Eusko Jaurlaritzaren Informatika Elkartea
Sociedad Informática del Gobierno Vasco

UDA - Utilidades de desarrollo de aplicaciones

Uso de WebDAV en UDA

Fecha: 08/01/2014 | **Referencia:**

EJIE S.A.
Mediterráneo, 14
Tel. 945 01 73 00*
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01080 Vitoria-Gasteiz
www.ejie.es

Control de documentación

Título de documento: Web Services

Histórico de versiones

Código:	Versión:	Fecha:	Resumen de cambios:
	1.0.0	08/01/2014	Primera versión

Cambios producidos desde la última versión

Primera versión

Control de difusión

Responsable: Ander Martínez

Aprobado por:

Firma:	Fecha:

Distribución:

Referencias de archivo

Autor:

Nombre archivo:

Localización:

Contenido

1 Introducción

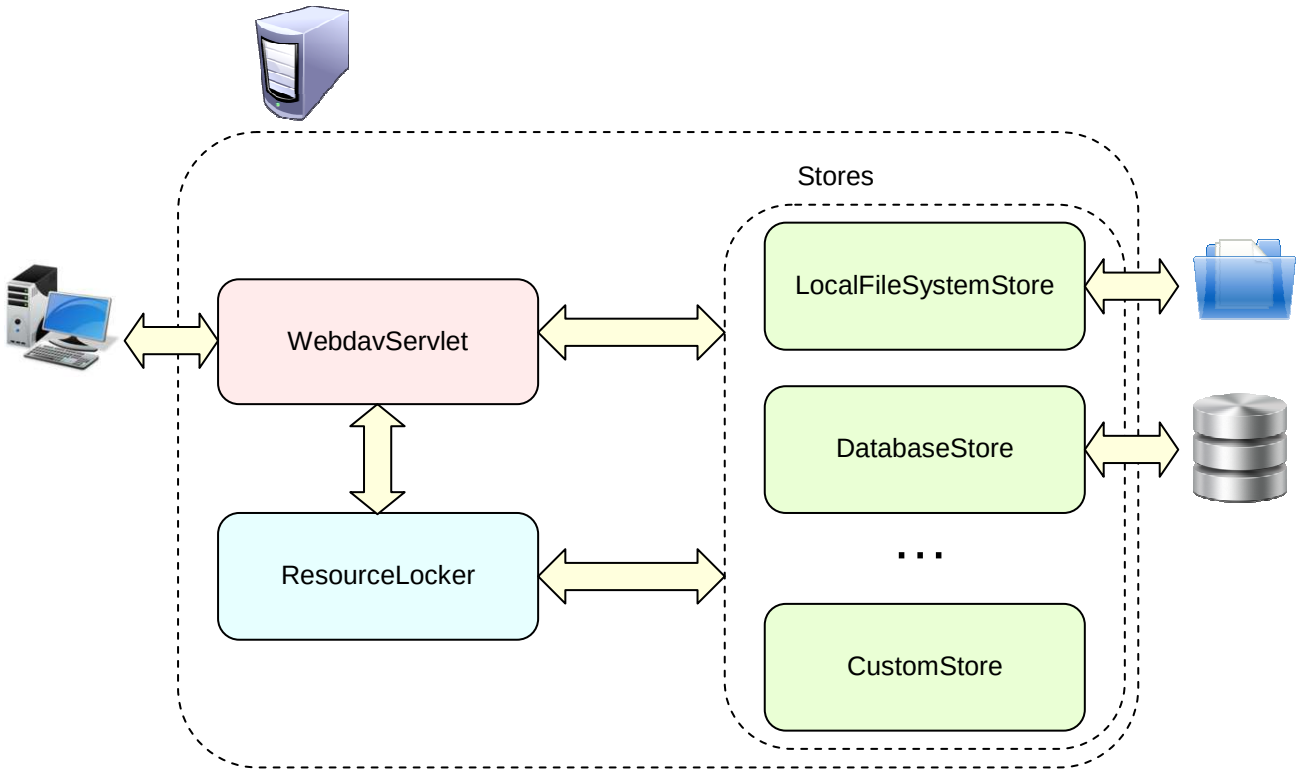
El presente documento tiene como objetivo definir y explicar de manera detallada como integrar en una aplicación el uso de WebDAV para el tratamiento de ficheros almacenados en servidor.

El objetivo es proporcionar una solución común a todas las aplicaciones que requieran el uso de WebDAV para la gestión de documentos.

La solución propuesta proporciona soporte para WebDAV nivel 2, la cual permite realizar bloqueos de directorios y ficheros para poder protegerlos y evitar que sean modificados por otros usuarios. Algunos clientes WebDAV como Microsoft Office o Microsoft Web Folders requieren de un servidor de WebDAV que implemente el nivel 2.

2 Arquitectura

Este sería el esquema de la arquitectura del componente WebDAV:



El componente WebDav consta de los siguientes módulos:

- ***Servlet***: Es el encargado de recibir y gestionar las peticiones WebDAV. Permite al usuario interactuar con los recursos accesibles mediante WebDAV.

  Como implementación del servlet se proporciona la clase **com.ejie.x38.webdav.WebdavSpringServlet,** preparada para ser integrada en Spring.

- ***Store***: Es el encargado de lograr la interacción entre las peticiones WebDAV y los documentos almacenados.

  Implementando diferentes *stores* se puede extender la capacidad del componente WebDAV para gestionar recursos almacenados en cualquier sistema de persistencia.

  Por defecto se proporciona el store **com.ejie.x38.webdav.LocalFileSystemStore** que permite gestionar recursos almacenados en un sistema de ficheros.

  Para utilizar un sistema de persistencia diferente para los recursos, se deberá de crear, por parte de las aplicaciones, una clase que implemente la interface **com.ejie.x38.webdav.IWebdavStore.**

- ***Locker***: Gestiona los bloqueos de los ficheros para evitar las modificaciones por parte de otros usuarios. El sistema de bloqueos es necesario para determinadas operaciones con WebDAV nivel 2.

  El componente permite gestionar los bloqueos mediante persistencia en memoria o uso de base de datos. Se recomienda el uso de base de datos ya que es la única que garantiza el correcto funcionamiento en entornos *cluster*.

  La clase que gestiona los bloqueos de los ficheros es **com.ejie.x38.webdav.locking.DataBaseResourceLocks.**

3 Configuración

Para utilizar el componente de WebDAV se deben de realizar los siguientes pasos:

3.1 Obtener dependencias

El componente WebDAV se proporciona a través de la librería x38ShLibClasses-dav. Para incorporarla a la aplicación bastará con añadir la siguiente dependencia en el fichero *pom.xml* de la aplicación.

```
	<dependency>
		<groupId>com.ejie.x38</groupId>
		<artifactId>x38ShLibClasses-dav</artifactId>
		<version>2.4.0-RELEASE</version>
	</dependency>
```

Se recomienda indicar la versión más reciente que se haya liberado.

3.2 Configuración de Spring

Se deberá de crear un nuevo *fichero webdav-config.xml* en el que se incluirá toda la configuración relacionada con el componente.

El contenido del fichero será el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

	<!-- Gestión de los locks de recursos de WebDAV mediante base de datos -->
	<bean id="webDavDataBaseLockDao"
class="com.ejie.x38.webdav.locking.DataBaseLockDaoImpl">
		<property name="lockingDataSource" ref="dataSource" />
		<property name="lockingTableName" value="Y64BWEBDAVLOCK" />
	</bean>

	<bean id="webDavDataBaseLockService"
class="com.ejie.x38.webdav.locking.DataBaseLockServiceImpl">
		<property name="dataBaseLockDao" ref="webDavDataBaseLockDao" />
	</bean>
```

```
	<bean id="dataBaseResourceLocks"
class="com.ejie.x38.webdav.locking.DataBaseResourceLocks">
		<property name="dataBaseLockService" ref="webDavDataBaseLockService"
/>
	</bean>

	<!-- Definición del almacén de datos de WebDAV -->
	<bean id="webDavStore" class="com.ejie.x38.webdav.LocalFileSystemStore">
		<property name="rootPath" value="/datos/xxx/tmp/" />
	</bean>

	<!-- Servlet que gestiona las peticiones WebDAV -->
	<bean id="webdavServlet" class="com.ejie.x38.webdav.WebdavSpringServlet">
		<property name="webDavStore" ref="webDavStore" />
		<property name="resourceLocks" ref="dataBaseResourceLocks" />
	</bean>
</beans>
```

Los beans que se definen en el fichero son los siguientes:

- ***webdavServlet:*** Definición del servlet que se encargará de escuchar las peticiones WebDAV. El identificador aquí especificado deberá coincidir con el utilizado en el fichero web.xml, tal y como se comenta más adelante.

  La definición del bean requiere informar los siguientes parámetros:

  - ***webDavStore:*** Bean de tipo `com.ejie.x38.webdav.IWebdavStore` que implementa la interacción entre las peticiones WebDav y el almacén de datos.
  - ***resourceLocks:*** Bean de tipo `com.ejie.x38.webdav.IResourceLocks` que implementa la gestión de bloqueos.

- ***webDavStore:*** Determina el bean encargado de permitir la interacción entre las peticiones de WebDAV y el almacén de datos.

  En x38 se proporciona una implementación por defecto mediante la clase `com.ejie.x38.webdav.LocalFileSystemStore`. Esta clase realiza las operaciones WebDAV contra un sistema de ficheros local. Mediante la propiedad `rootPath` se especifica la ruta sobre el sistema de ficheros local que se tomará como base para el componente WebDAV.

  En caso de necesitar interactuar con un sistema de almacenamiento diferente se deberá proporcionar una implementación que resuelva la interacción entre el propio almacén y las peticiones de WebDAV. Este punto se detalla en el apartado *3.Implementación del Store.*

Las siguientes definiciones de beans son necesarias para la gestión de los bloqueos de los documentos mediante el uso de base de datos.

- ***webDavDataBaseLockDao:*** Bean de tipo `com.ejie.x38.webdav.locking.DataBaseLockDaoImpl` que implementa las operaciones contra la base de datos.

  La definición del bean requiere informar los siguientes parámetros:

  - ***lockingDataSource:*** Datasource utilizado para conectar con la base de datos.

  - ***lockingTableName:*** Nombre de la tabla existente en la base de datos que será utilizada para gestionar los bloqueos.

- ***webDavDataBaseLockService:*** Bean de tipo `com.ejie.x38.webdav.locking.DataBaseLockServiceImpl` implementa los métodos encargados de gestionar las operaciones del bloqueo.

  La definición del bean requiere informar los siguientes parámetros:

  - ***dataBaseLockDao:*** Dao utilizado para conectar con la base de datos.

- ***dataBaseResourceLocks:*** Bean de tipo `com.ejie.x38.webdav.locking.DataBaseResourceLocks` implementa el recurso de bloqueos del componente WebDAV mediante base de datos.

  La definición del bean requiere informar los siguientes parámetros:

  - ***dataBaseLockService:*** Servicio utilizado para realizar las operaciones de bloqueo mediante base de datos.

A continuación se deberá añadir una entrada en el fichero *app-config,xml* de WAR de la aplicación para que se incluya en la carga de configuración de Spring.

```
<beans xmlns=http://www.springframework.org/schema/beans ...>
      <import resource="mvc-config.xml" />
      <import resource="log-config.xml"/>
      <import resource="validation-config.xml"/>
      <import resource="security-core-config.xml"/>
      <import resource="security-config.xml"/>
      <import resource="webdav-config.xml"/>
</beans>
```

3.3 web.xml

Para que el componente gestione las peticiones WebDAV se deberá incluir en el fichero web.xml del War de la aplicación la siguiente definición:

```
      <servlet>
          <display-name>WebDAVServlet</display-name>
          <servlet-name>webdavServlet</servlet-name>
          <servlet-class>
              org.springframework.web.context.support.HttpRequestHandlerServlet
          </servlet-class>
      </servlet>
      <servlet-mapping>
          <servlet-name>webdavServlet</servlet-name>
          <url-pattern>/webdavServlet/*</url-pattern>
      </servlet-mapping>
```

El valor indicado en la propiedad *servlet-name*, tanto para la definición del servlet como del servlet-mapping, se deberá corresponder con el que se ha utilizado como identificador en la definición del bean *`com.ejie.x38.webdav.WebdavSpringServlet`* en el fichero webdav-config.xml

```
    <servlet-name>webdavServlet</servlet-name>
```

En la propiedad *url-pattern* se definirá la url a partir de la cual se gestionarán las peticiones webdav.

```
    <url-pattern>/webdavServlet/*</url-pattern>
```

4 Implementación del Store

Mediante la librería x38 se proporciona un *store* por defecto que permite al componente WebDAV interactuar con archivos almacenados en un sistema de ficheros local.

En una aplicación puede ser habitual que los documentos se almacenen mediante un sistema distinto, ya sea en base de datos o en un gestor documental. En este caso se deberá proveer al componente WebDAV de una implementación del *store* que permita la interacción entre las peticiones WebDAV y los archivos almacenados.

Para implementar un *store* se deberá crear una clase que implemente la interfaz `IWebdavStore`, p. ej.:

```
package com.ejie.xxx.webdav;

public class CustomWebdavStore implements IWebdavStore {

}
```

La clase deberá implementar los siguientes métodos:

- ***begin:*** Indica que se ha iniciado una nueva petición o transacción sobre el *store* implementado.

  Parámetros de entrada:
  - *principal*: Representa la identidad que realiza el inicio de la petición. Puede ser nulo en caso de no estar disponible.

  Retorno.
  - Objeto identificador de la transacción iniciada.

```
@Override
public ITransaction begin(Principal principal) {
	// TODO Auto-generated method stub
	return null;
}
```

- ***checkAuthentication:*** Comprueba que la información de autenticación proporcionada es correcta. En caso de no serlo se lanza una excepción.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.

```
@Override
public void checkAuthentication(ITransaction transaction) {
	// TODO Auto-generated method stub

}
```

- ***commit:*** Determina que todos los cambios realizados pueden ser marcados como permanentes y todas las transacciones, conexiones y recursos temporales pueden ser liberados.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.

```
@Override
public void commit(ITransaction transaction) {
	// TODO Auto-generated method stub

}
```

- ***rollback:*** Determina que todos los cambios realizados pueden ser deshechos y todas las transacciones, conexiones y recursos temporales pueden ser liberados.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.

```
@Override
public void rollback(ITransaction transaction) {
	// TODO Auto-generated method stub

}
```

- ***createFolder:*** Crea un directorio en la ruta especificada por el parámetro *folderUri*.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.
  - *forlderUri*: Ruta del directorio.

```
@Override
public void createFolder(ITransaction transaction, String folderUri) {
	// TODO Auto-generated method stub

}
```

- ***createResource:*** Crea un contenedor del recurso en la ruta especificada por el parámetro *resourceUri*.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.
  - *resourceUri*: Ruta del contenedor del recurso.

```
@Override
public void createResource(ITransaction transaction, String resourceUri) {
	// TODO Auto-generated method stub

}
```

- ***getResourceContent:*** Devuelve el contenido del recurso especificado por el parámetro *resourceUri*.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.
  - *resourceUri*: Ruta del recurso que se desea recuperar.

  Retorno.
  - Objeto InputStream a partir del cual se puede leer el contenido del recurso.

```
@Override
public InputStream getResourceContent(ITransaction transaction,
            String resourceUri) {
      // TODO Auto-generated method stub
      return null;
}
```

- ***setResourceContent:*** Almacena el contenido del recurso especificado por el parámetro *resourceUri*.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.
  - *resourceUri*: Ruta del recurso que se desea almacenar.
  - *content*: InputStream a partir del cual se va a leer el recurso.
  - *contentType*: *ContentType* del recurso que se va a almacenar. Puede ser *null* en caso de ser desconocido.
  - *characterEncoding*: *CharacterEncoding* del recurso que se va a almacenar. Puede ser *null* en caso de ser desconocido.

  Retorno.
  - Tamaño del recurso almacenado en *bytes*.

```
@Override
public long setResourceContent(ITransaction transaction,
            String resourceUri, InputStream content, String contentType,
            String characterEncoding) {
      // TODO Auto-generated method stub
      return 0;
}
```

- ***getChildrenNames:*** Devuelve los nombres de los hijos del directorio especificado por el parámetro *folderUri.*

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.
  - *folderUri*: Ruta del directorio del cual hay que obtener los hijos.

  Retorno.
  - Array de *string*s que contiene la lista de hijos del directorio.

```
@Override
public String[] getChildrenNames(ITransaction transaction,String folderUri){
      // TODO Auto-generated method stub
```

```
        return null;
    }
```

- ***getResourceLength:*** Devuelve el tamaño, en *bytes* del recurso solicitado por el parámetro *resourceUri*.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.
  - *resourceUri*: Ruta del recurso que se desea conocer el tamaño.

  Retorno.
  - Tamaño en *bytes* del recurso solicitado.

```
    @Override
    public long getResourceLength(ITransaction transaction,String resourceUri) {
        // TODO Auto-generated method stub
        return 0;
    }
```

- ***removeObject:*** Elimina el objeto especificado por el parámetro *uri*.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.
  - *uri*: Ruta del recurso que se desea eliminar (fichero o directorio).

```
    @Override
    public void removeObject(ITransaction transaction, String uri) {
        // TODO Auto-generated method stub

    }
```

- ***getStoredObject:*** Devuelve un *StoredObject* correspondiente al recurso identificado mediante la ruta especificada por el parámetro *uri*.

  Parámetros de entrada:
  - *transaction*: Objeto identificador de la transacción.
  - *uri*: Ruta del recurso del que se quiere obtener el *StoredObject*.

  Retorno.
  - *StoredObject* del recurso correspondiente.

```
    @Override
    public StoredObject getStoredObject(ITransaction transaction, String uri) {
        // TODO Auto-generated method stub
        return null;
    }
```

Un objecto StoredObject contiene información relative a un recurso WebDAV. Las propiedades que contiene son las siguientes:

- *isFolder: Indica si es un directorio (true) o un fichero (false).*
- *lastModified: Fecha que determina en que momento se realizó la última modificación.*
- *creationDate: Fecha que determina el momento de creación del recurso.*
- *contentLength: Tamaño en bytes del recurso.*

Una vez implementado el *store* se deberá indicar al componente de WebDAV que lo utilice para interactuar con el almacén de ficheros. Esto se indicará en el fichero *webdav-config.xml* del siguiente modo:

```
<!-- Definición del almacén de datos de WebDAV -->
<bean id="customWebDavStore" class="com.ejie.xxx.webdav.CustomWebdavStore">
</bean>

<!-- Servlet que gestiona las peticiones WebDAV -->
<bean id="webdavServlet" class="com.ejie.x38.webdav.WebdavSpringServlet">
      <property name="webDavStore" ref=" customWebDavStore " />
      <property name="resourceLocks" ref="dataBaseResourceLocks" />
</bean>
```

5 Script de BBDD

Para gestionar correctamente los bloqueos de los ficheros mediante base de datos se deberá de crear una nueva tabla en el esquema de BBDD de la aplicación. En esta tabla se almacenará la información necesaria para realizar los bloqueos de los ficheros que estén siendo editados por parte de los usuarios de la aplicación.

El nombre de la tabla empleado deberá de coincidir con el indicado en la propiedad `lockingTableName` del bean `webDavDataBaseLockDao` en el fichero de configuración *webdav-config.xml.*

El script de creación de la tabla sería el siguiente:

```
CREATE TABLE "WEBDAV_LOCK"
  (	"ID" VARCHAR2(255 BYTE) NOT NULL ENABLE,
	"PATH" VARCHAR2(255 BYTE) NOT NULL ENABLE,
	"LOCK_DEPTH" VARCHAR2(10 BYTE),
	"EXPIRES_AT" NUMBER(19,0),
	"OWNER" VARCHAR2(255 BYTE) NOT NULL ENABLE,
	"EXCLUSIVE_LOCK" VARCHAR2(1 BYTE),
	"LOCK_TYPE" VARCHAR2(10 BYTE),
	"CHILDREN_ID" VARCHAR2(255 BYTE),
	"PARENT_ID" VARCHAR2(255 BYTE),
	"TEMP_LOCK" VARCHAR2(1 BYTE),
	 CONSTRAINT "WEBDAV_LOCK_PK" PRIMARY KEY ("ID", "OWNER")
);
```

El nombre de la tabla (resaltado en negrita) deberá de cumplir con la normativa de nomenclatura de tablas de EJIE.

6 Edición en línea de un documento mediante Microsoft Office

Uno de los escenarios más utilizado por las aplicaciones es la edición en línea de documentos mediante Microsoft Office.

Si se han seguido los pasos indicados en los apartados anteriores se dispondrá de un servidor WebDAV nivel 2 de acuerdo a lo requerido por Microsoft Office.

Dependiendo de la versión de Microsoft Office utilizada se podrá realizar la edición en línea desde diferentes navegadores:

- Microsoft Office 2003 e inferiores: Internet Explorer.
- Microsoft Office 2007 y superiores: Internet Explorer, Firefox y Chrome.

Para cubrir todos estos escenarios se deben seguir los siguientes pasos:

1. Se deberá incluir el siguiente código en la jsp correspondiente. Esto sirve para que se pueda utilizar el plugin FFWinPlugin que permite abrir y editar documentos mediante Microsoft Office desde Firefox y Chrome.

```
<object id="winFirefoxPlugin" type="application/x-sharepoint" width="0"
height="0" style="visibility: hidden;"></object>
```

2. En el fichero js se incluirá el siguiente *script* para gestionar cada uno de los diferentes tipos de navegadores utilizados por el usuario.

```
var url = baseUrlOfApp+"webdavServlet/webdav_file.doc", obj, hownowPlugin,
version;

if ($.browser.msie === true){
      // Gestión para Internet Explorer
      try{
            // Microsoft Office 2007 y superiores
            obj = new ActiveXObject("SharePoint.OpenDocuments.3");
            obj.EditDocument(url);
      }catch(e){
            try{
                  // Microsoft Office 2003 e inferiores
                  obj = new ActiveXObject('Word.Application');
                  obj.Visible = true;
                  obj.Documents.Open(encodeURI(url));
            }catch(e){
                  // No se puede editar en línea, se descarga el archivo
                  window.open("url_de_descarga_del_fichero", '_blank');
            }
      }
} else {
      // Gestión para Firefox y Chrome
      try{
```

```
            // Microsoft Office 2007 y superiores
            hownowPlugin = document.getElementById("winFirefoxPlugin");
          version = hownowPlugin.GetOfficeVersion();
          hownowPlugin.EditDocument(url, version);
      }catch(e){
            // No se puede editar en línea, se descarga el archivo
            window.open("url_de_descarga_del_fichero", '_blank');
      }
}
```

6.1 Componente RUP DAV

Para facilitar el uso de WebDAV desde una aplicación UDA se ha implementado el componente rup.dav.

Uno de los objetivos es abstraer al desarrollador de la gestión de la edición de un documento en línea dependiente del navegador utilizado por el usuario.

Este componente se ha incluido en la versión v2.4.1 de UDA por lo que se podrá hacer uso del mismo en las aplicaciones que dispongan de dicha versión o posteriores.

En caso de utilizar este componente, el proceso sería el siguiente:

1. Se deberá incluir el siguiente código en la jsp correspondiente. Esto sirve para que se pueda utilizar el plugin FFWinPlugin que permite abrir y editar documentos mediante Microsoft Office desde Firefox y Chrome.

```
<object id="winFirefoxPlugin" type="application/x-sharepoint" width="0"
height="0" style="visibility: hidden;"></object>
```

2. Para invocar al servidor de WebDAV y realizar la edición online se llamará al componente rup.dav del siguiente modo:

```
jQuery.rup_dav.editOnline({
      url:"webdavServlet/webdav.doc",
      xlnetsAuth:true,
      downloadOnError:true,
      alternateDownloadURL:"../upload?fileName=webdav.doc"
});
```

Las propiedades de configuración son las siguientes:

- **url**: Url utilizada para acceder al recurso deseado a través del servlet de WebDAV.
- **xlnetsAuth**: Determina si se debe realizar la petición teniendo en cuenta la autenticación del usuario en XLNets. Esto permite aplicar la seguridad de una aplicación UDA que utilice Spring Security con autenticación del usuario en XLNets.
- **downloadOnError**: Determina si se debe habilitar la descarga automática del fichero en caso de que no sea posible realizar la edición online.
- **alternateDownloadURL**: Esta URL será la utilizada para que el usuario pueda descargar el fichero en caso de que falle la edición online.

7 Seguridad

En un principio el componente WebDAV no proporciona mecanismos de seguridad adicionales a los que proporciona la aplicación.

Es tarea de la aplicación securizar las urls de acceso y los métodos de negocio utilizados para proveer del recurso sobre el que se desea trabajar mediante WebDAV.

En las aplicaciones UDA la securización de estos elementos se realiza mediante los mecanismos habituales que proporciona Spring Security, de modo que se podrá hacer uso tanto de las anotaciones como de la configuración declarativa en los ficheros de configuración de Spring Security.

La securización de la url del servlet del componente WebDAV se realiza del mismo modo que con las demás urls de la aplicación. Se añadirá en el fichero *security-config.xml* la regla correspondiente que determinará la securización de la url por parte de los usuarios.

Un ejemplo sería el siguiente:

```
<bean id="filterSecurityInterceptor"
class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor">
       <property name="authenticationManager" ref="authenticationManager" />
        <property name="accessDecisionManager" ref="affirmativeBased" />
        <property name="securityMetadataSource" >
            <security:filter-security-metadata-source use-expressions="true" request-matcher="regex">

                <security:intercept-url pattern="/webdavServlet/.*" access="hasRole('ROLE_XXX-IN-
0005')" />
              <!—Resto de reglas de securización -->
            </security:filter-security-metadata-source>
        </property>
</bean>
```

De este modo determinamos que el usuario que desee acceder a los recursos de WebDAV debe disponer del nivel de autorización correspondiente otorgado por el *rol* de seguridad *ROLE_XXX-IN-005.*

7.1 XLNets

En caso de utilizar XLNets para la autenticación y la securización de la aplicación, se deberá realizar una configuración adicional para permitir al componente WebDAV integrarse correctamente con XLNets.

Se deberá incluir en el fichero *security-core-config.xml* la siguiente definición:

```
<bean id="springSecurityFilterChain"
  class="org.springframework.security.web.FilterChainProxy">
  <security:filter-chain-map request-matcher="regex">
       <security:filter-chain pattern="/sessionInfo.*" filters="none"/>
       <security:filter-chain pattern="/mockLoginPage.*" filters="none"/>
       <security:filter-chain pattern="/mockLoginAjaxPage.*" filters="none"/>
       <security:filter-chain pattern="/error.*" filters="none"/>
       <security:filter-chain pattern="/accessDenied.*" filters="none"/>

       <security:filter-chain pattern="/webdavServlet/.*" filters="
              webDAVXLNetsAuthenticationFilter,
       exceptionTranslationFilter,
       securityContextPersistenceFilter,
```

```
        logoutFilter,
        preAuthenticateProcessingFilter,
        filterSecurityInterceptor" />

        <security:filter-chain pattern="**" filters="
        exceptionTranslationFilter,
        securityContextPersistenceFilter,
        logoutFilter,
        preAuthenticateProcessingFilter,
        filterSecurityInterceptor" />
  </security:filter-chain-map>
</bean>

<bean id="webDAVXLNetsAuthenticationFilter"
class="com.ejie.x21a.webdav.WebDAVXLNetsAuthenticationFilter"/>
```

El objetivo es definir para la url del servlet del componente WebDAV una cadena de ejecución de filtros de seguridad entre los que se incluye el filtro *webDAVXLNetsAuthenticationFilter*.

Mediante este filtro se logra que las peticiones que realiza el cliente de XLNets, por ejemplo el Microsoft Office, puedan ser securizadas mediante la misma configuración de seguridad empleada en la aplicación.