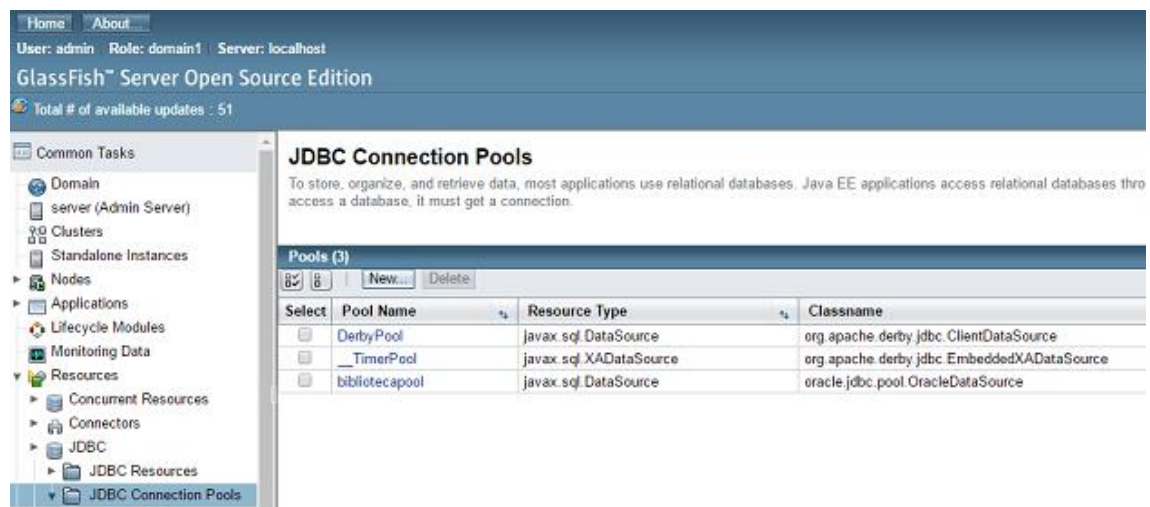
4. Creación del dominio de seguridad en Glassfish.

4.1. Crearemos un pool de conexiones que permita acceder al servidor Glassfish al esquema de base de datos que aloja las tablas que guardan los usuarios y grupos.

Para ello sigue los siguientes pasos:

4.1.1. Creación del pool.

Conectado al panel de administración de GlassFish, sitúate en JDBC Connection Pools y pulsa new.



4.1.2. Define el nombre, tipo de recurso e implementación del driver JDBC a utilizar

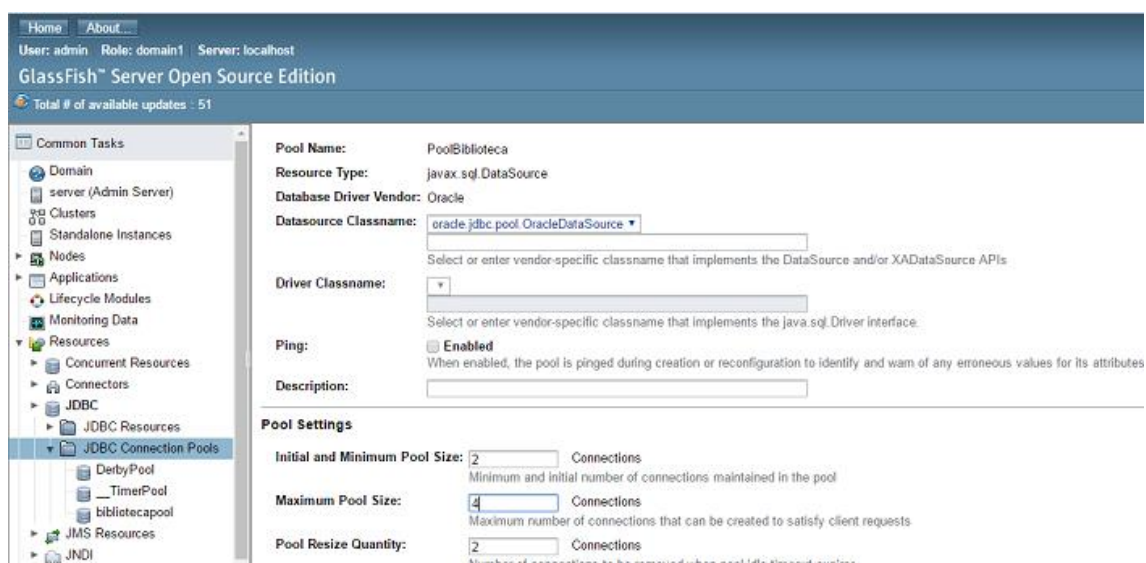
Muy importante:

*Hay que tener una copia de la implementación del driver en la carpeta:
C:\glassfish4\glassfish\domains\domain1\lib
En nuestro caso copiamos el archivo ojdbc6.jar en esta ubicación.
Si el servidor de aplicaciones estaba arrancado, habrá que reiniciarlo.*

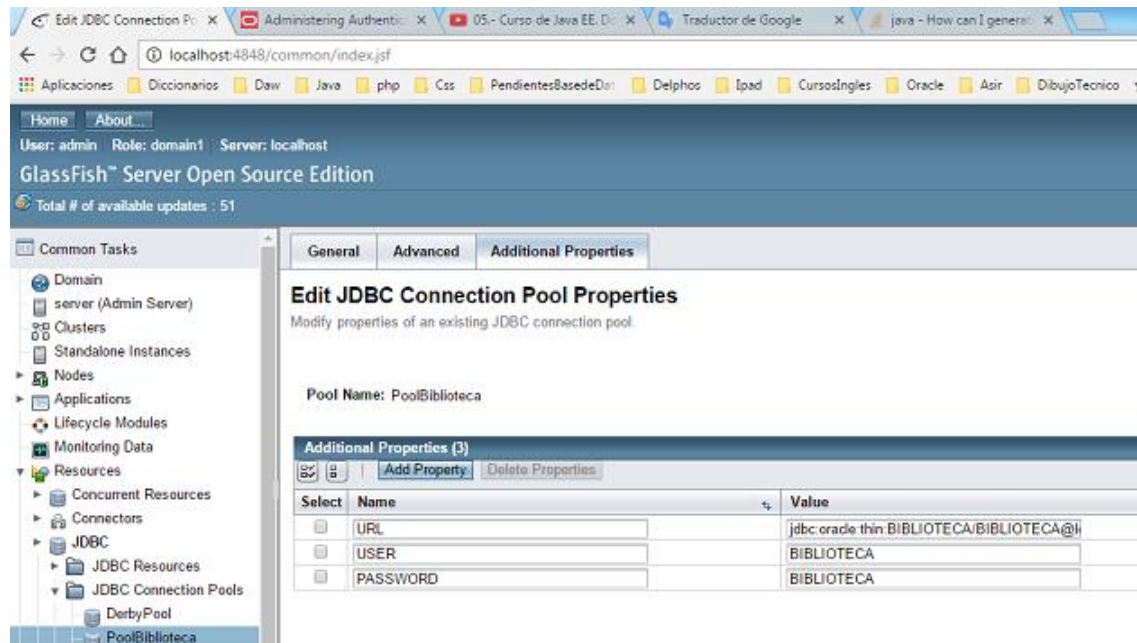


4.1.3 Ajustes del pool de conexiones. Tamaño inicial de conexiones, etc.

En la figura se muestra un pool de conexiones formado inicialmente por dos conexiones a nuestra base de datos, un máximo de 4 conexiones y un incremento en 2 conexiones del pool si las dos iniciales estuviesen ocupadas en un momento dado.

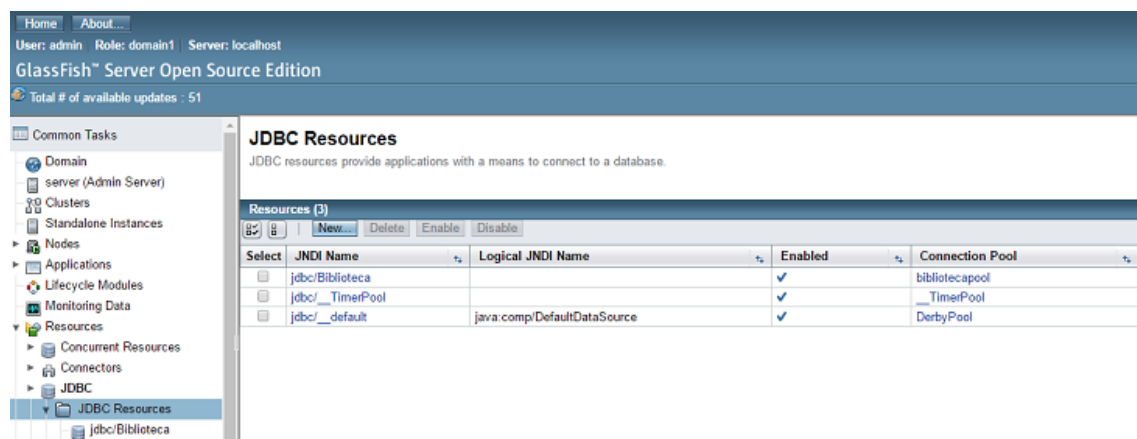


4.1.4 Definir las propiedades de conexión. Basta con las que muestra la figura. El resto las podemos borrar.

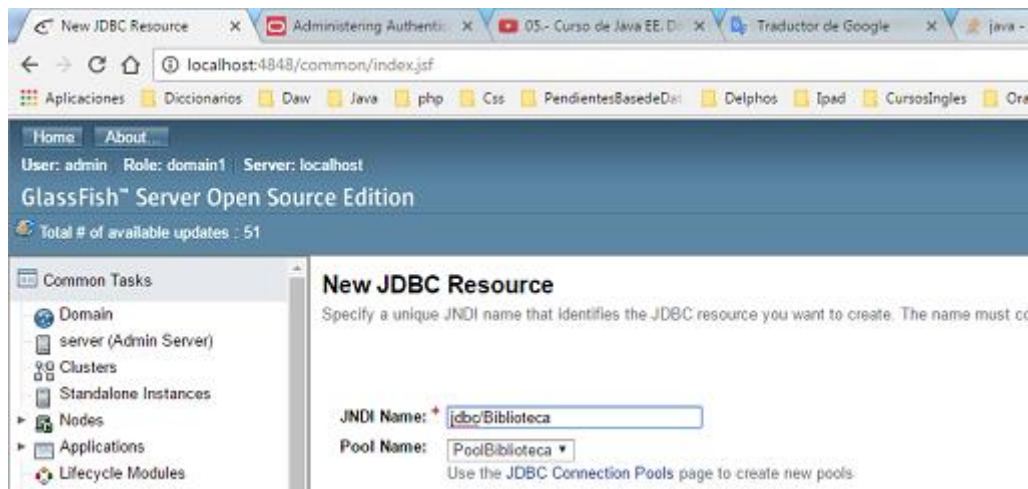


4.2. Crear un recurso JDBC asociado a este pool de conexiones.

4.2.1 Desde JDBC>>JDBC Resources pulsamos new



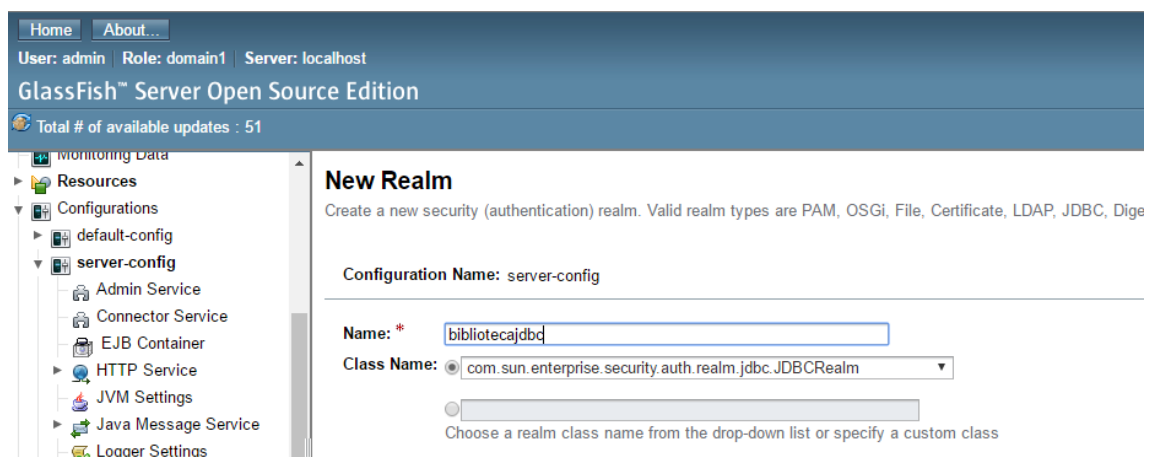
4.2.2. Configuramos el nuevo recurso JDBC. En el nombre usamos el formato estándar: jdbc/Nombre. Le asociamos el pool definido en el apartado anterior.



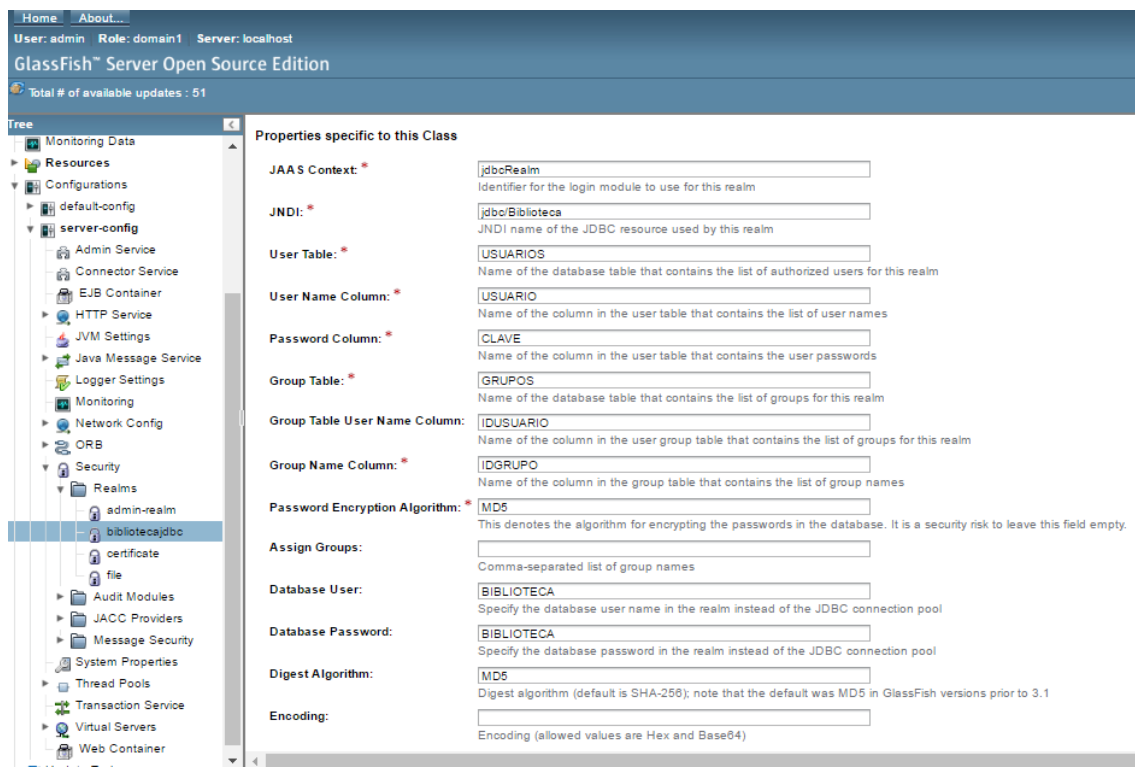
4.3. Crear el dominio de seguridad.

Con lo realizado en los apartados anteriores, ya estamos en disposición de crear nuestro dominio de seguridad de tipo jdbcRealm.

4.3.1. Desde server-config>>Security>>Realms pulsamos new para crear un nuevo dominio de seguridad y configuramos el nombre y el tipo:



4.3.2. Configurar las propiedades del dominio de seguridad.



- ✓ **JaasContext:** jdbcRealm (nuestro tipo de dominio de seguridad).
- ✓ **JNDI:** Recurso JDBC asociado al pool de conexiones que me permite acceder a las tablas que contienen los usuarios y grupos del dominio de seguridad.
- ✓ **User Table:** Tabla que contiene los usuarios autenticados.
- ✓ **User Name Column:** Columna que tiene el nombre del usuario.
- ✓ **Password Column:** Columna que aloja la clave del usuario.
- ✓ **Group Table:** Tabla que contiene los grupos.
- ✓ **Group Table User Name Column:** Columna de la tabla grupos que aloja el nombre del usuario.
- ✓ **Column Name Group:** Columna que aloja el nombre del grupo.
- ✓ **Algoritmo de encriptación de la clave:** MD5 en nuestro caso. Es el que hemos usado para encriptar la clave del usuario.
- ✓ **Database User:** Nombre del esquema de base de datos que alojan las tablas USUARIOS Y GRUPOS. En nuestro caso: BIBLIOTECA . Es opcional ya que está definido en el pool asociado al recurso JDBC.
- ✓ **Database Password:** Clave del esquema de base de datos que alojan las tablas USUARIOS Y GRUPOS. En nuestro caso: BIBLIOTECA . Es opcional ya que está definido en el pool asociado al recurso JDBC.
- ✓ **Digest Algorithm:** Volvemos a escribir el que hemos utilizado: MD5

Hasta aquí todo lo que necesitamos configurar en el servidor de aplicaciones. En el siguiente apartado veremos como configurar la seguridad de la aplicación.

5. Configurar la seguridad de nuestra aplicación Web.

Este proceso implica llevar a cabo los siguientes pasos:

5.1.: Selección del método de autenticación.

En el archivo web.xml definiremos el método de autenticación de los usuarios que se conecten a nuestra aplicación.

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>bibliotecajdbc</realm-name>
  <form-login-config>
    <form-login-page>/seguridad/identificate.jsp</form-login-page>
    <form-error-page>/seguridad/credencialesnovalidas.jsp</form-error-page>
  </form-login-config>
</login-config>
```

Mediante estos elementos del archivo web.xml estamos indicando:

El método de autenticación: Mediante formulario.

El dominio de seguridad: bibliotecajdbc

Página del formulario de login

Página de error de autenticación: Página a la que se dirigirá la aplicación si las credenciales no son válidas.

A continuación se muestra el contenido de las páginas identificate.jsp y credencialesnovalidas.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Pagina de identificacion</title>
<jsp:directive.include file=" ../includes/includefiles.jspf" />
</head>
<body>
  <div id="container">
    <div id="header"></div>
    <form action="j_security_check" method="POST">
      <table cellpadding="3" cellspacing="2" border="0" width="100%">
        <tr>
          <td colspan="2">
            <h2>
              Introduzca sus datos de usuario
            </h2>
          </td>
        </tr>
        <tr>
          <td colspan="2">
          </td>
        </tr>
        <tr>
          <td width="11%">*Username</td>
          <td>
            <input type="text" name="j_username"/>
          </td>
        </tr>
        <tr>
          <td>*Password</td>
          <td>
            <input type="password" name="j_password"/>
          </td>
        </tr>
      </table>
    </form>
  </div>
</body>
</html>
```

```

        </td>
      </tr>
      <tr>
        <td>&nbsp;</td>
        <td colspan="2">
          <input type="submit" name="login" value="Login"/>
          <a href="/Biblioteca/altasocio.jsp">Registrese</a>
        </td>
      </tr>
      <tr>
        <td colspan="2">
          <hr/>
        </td>
      </tr>
    </table>
  </form>
</div>
</body>
</html>

```

Cosas obligatorias en el formulario de acceso:

- Propiedad `action= j_security_check`

j_security_check es un servlet proporcionado por el API JAAS que se va a encargar de recoger las credenciales del usuario que quiere conectarse a la aplicación.

- **j_username (el nombre es obligatorio):** recoge el nombre del usuario que quiere acceder a nuestra aplicación.
- **j_password (obligatorio el nombre):** que recoge la clave de usuario que se está identificando.

j_security_check haciendo uso del dominio de seguridad bibliotecajdbc podrá comprobar que el usuario y la password se corresponden con un usuario de la tabla USUARIOS, tabla asociada a nuestro dominio de seguridad.

Si las credenciales no son válidas, j_security_check redirige al usuario a la página credencialesnovalidas.jsp. Esta página normalmente se configura de forma que se muestre el error en las credenciales y permita volver a identificarse en la aplicación.

```

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Pagina de identificacion</title>
<jsp:directive.include file="./includes/includefiles.jspf" />
</head>
<body>
    <div id="container">
        <div id="header"></div>
        <div id="diverror">
            <p>
                <strong><c:out value="Error" /></strong> <br>
                <c:out value="Usuario o password incorrectos " />
            </p>
        </div>
        <form action="j_security_check" method="POST">
            <table cellpadding="2" cellspacing="3" border="0" width="100%">
                <tr>
                    <td colspan="2">

```

```

        <h2>Introduzca sus datos de usuario</h2>
    </td>
</tr>
<tr>
    <td width="11%">*Username</td>
    <td><input type="text" name="j_username" /></td>
</tr>
<tr>
    <td>*Password</td>
    <td><input type="password" name="j_password" /></td>
</tr>
<tr>
    <td>&nbsp;</td>
    <td colspan="2"><input type="submit" name="Login"
        value="Login" /> <a href="/Biblioteca/altasocio.jsp">Registrese</a>
    </td>
</tr>
</table>
</form>
</div>
</body>
</html>

```

Como se observa la página es idéntica a `identificate.jsp`. La única diferencia es la inclusión del mensaje de error.

5.2. Definir los roles de la aplicación.

Recordemos que los roles serán permisos para acceder a un determinado recurso. Nuestra aplicación define los roles: socios y administrativos mediante los siguientes elementos del archivo `web.xml`.

```

<security-role>
    <description>Rol de usuarios registrados para la aplicacion</description>
    <role-name>socios</role-name>
</security-role>
<security-role>
    <description>Rol de administrativos para la aplicacion</description>
    <role-name>administrativos</role-name>
</security-role>

```

5.3. Una vez definidos los roles, definimos los recursos protegidos de nuestra aplicación. Realizamos esta tarea mediante los siguientes elementos del archivo `web.xml`:

```

<security-constraint>
    <display-name>Consulta Bibliografica</display-name>
    <web-resource-collection>
        <web-resource-name>Consulta Bibliografica</web-resource-name>
        <description></description>
        <url-pattern>/socios/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <description></description>
        <role-name>socios</role-name>
        <role-name>administrativos</role-name>
    </auth-constraint>
</security-constraint>

```

```

<security-constraint>
  <display-name>Paginas administrativas</display-name>
  <web-resource-collection>
    <web-resource-name>Paginas administrativas</web-resource-name>
    <description></description>
    <url-pattern>/admin/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <description></description>
    <role-name>administrativos</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint>
  <display-name>Acceso al controlador de operaciones de Socio</display-
name>
  <web-resource-collection>
    <web-resource-name>Acceso al controlador de operaciones de Socio</web-
resource-name>
    <description></description>
    <url-pattern>/controllersocio</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <description></description>
    <role-name>socios</role-name>
    <role-name>administrativos</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint>
  <display-name>Acceso al controlador de operaciones de
Administrativos</display-name>
  <web-resource-collection>
    <web-resource-name>Acceso al controlador de operaciones de
Administrativos</web-resource-name>
    <description></description>
    <url-pattern>/controlleradmin</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <description></description>
    <role-name>administrativos</role-name>
  </auth-constraint>
</security-constraint>

```

Recursos protegidos de nuestra aplicación:

- Páginas de la ruta /socios/*.
Pueden acceder a este recurso los usuarios que posean los roles socios o administrativos.
- Páginas de la ruta /admin/*.
Pueden acceder a este recurso los usuarios que posean el role administrativo.
- Servlet de la ruta /controllersocio
Pueden acceder a este recurso los usuarios que posean los roles socios o administrativos.
- Servlet de la ruta /controlleradmin
Pueden acceder a este recurso los usuarios que posean el rol de administrativos.

5.4. Establecer la correspondencia entre roles y grupos

web.xml ha establecido el acceso a recursos protegidos mediante el concepto de rol.

El servidor GlassFish dispone de usuarios autenticados y grupos de usuarios autenticados.

Para asignar los roles que tienen los usuarios de nuestra aplicación hacemos uso del archivo glassfish-web.xml.

Se pueden asignar de manera individual a usuarios o hacerlo por grupos. En el siguiente ejemplo se indica que el rol Mascot lo tiene el usuario Duke y el rol Admin lo tiene el grupo Director. Cuando el número de usuarios es alto, establecer la correspondencia mediante grupos es mucho más cómodo.

```
<glassfish-web-app>
...
<security-role-mapping>
<role-name>Mascot</role-name>
<principal-name>Duke</principal-name>
</security-role-mapping>
<security-role-mapping>
<role-name>Admin</role-name>
<group-name>Director</group-name>
</security-role-mapping>
...
</glassfish-web-app>
```

En nuestra aplicación de ejemplo hemos establecido una correspondencia 1 a 1 entre grupos y roles. Así hemos hecho corresponder el grupo sociosbiblioteca con el rol socios. Así cualquier usuario que pertenezca al grupo sociosbiblioteca dispondrá del rol socios que le permite el acceso al controlador /controllersocio y a los recursos situados en la ruta /socios/*.

```
<security-role-mapping>
  <role-name>socios</role-name>
  <group-name>sociosbiblioteca</group-name>
</security-role-mapping>
```

De la misma forma hemos hecho corresponder el grupo adminbiblioteca con el rol administrativos

```
<security-role-mapping>
  <role-name>administrativos</role-name>
  <group-name>adminbiblioteca</group-name>
</security-role-mapping>
```

Así cualquier usuario que pertenezca al grupo adminbiblioteca dispondrá del rol administrativos que le permite el acceso al controlador /controlleradmin y a los recursos situados en la ruta /admin/*. El role administrativos también permite el acceso a los recursos: /controllersocio y /socios/*

¿Cómo funciona la seguridad con el sistema que hemos implantado?

Cuando un usuario no registrado intenta acceder desde el navegador a un recurso protegido, el servicio JAAS lo redirige automáticamente a la página de identificación para que se identifique en el sistema.

Si un usuario registrado intenta acceder a una zona prohibida para él, porque no tiene el rol que le permite el acceso a dicho recurso, el sistema responde con un error 403

