CREACIÓN DE UN PORTAL EN OPENCMS 6

Hay muchas formas de crear portales en opencms, las combinaciones son muchas y en Internet podemos encontrar multitud de ellas pero nos vamos a centrar en una forma bastante simple para crear nuestro mini portal y dejaremos el resto de formas aparcadas.
Vamos a explicar los conceptos mas básicos de las jsp y taglibs de opencms para ello vamos a seguir una serie de pasos básicos. Siguiéndolos poco a poco iremos aprendiendo a crear un portal sin mucha dificultad, además de servir este documento como referencia básica.

1º - Nos situamos en /sites/default/ y creamos un recurso de tipo extended folder/microsite. Le especificamos el nombre de la carpeta y el titulo real de nuestro sitio web y finalizamos. Con esto ya tenemos nuestra carpeta de navegación de nuestro sitio. Si la abrimos observamos que hay varias subcarpetas que crea por defecto e incluso algunos archivos de configuraciones, realmente estas carpetas no son necesarias o incluso se pueden renombrar como queramos. Simplemente las inserta como ejemplo.

2º - Ahora necesitamos un template principal para nuestro sitio, que sera el que se cargue cuando abramos nuestro portal. Para crear nuestros templates necesitaremos antes crear un modulo para almacenarlo y tenerlo organizado. De esta forma separamos lo que es la programación de lo que son la parte de los datos del portal. Para ello nos vamos a la zona de administración de opencms al gestor de módulos y pinchamos sobre nuevo. Esto nos abrirá un formulario para que creemos nuestro modulo principal, ahí podemos especificar los datos del modulo pero de momento para nuestro ejemplo basta con que pongamos org.misitioweb

en el nombre del paquete, principal en nombre del modulo, y en la parte baja señalemos las opciones Create templates subfolder,Create elements subfolder y Create resources subfolder, así nos creara las carpetas que necesitamos para meter nuestros templates.

3º - Volvemos a nuestra vista de explorador para comprobar que todo se ha creado en la carpeta /system/modules/org.misitioweb/... en azul ya que no hemos publicado nada.

4º - Ahora en la carpeta /system/modules/org.misitioweb/resources/ podemos ir agregando los recursos que vayamos a usar (hojas de estilos, imagenes, etc...). Por ejemplo vamos a agregar todos los recursos de la carpeta resources de ejemplo en nuestra carpeta resources recién creada. Para ello nos situamos en la carpeta y pinchamos sobre nuevo/upload new file, señalamos todos y los subimos. Esta opción sube los archivos a opencms vía http, así que puede tardar un poco dependiendo del tamaño y la conexión.

5º - Una vez hecho esto ya podemos empezar a crear nuestros templates jsp en la carpeta /system/modules/org.misitioweb/templates/ y la carpeta que nos queda /system/modules/org.misitioweb/elements/ la dejaremos para guardar los trozos de templates que se repitan muchas veces como por ejemplo la cabecera, menús o cualquier otro elemento que tenga nuestro portal. Para comenzar a crear nuestro template debemos darle a nuevo y crear una jsp a la que llamaremos principal.jsp por ejemplo. El hecho de crear esta jsp en esta carpeta templates hace que opencms guarde la dirección de la misma como un nuevo template al cual nombrara con la propiedad title que le pongamos.

6º - Las jsp no son mas que trozos de código java suelto (sin clase) que es ejecutado directamente cuando se llama. Aunque puedan contener clases dentro de una jsp no es recomendable ya que puede liar mucho el código. Si necesitamos crear una clase es conveniente crearla fuera de opencms y luego importar el .jar creado (estos se pueden meter en la carpeta lib dentro de opencms en el disco duro del servidor). Las cabeceras mas usadas en jsp de opencms son las siguientes:

```
<%@page buffer="none" session="false" import="java.util.*, org.opencms.jsp.*" %>
<%@ taglib prefix="cms" uri="http://www.opencms.org/taglib/cms" %>
```

la primera @page especifica que son parámetros de la pagina jsp, buffer="none" especifica que no se cachee la jsp, session="false" especifica que no guarde la sesión al usuario que la este viendo y import se usa igual que en java para importar clases que podrán ser usadas en la jsp, si queremos especificar mas de un paquete o clase podemos separarlos por comas. En cuanto a la @taglib nos sirve para especificar que conjunto de taglibs vamos a usar y con que prefijo (prefix) que las vamos a llamar, tan solo tenemos que especificarle la dirección donde se encuentra la especificación de las taglibs que usaremos con el parámetro uri.

7º - Hay muchas taglibs de opencms que podemos encontrar en la pagina de documentación que viene de ejemplo con opencms, algunas de ellas son:

```
<cms:editable />
```

que inserta en el código html los javascripts necesarios para que se puedan modificar los recursos en offline si esta logado como usuario de opencms.

```
<cms:include file="../elements/cabecera.jsp" />
```

esta taglib incluye otra jsp dentro de la que se esta ejecutando, digamos que cuando la jsp A encuentra el include se va a ejecutar la B y cuando B acaba, continua la

ejecución de A por donde se quedo. La ruta especificada en este caso es relativa al template principal ya que usa .. para salir de la carpeta templates y luego especifica la ruta del template incluido. Toda esta información y mucha más se encuentra en el portal de documentación de opencms que da opción a agregar en la instalación.

8º - Bueno ya que sabemos varias cosas de jsp, podemos crear nuestro primer y simple template, un consejo para hacerlo seria poner primero la estructura de nuestro pagina en html simplemente sin nada de jsp...

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
	<head>
		<meta http-equiv="content-type" content="text/html; charset=iso-8859-1"/>
		<title>Mi Sitio Web</title>
		<link rel="STYLESHEET" type="text/css" href="Mi hoja de estilos.css"/>
	</head>
	<body>
		<!-- aqui ira la cabecera -->
		<p>Hola soy un template</p>
	</body>
</html>
```

9º - Y ahora podemos empezar a meter nuestro código jsp con algunas taglibs...

```
<%@page buffer="none" session="false" import="java.util.*" %>
<%@ taglib prefix="cms" uri="http://www.opencms.org/taglib/cms" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
	<head>
		<meta http-equiv="content-type" content="text/html; charset=iso-8859-1"/>
		<title>Mi Sitio Web</title>
		<link rel="STYLESHEET" type="text/css" href="<cms:link>../resources/estilos.css</cms:link>"/>
		<cms:editable />
	</head>
	<body>
		<cms:include file="../elements/cabecera.jsp" />
```

```
            <p>Hola soy un template</p>
        </body>
</html>
```

Hemos agregado la taglib "editable" dentro de la etiqueta "head" para que nos agregue lo necesario para que los recursos sean editables, también hemos especificado donde se encuentra nuestra hoja de estilos, hemos usado "link" para que no use la ruta relativa al html sino la ruta relativa al template jsp, "link" se encarga de componer la ruta completa considerando la ruta de la jsp. Y por ultimo hemos usado la etiqueta include (en esta no se pone el link ya que esta etiqueta siempre considera la ruta relativa a la jsp), como ya dijimos anteriormente esta etiqueta ejecuta la jsp que indique y vuelve. No olvidemos definir la propiedad title de nuestra template ya que nos servirá en un futuro, pongamos "template principal de mi sitio web".

10º - Bueno pues ya tenemos casi nuestro primer template pero nos falta crear el elemento al que hemos llamado "../elements/cabecera.jsp" que tendrá nuestra imagen de cabecera de momento. Para ello nos situamos en esa carpeta y volvemos a crear otra jsp que se llame igual y editamos su código y escribimos lo siguiente:

```
<%@page buffer="none" session="false" import="java.util.*" %>
<%@ taglib prefix="cms" uri="http://www.opencms.org/taglib/cms" %>
<div id="cabecera">
</div>
```

En esta jsp solamente incluimos el div de la cabecera que llama a la imagen de cabecera y la muestra mediante css (esta forma de trabajar simplifica y separa el trabajo del programador, maquetador y editor de contenido, ya que el programador escribe solo lo justo de html y el diseñador se encarga de crear el diseño en css).

11º - Y ahora que tenemos nuestro template creado necesitaremos una pagina (page en opencms) que lo use. Para ello nos situamos en la carpeta principal de nuestro

microsite y creamos un recurso de tipo "page with free text" esto es una pagina con texto libre, al crearlo nos pregunta que template queremos aplicarle, si los pasos anteriores los hicimos correctamente en esa lista debe aparecer nuestro template "template principal de mi sitio web", la lista de abajo nos permite seleccionar el tipo de estructura que va a tener el xml de nuestra pagina podemos usar "1 column, 1 row" especificamos el nombre "inicio.html" y damos a siguiente. Aquí podemos seguir especificando propiedades de la pagina que luego más tarde podremos leer dentro del template para hacer determinadas cosas como poner un titulo o configurar ciertos datos del menú o cualquier cosa que se nos ocurra, en principio no vamos a necesitar ninguna más. Guardamos y salimos.

12º - A partir de ahora cada vez que pinchemos sobre esa pagina se nos abrirá un explorador con una web que usara el código de nuestro template y podremos ver el código html que hemos hecho. ¿Si el código que se ve es el que hicimos en la jsp, para que hemos creado una pagina html? ¿Por qué no hemos llamado directamente a la jsp? Pues vamos a explicarlo, en principio si llamamos directamente a la jsp debería de funcionar también pero no es lo que queremos. Queremos tener un template que nos valga para mostrar cualquier texto simple. ¿Donde especificamos el texto que vamos a mostrar? En la pagina, si nos situamos en la carpeta del microsite y pinchamos sobre "edit page" sobre el icono del recurso page, para entrar en el editor de textos de opencms, aquí escribimos nuestro texto y lo guardamos. El editor da muchas posibilidades para insertar imagenes, objetos y demás, pero de momento solo meteremos un texto simple. Una vez escrito el texto guardamos y salimos.

13º - Si pinchamos en Edit controlcode sobre la pagina podremos ver algo parecido a esto...

```
<?xml version="1.0" encoding="UTF-8"?>
<pages                                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.opencms.org/dtd/6.0/xmlpage.xsd">
  <page language="de">
    <element name="text1">
      <links/>
      <content/>
    </element>
  </page>
  <page language="en">
    <element name="text1">
      <links/>
      <content><![CDATA[hola soy la primera pagina]]></content>
    </element>
  </page>
</pages>
```

Si os fijáis en la estructura del xml ha guardado un "element" en dos idiomas "en" y "de", aunque lo que hemos escrito lo ha guardado en la zona "en" ya que teníamos configurado el editor en "english". Ahora nuestra pagina tiene un texto concreto que poder mostrar que no depende del template en ningún sentido. Si pinchamos de nuevo sobre la pagina podremos observar que no ha salido el texto en ninguna parte del html generado, esto ocurre porque en el template no hemos especificado donde debe aparecer dicho texto. Así que nos vamos a la carpeta de nuestro template principal y lo editamos. Y donde queramos incluir nuestro texto ponemos la siguiente taglib

```
<cms:include element="text1" />
```

este taglib escrito de esta forma cojera el element "text1" en el idioma en el que estemos, por defecto "en".

14º - Bueno ya que esto empieza a funcionar probablemente nos entren ganas de meter más paginas, para ello podéis ir creando varias paginas en el microsite por ejemplo productos.html, quienessomos.html, novedades.html y contacto.html y le especificamos que coja nuestro template principal y escribimos algún texto distinto para las cuatro paginas. Una vez guardadas las probamos para comprobar que el template funciona correctamente y

se muestra todo exactamente igual menos el texto de nuestras paginas.

15º - Esto esta muy bien pero de alguna forma tendremos que ir de una pagina a otra ¿verdad?, y poner todos los enlaces en el template a pelo no es muy buena idea, ya que tendremos que modificar el template cada vez que insertemos una pagina nueva. Así que será mejor que creemos un menú dinámico, para ello vamos a crear una lista de enlaces en una carpeta. Para crearla hacemos lo siguiente, primero nos creamos una carpeta en nuestro microsite que se llame "menuIzquierdo" dentro de la carpeta "/miSitioWeb/modules/" por ejemplo. La abrimos y creamos un recurso de tipo "Structured content/Linklist item" y lo llamamos "linklist_0001.html", este nombre es importante ya que todos los recursos que agreguemos tienen que seguir un patrón concreto para luego poder recorrerlos en este caso el patrón es "linklist_$number.html" donde $number es un numero cualquiera de 4 dígitos. Con que creemos uno nos vale, lo editamos y ponemos la url a una de nuestras paginas como por ejemplo "/miSitioWeb/inicio.html" o "/miSitioWeb/contacto.html". Perfecto ya tenemos nuestra lista de enlaces (aunque solo haya uno) solo nos queda que nuestro template nos muestre esta lista de forma dinámica.

16º - Para ello nos vamos a crear otro elements para el menú izquierdo, nos vamos a la carpeta del modulo elements y creamos menuizquierdo.jsp y lo editamos con esto:

```
<%@page buffer="none" session="false" import="java.util.*" %><%@ taglib prefix="cms" uri="http://www.opencms.org/taglib/cms" %>

<div id="bloqueIzq">

        <cms:contentload collector="allInFolderNavPos" param="/miSitioWeb/modules/menuIzquierdo/linklist_${number}.html|linklist|12" editable="true">

                <div class="cajaMenuIzq"><div class="tituloMenuIzq"><a href="<cms:contentshow element="Url" />" target="<cms:contentshow element="Target" />"><cms:contentshow element="Title" /></a></div></div>
```

```
        </cms:contentload>
</div>
```

La taglib contentload carga un archivo xml en memoria para extraer datos y escribirlos en el html de salida. También puede cargar todos o algunos archivos de una carpeta y hacer un bucle sobre ellos para ir recorriendo los recursos de uno en uno como es nuestro caso. En el portal de ayuda de opencms vienen todas las posibilidades que ofrece esta taglib pero nosotros vamos a ver las más usadas. El parámetro "collector" especifica la forma de recorrer la carpeta en este caso le estamos diciendo que coja todos los recursos de la carpeta ordenados por la propiedad NavPos (esta propiedad es un numero con decimales ej: 1.3) esta forma nos da mucho juego ya que nos permite situar cada elemento en la posición que queramos simplemente cambiándole la propiedad NavPos al recurso que queramos. Otros ejemplos son "allInFolderDateReleasedDesc" que lee todos los recursos de la carpetas ordenados por la fecha de creación. Y otro importante es "allInSubTree" que lee la carpeta y además los recursos de las subcarpetas, es ideal para hacer una navegación de descargas por ejemplo. En "param" especificamos la carpeta con el patrón de archivos que tiene que leer. Si solo queremos coger los datos de un solo xml podemos usar "singleFile" y especificar el archivo completo en el parametro "param". Como veis "linklist_${number}.html" es el patrón que ya comentamos antes, que podemos modificar según nos convenga, incluso usar patrones diferentes en una misma carpeta. Lo que sigue al patrón "|linklist|12" es el tipo de recurso que queremos que lea (tiene que especificarse correctamente sino no leerá nada) y el numero de elementos que queremos que recorra, en este caso recorrerá 12 aunque existieran más, o los que existieran en caso de haber menos de 12. Y por ultimo pero no por ello menos importante el parámetro "editable" que especifica si podemos editar nuestro recurso viendo el portal en offline o tenemos que irnos al opencms para editarlo. Si lo ponemos a true y dentro del head del html ponemos el

<cms:editable /> y vemos el portal desde dentro de opencms aparecerá al lado de lo que mostremos del recurso varios iconos, uno que parece una diana es para ir a la pantalla de modificar ese recurso, el icono más es para agregar un recurso nuevo siguiendo el patrón especificado en el contentload, y la X elimina el recurso de opencms.

17º - En este punto vamos a rellenar el template principal para que vaya usando todo lo que hemos estado haciendo hasta ahora para ello nos vamos al modulo, carpeta templates y editamos principal.jsp y lo editamos con lo siguiente:

```
<%@page buffer="none" session="false" import="java.util.*" %><%@ taglib prefix="cms"
uri="http://www.opencms.org/taglib/cms" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
	<head>
		<meta http-equiv="content-type" content="text/html; charset=iso-8859-
1"/>
		<title>Mi Sitio Web</title>
		<link rel="STYLESHEET" type="text/css"
href="<cms:link>../resources/estilos.css</cms:link>"/>
		<cms:editable />
	</head>
	<body>
		<div id="contenedor">
			<cms:include file="../elements/cabecera.jsp" />
			<div id="cuerpo">
				<cms:include file="../elements/menuizquierdo.jsp" />
				<div id="bloquedcha">
					<cms:include element="text1" />
				</div>
			</div>
		</div>
	</body>
</html>
```

18º - Nos paramos un poco en este punto para ver que hemos hecho, guardamos la jsp y nos vamos a la carpeta miSitioWeb y pinchamos sobre inicio.html, si todo lo hemos realizado correctamente se nos abrirá la parte del portal que llevamos hecha hasta el momento. Podremos ver la cabecera, el menú izquierdo y el texto del html en el centro. Si nos fijamos el menú izquierdo el enlace que metimos nos sale varios iconos para agregar, editar y

eliminar los enlaces, pinchando sobre ellos opencms nos dejara modificar la lista de forma directa, él realizará por detrás los cambios en los archivos necesarios. De esta forma nos ahorramos bastante tiempo, en vez de irnos al sitio donde se encuentren los enlaces tener que modificarlos y demás. Y por supuesto todo esto separando diseño, programación y datos de forma sencilla y evidente. Intentad agregar algún elemento al menú aunque el enlace no sea correcto (posteriormente lo corregiremos) para ello pulsad en el símbolo más de nuestra lista del menú. Se nos abrirá el editor del recurso linklist y podremos agregar un enlace interno o externo. Al guardar volveremos de nuevo a la pagina que estábamos viendo con el cambio visible.

19º - Aquí es realmente donde se ve la potencia de opencms 6 ya que ahora no necesitamos irnos a la carpeta a crear un nuevo recurso sino que viendo el portal podemos agregar recursos rápidamente sin importarnos realmente donde se encuentren físicamente esos recursos, si queremos corregir una falta de ortografía, de esta forma tardaríamos un minuto, mientras que si tuvieramos que entrar en opencms, ir a la carpeta del recurso, abrir el recurso y modificarlo tardaríamos mucho más y es más engorroso. Ya solo nos queda saber que hace la etiqueta contentshow que lo único que hace es leer el valor de una etiqueta del xml que tengamos cargado en ese momento en memoria y mostrarla. El parámetro "element" es el que especifica que mostramos.

20º - Básicamente opencms se basa en esto, con la practica se van aprendiendo diferentes formas de crear los listados y diferentes usos para los mismos. Una vez que se domina esta técnica se puede aprender a filtrar los listados de forma que solo aparezcan los recursos que tengan un cierto element (con contentcheck) o hacer un subbucle sobre un elemento que se repite (con contentloop, por ejemplo si un recurso tiene varias imagenes para recorrerlas todas).

21º - Para acabar de momento el template le vamos a

agregar una cosa mas. El pie de la pagina, usaremos otro elements al que llamaremos pie.jsp. Para ellos nos vamos a la carpeta elements y creamos pie.jsp y lo editamos con los siguiente:

```
<%@page buffer="none" session="false" import="java.util.*" %><%@ taglib prefix="cms"
uri="http://www.opencms.org/taglib/cms" %>
<div id="pie">
    <div class="margenBtnlinkpie">
        <div class="linkpie"><a href=""><img alt="" src="<cms:link>../resources/W3C-
WAI-AA.gif</cms:link>" border="0"></img></a></div>
        <div class="linkpie"><a href=""><img alt="" src="<cms:link>../resources/W3C-
CSS.gif</cms:link>" border="0"></img></a></div>
    </div>
    <div class="margentxtlinkpie">
        <div class="linkpie"><a href="">E-mail</a></div>
        <div class="linkpie">|</div>
        <div class="linkpie"><a href="">Privacidad</a></div>
        <div class="linkpie">|</div>
        <div class="linkpie"><a href="">Aviso Legal</a></div>
    </div>
</div>
```

Y en nuestro template principal llamamos al pie de la siguiente forma:

```
...
                        </div>
                <cms:include file="../elements/pie.jsp" />
        </div>
    </body>
...
```

Con esto ya nos aparecerá el pie en nuestras paginas. Si veis que algún bloque se descuadra un poco o se coloca en un sitio extraño puede ser debido a dos cosas. La primera es que opencms en offline si tenemos la etiqueta de editable nos crea capas y estructuras para colocar los iconos de edición, agregar y eliminar, opencms los agrega con capas (div) y varios elementos mas y lo normal es que nos descuadre un poco la pagina. Pero esto no es ningún problema ya que solo pasa en offline, cuando nos vamos al portal online sin estar logados todas esas capas desaparecen y se muestra el portal correctamente como lo escribimos en nuestras jsp. Otro problema puede ser que opencms al hacer el include en ciertos casos puede dejar huecos en blanco entre div y div dependiendo de nuestro

código pero se puede solucionar quitando los saltos de lineas que nos molesten o uniendo los en determinados casos concretos.

22º - Otra posibilidad que nos ofrecen las jsp aparte de las potentes taglibs es usar código java directamente. Para ello podemos escribir nuestro código java entre <% y %> . Todo lo que este entre estas llaves lo ejecutara el servidor antes de enviar el html al cliente. En todas las jsp tendremos varios objetos ya definidos nada mas entrar como por ejemplo el objeto "out", es el buffer de salida del servidor, que es el equivalente a "System.out" de la consola java pero para las jsp, con este objeto podemos escribir parte de nuestro html usándolo de esta forma:

```
out.println("<div>hola mundo</div>");
```

También tenemos otros objetos como request, response o session. Un ejemplo practico del objeto request es recoger una variable enviada en un formulario:

```
String var=request.getParameter("miCampo");
```

A partir de esa linea tendremos en var lo que se envió en el input con name="miCampo" o tendremos un null si no se ha enviado nada aun.

23º - Antes de seguir necesitaremos crearnos por lo menos dos galerías, una para imagenes y otra para descargas. Las galerías no son ni mas ni menos que una carpeta de tipo extended folder/image gallery, download gallery o external links gallery (aunque hay varias más), así pues creamos una "image gallery" y la nombramos "/miSitioWeb/modules/imagenesNoticias/" y un download gallery llamada "/miSitioWeb/modules/descargasNoticias/" y le especificamos la propiedad title a las dos para identificarlas posteriormente. Estas carpetas nos ayudaran para tener organizadas las imagenes y las descargas de nuestras noticias, también es una forma de no almacenar

múltiples veces una imagen o una descarga ya que al encontrarse en la misma carpeta no sera necesario duplicar el archivo.

24º - Ahora vamos a continuar mostrando otro listado pero esta vez sera de noticias e iremos añadiendo varias opciones más que aun no hemos visto. Para ello nos creamos nuestra carpeta que las contendrá "/miSitioWeb/modules/noticias/" y creamos un structured content/news article la llamamos por ejemplo news_0001.html (sera nuestro patrón de búsqueda), la editamos y rellenamos algunos datos. Una vez hecho esto pulsamos sobre el más de imagenes, y luego sobre el icono de galería de imagenes, esto nos abre el visor de imagenes de galerías, aquí podremos seleccionar nuestra galería que creamos anteriormente y seleccionar la imagen que mas nos guste o si no tenemos todavía imagenes subirlas. Subimos dos imagenes que tengamos en nuestro disco duro pinchando sobre el icono izquierdo, y las agregamos a la noticia (podemos agregar el numero de imagenes que deseemos).
Y seguimos haciendo exactamente lo mismo pero con las descargas pulsamos en el icono mas, y metemos en la galería dos archivos cualquiera, los adjuntamos a la noticia. Con dos bastara para el ejemplo.

25º - Ahora nos vamos a nuestra carpeta de templates y copiamos nuestro template principal con otro nombre "noticias.jsp" y lo editamos para cambiarle el title y el código. Debajo de

```
<div class="bloquedcha">
    <div class="txtruta"><cms:include element="text1" /></div>
</div>
```

ponemos lo siguiente:

```
<div class="bloquesDestacados">
        <div class="cabeceraBloqLarga">LISTADO NOTICIAS</div>
                <cms:contentload                    collector="allInFolderDateReleasedDesc"
param="/miSitioWeb/modules/noticias/news_${number}.html|news|4" editable="true">
                        <div class="fechaYTitulo">
```

```
                        <div class="fechaDestacada">Fecha |</div>
                        <div          class="tituloDestacado"><cms:contentshow
element="Title" /></div>
                    </div>
                    <cms:contentcheck ifexists="Teaser" >
                        <div          class="subtituloDestacado"><cms:contentshow
element="Teaser" /></div>
                    </cms:contentcheck>
                    <div class="sepmasInfoLargo">
                        <div class="masInfo">
                            <a          href="<cms:link><cms:contentshow
element="${opencms.filename}" /></cms:link>">Mas info</a>
                        </div>
                    </div>
                </cms:contentload>
        <div class="sepBloq">
            <img alt="" src="<cms:link>../resources/transpa.gif</cms:link>"/>
        </div>
</div>
```

hemos hecho lo mismo que hicimos para la lista de enlaces pero esta vez hemos usado allInFolderDateReleasedDesc (lee los recursos de la carpeta ordenados por fecha de release descententemente), hemos usado el contentcheck para comprobar si la noticia tiene Teaser, si la noticia no tuviese no Teaser no entraría dentro del contentcheck, así por ejemplo no se escribiría asunto de la noticia. Para los enlaces de Más Info hemos usado:

```
<cms:contentshow element="${opencms.filename}" />
```

Esto nos devuelve la ruta del archivo actual por el que va el bucle, así pues cuando pulsemos sobre mas info simplemente redirigimos la pagina directamente al recurso, y ya opencms se encargara del resto.

26º - Ahora vamos a crear el template que nos servirá para mostrar los detalles de cada noticia, ya que en la pagina de noticias solo mostramos el titulo y el asunto, cuando pulsemos en "mas info" deberá saltar este template mostrando todos los datos de la noticia o solo los que nos interesen. Para ello nos vamos a los templates y creamos uno que se llame "detallenoticia.jsp", le ponemos el titulo "detalle de noticias de mi sitio web". Ahora la editamos y la dejamos de esta forma:

```
<%@page buffer="none" session="false" import="java.util.*" %>
<%@ taglib prefix="cms" uri="http://www.opencms.org/taglib/cms" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<html xmlns="http://www.w3.org/1999/xhtml">
	<head>
		<meta http-equiv="content-type" content="text/html; charset=iso-8859-
1"/>
		<title>Mi Sitio Web</title>
		<link rel="STYLESHEET" type="text/css"
href="<cms:link>../resources/estilos.css</cms:link>"/>
		<cms:editable />
	</head>
	<body>
		<div id="contenedor">
			<cms:include file="../elements/cabecera.jsp" />
			<div id="cuerpo">
				<cms:include file="../elements/menuizquierdo.jsp" />
				<div id="bloquedcha">
					<div class="ruta">
						<cms:contentload collector="singleFile"
param="${opencms.uri}" editable="true">
							<div
class="txtruta"><cms:contentshow element="Title" /></div>
							<div
class="txtruta"><cms:contentshow element="Author" /></div>
							<c:set
var="dateString"><cms:contentshow element="Date"/></c:set><%
								java.util.Date date = new
java.util.Date();
								try{

	date.setTime(Long.parseLong(pageContext.getAttribute("dateString").toString()));

	pageContext.setAttribute("date", date);
								}catch(Exception e){

	pageContext.setAttribute("date", "");
								}
							%><div
class="txtruta"><fmt:formatDate value="${date}" type="date" pattern="dd/MM/yyyy"
/></div>
							<div
class="txtruta"><cms:contentshow element="Teaser" /></div>
							<cms:contentloop
element="Attachment">
								<div class="txtruta"><a
href="<cms:link><cms:contentshow element="File" /></cms:link>"><cms:contentshow
element="Description" /></a></div>
							</cms:contentloop>
							<cms:contentloop element="Image">
								<div><img
src="<cms:link><cms:contentshow element="Image" /></cms:link>" /></div>
								<div
class="txtruta"><cms:contentshow element="Description" /></div>
							</cms:contentloop>
						</cms:contentload>
					</div>
				</div>
			</div>
			<cms:include file="../elements/pie.jsp" />
		</div>
```

```
        </body>
</html>
```

Recordad ponedle la propiedad title a las dos nuevas jsp para distinguir los templates.
Si os fijáis os daréis cuenta que hemos usado la misma técnica que para el template principal usando los <cms:include element=”elemento” /> solo que esta vez hemos agregado algunos componentes a la jsp para transformar la fecha. La fecha en los archivos xml de opencms los guarda en formato “long” por lo tanto si lo cogemos como tal veremos un numero grande. Para solucionar esto hemos agregado dos componentes a la jsp:

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
```

El componente fmt contiene taglibs para formatear números,cadenas, fechas, etc... Y el componente core nos ayuda a agregar y extraer variables en el contexto desde otra taglib por ejemplo:

```
<c:set var="dateString"><cms:contentshow element="Date"/></c:set>
```

Esto guarda en la variable de contexto dateString el valor que devuelva la etiqueta contentshow. Mas tarde para extraerla y usarla en java hacemos lo siguiente:

```
date.setTime(Long.parseLong(pageContext.getAttribute("dateString").toString()));
```

Cogemos la variable dateString del contexto, la pasamos a cadena, luego la transformamos a long y lo metemos en el objeto date de tipo Date que teníamos arriba declarado. Una vez hecho esto usamos el formateador para ponerlo como queremos:

```
<% pageContext.setAttribute("date", date); %>
<fmt:formatDate value="${date}" type="date" pattern="dd/MM/yyyy"  />
```

Donde ${date} es la variable del contexto que transformamos.

27º - Además de formatear la fecha hemos usado varias

etiquetas más por ejemplo en esta parte del código:

```
<cms:contentloop element="Attachment">
        <div class="txtruta">
                <a              href="<cms:link><cms:contentshow              element="File"
/></cms:link>"><cms:contentshow element="Description" /></a>
        </div>
</cms:contentloop>
```

Donde hemos recorrido todos las etiquetas Attachment que tuviese la noticia, ya que el editor permite meter todos los que se deseen. Y para las imagenes hemos hecho exactamente lo mismo. Si no estamos seguros de los nombres de las etiquetas podemos irnos a la noticia y editarle el código de control así podremos ver el xml que genera el editor, este xml nos vale para ver la estructura que tenemos que seguir.

28º - Si vais a la carpeta donde guardamos las noticias y pincháis sobre alguna os daréis cuenta que no sale nuestra pagina de detalle, sino que sale alguna de opencms que ya estuviese definida anteriormente. De alguna forma tenemos que decirle que esas noticias deben abrirse con nuestro template de detalle. La forma mas sencilla es cambiando la propiedad “template-elements” de la carpeta de noticias. Al partir de hacer este cambio cada vez que opencms intente abrir una noticia de esa carpeta mirara la propiedad template-elements del recurso noticia, al no encontrarlo (esta en blanco) buscara esa propiedad en el padre (nuestra carpeta noticias) como el padre si la tiene la usa y nos redirige a nuestro template de detalle. Resumiendo hemos especificado a opencms que todo lo que todos los elementos de esa carpeta han de abrirse con nuestro template de detalle de la noticia. Hay que eliminar la propiedad template-elements de la noticia si la tuviese ya definida de antes.

29º - Ahora vamos a darle algunos detalles mas a la web, por ejemplo vamos a poner dos banners debajo del menú izquierdo. Empezaremos por crear otra galería de imagenes que contendrá las imagenes de nuestros baners. La

creamos en "/miSitioWeb/modules/imagenesBaners" y le ponemos algún titulo (propiedad title) "imagenes de los banners de mi sitio web" por ejemplo. Y creamos la carpeta "/miSitioWeb/modules/baners/" para meter los banners ahí.

30º - Creamos un banner_0001.html dentro de la carpeta (nos servirá de patrón) de tipo linklist (como en el menú izquierdo). No tendremos problemas con las imagenes ya que el tipo linklist deja insertar imagenes. Cogemos las dos imagenes de banners que tenemos en la red y editamos el banner subiendo las dos imagenes a la galería como ya comentamos anteriormente. Seleccionamos una de las dos y guardamos nuestro enlace.

31º - Como no necesitamos que nuestros baner se abran con ningun template no es necesario que definamos la propiedad template-elements ni nada más. Nos vamos a editar el menuizquierdo.jsp de nuestra carpeta elements en el modulo. Lo modificamos para que quede así:

```
<%@page buffer="none" session="false" import="java.util.*" %><%@ taglib prefix="cms"
uri="http://www.opencms.org/taglib/cms" %>
<div id="bloqueIzq">
        <cms:contentload                                        collector="allInFolderNavPos"
param="/miSitioWeb/modules/menuIzquierdo/linklist_${number}.html|linklist|12"
editable="true">
                <div class="cajaMenuIzq">
                        <div        class="tituloMenuIzq"><a        href="<cms:contentshow
element="Url" />" target="<cms:contentshow element="Target" />"><cms:contentshow
element="Title" /></a></div>
                </div>
        </cms:contentload>
        <cms:contentload                                        collector="allInFolderNavPos"
param="/miSitioWeb/modules/baners/banner_${number}.html|linklist|2" editable="true">
                <div class="banner">
                        <a          href="<cms:contentshow          element="Url"          />"
target="<cms:contentshow element="Target" />">
                                <img      alt="<cms:contentshow      element="Title"      />"
src="<cms:link><cms:contentshow element="Image/Image" /></cms:link>" border="0"/>
                        </a>
                </div>
        </cms:contentload>
</div>
```

Hemos hecho lo mismo de siempre pero cambiando para que solo salgan dos banners y cambiando el nombre del patrón para que nos coja banner_0001.html. Una vez hecho esto nos vamos a portal y si vemos un banner lo

hemos hecho correctamente. Le damos al más y agregamos el otro banner con el editor desde la misma web. Si os fijáis hemos configurado el tamaño de la imagen width="141" height="74" por si alguien mete una imagen mas grande que no se descuadre el portal.

32º - Como se puede observar sin mucho esfuerzo de programación hemos realizado un portal dinámico sin muchos problemas y viendo solo algunas taglibs, si queréis meteros más en profundidad y sacar realmente todo el potencial de opencms tendréis que usar su api java que podéis encontrar en esta url:

http://www.opencms.org/export/javadoc/core/index.html

Todo lo que nosotros hemos realizado con taglibs se puede realizar con el código java, en los ejemplos que trae opencms vienen algunos donde se compara las taglib con el código java haciendo lo mismo. Por poner algunos ejemplos del uso de la api de opencms que no podremos conseguir con taglibs son:

- poder logarse desde una jsp en opencms
- modificar el contenido de los archivos desde una jsp directamente
- hacer busquedas de archivos usando el motor de Lucene que trae incorporado opencms 6
- cambiar configuraciones de opencms
- crear newsletters
- exportar/importar recursos de la BD al disco o viceversa
- bloquear/desbloquear/cambiar permisos de archivos de la BD
- y un largo etc...

Si estais interesados en esta herramienta y teneis alguna duda podeis consultar la web oficial de opencms:

http://www.opencms.org

o nuestra web en castellano:

http://www.opencmshispano.com
http://www.opencmshispano.org

33º - Y por ultimo actividades:

- Modifica los estilos del detalle de la noticia para que no quede tan simple
- Crea una sección nueva de articulos, usando el recurso tipo article (este recurso es muy parecido al de noticia), haz el template que lista los articulos y después el template del detalle que muestra todos los datos de los articulos.