**7.3. Validación**

La principal utilidad de JavaScript en el manejo de los formularios es la validación de los datos introducidos por los usuarios. Antes de enviar un formulario al servidor, se recomienda validar mediante JavaScript los datos insertados por el usuario. De esta forma, si el usuario ha cometido algún error al rellenar el formulario, se le puede notificar de forma instantánea, sin necesidad de esperar la respuesta del servidor.

Notificar los errores de forma inmediata mediante JavaScript mejora la satisfacción del usuario con la aplicación (lo que técnicamente se conoce como *"mejorar la experiencia de usuario"*) y ayuda a reducir la carga de procesamiento en el servidor.

Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario. En esta función, se comprueban si los valores que ha introducido el usuario cumplen las restricciones impuestas por la aplicación.

Aunque existen tantas posibles comprobaciones como elementos de formulario diferentes, algunas comprobaciones son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que la dirección de email indicada sea correcta, que la fecha introducida sea lógica, que se haya introducido un número donde así se requiere, etc.

A continuación se muestra el código JavaScript básico necesario para incorporar la validación a un formulario:

<form action="" method="" id="" name="" onsubmit="return validacion()">

...

</form>

Y el esquema de la función validacion() es el siguiente:

function validacion() {

if (condicion que debe cumplir el primer campo del formulario) {

*// Si no se cumple la condicion...*

alert('[ERROR] El campo debe tener un valor de...');

return false;

}

else if (condicion que debe cumplir el segundo campo del formulario) {

*// Si no se cumple la condicion...*

alert('[ERROR] El campo debe tener un valor de...');

return false;

}

...

else if (condicion que debe cumplir el último campo del formulario) {

*// Si no se cumple la condicion...*

alert('[ERROR] El campo debe tener un valor de...');

return false;

}

*// Si el script ha llegado a este punto, todas las condiciones*

*// se han cumplido, por lo que se devuelve el valor true*

return true;

}

El funcionamiento de esta técnica de validación se basa en el comportamiento del evento onsubmit de JavaScript. Al igual que otros eventos como onclick y onkeypress, el evento 'onsubmit varía su comportamiento en función del valor que se devuelve.

Así, si el evento onsubmit devuelve el valor true, el formulario se envía como lo haría normalmente. Sin embargo, si el evento onsubmit devuelve el valor false, el formulario no se envía. La clave de esta técnica consiste en comprobar todos y cada uno de los elementos del formulario. En cuando se encuentra un elemento incorrecto, se devuelve el valor false. Si no se encuentra ningún error, se devuelve el valor true.

Por lo tanto, en primer lugar se define el evento onsubmit del formulario como:

onsubmit="return validacion()"

Como el código JavaScript devuelve el valor resultante de la función validacion(), el formulario solamente se enviará al servidor si esa función devuelve true. En el caso de que la función validacion() devuelva false, el formulario permanecerá sin enviarse.

Dentro de la función validacion() se comprueban todas las condiciones impuestas por la aplicación. Cuando no se cumple una condición, se devuelve false y por tanto el formulario no se envía. Si se llega al final de la función, todas las condiciones se han cumplido correctamente, por lo que se devuelve true y el formulario se envía.

La notificación de los errores cometidos depende del diseño de cada aplicación. En el código del ejemplo anterior simplemente se muestran mensajes mediante la función alert() indicando el error producido. Las aplicaciones web mejor diseñadas muestran cada mensaje de error al lado del elemento de formulario correspondiente y también suelen mostrar un mensaje principal indicando que el formulario contiene errores.

Una vez definido el esquema de la función validacion(), se debe añadir a esta función el código correspondiente a todas las comprobaciones que se realizan sobre los elementos del formulario. A continuación, se muestran algunas de las validaciones más habituales de los campos de formulario.

**7.3.1. Validar un campo de texto obligatorio**

Se trata de forzar al usuario a introducir un valor en un cuadro de texto o textarea en los que sea obligatorio. La condición en JavaScript se puede indicar como:

valor = document.getElementById("campo").value;

if( valor == null || valor.length == 0 || /^\s+$/.test(valor) ) {

return false;

}

Para que se de por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que cero y que no se hayan introducido sólo espacios en blanco.

La palabra reservada null es un valor especial que se utiliza para indicar *"ningún valor"*. Si el valor de una variable es null, la variable no contiene ningún valor de tipo objeto, array, numérico, cadena de texto o booleano.

La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a cero caracteres, esto es, que no sea un texto vacío.

Por último, la tercera parte de la condición (/^\s+$/.test(valor)) obliga a que el valor introducido por el usuario no sólo esté formado por espacios en blanco. Esta comprobación se basa en el uso de *"expresiones regulares"*, un recurso habitual en cualquier lenguaje de programación pero que por su gran complejidad no se van a estudiar. Por lo tanto, sólo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.

**7.3.2. Validar un campo de texto con valores numéricos**

Se trata de obligar al usuario a introducir un valor numérico en un cuadro de texto. La condición JavaScript consiste en:

valor = document.getElementById("campo").value;

if( isNaN(valor) ) {

return false;

}

Si el contenido de la variable valor no es un número válido, no se cumple la condición. La ventaja de utilizar la función interna isNaN() es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

A continuación se muestran algunos resultados de la función isNaN():

isNaN(3); *// false*

isNaN("3"); *// false*

isNaN(3.3545); *// false*

isNaN(32323.345); *// false*

isNaN(+23.2); *// false*

isNaN("-23.2"); *// false*

isNaN("23a"); *// true*

isNaN("23.43.54"); *// true*

**7.3.3. Validar que se ha seleccionado una opción de una lista**

Se trata de obligar al usuario a seleccionar un elemento de una lista desplegable. El siguiente código JavaScript permite conseguirlo:

indice = document.getElementById("opciones").selectedIndex;

if( indice == null || indice == 0 ) {

return false;

}

<select id="opciones" name="opciones">

<option value="">- Selecciona un valor -</option>

<option value="1">Primer valor</option>

<option value="2">Segundo valor</option>

<option value="3">Tercer valor</option>

</select>

A partir de la propiedad selectedIndex, se comprueba si el índice de la opción seleccionada es válido y además es distinto de cero. La primera opción de la lista (- Selecciona un valor –) no es válida, por lo que no se permite el valor 0 para esta propiedad selectedIndex.

**7.3.4. Validar una dirección de email**

Se trata de obligar al usuario a introducir una dirección de email con un formato válido. Por tanto, lo que se comprueba es que la dirección *parezca* válida, ya que no se comprueba si se trata de una cuenta de correo electrónico real y operativa. La condición JavaScript consiste en:

valor = document.getElementById("campo").value;

if( !(/\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)/.test(valor)) ) {

return false;

}

La comprobación se realiza nuevamente mediante las expresiones regulares, ya que las direcciones de correo electrónico válidas pueden ser muy diferentes. Por otra parte, como el estándar que define el formato de las direcciones de correo electrónico es muy complejo, la expresión regular anterior es una simplificación. Aunque esta regla valida la mayoría de direcciones de correo electrónico utilizadas por los usuarios, no soporta todos los diferentes formatos válidos de email.

**7.3.5. Validar una fecha**

Las fechas suelen ser los campos de formulario más complicados de validar por la multitud de formas diferentes en las que se pueden introducir. El siguiente código asume que de alguna forma se ha obtenido el año, el mes y el día introducidos por el usuario:

var ano = document.getElementById("ano").value;

var mes = document.getElementById("mes").value;

var dia = document.getElementById("dia").value;

valor = new Date(ano, mes, dia);

if( !isNaN(valor) ) {

return false;

}

La función Date(ano, mes, dia) es una función interna de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. Es muy importante tener en cuenta que el número de mes se indica de 0 a 11, siendo 0 el mes de Enero y 11 el mes de Diciembre. Los días del mes siguen una numeración diferente, ya que el mínimo permitido es 1 y el máximo 31.

La validación consiste en intentar construir una fecha con los datos proporcionados por el usuario. Si los datos del usuario no son correctos, la fecha no se puede construir correctamente y por tanto la validación del formulario no será correcta.

**7.3.6. Validar un número de DNI**

Se trata de comprobar que el número proporcionado por el usuario se corresponde con un número válido de Documento Nacional de Identidad o DNI. Aunque para cada país o región los requisitos del documento de identidad de las personas pueden variar, a continuación se muestra un ejemplo genérico fácilmente adaptable. La validación no sólo debe comprobar que el número esté formado por ocho cifras y una letra, sino que también es necesario comprobar que la letra indicada es correcta para el número introducido:

valor = document.getElementById("campo").value;

var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

if( !(/^\d{8}[A-Z]$/.test(valor)) ) {

return false;

}

if(valor.charAt(8) != letras[(valor.substring(0, 8))%23]) {

return false;

}

La primera comprobación asegura que el formato del número introducido es el correcto, es decir, que está formado por 8 números seguidos y una letra. Si la letra está al principio de los números, la comprobación sería /^[A-Z]\d{8}$/. Si en vez de ocho números y una letra, se requieren diez números y dos letras, la comprobación sería /^\d{10}[A-Z]{2}$/ y así sucesivamente.

La segunda comprobación aplica el algoritmo de cálculo de la letra del DNI y la compara con la letra proporcionada por el usuario. El algoritmo de cada documento de identificación es diferente, por lo que esta parte de la validación se debe adaptar convenientemente.

**7.3.7. Validar un número de teléfono**

Los números de teléfono pueden ser indicados de formas muy diferentes: con prefijo nacional, con prefijo internacional, agrupado por pares, separando los números con guiones, etc.

El siguiente script considera que un número de teléfono está formado por nueve dígitos consecutivos y sin espacios ni guiones entre las cifras:

valor = document.getElementById("campo").value;

if( !(/^\d{9}$/.test(valor)) ) {

return false;

}

Una vez más, la condición de JavaScript se basa en el uso de expresiones regulares, que comprueban si el valor indicado es una sucesión de nueve números consecutivos. A continuación se muestran otras expresiones regulares que se pueden utilizar para otros formatos de número de teléfono:

| **Número** | **Expresión regular** | **Formato** |
| --- | --- | --- |
| 900900900 | /^\d{9}$/ | 9 cifras seguidas |
| 900-900-900 | /^\d{3}-\d{3}-\d{3}$/ | 9 cifras agrupadas de 3 en 3 y separadas por guiones |
| 900 900900 | /^\d{3}\s\d{6}$/ | 9 cifras, las 3 primeras separadas por un espacio |
| 900 90 09 00 | /^\d{3}\s\d{2}\s\d{2}\s\d{2}$/ | 9 cifras, las 3 primeras separadas por un espacio, las siguientes agrupadas de 2 en 2 |
| (900) 900900 | /^\(\d{3}\)\s\d{6}$/ | 9 cifras, las 3 primeras encerradas por paréntesis y un espacio de separación respecto del resto |
| +34 900900900 | /^\+\d{2,3}\s\d{9}$/ | Prefijo internacional (+ seguido de 2 o 3 cifras), espacio en blanco y 9 cifras consecutivas |

**7.3.8. Validar que un checkbox ha sido seleccionado**

Si un elemento de tipo *checkbox* se debe seleccionar de forma obligatoria, JavaScript permite comprobarlo de forma muy sencilla:

elemento = document.getElementById("campo");

if( !elemento.checked ) {

return false;

}

Si se trata de comprobar que todos los *checkbox* del formulario han sido seleccionados, es más fácil utilizar un bucle:

formulario = document.getElementById("formulario");

for(var i=0; i<formulario.elements.length; i++) {

var elemento = formulario.elements[i];

if(elemento.type == "checkbox") {

if(!elemento.checked) {

return false;

}

}

}

**7.3.9. Validar que un radiobutton ha sido seleccionado**

Aunque se trata de un caso similar al de los *checkbox*, la validación de los *radiobutton* presenta una diferencia importante: en general, la comprobación que se realiza es que el usuario haya seleccionado algún *radiobutton* de los que forman un determinado grupo. Mediante JavaScript, es sencillo determinar si se ha seleccionado algún *radiobutton* de un grupo:

opciones = document.getElementsByName("opciones");

var seleccionado = false;

for(var i=0; i<opciones.length; i++) {

if(opciones[i].checked) {

seleccionado = true;

break;

}

}

if(!seleccionado) {

return false;

}

El anterior ejemplo recorre todos los *radiobutton* que forman un grupo y comprueba elemento por elemento si ha sido seleccionado. Cuando se encuentra el primer *radiobutton* seleccionado, se sale del bucle y se indica que al menos uno ha sido seleccionado.

# 8.1. Relojes, contadores e intervalos de tiempo

En ocasiones, algunas páginas web muestran un reloj con la hora actual. Si el reloj debe actualizarse cada segundo, no se puede mostrar la hora directamente en la página HTML generada por el servidor. En este caso, aunque existen alternativas realizadas con Java y con Flash, la forma más sencilla de hacerlo es mostrar la hora del ordenador del usuario mediante JavaScript.

Para crear y mostrar un reloj con JavaScript, se debe utilizar el objeto interno Date() para crear fechas/horas y las utilidades que permiten definir contadores, para actualizar el reloj cada segundo.

El objeto Date() es una utilidad que proporciona JavaScript para crear fechas y horas. Una vez creado un objeto de tipo fecha, es posible manipularlo para obtener información o realizar cálculos con las fechas. Para obtener la fecha y hora actuales, solamente es necesario crear un objeto Date() sin pasar ningún parámetro:

var fechaHora = new Date();

Utilizando el código anterior, se puede construir un reloj muy básico que no actualiza su contenido:

var fechaHora = new Date();

document.getElementById("reloj").innerHTML = fechaHora;

<div id="reloj" />

Cuando se carga la página, el ejemplo anterior mostraría un texto parecido al siguiente en el <div> reservado para el reloj:

Mon May 04 2009 13:36:10 GMT+0200 (Hora de verano romance)

Este primer reloj construido presenta muchas diferencias respecto al reloj que se quiere construir. En primer lugar, muestra más información de la necesaria. Además, su valor no se actualiza cada segundo, por lo que no es un reloj muy práctico.

El objeto Date() proporciona unas funciones muy útiles para obtener información sobre la fecha y la hora. Concretamente, existen funciones que devuelven la hora, los minutos y los segundos:

var fechaHora = new Date();

var horas = fechaHora.getHours();

var minutos = fechaHora.getMinutes();

var segundos = fechaHora.getSeconds();

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

<div id="reloj" />

Utilizando las funciones getHours(), getMinutes() y getSeconds() del objeto Date, el reloj solamente muestra la información de la hora. El resultado del ejemplo anterior sería un reloj como el siguiente:

20:9:21

Si la hora, minuto o segundo son menores que 10, JavaScript no añade el 0 por delante, por lo que el resultado no es del todo satisfactorio. El siguiente código soluciona este problema añadiendo un 0 cuando sea necesario:

var fechaHora = new Date();

var horas = fechaHora.getHours();

var minutos = fechaHora.getMinutes();

var segundos = fechaHora.getSeconds();

if(horas < 10) { horas = '0' + horas; }

if(minutos < 10) { minutos = '0' + minutos; }

if(segundos < 10) { segundos = '0' + segundos; }

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

<div id="reloj" />

Ahora el reloj muestra correctamente la hora:

20:14:03

Si se quiere mostrar el reloj con un formato de 12 horas en vez de 24, se puede utilizar el siguiente código:

var fechaHora = new Date();

var horas = fechaHora.getHours();

var minutos = fechaHora.getMinutes();

var segundos = fechaHora.getSeconds();

var sufijo = ' am';

if(horas > 12) {

horas = horas - 12;

sufijo = ' pm';

}

if(horas < 10) { horas = '0' + horas; }

if(minutos < 10) { minutos = '0' + minutos; }

if(segundos < 10) { segundos = '0' + segundos; }

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos+sufijo;

<div id="reloj" />

Utilizando el código anterior, el reloj ya no muestra la hora como 20:14:03, sino que la muestra en formato de 12 horas y con el sufijo adecuado:

08:14:03 pm

Para completar el reloj, sólo falta que se actualice su valor cada segundo. Para conseguirlo, se deben utilizar unas funciones especiales de JavaScript que permiten ejecutar determinadas instrucciones cuando ha transcurrido un determinado espacio de tiempo.

La función setTimeout() permite ejecutar una función una vez que haya transcurrido un periodo de tiempo indicado. La definición de la función es:

setTimeout(nombreFuncion, milisegundos);

La función que se va a ejecutar se debe indicar mediante su nombre sin paréntesis y el tiempo que debe transcurrir hasta que se ejecute se indica en milisegundos. De esta forma, si se crea una función encargada de mostrar la hora del reloj y se denomina muestraReloj(), se puede indicar que se ejecute dentro de 1 segundo mediante el siguiente código:

function muestraReloj() {

var fechaHora = new Date();

var horas = fechaHora.getHours();

var minutos = fechaHora.getMinutes();

var segundos = fechaHora.getSeconds();

if(horas < 10) { horas = '0' + horas; }

if(minutos < 10) { minutos = '0' + minutos; }

if(segundos < 10) { segundos = '0' + segundos; }

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

}

setTimeout(muestraReloj, 1000);

<div id="reloj" />

No obstante, el código anterior simplemente muestra el contenido del reloj 1 segundo después de que se cargue la página, por lo que no es muy útil. Para ejecutar una función de forma periódica, se utiliza una función de JavaScript muy similar a setTimeout() que se denomina setInterval(). Su definición es:

setInterval(nombreFuncion, milisegundos);

La definición de esta función es idéntica a la función setTimeout(), salvo que en este caso, la función programada se ejecuta infinitas veces de forma periódica con un lapso de tiempo entre ejecuciones de tantos milisegundos como se hayan establecido.

Así, para construir el reloj completo, se establece una ejecución periódica de la función muestraReloj() cada segundo:

function muestraReloj() {

var fechaHora = new Date();

var horas = fechaHora.getHours();

var minutos = fechaHora.getMinutes();

var segundos = fechaHora.getSeconds();

if(horas < 10) { horas = '0' + horas; }

if(minutos < 10) { minutos = '0' + minutos; }

if(segundos < 10) { segundos = '0' + segundos; }

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

}

window.onload = function() {

setInterval(muestraReloj, 1000);

}

<div id="reloj" />

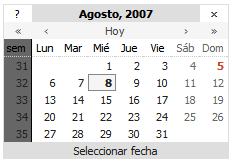
Empleando el objeto Date y sus funciones, es posible construir "cuentras atrás", es decir, relojes que muestran el tiempo que falta hasta que se produzca un evento. Además, las funciones setTimeout() y setInterval() pueden resultar muy útiles en otras técnicas de programación.

# 8.2. Calendario

Cuando una aplicación muestra un formulario para que el usuario inserte sus datos, la información más complicada de obtener siempre suelen ser los números de teléfono y las fechas. El motivo es que existen muchas formas diferentes de indicar estos datos. Las fechas por ejemplo se pueden indicar como dd/mm/aa, dd/mm/aaaa, aaaa/mm/dd, dd-mm-aa, dd mm aaaa, etc.

Los métodos más utilizados para que el usuario introduzca la información relativa a una fecha suelen ser un cuadro de texto en el que tiene que insertar la fecha completa (indicándole el formato que debe seguir) o un cuadro de texto para el día, una lista desplegable para el mes y otro cuadro de texto para el año.

En cualquier caso, para el usuario suele ser más cómodo que la aplicación incluya un calendario en el que pueda indicar la fecha pinchando sobre el día elegido:



**Figura 8.1** Aspecto de un calendario creado con JavaScript

Además, este método es menos propenso a cometer errores, ya que si el calendario está bien construido, no es posible introducir fechas inválidas y tampoco es posible insertar las fechas en un formato incorrecto.

No obstante, realizar un calendario completo en JavaScript no es una tarea trivial, por lo que no se va a estudiar su desarrollo completo. Afortunadamente, existen scripts gratuitos para mostrar calendarios y que permiten su uso libre incluso en aplicaciones comerciales.

De entre todos los calendarios disponibles, uno de los más completos es el desarrollado por la empresa DynArch, que se puede acceder desde su página web oficial: http://www.dynarch.com/projects/calendar/ y que se puede descargar gratuitamente desde http://sourceforge.net/projects/jscalendar/ El archivo descargado incluye todos los scripts necesarios, su documentación, ejemplos de uso, diferentes estilos CSS para el calendario, etc.

A continuación se indican los pasos necesarios para incluir un calendario básico en cualquier página web:

1) Enlazar los archivos JavaScript y CSS requeridos:

Se descomprime el archivo descargado, se guardan los archivos JavaScript y CSS en el sitio adecuado (en este ejemplo se supone que los archivos JavaScript se guardan en el directorio js/ y los archivos CSS en el directorio css/) y se enlazan directamente desde la cabecera de la página HTML.

<head>

...

<style type="text/css">@import url("css/calendar-estilo.css");</style>

<script type="text/javascript" src="js/calendar.js" />

<script type="text/javascript" src="js/calendar-es.js" />

<script type="text/javascript" src="js/calendar-setup.js" />

...

</head>

El calendario incluye traducciones a más de 40 idiomas, entre los que se encuentra el español. Para mostrar el calendario en un determinado idioma, tan solo es necesario enlazar el archivo del idioma correspondiente. Las traducciones se encuentran en el directorio lang y su formato es calendar-XX.js, donde XX es el código del idioma.

2) Añadir el código XHTML necesario para el elemento que va a mostrar el calendario:

<p>Introduce la fecha pulsando sobre la imagen del calendario</p>

<input type="text" name="date" id="fecha" readonly="readonly" />

<img src="calendario.png" id="selector" />

En este caso, el calendario está formado por dos elementos:

* Un cuadro de texto llamado fecha y que almacena el valor introducido por el usuario. Como se le ha asignado un atributo readonly="readonly", el usuario no puede modificar su valor.
* Una pequeña imagen o icono que se utiliza para activar el calendario. Cuando el usuario pincha con el ratón sobre la imagen, se muestra el calendario de JavaScript.

3) Configurar el calendario:

<script type="text/javascript">

window.onload = function() {

Calendar.setup({

inputField: "fecha",

ifFormat: "%d / %m / %Y",

button: "selector"

});

}

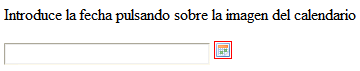
</script>

Una vez enlazados los archivos del calendario y preparado el código XHTML, es necesario inicializar y configurar el calendario. La configuración del calendario consiste en establecer el valor de alguna de sus propiedades. Las propiedades básicas imprescindibles son:

* inputField: se trata del atributo id del elemento en el que se mostrará la fecha seleccionada, en este ejemplo sería fecha.
* ifFormat: formato en el que se mostrará la fecha una vez seleccionada (en este caso se ha optado por el formato dd / mm / aaaa).
* button: atributo id del elemento que se pulsa (botón, imagen, enlace) para mostrar el calendario de selección de fecha. En este ejemplo, el id de la imagen es selector.

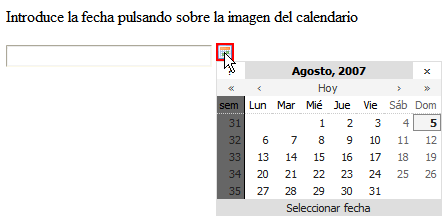
Después de esta configuración básica, el calendario ya puede utilizarse en la aplicación:

1) Aspecto por defecto del selector de calendario:



**Figura 8.2** Elementos XHTML del calendario JavaScript: cuadro de texto e icono

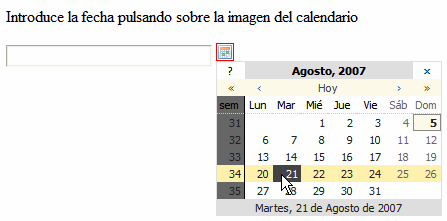
2) Al pinchar una vez sobre la imagen del calendario:



**Figura 8.3** Aspecto inicial del calendario JavaScript cuando se abre por primera vez

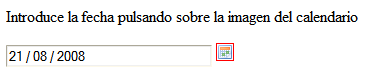
Se muestra el calendario con el aspecto e idioma establecidos y aparece resaltada la fecha del día actual.

3) Se selecciona la fecha deseada:



**Figura 8.4** Seleccionando una fecha en el calendario JavaScript

4) Después de pinchar con el ratón sobre la fecha deseada:



**Figura 8.5** El cuadro de texto muestra la fecha establecida por el usuario con el calendario JavaScript

Al pulsar con el ratón sobre la fecha deseada, el calendario se cierra automáticamente y el cuadro de texto muestra la fecha seleccionada con el formato indicado en la configuración del calendario. Si se vuelve a abrir el calendario para seleccionar otra fecha, se mostrará resaltada la fecha del día seleccionado y no la fecha del día actual.