



Groovy & Grails: Desarrollo rápido de aplicaciones

Sesión 11: Seguridad



Seguridad

- Autenticación
- Registro de usuarios con CAPTCHAS
- Control de acceso: Filtros
- JSecurity



Autenticación

- Comprobar que el usuario es realmente quien dice ser
- También se tiene en cuenta cuando el usuario abandona la aplicación
- El método de autenticación de nuestra aplicación es demasiado simple



Autenticación

- Vamos a cambiar el método de autenticación de nuestra aplicación para solicitar la contraseña
- Debemos cambiar la vista de la página de login:
grails-app/views/usuario/login.gsp



Autenticación

```
....  
<form>  
  <br/><label for="login">Nombre de usuario</label>  
  <br/><input type="text" maxlength="20" id="login"  
name="login" value="{fieldValue(bean:usuarioInstance,field:'login')}" />  
  <br/><label for="login">Contraseña</label>  
  <br/><input type="password" maxlength="20" id="password"  
name="password" />  
  <br/><div class="buttons"><span class="button">  
<g:actionSubmit value="Login" action="handleLogin" />  
</span></div>  
</form>  
....
```



Autenticación

- Debemos también modificar el controlador de la clase Usuario para que se compruebe también la contraseña
- El método a modificar sería *handleLogin()*



Autenticación

```
def handleLogin = {  
    def usuario =  
    Usuario.findByLoginAndPassword(params.login,params.password)  
    if (!usuario) {  
        flash.message = "El usuario ${params.login} no existe"  
        redirect(controller: 'usuario', action:'login')  
        return  
    }  
    else {  
        session.usuario = usuario  
        redirect(controller:'operacion')  
    }  
}
```



Autenticación

- Hasta el momento, la contraseña de los usuarios se almacena en forma de texto plano
- Esto se considera una falta muy grave
- Debemos encriptar la contraseña mediante técnicas *hash*



Autenticación

- Utilizaremos el paquete *org.apache.commons.codec.digest.DigestUtils* que tiene los siguientes métodos:
 - *DigestUtils.md5(java.lang.String cadena)*
 - *DigestUtils.md5Hex(java.lang.String cadena)*
 - *DigestUtils.shaHex(java.lang.String cadena)*



Autenticación

- Modificamos el método *save()* de la clase Usuario para encriptar la contraseña antes de persistir el objeto
- No olvidar importar el paquete *org.apache.commons.codec.digest.DigestUtils*



Autenticación

```
def save = {  
    params.password = DigestUtils.md5Hex(params.password)  
    def u = usuarioService.altaUsuario(params)  
    if(!u.hasErrors()) {  
        flash.message = "Usuario ${u.nombre} ${u.apellidos} created"  
        redirect(action:show,id:u.id)  
    }  
    else {  
        render(view:'create',model:[usuarioInstance:u])  
    }  
}
```



Autenticación

- Debemos modificar también el fichero *BootStrap.groovy* para que la contraseña sea encriptada antes de ser persistido

```
def usuario1 = new Usuario(  
    login:'frangarcia',  
    password:DigestUtils.md5Hex('mipassword'),  
    nombre:'Francisco José',  
    apellidos:'García Rico',  
    tipo:'administrador'  
)
```



Autenticación

- También deben introducirse algunos cambios en el método *handleLogin()* para comprobar la contraseña encriptada

```
def usuario = Usuario.findByLoginAndPassword(  
    params.login,  
    DigestUtils.md5Hex(params.password)  
)
```



Registro de usuarios con CAPTCHAS

- Vamos a crear un método en la aplicación para que sean los propios usuarios quienes se registren en el sistema
- Para evitar el registro automatizado de nuevos usuarios utilizaremos la técnica CAPTCHA



Registro de usuarios con CAPTCHAS

- **CAPTCHA** es una prueba desafío-respuesta utilizada en computación para determinar cuando el usuario es o no humano
- La típica prueba es aquella en la que se muestra al usuario una imagen distorsionada con unos caracteres, y que sólo un humano sería capaz de reescribir



Registro de usuarios con CAPTCHAS

- Empecemos instalando un plugin que nos facilitará la labor

```
grails install-plugin jcaptcha
```




Registro de usuarios con CAPTCHAS

- Debemos editar el archivo *grails-app/conf/Config.groovy* para que la aplicación pueda generar las imágenes distorsionadas



Registro de usuarios con CAPTCHAS

- Posteriormente, debemos crear la vista para el registro de los nuevos usuarios *grails-app/views/usuario/register.gsp*
- Esta vista será muy similar al archivo *create.gsp* con lo que podemos copiarlo e introducir posteriormente algunas modificaciones
- En este registro solicitaremos también al usuario que repita la contraseña



Registro de usuarios con CAPTCHAS

```
<g:form action="handleRegistration" method="post" >
  ...
  <tr class="prop"><td valign="top" class="name">
    <label for="password">Confirm Password:</label>
  </td> <td valign="top">
    <input type="password" maxlength="20" id="confirm"
name="confirm"/>
  </td> </tr>
  ....
</g:form>
```



Registro de usuarios con CAPTCHAS

```
<g:form action="handleRegistration" method="post" >
  ...
  <tr class="prop"><td valign="top" class="name">
    <label for="tipo">Captcha:</label>
  </td> <td valign="top" class="value $
{hasErrors(bean:usuarioInstance,field:'responseCaptcha','errors')} ">
    <jcaptcha:jpeg name="image" />
    <br>
    <g:textField name="responseCaptcha" value="" />
    <br>
  </td> </tr>
  ...
</g:form>
```



Registro de usuarios con CAPTCHAS

- Debemos añadir también los métodos *register()* y *handleRegister()* al controlador de la clase Usuario
- *handleRegister()* se encargará de comprobar si la pregunta del captcha ha sido correctamente contestada
- Debemos crear la variable *captchaService*



Registro de usuarios con CAPTCHAS

```
def captchaService
    ....
    def register = {
        def usuarioInstance = new Usuario()
        usuarioInstance.properties = params
        return ['usuarioInstance':usuarioInstance]
    }
```



Registro de usuarios con CAPTCHAS

```
def handleRegistration = {
    def usuarioInstance = new Usuario()
    //Proceso el captcha
    if (!jcaptchaService.validateResponse("image", session.id,
params.responseCaptcha)){
        flash.message = "El captcha no es correcto"
        redirect(controller:'usuario', action:'register')
    }
    else{
        ....
    }
}
```



Registro de usuarios con CAPTCHAS

```
//Compruebo la confirmación de la contraseña
if(params.password != params.confirm) {
    flash.message = "La confirmación de la contraseña no coincide
con la contraseña"
    redirect(action:register)
}
else{
    params.password = DigestUtils.md5Hex(params.password)
    params.tipo = "socio"
    usuarioInstance.properties = params
    ....
}
```




Registro de usuarios con CAPTCHAS

```
if(usuarioInstance.save()) {  
    session.usuario = usuarioInstance  
    redirect(controller:'operacion')  
}  
else {  
    render(view:'register',model:[usuarioInstance:usuarioInstance])  
}
```



Registro de usuarios con CAPTCHAS

- Se comprueba que la respuesta al desafío es correcto
- Se chequea que la confirmación de la contraseña es correcta
- Si se superan ambos tests, se persiste el usuario en la base de datos
- Se autentica directamente al usuario en la aplicación



Registro de usuarios con CAPTCHAS

- Debemos mostrar al usuario un enlace para que pueda registrarse
- Modificamos el archivo *grails-app/views/common/_header.gsp*



Registro de usuarios con CAPTCHAS

```
<div id="menu">
  <nobr>
    <g:if test="${session.usuario}">
      <b>${session.usuario?.nombre} ${session.usuario?.apellidos}</b>
      | <g:link controller="usuario" action="logout">
        <g:message code="encabezado.logout"/></g:link>
    </g:if>
    <g:else>
      <g:link controller="usuario" action="login">
        <g:message code="encabezado.login"/></g:link>
      | <g:link controller="usuario" action="register">Registrarse</g:link>
    </g:else>
  </nobr>
</div>
```



Registro de usuarios con CAPTCHAS

[Identificarse](#) | [Registrarse](#)

Biblioteca

[Home](#) [Usuario List](#)

Register Usuario

Login:	<input type="text"/>
Password:	<input type="password"/>
Confirm Password:	<input type="password"/>
Nombre:	<input type="text"/>
Apellidos:	<input type="text"/>
Captcha:	 <input type="text"/>

[Create](#)



Control de acceso: Filtros

- El control de acceso se refiere a “*quien puede hacer que*” en la aplicación
- Por ejemplo, que los usuarios de tipo socio no puedan crear usuarios
- Podemos utilizar los filtros de Grails para conseguir este tipo de restricciones



Control de acceso: Filtros

- Los filtros deben crearse en el directorio *grails-app/conf* y su nombre deben terminar con la palabra *Filters*

```
class SecurityFilters {  
    def filters = {  
        .....  
    }  
}
```



Control de acceso: Filtros

- Podemos definir los filtros de dos formas:
 - Especificando el controlador y la acción
 - Especificando la URI



Control de acceso: Filtros

- Podemos también especificar cuando queremos que se ejecute el filtro:
 - *before*, el filtro se ejecuta antes que la acción dada
 - *after*, el filtro se ejecuta después de la acción dada
 - *afterView*, el filtro se ejecuta después de que la vista sea renderizada



Control de acceso: Filtros

```
class SecurityFilters {
  def filters = {
    bibliotecaFilter(controller:'*', action:'*') {
      before = {
        if (!session.usuario
            && (controllerName.equals('usuario') && !actionName.equals('login'))
            && (controllerName.equals('usuario') && !actionName.equals('register'))
            && !controllerName.equals('jcaptcha') ){
          redirect(controller:'usuario', action:'login')
          return false
        }
      }
    }
  }
}
```



Control de acceso: Filtros

- La variable *controllerName* nos indica el *controlador* al que estamos accediendo
- La variable *actionName* nos indica la *acción* del controlador referida



Control de acceso: Filtros

- También es posible especificar la URI

```
otroFilter(uri:'/usuario/register') {  
    before = {  
        redirect(controller:'usuario', action:'registernew')  
    }  
}
```



Control de acceso: Filtros

- Grails inyecta una serie de propiedades para hacerlas accesibles en los filtros:
 - *request*
 - *response*
 - *session*
 - *servletContext*
 - *applicationContext*
 - *params*



JSecurity

- Grails incorpora una solución predefinida para la gestión de usuarios, permisos y sus relaciones
- Esta solución consiste en la utilización de un plugin llamado JSecurity



JSecurity

- En primer lugar debemos instalar el plugin

```
grails install-plugin jsecurity
```

- En segundo lugar ejecutaremos un comando que creará una serie de clases de dominio, controladores y vistas en nuestra aplicación

```
grails quick-start
```



Clases de dominio de JSecurity

- **JsecUser**
 - Se encarga de la gestión de los usuarios del sistema

Atributo	Descripción
<i>username</i>	Nombre de usuario
<i>passwordHash</i>	Contraseña



Clases de dominio de JSecurity

- **JsecRole**
 - Esta clase especifica los roles de la aplicación

Atributo	Descripción
<i>name</i>	Nombre del rol o perfil



Clases de dominio de JSecurity

- **JsecPermission**
 - Esta clase se utiliza para establecer grupos de permisos

Atributo	Descripción
<i>type</i>	Nombre para el permiso
<i>possibleActions</i>	Listado de acciones posibles para el permisos separadas por comas



Clases de dominio de JSecurity

- **JsecRolePermissionRel**
 - Esta clase se utiliza para establecer las relaciones entre los permisos y los roles del sistema

Atributo	Descripción
<i>role</i>	JsecRole relacionado
<i>permission</i>	JsecPermission relacionado
<i>target</i>	Controlador asociado con la relación
<i>actions</i>	Acciones objetivo de los permisos



Clases de dominio de JSecurity

- **JsecUserRoleRel**
 - Esta clase se utiliza para establecer las relaciones entre los usuarios y los roles del sistema

Atributo	Descripción
<i>user</i>	JsecUser relacionado
<i>role</i>	JsecRole relacionado



Clases de dominio de JSecurity

- **JsecUserPermissionRel**
 - Esta clase se utiliza para establecer las relaciones entre los permisos y los usuarios del sistema

Atributo	Descripción
<i>user</i>	JsecUser relacionado
<i>permission</i>	JsecPermission relacionado
<i>target</i>	Controlador asociado con la relación
<i>actions</i>	Acciones objetivo de los permisos



Clases de dominio de JSecurity

- **Carga de datos**

- Creamos algunos datos de ejemplo desde *grails-app/conf/BootStrap.groovy*

```
def admin = new JsecUser(username:'admin',  
passwordHash:DigestUtils.shaHex('password')).save()  
  
def profe = new JsecUser(username:'profesor',  
passwordHash:DigestUtils.shaHex('password')).save()
```



Clases de dominio de JSecurity

- **Carga de datos**

```
def adminRole = new JsecRole(name:'Administrador').save()
def profeRole = new JsecRole(name:'Profesor').save()

def perm = new JsecPermission(type:'BasicPermission',
possibleActions:'create,delete,update').save()

new JsecUserRoleRel(user:admin, role:adminRole).save()
new JsecUserRoleRel(user:profe, role:profeRole).save()
```



Clases de dominio de JSecurity

- **Carga de datos**
 - `http://localhost:8080/auth`
 - A continuación definiremos los filtros para definir que vamos a poder hacer con cada uno de esos roles



Clases de dominio de JSecurity

- **Carga de datos**

```
auth(controller: "*", action: "*") {  
    before = {  
        // Necesitamos acceder a estas acciones para identificarnos  
        if (controllerName=="auth" && (actionName == "login")  
            || (actionName=="signin"))  
            return true  
  
        // Para cualquier otra operación el usuario debe estar identificado  
        accessControl { true }  
    }  
}
```



Clases de dominio de JSecurity

- **Carga de datos**

```
usuarioControl(controller: "usuario",  
              action: "(create|edit|save|update|delete|show|list)") {  
    before = {  
        accessControl {  
            role("Administrador")  
        }  
    }  
}
```



Clases de dominio de JSecurity

- **Carga de datos**

```
operacionControl(controller: "operacion",
    action: "(create|edit|save|update|delete|show|list)") {
    before = {
        accessControl {
            role("Profesor")
        }
    }
}
```



Clases de dominio de JSecurity

- **Carga de datos**

```
libroControl(controller: "libro",  
             action: "(create|edit|save|update|delete|show|list)") {  
    before = {  
        accessControl {  
            role("Administrador")  
        }  
    }  
}
```



Clases de dominio de JSecurity

- **Carga de datos**
 - Los administradores controlarán la gestión de los usuarios
 - Los profesores tendrán la posibilidad de realizar operaciones



Clases de dominio de JSecurity

- **Etiquetas GSP**

Atributo	Descripción
<i>isLoggedIn</i>	Comprueba si el usuario está identificado en el sistema
<i>isNotLoggedIn</i>	Comprueba si el usuario no está identificado en el sistema
<i>authenticated</i>	Etiqueta sinónima a <i>isLoggedIn</i>
<i>notAuthenticated</i>	Etiqueta sinónima a <i>isNotLoggedIn</i>
<i>user</i>	Sólo se muestra el contenido si el usuario es reconocido con la utilización de la opción <i>Remember me</i>



Clases de dominio de JSecurity

- **Etiquetas GSP**

Atributo	Descripción
<i>principal</i>	Muestra el nombre de usuario del usuario registrado
<i>hasRole</i>	Sólo se muestra el contenido si el usuario está autenticado y además tiene un rol asignado
<i>lacksRole</i>	El método inverso a <i>hasRole</i>
<i>hasPermission</i>	Sólo se muestra el contenido si el usuario está autenticado y además tiene permisos asignados
<i>lacksPermission</i>	El método inverso a <i>hasPermission</i>



Clases de dominio de JSecurity

- **Etiquetas GSP**

- Modificamos el contenido de la plantilla `_header.gsp`

```
<div id="menu"><nobr>
  <jsec:isLoggedIn><div>
    <jsec:principal/>(<g:link controller="auth"
      action="signOut"><g:message code="encabezado.logout"/></g:link>)
  </div> </jsec:isLoggedIn>
  <jsec:isNotLoggedIn> <div>
    <g:link controller="auth" action="login"><g:message
code="encabezado.login"/></g:link>)
  </div> </jsec:isNotLoggedIn>
</nobr> </div>
```