



**GRAILS**

# Groovy & Grails: Desarrollo rápido de aplicaciones

Sesión 8: Construir la interfaz de usuario (II)



# Construir la interfaz de usuario (II)

- Tests
- Validación y errores
- Externalización de cadenas de caracteres



# Tests

- Tests de integración
- Tests funcionales



# Tests

- Grails utiliza dos conocidos frameworks de tests que son JUnit y Canoo
- JUnit se utiliza para los tests de integración
- Canoo se utiliza para realizar tests funcionales



# Tests de integración

- Implementaremos tests de integración para comprobar los métodos *handleLogin()* y *logout()*

```
grails create-integration-test Usuario
```

- Se creará un archivo en el directorio *test/integration* llamado *UsuarioTests.groovy*



# Tests de integración

```
import grails.test.*
class UsuarioTests extends GrailsUnitTestCase {
    protected void setUp() {
        super.setUp()
    }
    protected void tearDown() {
        super.tearDown()
    }
    void testSomething() {
    }
}
```



# Tests de integración

- El método *setUp()* se ejecuta antes que cualquier test de la clase en cuestión
- El método *tearDown()* será lo último que se ejecute
- El método *testSomething()* es un método de ejemplo



# Tests de integración

- Crearemos un método para comprobar el funcionamiento del método *handleLogin()*
- Implementaremos un test positivo con un usuario correcto
- Y un test negativo con un usuario incorrecto



# Tests de integración

- Los métodos deben comenzar por la palabra *test*
- El primer método que crearemos será *testHandleLogin()*



# Tests de integración

```
Usuario u
UsuarioController uc
protected void setUp() {
    //Creo el usuario
    u = new Usuario(login:'frangarcia2',
        password:'mipassword', nombre:'Francisco José',
        apellidos:'García Rico', tipo:'administrador', email:'fgarcia@ua.es')
    u.save()
    //Inicializo el controlador
    uc = new UsuarioController()
}
protected void tearDown() {
    u.delete()
}
```



# Tests de integración

```
void testHandleLogin() {
    // Establece los parámetros del usuario
    uc.params.login = u.login
    // Invoca la acción
    uc.handleLogin()
    // Si la acción ha funcionado correctamente, la variable
    // session tendrá los datos del usuario
    def sessUsuario = uc.session.usuario
    //Compruebo que la sesión del usuario se ha creado bien y que no es null
    assert sessUsuario
    assertEquals(u.login, sessUsuario.login)
    // Y el usuario se redirige a la página para realizar operaciones
    assertEquals "/operacion", uc.response.redirectedUrl
}
```



# Tests de integración

- Comprobamos que la aplicación pasa la batería de tests

```
grails test-app Usuario
```

- Se generará un informe en el directorio *test/reports*
- Creamos también el método *testHandleLoginInvalidUser()*



# Tests de integración

```
void testHandleLoginInvalidUser() {
    // Establece los parámetros del usuario
    uc.params.login = "loginincorrecto"
    // Invoca la acción
    uc.handleLogin()
    //Compruebo que la acción ha redireccionado de nuevo a la
    //página de login
    assertEquals "/usuario/login", uc.response.redirectedUrl
    //Compruebo el mensaje flash devuelto por el controlador
    def message = uc.flash.message
    assert message
    assert message.startsWith("El usuario ${uc.params.login} no existe")
}
```



# Tests de integración

- Comprobamos que la aplicación pasa la batería de tests

```
grails test-app Usuario
```

- Creamos también el método *testLogout()*



# Tests de integración

```
void testLogout() {  
    // Simulamos que el usuario ya se ha identificado en el sistema  
    //copiando sus datos en la variable session  
    uc.session.usuario = u  
    // Abandonamos la aplicación  
    uc.logout()  
    def sessUsuario = uc.session.usuario  
    //Comprueba que la variable sessUsuario es null  
    assertNull("Expected session user to be null", sessUsuario)  
    //Comprueba que el usuario es redirigido a la página usuario/login  
    assertEquals "/usuario/login", uc.response.redirectedUrl  
}
```



# Tests de integración

grails test-app Usuario

Unit Test Results

Designed for use with [JUnit](#) and [Ant](#).

### Summary

Tests	Failures	Errors	Success rate	Time
<a href="#">3</a>	<a href="#">0</a>	<a href="#">0</a>	100.00%	1.154

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

### Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
<a href="#">&lt;none&gt;</a>	3	0	0	1.154	2009-05-26T08:09:55	macbook-pro-de-francisco-jose-garcia-rico.local



# Tests de integración

- Es conveniente desarrollar tests de integración al mismo tiempo que se desarrolla la aplicación
- Se ahorrará tiempo y esfuerzo en detectar y solucionar futuros problemas



# Tests funcionales

- Pretenden comprobar la funcionalidad de la aplicación
- Se imitan las acciones que pueden realizar los usuarios interactuando con la aplicación
- Se utiliza Canoo, un popular framework de código abierto



# Tests funcionales

- Se instala como un plugin externo

```
grails install-plugin webtest
```

- Comprobaremos que la aplicación lista, crea y elimina usuarios correctamente
- Creamos un nuevo test funcional

```
grails create-webtest Usuario
```



# Tests funcionales

- Se creará un nuevo directorio llamado *webtest* que a su vez contendrá tres subdirectorios:
  - conf
  - reports
  - tests



# Tests funcionales

- El archivo *webtest/conf/webtest.properties*

Nombre	Valor inicial
wt.config.host	localhost
wt.config.port	8080
wt.config.protocol	http
wt.config.summary	true
wt.config.saveresponse	true
wt.config.resultpath	webtest/reports
wt.config.resultfile	WebTestOverview.xml
wt.config.haltonerror	False



# Tests funcionales

Nombre	Valor inicial
wt.config.errorproperty	webTestError
wt.config.haltonfailure	false
wt.config.failureproperty	webTestFailure
wt.config.showhtmlparseroutput	true



# Tests funcionales

- El archivo *webtest/tests/UsuarioTest.groovy* generado automáticamente realiza una serie de comprobaciones sobre la aplicación
- Comprueba que listar, crear y eliminar usuarios funciona correctamente
- Debemos hacer algunos cambios sobre el archivo original



# Tests funcionales

- Los test funcionales simulan hacer clic sobre enlaces (*clickLink*) o botones (*clickButton*)
- Comprobar que un texto aparece en la página con *verifyText*
- Comprobar el contenido encerrado entre etiquetas del código HTML con *verifyXPath*



# Tests funcionales

```
grails run-webtest
```

- Repasando los informes, comprobamos que el test funcional original falla en determinados puntos que debemos solucionar
- El informe nos indica cuantos tests se han ejecutado correctamente y los pasos dentro de estos tests que se han pasado sin problemas



## Tests funcionales

- El método que ha dado problemas es *verifyListSize()*
- Los tests funcionales cuentan con la información añadida en el archivo *conf/BootStrap.groovy* con lo que la aplicación empieza con 5 usuarios
- Debemos modificar las líneas donde aparezca *verifyListSize* para sumarle esos 5 usuarios



# Tests funcionales

- Volvemos a ejecutar los tests funcionales con *grails run-webtest*
- El siguiente problema con el que nos encontramos es que el usuario no está identificado en el sistema
- Debemos añadir el código necesario para identificarnos en el sistema



# Tests funcionales

- Justo después de *verifyText 'Home'*

```
group(description:'intento identificarme en el sistema') {  
    showFirstElementDetails()  
    clickLink 'Login'  
    verifyText 'Login'  
    clickButton 'Login'  
    verifyText 'Logout'  
    clickLink 'Home', description:'Back home'  
    verifyText 'UsuarioController'  
    clickLink 'UsuarioController'  
}
```



# Tests funcionales

- Volvemos a ejecutar los tests funcionales con *grails run-webtest*
- El siguiente problema con el que nos encontramos es que hemos dejado campos en blanco
- Debemos rellenar estos campos antes de enviar la información



# Tests funcionales

- Justo después de *verifyText 'Create Usuario'*

```
setInputField(name:'login','usuario2')  
setInputField(name:'password','mipassword')  
setInputField(name:'nombre','Usuario')  
setInputField(name:'apellidos','Dos')
```



# Tests funcionales

- Hay que tener muy en cuenta que el orden en el que se ejecutan los tests funcionales es alfabético
- Se pueden numerar los tests al estilo de 0001, 0002, 0003, etc. para que los tests se ejecuten en el orden que nosotros queremos



## Validación y errores

- Hasta ahora hemos visto funciones como *hasErrors()* que sirven para detectar la presencia de errores en los controladores
- También hemos visto algunas características de los *mensajes flash*



## Validación y errores

- Si intentamos crear un usuario y no lo indicamos todos los datos obligatorios, el sistema nos indica que se han cometido una serie de errores

```
<g:hasErrors bean="{usuarioInstance}">
  <div class="errors">
    <g:renderErrors bean="{usuarioInstance}" as="list" />
  </div>
</g:hasErrors>
```



## Validación y errores

- Con la etiqueta `<g:hasErrors>` se comprueba si se han producido errores en el *bean* pasado
- En caso afirmativo, se imprimen todos los errores gracias a la etiqueta `<g:renderErrors>`
- Esta etiqueta permite el uso del parámetro *field* para imprimir sólo el error de un determinado campo

```
<g:renderErrors bean="\${book}" as="list" field="title"/>
```



## Validación y errores

- Cuando creamos un usuario correctamente, se muestra en la parte superior un texto indicativo
- Esto se consigue con los *mensajes flash* que se rellenan en los controladores y se muestran en la vista

```
def save = {
    def usuarioInstance = new Usuario(params)
    if(!usuarioInstance.hasErrors() && usuarioInstance.save()) {
        flash.message = "Usuario ${usuarioInstance.id} created"
        redirect(action:show,id:usuarioInstance.id)
    }
    else { render(view:'create',model:[usuarioInstance:usuarioInstance]) }
}
```



## Validación y errores

- El *mensaje flash* se muestra en la vista

```
<g:if test="${flash.message}">  
    <div class="message">${flash.message}</div>  
</g:if>
```

- Los *mensajes flash* se utilizan cuando se redirecciona a otras páginas
- Estos mensajes se borran en la siguiente petición, con lo que debemos hacerlo nosotros



# Externalización de cadenas de caracteres

- Hasta ahora todas las cadenas de texto se incluyen directamente en el código, tanto de los controladores como de las vistas
- Lo ideal es externalizar estas cadenas
- Al externalizar, podemos disponer de una misma aplicación en varios idiomas de forma rápida y sencilla



# Externalización de cadenas de caracteres

- En el directorio *grails-app/i18n/* se pueden gestionar los ficheros de traducción de cadenas de texto
- Existen ejemplos de ficheros creados para varios idiomas
- El archivo *message.properties* contiene las cadenas de texto que aparecen cuando se produce algún error en la aplicación



# Externalización de cadenas de caracteres

default.doesnt.match.message=Property [{0}] of class [{1}] with value [{2}] does not match the required pattern [{3}]

default.invalid.url.message=Property [{0}] of class [{1}] with value [{2}] is not a valid URL

default.invalid.creditCard.message=Property [{0}] of class [{1}] with value [{2}] is not a valid credit card number

default.invalid.email.message=Property [{0}] of class [{1}] with value [{2}] is not a valid e-mail address

default.invalid.range.message=Property [{0}] of class [{1}] with value [{2}] does not fall within the valid range from [{3}] to [{4}]



# Externalización de cadenas de caracteres

- Podemos añadir un par de entradas para los textos referentes a *Login* y *Logout*

```
encabezado.login = Login  
encabezado.logout = Logout
```



# Externalización de cadenas de caracteres

- Modificaremos también la plantilla `_header.gsp` para contemplar la nueva metodología
- Utilizaremos la etiqueta `<g:message>`



# Externalización de cadenas de caracteres

```
<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:else>
  </nobr>
</div>
```



# Externalización de cadenas de caracteres

- Grails intenta saber cual es el idioma de nuestro navegador por defecto, con lo que es probable que necesitemos modificar también el archivo *message\_es.properties*

```
encabezado.login = Identificarse  
encabezado.logout = Salir
```



# Externalización de cadenas de caracteres

- Las cadenas de texto pueden especificar parámetros

```
usuario.updated.message = El usuario {0} {1} ha sido modificado correctamente
```

- Los valores `{0}` y `{1}` serán sustituidos por el nombre y apellidos del usuario
- Debemos modificar el fragmento de código del método `update()` que indica que el usuario se ha creado correctamente



# Externalización de cadenas de caracteres

```
if(!usuarioInstance.hasErrors() && usuarioInstance.save()) {  
    flash.message = "usuario.updated.message"  
    flash.args = [usuarioInstance.nombre, usuarioInstance.apellidos]  
    flash.defaultMsg = "Usuario modificado correctamente"  
    redirect(action:show,id:usuarioInstance.id)  
}
```



# Externalización de cadenas de caracteres

- Debemos también modificar la vista *show.gsp*

```
<g:if test="{flash.message}">
    <div class="message">{flash.message}</div>
</g:if>
```

por

```
<div class="message">
    <g:message code="{flash.message}" args="{flash.args}"
default="{flash.defaultMsg}"/>
</div>
```



# Externalización de cadenas de caracteres

## Show Usuario

 El usuario Pablo Mar Mol ha sido modificado correctamente

Id:	2
Login:	pablomarmol
Password:	marmol
Nombre:	Pablo
Apellidos:	Mar Mol
Tipo:	bibliotecario
Operaciones:	



**Edit**



**Delete**