



GRAILS

Groovy & Grails: Desarrollo rápido de aplicaciones

Sesión 13: Web 2.0



Web 2.0

- Texto enriquecido
- Tooltips
- Plugin de búsqueda de contenido
- Carga de archivos
- Exportar a varios formatos
- Crear fuentes RSS
- Librería de etiquetas



Texto enriquecido

- Podemos permitir a los usuarios introducir texto con formato
- Varios editores disponibles: *FCK Editor*, *TinyMCE* o un editor tipo *Wiki*
- Utilizaremos *FCK Editor*



Texto enriquecido

- Necesitamos de un plugin llamado *RichUI*
- Instalamos el plugin

```
grails install-plugin richui
```

- Este plugin dispone de varios componentes. Donde vayamos a utilizar un componente, debemos escribir

```
<resource: nombreComponente/>
```



Texto enriquecido

- El componente necesario para el editor de texto enriquecido

```
<resource: richTextEditor/>
```

- La etiqueta que implementa el editor de texto enriquecido es

```
<richui: richTextEditor/>
```



Texto enriquecido

- La etiqueta `<richui:richTextEditor/>` soporta los siguientes atributos:
 - *name*, nombre del elemento del formulario
 - *id*, identificador HTML del elemento del formulario
 - *value*, valor del elemento del formulario
 - *height*, altura del editor
 - *width*, anchura del editor



Texto enriquecido

- En la etiqueta `<resource:richTextEditor/>` podemos especificar el tipo de editor que deseamos mediante el atributo *type*:
 - *simple*
 - *medium*
 - *advanced*
 - *full*



Texto enriquecido

- ***Editor simple***

Lorem **ipsum** dolor sit amet, consetetur sadipscing elit, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd *gubergren*, no sea takimata sanctus est Lorem ipsum dolor sit amet.

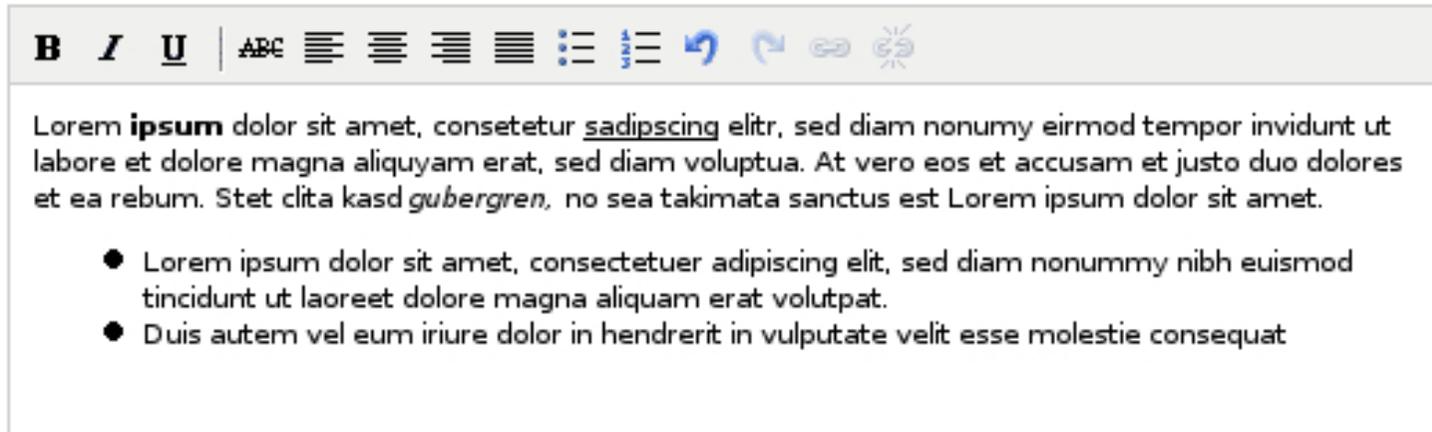
- Lorem ipsum dolor sit amet, consetetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.
- Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat





Texto enriquecido

- ***Editor medium***



The screenshot shows the Editor medium interface. At the top is a toolbar with icons for bold (B), italic (I), underline (U), text color (ABC), background color (three horizontal lines), text color (three horizontal lines), text background color (three horizontal lines), bulleted list (three vertical lines), numbered list (three vertical lines), undo (curved arrow), redo (curved arrow), link (chain), and unlink (chain with slash). Below the toolbar is a text area containing the following content:

Lorem **ipsum** dolor sit amet, consetetur sadipscing elit, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd *gubergren*, no sea takimata sanctus est Lorem ipsum dolor sit amet.

- Lorem ipsum dolor sit amet, consetetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.
- Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat



Texto enriquecido

- ***Editor advanced***

The screenshot shows the 'Editor advanced' toolbar and content area. The toolbar includes buttons for Bold (B), Italic (I), Underline (U), ABC, bulleted list, numbered list, and indent. It also features dropdown menus for Styles, Format, Font family, and Font size, along with icons for bulleted list, numbered list, undo, redo, link, and unlink.

Below the toolbar, the content area displays the following text:

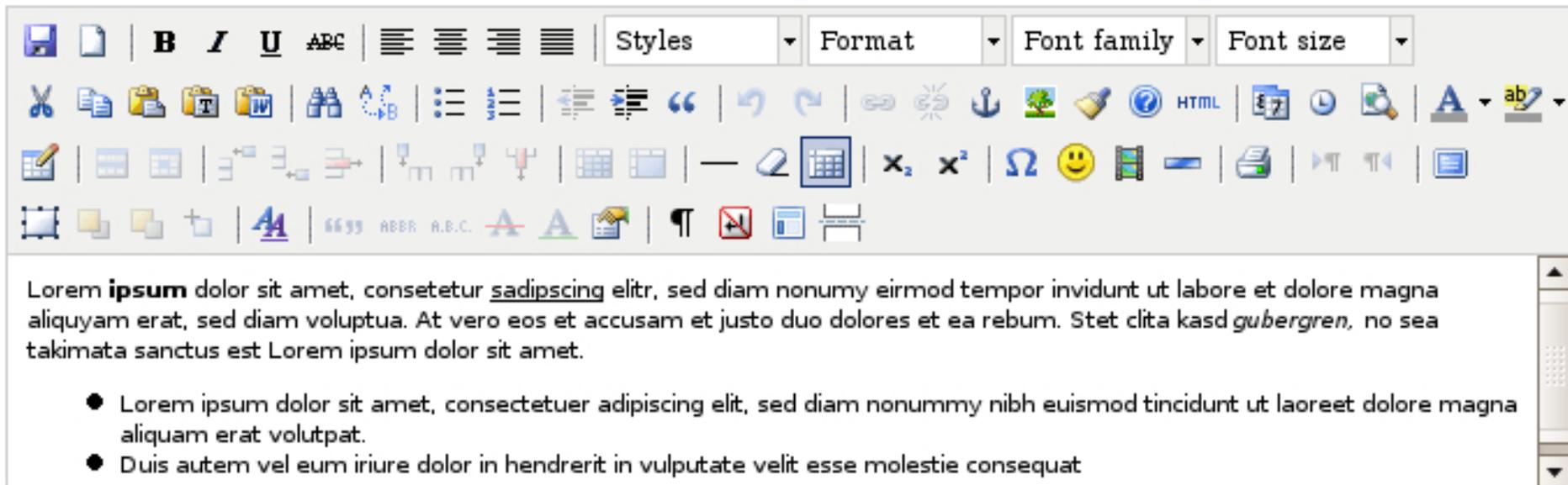
Lorem **ipsum** dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd *gubergren*, no sea takimata sanctus est Lorem ipsum dolor sit amet.

- Lorem ipsum dolor sit amet, consetetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.
- Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat



Texto enriquecido

- ***Editor full***





Texto enriquecido

- Introduciremos un editor de texto enriquecido en la descripción de los libros
- Debemos editar los archivos *create.gsp* y *edit.gsp*
- El elemento de formulario de la *descripción* del archivo *edit.gsp* quedaría así



Texto enriquecido

```
<tr class="prop">
  <td valign="top" class="name">
    <label for="descripcion">Descripcion:</label>
  </td>
  <td valign="top" class="value $
{hasErrors(bean:libroInstance,field:'descripcion','errors')}}">
    <richui:richTextEditor name="descripcion" value="$
{fieldValue(bean:libroInstance, field:'descripcion').decodeHTML()}"
width="525" />
  </td>
</tr>
```



Texto enriquecido

- Debemos utilizar la función *decodeHTML()* para que no se conviertan los caracteres `<` y `>` por *<* y *>*;



Texto enriquecido

Edit Libro

Isbn:	<input type="text" value="0844273619"/>
Título:	<input type="text" value="El ingenioso hidalgo don C"/>
Autor:	<input type="text" value="Miguel de Cervantes Saav"/>
Editorial:	<input type="text" value="Anaya"/>
Año:	<input type="text" value="1605"/>
Fecha:	<input type="text" value="21"/> <input type="text" value="junio"/> <input type="text" value="2009"/> <input type="text" value="12"/> : <input type="text" value="26"/>
Descripción:	<div><p>B I U ABC     </p><p>Don Quijote de la Mancha es una novela escrita por el español <i>Miguel de Cervantes Saavedra</i>. Publicada su primera parte con el título de <i>El Ingenioso hidalgo don Quijote de la Mancha</i> a comienzos de 1605, es una de las obras más destacadas de la literatura española y la literatura universal, y una de las más traducidas. En <u>1615</u> aparecería la segunda parte del Quijote de Cervantes con el título de <i>El ingenioso caballero don Quijote de la Mancha</i>.</p></div>
Operaciones:	<ul style="list-style-type: none">● prestamo (true) [2009-06-22 12:26:29.614 - 2009-06-23 12:26:29.614] Add Operacion



Texto enriquecido

- En el archivo *show.gsp* debemos utilizar también la función *decodeHTML()* para evitar que la salida se muestre correctamente

```
<tr class="prop">
  <td valign="top" class="name">Descripcion:</td>
  <td valign="top" class="value">
    ${fieldValue(bean:libroInstance, field:'descripcion').decodeHTML()}
  </td>
</tr>
```



Texto enriquecido

- El plugin instalado *RichUI* tiene además otras funcionalidades interesantes
- <http://www.grails.org/RichUI+Plugin>



Tooltips

- Otra de las funcionalidades del plugin *RichUI* es la posibilidad de añadir *tooltips* en nuestras páginas
- Los *tooltips* son pequeños textos de ayuda que aparecen sobre las imágenes o los enlaces
- Para utilizar este plugin, debemos añadir en las páginas

```
<resource: tooltip/>
```



Tooltips

- Vamos a indicar una ayuda al usuario que quiera registrarse en la aplicación
- Le indicaremos mediante un *tooltip* el número mínimo de caracteres que debe tener la contraseña

```
<span id="ayudaPassword" title="La contraseña debe tener al menos 6 caracteres">  
    Ayuda  
</span>  
<richui:tooltip id="ayudaPassword" />
```



Plugin de búsqueda de contenido

- Con este plugin vamos a poder buscar información en nuestras clases de dominio
- Sería interesante que los usuarios pudieran disponer de un buscador de libros
- Grails cuenta con un plugin llamada *searchable*, que está basado en *OpenSymphony Compass Search Engine*



Plugin de búsqueda de contenido

- Instalamos el plugin *Searchable*

```
grails install-plugin searchable
```

- <http://localhost:8080/biblioteca/searchable>
- Debemos indicarle donde debe buscar la información
- En las clases de dominio se debe añadir una propiedad llamada *searchable*



Plugin de búsqueda de contenido

```
class Libro {  
    String isbn  
    String titulo  
    String autor  
    String editorial  
    Integer anyo  
    String descripcion  
    Date fecha  
  
    static searchable = true  
  
    .....  
}
```



Plugin de búsqueda de contenido

Grails Searchable Plugin

Search

Showing 1 - 1 of 1 results for **hidalgo**

Libro #3

El ingenioso hidalgo don Quijote de la Mancha
</biblioteca/libro/show/3>

Page: 1



Plugin de búsqueda de contenido

- El plugin instala un nuevo *controlador* y una *vista* que podemos modificar a nuestro gusto
- Al añadir la propiedad *searchable* en la clase de dominio, permitimos buscar en todas sus propiedades
- Podemos afinar un poco más la búsqueda



Plugin de búsqueda de contenido

```
class Libro {  
    String isbn  
    String titulo  
    String autor  
    String editorial  
    Integer anyo  
    String descripcion  
    Date fecha  
  
    static searchable = [only: ['titulo', 'autor', 'descripcion']]  
  
    .....  
}
```



Plugin de búsqueda de contenido

```
class Libro {  
    String isbn  
    String titulo  
    String autor  
    String editorial  
    Integer anyo  
    String descripcion  
    Date fecha  
  
    static searchable = [except: 'isbn']  
  
    .....  
}
```



Carga de archivos

- La carga de archivos permitirá a los usuarios subir archivos utilizando la aplicación
- Vamos a permitir a los usuarios subir imágenes en su perfil
- Debemos crear una propiedad de tipo *byte[]* en la clase de dominio correspondiente



Carga de archivos

```
class Usuario {  
    String login  
    String password  
    String nombre  
    String apellidos  
    String tipo  
    byte[] imagen  
    String nombreImagen  
    String contentTypeImagen .  
  
    .....  
}
```



Carga de archivos

- Modificamos el archivo *edit.gsp* de los usuarios

```
<g:form method="post" enctype="multipart/form-data">
.....
<tr class="prop"><td valign="top" class="name">
    <label for="tipo">Imagen:</label>
  </td><td valign="top" class="value $
{hasErrors(bean:usuarioInstance,field:'imagen','errors')}">
    <input type="file" name="imagen"/>
  </td>
</tr>
.....
</g:form>
```



Carga de archivos

- Modificamos también el método *update()* del controlador de la clase *Usuario* para almacenar la información necesaria

```
def update = {  
    ....  
    params.contentTypeImagen = params.imagen.contentType  
    params.nombreImagen = params.imagen.originalFilename  
    usuarioInstance.properties = params  
    if(!usuarioInstance.hasErrors() && usuarioInstance.save()) {  
        ....  
    }  
}
```



Carga de archivos

- Para mostrar las imágenes subidas, creamos un nuevo método llamado *showImagen* en el controlador de la clase *Usuario*

```
def showImagen = {
    def usuarioInstance = Usuario.get( params.id )
    response.setHeader("Content-disposition", "inline; filename='${
usuarioInstance.nombreImagen}')"
    response.contentType="${usuarioInstance.contentTypeImagen}"
    response.outputStream << usuarioInstance.imagen
}
```



Carga de archivos

- `http://localhost:8080/biblioteca/usuario/showImagen/1`
- Con *Content-disposition* con el valor *attachment*, el navegador intentaría descargar la imagen en lugar de mostrarla directamente
- Modificamos el archivo *edit.gsp* para mostrar la imagen en caso de que exista



Carga de archivos

```
<tr class="prop">
  <td valign="top" class="name">
    <label for="tipo">Imagen:</label>
  </td>
  <td valign="top" class="value $
{hasErrors(bean:usuarioInstance,field:'imagen','errors')}">
    <input type="file" name="imagen"/>
    <g:if test="{usuarioInstance?.contentTypeImagen != ''}">
      
    </g:if>
  </td>
</tr>
```



Exportar a varios formatos

- La aplicación permitirá exportar nuestras páginas a otros formatos
- Los formatos más habituales serán *pdf*, *hojas de cálculo excel*, *csv* u *ods*
- Debemos instalar el archivo *export*

```
grails install-plugin export
```



Exportar a varios formatos

- Añadimos algunos *mime types* en la variable *grails.mime.types* del archivo *Config.groovy*

```
grails.mime.types = [  
    ....  
    csv: 'text/csv',  
    pdf: 'application/pdf',  
    excel: 'application/vnd.ms-excel',  
    ods: 'application/vnd.oasis.opendocument.spreadsheet',  
    csv: 'text/csv',  
    ....  
]
```



Exportar a varios formatos

- Debemos incluir la etiqueta `<export:resource/>` en las páginas que queramos incluir las opciones de exportar
- Posteriormente debemos incluir la etiqueta `<export:formats/>` para que muestre la barra con las diversas opciones
- Esta etiqueta acepta el parámetro *formats* para que le indiquemos que formatos deseamos permitir



Exportar a varios formatos

- Añadimos la posibilidad de exportar el listado de los libros
- Editamos el archivo *grails-app/views/libro/list.gsp*



Exportar a varios formatos

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    <meta name="layout" content="main" />
    <g:javascript library="scriptaculous"/>
    <export:resource />
    <title>Libro List</title>
  </head>
  ....
  <g:if test="{flash.message}"><div class="message">{flash.message}</div>
</g:if>
  <export:formats formats="['csv', 'excel', 'ods', 'pdf', 'xml']" />
  <div class="list">
    ....
</html>
```



Exportar a varios formatos

Libro List

CSV EXCEL ODS PDF XML					
Id	Isbn	Título	Autor	Editorial	Año
1	843992688X	La colmena	Camilo José Cela Trulock	Anaya	1951
2	0936388110	La galatea	Miguel de Cervantes Saavedra	Anaya	1585
3	0844273619	El ingenioso hidalgo don Quijote de la Mancha	Miguel de Cervantes Saavedra	Anaya	1605
4	847039360X	La dorotea	Félix Lope de Vega y Carpio	Anaya	1632
5	8437624045	La dragontea	Félix Lope de Vega y Carpio	Anaya	1602



Exportar a varios formatos

- Debemos modificar también el método *list()* del controlador de la clase *Libro* para que acepte los nuevos formatos



Exportar a varios formatos

```
import org.codehaus.groovy.grails.commons.ConfigurationHolder
class LibroController {
    // Servicio para exportar a varios formatos ofrecidos por el plugin Export
    def exportService
    ....
    def list = {
        params.max = Math.min( params.max ?
        params.max.toInteger() : 10, 100)
        if(params?.format && params.format != "html"){
            response.contentType =
ConfigurationHolder.config.grails.mime.types[params.format]
            response.setHeader("Content-disposition",
"attachment; filename=libros.${params.format}")
            exportService.export(params.format,
response.outputStream, Libro.list(params), [:], [:])
        } ....
    }
```



Exportar a varios formatos

- En los archivos exportados aparecen todas las propiedades de la clase de dominio a excepción de *version*
- Es aconsejable controlar que propiedades queremos mostrar al usuario final



Exportar a varios formatos

```
def list = {  
    ....  
    response.setHeader("Content-disposition", "attachment;  
filename=libros.${params.format}")  
    List propiedades = ["isbn", "titulo", "autor", "editorial",  
"anyo", "descripcion"]  
    Map etiquetas = ["isbn":"ISBN", "titulo":"Título",  
"autor":"Autor", "editorial":"Editorial", "anyo":"Año",  
"descripcion":"Descripción"]  
    // Closure formateador  
    def mayusculas = { value -> return value.toUpperCase() }  
    Map formateadores = [isbn: mayusculas]  
    Map parametros = [title: "LISTADO DE LIBROS"]  
    exportService.export(params.format, response.outputStream,  
Libro.list(params), propiedades, etiquetas, formateadores, parametros)  
}
```



Exportar a varios formatos

- Podemos modificar los textos asociados a cada uno de los formatos que aparecen en la barra de exportar
- Debemos crear nuevas entradas en el archivo *message.properties*

```
default.csv = CSV  
default.excel = EXCEL  
default.pdf = PDF  
default.xml = XML  
default.ods = ODS
```



Crear fuentes RSS

- Mediante el formato *RSS* vamos a poder intercambiar información en nuestra aplicación
- Significan *Really Simple Syndication*
- Simplemente es un archivo XML



Crear fuentes RSS

- Vamos a crear una fuente RSS para informar a nuestros usuarios de los nuevos libros adquiridos por la biblioteca
- Instalamos un plugin llamado *feeds*

```
grails install-plugin feeds
```



Crear fuentes RSS

- Debemos crear un nuevo método llamado *feed()* en el controlador de la clase *Libro*
- El método realizará una llamada a la función *render()* indicándole el formato del archivo RSS generado
- Con un closure construiremos el contenido de la fuente RSS



Crear fuentes RSS

```
def feed = {
  render(feedType:"rss", feedVersion:"2.0") {
    title = "Los nuevos libros"
    link = "http://localhost:8080/biblioteca/libro/feed"
    description = "Fuente RSS de los nuevos libros adquiridos por la biblioteca"
    Libro.list().each() {
      libro -> entry(libro.titulo) {
        link = "http://localhost:8080/biblioteca/libro/show/${libro.id}"
        author = libro.autor
        publishedDate = libro.fecha
        libro.descripcion
      }
    }
  }
}
```



Crear fuentes RSS

- El parámetro *feedType* es obligatorio y puede tomar los valores *rss* o *atom*
- El parámetro *feedVersion* es opcional y si no se especifica toma los valores *2.0* para *rss* y *1.0* para *atom*
- <http://localhost:8080/biblioteca/libro/feed>



Librería de etiquetas

- Etiquetas simples
- Etiquetas lógicas
- Etiquetas iteradoras
- Generador de código HTML



Librería de etiquetas

- Grails permite utilizar tanto etiquetas *JSP* como *GSP*
- También podemos crear nuestras propias librerías de etiquetas
- Las librerías permiten la realización de tareas repetitivas de forma rápida y sencilla



Librería de etiquetas

- Podemos crear librerías de etiquetas mediante un comando

```
grails create-tag-lib
```

- O bien creando una nueva clase en el directorio *grails-app/taglib* cuyo nombre termine en *TagLib*



Etiquetas simples

- Vamos a crear una etiqueta que permita la inclusión de archivos de funciones *javascript*
- Creamos la librería de etiquetas llamada *Biblioteca TagLig* con la siguiente función

```
def includeJs = {  
    attrs -> out << "<script src='scripts/${attrs['script']}.js' />"  
}
```



Etiquetas simples

- La etiqueta acepta sólo un parámetro que indica el archivo *javascript* a importar
- Ahora, donde necesitemos incluir un archivo de funciones javascript, podemos escribir

```
<g:includeJs script="miscscript"/>
```



Etiquetas simples

- Se utiliza el espacio de nombres genérico `<g>`
- Podemos crear nuestro propio espacio de nombres definiendo la variable *namespace*

```
class BibliotecaTagLib {  
    static namespace = 'me'  
    def includeJs = {  
        attrs -> out << "<script src='scripts/${attrs['script']}.js'>"  
    }  
}
```

- Podemos utilizar la etiqueta

```
<me:includeJs script="miscrpt"/>
```



Etiquetas lógicas

- Podemos crear etiquetas que evalúen una cierta condición
- Creamos una etiqueta que compruebe si el usuario autenticado es un administrador

```
def esAdmin = {  
    attrs, body ->  
    def usuario = attrs['usuario']  
    if(usuario != null && usuario.tipo=="administrador") {  
        out << body()  
    }  
}
```



Etiquetas lógicas

- La etiqueta no sólo el atributo *attrs* sino que también necesita del atributo *body*
- El atributo *body* se refiere a todo lo que esté encerrado entre la apertura y el cierre de la etiqueta creada
- En la página GSP correspondiente

```
<me:esAdmin usuario="\${session.usuario}">  
    //Acciones restringidas a los administradores  
</me:esAdmin>
```



Etiquetas iteradoras

- Las etiquetas iteradoras nos facilitarán las tareas repetitivas
- Crearemos una etiqueta que imprima un párrafo un determinado número de veces pasado por parámetro



Etiquetas iteradoras

- Creamos un número método en la librería de etiquetas

```
def repetir = { attrs, body ->
    def i = Integer.valueOf( attrs["times"] )
    def actual = 0
    i.times {
        out << body( ++actual )
    }
}
```



Etiquetas iteradoras

- Podemos incluso pasarle parámetros al *body* con la variable *it*

```
<me:repetir times="4">  
    Este sería el párrafo número ${it}<br/>  
</me:repetir>
```



Generador de código HTML

- Gracias al builder *MarkupBuilder* vamos a poder generar código HTML sencillamente
- Vamos a crear una nueva etiqueta que imprima un enlace cuyo atributo *href* coincida con el título
- `http://www.google.com`



Generador de código HTML

- Creamos el método *printLink()* en la librería de etiquetas

```
def printLink = { attrs, body ->
    def mkp = new groovy.xml.MarkupBuilder(out)
    mkp {
        a(href:body(),body())
    }
}
```



Generador de código HTML

- Ahora en la vista podemos tener

```
<me:printLink>http://www.google.com</me>
```

- Que será sustituido por

```
<a href="http://www.google.com">http://www.google.com</a>
```