DWEC
Unidad 4

Formularios, validación y almacenamiento.

INIDICE

1. Utilización de formularios desde código

Un formulario web dentro de una página web permite al usuario introducir datos que son enviados a un servidor para ser procesados.

Cada elemento del formulario sirve para almacenar un tipo de dato o para accionar las distintas funcionalidades que tiene el formulario.

Los formularios disponen de una arquitectura, que habitualmente va ligada a otra más general. En nuestro caso los formularios están enmarcados en un lenguaje de marcado llamado HTML.

1.1. Estructura de un formulario

Un formulario está compuesto por muchos elementos, pero para generar una estructura básica de un formulario, basta con utilizar dos de estos elementos. La etiqueta principal de un formulario es
<form>, esta etiqueta engloba el comienzo y el final del formulario. Por tanto el resto de elementos estarán definidos dentro de la apertura y cierre de la etiqueta form, ejemplo, <form>...elementos...</form>.

La etiqueta <form> necesita inicializar dos atributos para que el formulario sea funcional. El primer atributo es action. Este atributo contendrá la URL donde se redirigirán los datos del formulario. La segunda etiqueta básica es method, esta etiqueta indica el método por el cual el formulario envía los datos. Las opciones de esta etiqueta son **POST** ó **GET** e indican la forma en la que van a ser enviados los datos del formulario. En el caso de **POST**, los datos se enviarán ocultos (no encriptados), en el método GET en cambio los datos viajan directamente en la URL. La forma de indicar cuándo finaliza la URL y cuándo comienzan a ser enviados los datos es separándolo con una interrogación cerrada (?) después de la finalización de la dirección. Conviene saber que el máximo de información a enviar por el método **GET** son unos **500 bytes** y que este método no permite el envío de ficheros adjuntos.

El método **POST** permite un **mayor envío de datos**, además es algo más seguro al ser más difícil acceder a los mismos. No debemos pensar que el método POST es mejor puesto que hay casos en los que resulta útil utilizar GET, por ejemplo, cuando tenemos un gran número de fotografías o vídeos para pasar el identificador de cada uno de ellos. También es muy útil el método GET para los desarrolladores de aplicaciones web puesto que en tiempo de desarrollo pueden usar este método para validar los datos y comprobar errores de una forma más visual.

La etiqueta <form> de los formularios también tiene atributos como enctype, que define el tipo de codificación que empleamos para enviar el formulario al servidor. Este atributo se indica en el formulario cuando el formulario permite enviar archivos adjuntos. Otro atributo de un formulario es accept (para formularios que aceptan ficheros adjuntos), indica el tipo de archivos que acepta el servidor.

También existen los atributos; accept-charset, onsubmit u onreset, que amplían algunas posibilidades para los formularios aunque son algo menos utilizados.

A continuación vamos a ver cómo queda encuadrada la estructura <form> en una página de código HTML:

```
<html>
  <head>
    <title>Ejemplo de formulario</title>
  </head>
  <body>
    <h3>Formulario</h3>
    <form action="www.Web.es/formulario.php" method="post"> </form>
  </body>
</html>
```

El en código anterior vemos como dentro de body de una página web introducimos una estructura form. En el action enviamos los datos del formulario para que sean procesados en el servidor. En esta caso los datos serán enviados de forma oculta puesto que hemos utilizado el método POST.

1.2. Elementos de un formulario

Según hemos visto en el punto anterior, un formulario sirve para enviar datos, pero... ¿cómo vamos a enviar esos datos? Y, además, ¿cómo vamos a indicar cuando queremos enviar esos datos? Para poder definir los datos que queremos enviar y cuando los vamos a enviar, tenemos los elementos del formulario. El elemento principal del formulario se denomina con la etiqueta <**input**>. Este elemento tiene una serie de atributos que modifican su funcionalidad. Los tipos de input según su funcionalidad se llaman controles de formulario y campos de formulario. Estos son los encargados de guardar los datos que vamos a enviar en el formulario. Los hay de varios tipos y a continuación vamos a definir cómo guardan y procesan los datos cada uno de ellos. También se encargan de accionar sucesos, por lo general son botones y al ser pulsados realizan la acción para la que están encomendados.

1.3. Estructura de una etiqueta input

El formato por defecto de una etiqueta input es el siguiente **<input ... / >**, como es una etiqueta unaría se cierra a sí misma con la barra (/) al final. Esta etiqueta se transforma en distintos elementos que a su vez almacenan un tipo de dato. Para definir el tipo de elemento, la etiqueta input se sirve del atributo type. De esta forma si igualamos uno de los tipos validos al atributo type, modelamos el input al tipo asignado. A continuación vamos a ver el atributo type y el resto de atributos que puede tener una etiqueta input:

- **Type**. Es el que indica el tipo de elemento que vamos a definir. De él dependen el resto de parámetros puesto que el elemento cambia según sea de un tipo o de otro. Los valores posibles que acepta el atributo type son; text (cuadro de texto), password (cuadro de contraseña, los caracteres aparece ocultos tras asteriscos), checkbox (casilla de verificación), radio (opción de entre dos o más), submit (botón de envío del formulario), reset (botón de vaciado de campos), file (botón par a buscar ficheros), hidden (campo oculto para, el usuario no lo visualiza en el formulario), image (botón de imagen en el formulario), button (botón del formulario). Debido a las peculiaridades de cada uno de los tipos procederemos a explicar detalladamente cada tipo en el punto siguiente, indagando en cómo afectan al resto de atributos.

- **Name**. El atributo name asigna un nombre al elemento. Si no le asignamos nombre a un elemento, el servidor no podrá identificarlo y, por tanto, no podrá tener acceso al elemento.
- **Value**. El atributo value inicializa el valor del elemento. Los valores dependerán del tipo de dato, en ocasiones los posibles valores a tomar serán verdadero o falso.
- **Size**. Este atributo asigna el tamaño inicial del elemento. El tamaño se indica en pixeles. En los campos text y password hace referencia al número de caracteres.
- **Maxlength**. Este atributo indica el número máximo de caracteres que pueden contener los elementos text y password. Es conveniente saber que el tamaño de los campos text y password es independiente del número de caracteres que acepta el campo.
- **Checked**. Este atributo es exclusivo de los elementos checkbox y radio. En el definimos que opción por defecto queremos seleccionar.
- **Disable**. Este atributo hace que el elemento aparezca deshabilitado. En este caso el dato no se envía al servidor.
- **Readonly**. Este atributo sirve para bloquear el contenido del control, por tanto, el valor del elemento no se podrá modificar.
- **Src**. Este atributo es exclusivo para asignar una URL a una imagen que ha sido establecida como botón del formulario.
- **Alt**. El atributo alt incluye una pequeña descripción del elemento. Habitualmente, y si no lo hemos desactivado cuando posicionamos el ratón (sin pulsar ningún botón) encima del elemento, podemos visualizar la descripción del mismo.

1.4. Tipos de input

Como ya hemos visto en el punto anterior, la mutación que sufre cada tipo de input hace que sean elementos diferentes dentro del formulario. Por esta razón el estudio detallado de cada tipo, es necesario, para comprender las opciones que nos brindan los formularios. El objetivo de los tipos es adecuar los datos para facilitar la recopilación, manejo y envío de datos a través del formulario. También para la recuperación y comprensión de los mismos cuando el usuario recibe un formulario del servidor. A continuación vamos a detallar cada uno de estos tipos viendo un ejemplo de cada uno de ellos.

Cuadro de texto

El cuadro de texto muestra un cuadro de texto vacío en el que el usuario puede introducir un texto. Este es uno de los elementos más usados. La forma de indicar que es un campo de texto es la siguiente **type="text"**.

Para que el cuadro de texto pueda ser accedido deberá tener inicializado el atributo name, **name**="nombreCuadro ". El valor que indicamos en el atributo name es el que emplea el servidor para identificar este campo una vez se ha realizado el envío del formulario. El valor correspondiente a name deberá corresponder a un valor de variable como en cualquier otro lenguaje de programación. Respetando el no poner espacios en blanco, etc. El atributo valué inicializará con un valor el cuadro de texto, **value**="Javier". En el caso de que no mostremos el tamaño, el navegador mostrará el cuadro de texto con un tamaño predeterminado. De esta forma adaptaremos el tamaño del cuadro de

texto al tipo de dato que va a incluir. Con el atributo maxlength podemos limitar el número de caracteres que podrá incluir el usuario, así, realizamos una primera limitación a la hora de validar el formulario. Es muy útil a la hora de indicar que un dato como, por ejemplo, un teléfono, no permita introducir más de 9 caracteres. El atributo readonly permite que en el formulario visualicemos el cuadro de texto, pero no nos permite modificarlo. Este atributo es útil cuando realizamos una consulta de datos que residen en el servidor pero existen datos que no podemos modificar, por ejemplo el nombre de usuario a la hora de modificar el resto de datos. **Disabled** como el atributo anterior no permite modificar el campo de texto y además no envía los datos al servidor. A continuación vamos a ver cómo quedaría el código de una etiqueta input con los atributos anteriores:

```
<input type="text" name="nombre" value="Javier" readonly />
```

En el código anterior vemos un input de tipo text, que se llama nombre, está inicializado con la cadena de caracteres "Javier" y no se puede modificar por el usuario en el formulario.

Cuadro de contraseña.

El cuadro de contraseña es como el cuadro de texto, con la diferencia de que en el de contraseña, los caracteres que escribe el usuario no se ven en pantalla. En su lugar los navegadores muestran asteriscos o puntos con el fin de orientar al usuario del número de caracteres que va escribiendo. Su utilidad es escribir datos privados del usuario, como contraseñas u otros datos sensibles.

```
<input type="password" name="contrasenia" />
```

En el código anterior mostramos un input de tipo password que se llama contrasenía.

Los atributos indicados en el cuadro de texto pueden ser utilizados de igual forma en el cuadro de contraseña.

Casilla de verificación.

Estos elementos permiten al usuario activar o desactivar la selección de cada una de las casillas de forma individual. Su uso habitual es para indicar afirmación o negación sobre la casilla a la que se refiera.

```
<p>Colores  favoritos</p>
<input name="rojo" type="checkbox" value="ro"/> Rojo </br>
<input name="azul" type="checkbox" value="az"/> Azul </br>
<input name="verde" type="checkbox" value="ve"/> Verde </br>
```

En el código anterior mostramos los 3 input de tipo checkbox, estos se llaman rojo, azul y verde, los valores que enviarán al servidor en caso de ser verificados son ro, az, y ve, respectivamente.

En el caso de querer mostrar que uno de los input de tipo checkbox este seleccionado por defecto lo tendríamos que hacer con el atributo checked.

A continuación mostramos un ejemplo del código correspondiente:

```
<input name="rojo" type="checkbox" value="ro" checked />  Rojo
```

Opción de radio.

Este tipo de elemento agrupa una serie de opciones excluyentes entre sí. De esta forma el usuario solo puede coger una opción de entre todas las que tiene establecidas un grupo de botones radio. Al seleccionar una opción el usuario, automáticamente se deselecciona la que estaba seleccionada.

A continuación mostramos el código correspondiente la opción de radio que hemos mostrado anteriormente:

```
Género
</br>
<input type="radio" name="género" value="M"> Hombre </br>
<input type="radio" name="género" value="F"> Mujer
```

En el código anterior observamos que el nombre del input es el mismo. El nombre de los elementos que pertenezcan a un mismo grupo de opciones de radio deberá ser el mismo.

Es posible fijar un valor inicial con el atributo checked. En ese caso el input que deseemos que quede seleccionado por defecto deberá llevar dentro el atributo checked, **checked**="**checked**".

Botón de envío.

Este elemento es el encargado de enviar los datos del formulario al servidor. En este caso el type toma el valor **submit**. El valor del atributo value se mostrará en este caso en el botón generado. El botón enviará los datos del formulario a la dirección que esté dispuesta en el action.

Botón de reset.

Este elemento es un botón que establece el formulario a su estado original. En este botón no hemos añadido el atributo value. Es el navegador el encargado de asignar un valor por defecto, el valor por defecto en el caso anterior es "Restablecer".

Ficheros adjuntos.

Este tipo de **input** permite adjuntar **ficheros adjuntos**. Las limitaciones sobre cuántos ficheros y el tamaño no están definidas por el elemento. Por lo tanto, deberá controlarse desde otra parte del código para evitar desbordamientos o colapsos en el servidor. El elemento añade de forma automática un cuadro de texto que se dispondrá para almacenar la dirección del fichero adjunto seleccionado. Pulsando un botón situado a la derecha del cuadro de texto navegaremos por el árbol de directorios en busca del fichero a adjuntar y una vez seleccionado, éste se incluirá en el cuadro que hemos indicado antes.

```
<input type="file" name="fichero"/>
```

Vemos como en el código anterior no es necesario incluir nada más que el tipo de input como file para que en la Figura nos muestre el campo de texto y el botón de examinar.

Es importante saber que para adjuntar archivos debemos indicar el tipo de envío que vamos a realizar. Esto lo indicamos en la estructura del form, inicializando el atributo enctype. Habitualmente el valor que toma este atributo es **multipart/form-data**. Por lo tanto, el código en la cabecera del formulario quedaría de la siguiente manera.

```
<form action="..." method="post" enctype="multipart/form-data">
</form>
```

Campos ocultos.

Los campos ocultos no son visibles en el formulario por el usuario. Estos elementos son útiles para enviar información de forma oculta que no tenga que ser tratada por el usuario. Son usadas a menudo para que el servidor pueda tratar la información o hacer alguna acción determinada.

```
<input type="hidden" name="campoOculto" value="cambiar"/>
```

En el código anterior vemos como el campo con nombre campoOculto, toma el valor cambiar. Esto podría usarse para que el servidor realice un cambio con respecto a algo.

Botón de imagen.

Este elemento es una personalización de un botón, cambiando el aspecto por defecto que tienen los botones de un formulario por una imagen. El aspecto del botón sería el de la imagen a la que hayamos hecho referencia.

```
<input type="image" name="enviar" src="imagen mundo.jpg"/>
```

El elemento botón de imagen se tipa igualando a image el tipo de input como vemos en el código anterior.

Botón.

Existe un elemento botón, al que podemos asociar diferentes funcionalidades. De esta forma no nos tenemos que ceñir los botones de **submit** o **reset** que nos ofrecen los formularios.

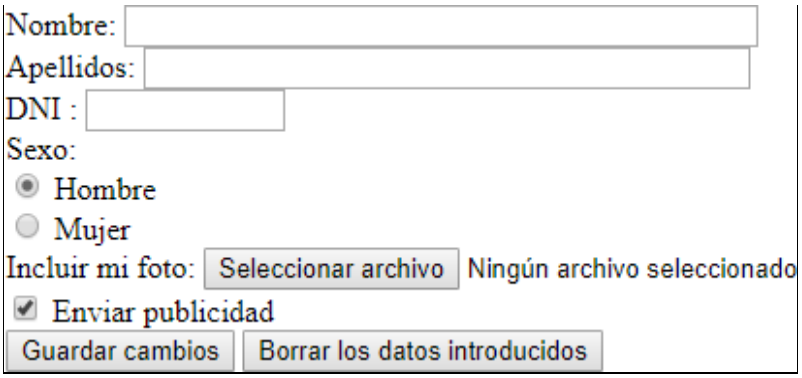
El botón podrá tener cualquier texto en su interior a través de la inicialización del atributo value.

```
<input type="button" name="opcion" value="Opcion validar"/>
```

Como vemos en el código anterior inicializamos el atributo type al valor button, de esta manera el elemento se convierte en un botón. Estos botones por sí mismos no tienen funcionalidad. Habrá que aplicar una funcionalidad por medio de JavaScript para que se accione alguna tarea.

A continuación vamos a mostrar el código de un formulario con varios componentes para ver cómo se integran los componentes del mismo entre sí.

```
<form action="pagina .php" method="post" enctype="multipart/form-data" />
  <br/>
  Nombre: <input type="text" name= "nombre" value="" size="42 maxlength="30"
/> <br/>
  Apellidos: <input type="text" name="apellidos" value="" size="40"
maxlength="80" />
  <br/>
  DNI : <input type="text" name="dni" value="" size="10" maxlength="9" />
  <br/> Sexo: <br/>
  <input type="radio" name="sexo" value="hombre" checked="checked" /> Hombre
<br/>
  <input type="radio" name="sexo" value="mujer" /> Mujer  <br/>
  Incluir mi foto: <input type="file" name="foto" /> <br/>
  <input name="publicidad" type="checkbox" value="publicidad"
checked="checked"/> Enviar publicidad <br/>
  <input type="submit" name="enviar" value="Guardar cambios">
  <input type="reset" name="limpiar" value="Borrar los datos introducidos" />
</form>
```



Formulario completo

1.5. Modificación del comportamiento de un formulario

Los formularios tienen unas acciones predeterminadas por defecto. Estas acciones están asociadas a los elementos así como a la estructura principal del formulario. En el caso de que queramos darle otro comportamiento a un elemento del formulario, tenemos que modificar el elemento para que este no realice su función habitual. Imaginemos que tenemos la estructura principal de un formulario pero queremos que los datos se envíen a URL diferentes dependiendo de un dato que introduzca el usuario. En este caso deberíamos modificar el atributo action indicado en la estructura principal. Para realizar el envío a la URL podríamos accionar en un script la petición al servidor. A continuación mostramos un ejemplo de cómo se enviaría el formulario a una URL diferente:

```
function enviar(form)
{
  if (form.alta.checked == true)
  {
    form.action = "paginas/alta.html";
  }
  if (form.alta.checked == false)
  {
    form.action = "paginas/baja.html";
  }
  form.submit()
}
```

En el código anterior comprobamos si la casilla alta ha sido chequeada. En caso de haber sido pulsada el action del form se inicia a un valor, si no ha sido pulsada se inicia a otro. Al final se envía el formulario por medio de un submit. Para que la función enviar sea ejecutada, debemos incluir un input de tipo button que contenga el evento onclick. En el manejador de este evento realizaremos la llamada de la siguiente forma: onclick="enviar (this.form) ". El objeto this hace que podamos enviar el formulario a través de la función. De esta forma nunca llamamos al action de la estructura principal del formulario.

1.6. Acceso a los campos más importantes

Cuadro de texto y textarea

El valor del texto mostrado por estos elementos se obtiene y se establece directamente mediante la propiedad value.

```
<input type="text" id="texto" />
var valor = document.getElementById("texto").value;


<textarea id="parrafo"></textarea>
var valor = document.getElementById("parrafo").value;
```

Radiobutton

Cuando se dispone de un grupo de radiobuttons, generalmente no se quiere obtener el valor del atributo value de alguno de ellos, sino que lo importante es conocer cuál de todos los radiobuttons se ha seleccionado.

La propiedad checked devuelve true para el radiobutton seleccionado y false en cualquier otro caso. Si por ejemplo se dispone del siguiente grupo de radiobuttons:

```
<input type="radio" value="si" name="pregunta" id="pregunta_si"/> SI
<input type="radio" value="no" name="pregunta" id="pregunta_no"/> NO
<input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc"/>
NS/NC
```

El siguiente código permite determinar si cada radiobutton ha sido seleccionado o no:

```
var elementos = document.getElementsByName("pregunta");
for(var i=0; i<elementos.length; i++)
{
  alert("Elemento:" + elementos[i].value + "\nSeleccionado:"+
elementos[i].checked);
}
```

Checkbox

Los elementos de tipo checkbox son muy similares a los radiobutton, salvo que en este caso se debe comprobar cada checkbox de forma independiente del resto. El motivo es que los grupos de radiobutton son mutuamente excluyentes y sólo se puede seleccionar uno de ellos cada vez. Por su parte, los checkbox se pueden seleccionar de forma independiente respecto de los demás.

Si se dispone de los siguientes checkbox:

```
<input type="checkbox" value="condiciones" name="condiciones"
id="condiciones"/> He leído y acepto las condiciones
<input type="checkbox" value="privacidad" name="privacidad"
id="privacidad"/> He leído la política de privacidad
```

Utilizando la propiedad checked, es posible comprobar si cada checkbox ha sido seleccionado:

```
var elemento = document.getElementById("condiciones");

alert(" Elemento: " + elemento.value + "\n Seleccionado: " +
elemento.checked);

elemento = document.getElementById("privacidad");
alert(" Elemento: " + elemento.value + "\n Seleccionado: " +
elemento.checked);
```

Select

Las listas desplegables (<select>) son los elementos en los que es más difícil obtener su valor. Si se dispone de una lista desplegable como la siguiente:

```
<select id="opciones" name="opciones">
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
  <option value="4">Cuarto valor</option>
</select>
```

En general, lo que se requiere es obtener el valor del atributo value de la opción (<option>) seleccionada por el usuario. Obtener este valor no es sencillo, ya que se deben realizar una serie de pasos. Además, para obtener el valor seleccionado, deben utilizarse las siguientes propiedades:

- **options**, es un array creado automáticamente por el navegador para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista. De esta forma, la primera opción de una lista se puede obtener mediante document.getElementById("id_de_la_lista").options[0].
- **selectedIndex**, cuando el usuario selecciona una opción, el navegador actualiza automáticamente el valor de esta propiedad, que guarda el índice de la opción seleccionada. El índice hace referencia al array options creado automáticamente por el navegador para cada lista.

```
// Obtener la referencia a la lista
var lista = document.getElementById("opciones");

// Obtener el índice de la opción que se ha seleccionado
var indiceSeleccionado = lista.selectedIndex;
// Con el índice y el array "options", obtener la opción seleccionada
var opcionSeleccionada = lista.options[indiceSeleccionado];

// Obtener el valor y el texto de la opción seleccionada
var textoSeleccionado = opcionSeleccionada.text;
var valorSeleccionado = opcionSeleccionada.value;

alert("Opción seleccionada: " + textoSeleccionado + "\n Valor de la
opción: " + valorSeleccionado);
```

Como se ha visto, para obtener el valor del atributo value correspondiente a la opción seleccionada por el usuario, es necesario realizar varios pasos. No obstante, normalmente se abrevian todos los pasos necesarios en una única instrucción:

```
var lista = document.getElementById("opciones");

// Obtener el valor de la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].value;

// Obtener el texto que muestra la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].text;
```

Lo más importante es no confundir el valor de la propiedad selectedIndex con el valor correspondiente a la propiedad value de la opción seleccionada. En el ejemplo anterior, la primera opción tiene un value igual a 1. Sin embargo, si se selecciona esta opción, el valor de selectedIndex será 0, ya que es la primera opción del array options (y los arrays empiezan a contar los elementos en el número 0).

1.7. Establecer el foco en un elemento

Cuando un elemento está seleccionado y se puede escribir directamente en él o se puede modificar alguna de sus propiedades, se dice que tiene el foco del programa.

Si un cuadro de texto de un formulario tiene el foco, el usuario puede escribir directamente en el sin necesidad de pinchar previamente con el ratón en el interior del cuadro. Igualmente, si una lista desplegable tiene el foco, el usuario puede seleccionar una opción directamente subiendo y bajando con las flechas del teclado.

Al pulsar repetidamente la tecla **TABULADOR** sobre una página web, los diferentes elementos (enlaces, imágenes, campos de formulario, etc.) van obteniendo el foco del navegador (el elemento seleccionado cada vez suele mostrar un pequeño borde punteado).

Si en una página web el formulario es el elemento más importante, como por ejemplo en una página de búsqueda o en una página con un formulario para registrarse, se considera una buena práctica de usabilidad el asignar automáticamente el foco al primer elemento del formulario cuando se carga la página.

Para asignar el foco a un elemento de HTML, se utiliza la función **focus()**. El siguiente ejemplo asigna el foco a un elemento de formulario cuyo atributo id es igual a primero:

```
document.getElementById("primero").focus();
```

```
<form id="formulario" action="#">
  <input type="text" id="primero" />
</form>
```

Ampliando el ejemplo anterior, se puede asignar automáticamente el foco del programa al primer elemento del primer formulario de la página, independientemente del id del formulario y de los elementos:

```
if(document.forms.length > 0)
{
  if(document.forms[0].elements.length > 0)
  {
    document.forms[0].elements[0].focus();
  }
}
```

El código anterior comprueba que existe al menos un formulario en la página mediante el tamaño del array forms. Si su tamaño es mayor que 0, se utiliza este primer formulario.

Empleando la misma técnica, se comprueba que el formulario tenga al menos un elemento (**if(document.forms[0].elements.length > 0**)). En caso afirmativo, se establece el foco del navegador en el primer elemento del primer formulario (**document.forms[0].elements[0].focus();**).

Para que el ejemplo anterior sea completamente correcto, se debe añadir una comprobación adicional. El campo de formulario que se selecciona no debería ser de tipo **hidden**:

```
if(document.forms.length > 0)
{
  for(var i=0; i < document.forms[0].elements.length; i++)
  {
    var campo = document.forms[0].elements[i];
    if(campo.type != "hidden")
    {
      campo.focus();
      break;
    }
  }
}
```

1.8. Evitar el envío duplicado de un formulario

Uno de los problemas habituales con el uso de formularios web es la posibilidad de que el usuario pulse dos veces seguidas sobre el botón "Enviar". Si la conexión del usuario es demasiado lenta o la respuesta del servidor se hace esperar, el formulario original sigue mostrándose en el navegador y por ese motivo, el usuario tiene la tentación de volver a pinchar sobre el botón de "Enviar".

En la mayoría de los casos, el problema no es grave e incluso es posible controlarlo en el servidor, pero puede complicarse en formularios de aplicaciones importantes como las que implican transacciones económicas.

Por este motivo, una buena práctica en el diseño de aplicaciones web suele ser la de deshabilitar el botón de envío después de la primera pulsación. El siguiente ejemplo muestra el código necesario:

```
<form id="formulario" action="#">
...
<input type="button" value="Enviar" onclick="this.disabled=true;
this.value='Enviando...'; this.form.submit()" />
</form>
```

Cuando se pulsa sobre el botón de envío del formulario, se produce el evento onclick sobre el botón y por tanto, se ejecutan las instrucciones JavaScript contenidas en el atributo onclick:

En primer lugar, se deshabilita el botón mediante la instrucción this.disabled = true;. Esta es la única instrucción necesaria si sólo se quiere deshabilitar un botón.

A continuación, se cambia el mensaje que muestra el botón. Del original "**Enviar**" se pasa al más adecuado **"Enviando..."**

Por último, se envía el formulario mediante la función **submit ()** en la siguiente instrucción: **this.form.submit()**

El botón del ejemplo anterior está definido mediante un botón de **tipo <input type="button" />**, ya que el código JavaScript mostrado no funciona correctamente con un botón de tipo **<input type="submit" />**. Si se utiliza un botón de tipo submit, el botón se deshabilita antes de enviar el formulario y por tanto el formulario acaba sin enviarse.

1.9. Limitar el tamaño de caracteres de un textarea

Se comprueba si se ha llegado al máximo número de caracteres permitido y en caso afirmativo se evita el comportamiento habitual del evento y por tanto, los caracteres adicionales no se añaden al textarea:

```
function limita(maximoCaracteres)
{
  var elemento = document.getElementById("texto");
  if(elemento.value.length >= maximoCaracteres )
  {
    return false;
  }
  else
  {
    return true;
  }
}
```

```
<textarea id="texto" onkeypress="return limita(100);"></textarea>
```

En el ejemplo anterior, con cada tecla pulsada se compara el número total de caracteres del textarea con el máximo número de caracteres permitido. Si el número de caracteres es igual o mayor que el límite, se devuelve el valor false y por tanto, se evita el comportamiento por defecto de onkeypress y la tecla no se añade.

1.10. Restringir los caracteres permitidos en un cuadro de texto

En ocasiones, puede ser útil bloquear algunos caracteres determinados en un cuadro de texto. Si por ejemplo un cuadro de texto espera que se introduzca un número, puede ser interesante no permitir al usuario introducir ningún carácter que no sea numérico.

Igualmente, en algunos casos puede ser útil impedir que el usuario introduzca números en un cuadro de texto. Utilizando el evento onkeypress y unas cuantas sentencias JavaScript, el problema se resuelve fácilmente:

```
function permite(elEvento, permitidos)
{
  // Variables que definen los caracteres permitidos
  var numeros = "0123456789";
  var caracteres =
"abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
  var numeros_caracteres = numeros + caracteres;
  var teclas_especiales = [8, 37, 39, 46];

  // 8= BackSpace, 46= Supr, 37= flecha izquierda, 39= flecha derecha
  // Seleccionar los caracteres a partir del parámetro de la función
  switch(permitidos)
  {
    case 'num':
      permitidos = numeros;
      break;
    case 'car':
      permitidos = caracteres;
      break;
    case 'num_car':
      permitidos = numeros_caracteres;
      break;
  }

  // Obtener la tecla pulsada
  var evento = elEvento || window.event;
  var   codigoCaracter  =   evento.charCode   ||  evento.keyCode;
  var   caracter  = String.fromCharCode(codigoCaracter);

  // Comprobar si la tecla pulsada es alguna de las teclas especiales
  // (teclas de borrado y flechas horizontales)
  var tecla_especial = false;
  for(var i in teclas_especiales)
  {
    if(codigoCaracter == teclas_especiales[i])
    {
      tecla_especial = true;
      break;
    }
  }
 // Comprobar si la tecla pulsada se encuentra en los caracteres
// permitidos  o si es una tecla especial
  return permitidos.indexOf(caracter) != -1 || tecla_especial;
}


<!-- Sólo números -->
<input type="text" id="texto" onkeypress="return permite(event,
'num')" />

<!--  Sólo letras -->
<input type="text" id="texto" onkeypress="return permite(event,
'car')" />

<!--  Sólo letras o números -->
<input type="text" id="texto" onkeypress="return permite(event,
'num_car')" />
```

El funcionamiento del script anterior se basa en permitir o impedir el comportamiento habitual del evento onkeypress. Cuando se pulsa una tecla, se comprueba si el carácter de esa tecla se encuentra dentro de los caracteres permitidos para ese elemento <**input**>.

Si el carácter se encuentra dentro de los caracteres permitidos, se devuelve true y por tanto el comportamiento de onkeypress es el habitual y la tecla se escribe. Si el carácter no se encuentra dentro de los caracteres permitidos, se devuelve false y por tanto se impide el comportamiento normal de onkeypress y la tecla no llega a escribirse en el input.

Además, el script anterior siempre permite la pulsación de algunas teclas especiales. En concreto, las teclas BackSpace y Supr para borrar caracteres y las teclas Flecha Izquierda y Flecha Derecha para moverse en el cuadro de texto siempre se pueden pulsar independientemente del tipo de caracteres permitidos.

2. Validación y envío de formularios

A la hora de enviar un formulario, no podemos garantizar que el usuario haya insertado cada dato del formulario como esperamos. Supongamos que en un campo de texto el usuario tiene que incluir un código postal, pero el usuario en ese campo ha incluido una cadena de texto, por ejemplo, Madrid. Si el formulario se envía con ese dato se producirá un error. Para controlar si los campos de un formulario han sido rellenados correctamente haremos uso de las validaciones. Existen varios tipos de validaciones. Podemos realizar validaciones en el servidor, a nivel de servidor y también en la base de datos. Realizar validaciones a nivel de servidor supone que se recibe una petición, el servidor tiene que procesarla y emitir una respuesta. En el caso de que el usuario incluya muchos datos que no son correctos el servidor se puede ver sobrecargado. Para solucionar esta sobrecarga del servidor, realizamos un primer nitrado de los datos en el cliente. Estas validaciones son las que vamos a ver a continuación.

Este tipo de validaciones se suelen realizar llamando a una función que analice si el dato cumple con las restricciones que se han impuesto como, por ejemplo, en el caso anterior, donde el dato tenía que cumplir con los caracteres que puede tener un código postal. A continuación vamos a ver algunos de los ejemplos de validación más habituales, los tipos de validación que pueden existir son de lo más variado y basta con definirlas dentro de cada una de las funciones que valida un dato.

2.1. Estructura del form para validar datos

Para que el formulario detecte que tiene que realizar una validación de los datos antes de enviar la petición al servidor, debemos indicárselo en su estructura. Para ello utilizaremos el evento onsubmit que ejecuta una acción cuando el action del formulario es llamado. Dentro del evento onsubmit, debemos llamar a una función, dependiendo de la respuesta de la función, el formulario enviará los datos al servidor (si los datos son correctos) o mostrará los diferentes mensajes de error, para los controles del formulario que se hayan realizado validaciones. El código correspondiente sería el siguiente:

```
<form action="URL" method="post" name="formValidado"
onsubmit="return validar()">
</form>
```

En el código anterior observamos como la función **validar ()** es llamada dentro del evento **onsubmit** y que dependiendo de la respuesta de validar el servidor envía o no los datos del formulario al servidor. Al realizarse un submit desde algún botón del formulario (petición de envío del formulario al servidor), siempre se realiza la llamada a la función validar () . Para que los datos del formulario sean validados, debemos tener implementada la función validar (). En esta función indicaremos cuáles son los datos del formulario que vamos a validar. A continuación mostramos algunos ejemplos de cómo validar datos de un formulario.

Validar un campo como obligatorio.

En ocasiones puede que nos interese que el usuario ingrese un campo como obligatorio en la aplicación web. En esta situación si el usuario no introduce el dato y antes de que el formulario sea enviado debemos comprobar si el campo está vacío.

Habiendo llamado en la estructura del form a una función, la implementación de la función será la siguiente:

```
function validación()
{
  valor = document.getElementById("campo").value;
  if( valor == null || valor.length == 0 )
  {
    alert("El campo no puede ser vacio");
    return false;
  }
  return true;
}
```

Este código deber ir integrado en la cabecera de la página **HTML (<head>...código JavaScript...</head>)**. En primer lugar, recogemos el valor a través del objeto document, el nombre del input es campo. En la estructura if comprobamos que el valor no sea nulo y que su longitud no sea cero. En caso de que el valor del campo input sea nulo o igual a cero, mostramos un mensaje de alerta y, posteriormente, devolvemos falso para que sea recuperado por el evento onsubmit. En ese caso no se enviaría el formulario.

Validar un campo de texto como numérico.

A menudo en los formularios nos encontramos con campos de texto que solo admiten números. Los teléfonos, código postal, DNI (sin letra) y otros muchos datos que obligatoriamente tienen una nomenclatura numérica. Para advertir al usuario de que el dato que ha introducido no es correcto utilizaremos el siguiente código implementado en una función:

```
function validaNum()
{
  valor = document.getElementById("telefono").value;
  if( isNaN(valor) )
  {
    alert("El campo tiene que ser numérico");
    return false;
  }
  return true;
}
```

En el código anterior, tenemos una variable que se llama valor. Esta variable tomará el valor del teléfono o en su defecto el dato que haya introducido el usuario en la aplicación web. Con la función **isNaN ()** comprobamos si el valor es numérico. En caso de que no lo sea devolverá un mensaje de alerta indicando que el dato tiene que ser numérico. Si por el contrario el dato es numérico la función devolverá **verdadero**.

Validar un checkbox.

Otra de las validaciones que podemos necesitar habitualmente, es comprobar si un checkbox ha sido seleccionado. A menudo en la aceptación de ciertas condiciones para acceder a servicios nos encontramos con esta comprobación, sin la que el formulario no tiene mucho sentido que continúe. En el siguiente código vamos a ver cómo se implementa la comprobación de si se ha activado un **checkbox**:

```
function validaCheck()
{
  elemento = document.getElementById("campoCondiciones");
  if( !elemento.checked )
  {
    return false;
  }
  return true;
}
```

En el código anterior recuperamos el valor que ha tomado el elemento checkbox del formulario. En la siguiente condición comprobamos si éste ha sido chequeado, en caso de que no se haya pulsado la función devolverá falso. Si se ha chequeado la función devuelve verdadero.

Conviene tener en cuenta que si somos demasiado estrictos a la hora de validar o damos poca información, podemos bloquear al usuario de la aplicación web.

Validar la selección de una opción en 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.

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;
    }
  }
}
```

Validar que una radiobutton se ha 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.

```
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.

3. Expresiones regulares

Las expresiones regulares describen un conjunto de elementos que siguen un patrón. Dicho de otra forma, es una regla que identifica a una serie de elementos que tiene algo en común. Un ejemplo de expresión regular podrían ser todas las palabras que comienzan por la letra "a" minúscula. La expresión regular para este patrón debe asegurarse de que la palabra comienza por "a" minúscula, el resto de palabras que precedan a la cadena podrán ser cualquier carácter.

La mayoría de los lenguajes de programación incluyen la implementación de expresiones regulares. En el caso que nos atañe, JavaScript implementa una configuración para implementar expresiones regulares y así facilitar las comprobaciones de ciertos datos que deben seguir una estructura concreta. Este tipo de expresiones es muy útil a la hora de evaluar algunos tipos de datos que normalmente utilizamos en los formularios.

3.1. Caracteres especiales de las expresiones regulares

A continuación vamos describir algunos de los elementos que utiliza JavaScript para crear expresiones regulares. En el siguiente apartado vamos a representar la simbología que nos presta la expresión regular y, posteriormente, significado o interpretación:

^ **Principio de entrada o línea**. Este carácter indica que las cadenas deberán comenzar por el siguiente carácter. Si éste fuera una "a" minúscula, como indicamos en el punto anterior, la expresión regular seria sería: ^a.

$ **Fin de entrada o línea**. Indica que la cadena debe terminar por el elemento precedido al dólar.

***** **El carácter anterior 0 o más veces**. El asterisco indica que el carácter anterior se puede repetir en la cadena 0 o más veces.

+ **El carácter anterior 1 o más veces**. El símbolo más indica que el carácter anterior se puede repetir en la cadena una o más veces.

? **El carácter anterior una vez como máximo**. El símbolo interrogación indica que el carácter anterior se puede repetir en la cadena cero o una vez.

. **Cualquier carácter individual**. El símbolo punto indica que puede haber cualquier carácter individual salvo el de salto de línea.

x | y **x ó y**. La barra vertical indica que puede ser el carácter x o el y.

{n} **n veces el carácter anterior**. El carácter anterior a las llaves tiene que aparecer exactamente n veces.

{n,m} **Entre n y m veces el carácter anterior**. El carácter anterior a las llaves tiene que aparecer como mínimo n y como máximo m veces.

[abc] **Cualquier carácter de los corchetes**. En la cadena puede aparecer cualquier carácter que esté incluido en los corchetes. Además, podemos especificar rangos de caracteres que sigan un orden. Si se especifica el rango [a-z] equivaldría a incluir todas las letras minúsculas del abecedario.

[^abc] **Un carácter que no esté en los corchetes**. En la cadena pueden aparecer todos los caracteres que no estén incluidos en los corchetes. También podemos especificar rangos de caracteres como en el punto anterior.

\b **Fin de palabra**. El símbolo \b indica que tiene que haber un fin de palabra o retorno de carro.

\B **No fin de palabra**. El símbolo \B indica cualquiera que no sea un límite de palabra.

\d **Cualquier carácter dígito**. El símbolo \d indica que puede haber cualquier carácter numérico, de 0 a 9.

\D **Carácter que no es dígito**. El símbolo \D indica que puede haber cualquier carácter siempre que no sea numérico.

\f **Salto de página.** El símbolo \f indica que tiene que haber un salto de página.

\n **Salto de línea**. El símbolo \n indica que tiene que haber un salto de línea.

\r **Retorno de carro**. El símbolo \r indica que tiene que haber un retorno de carro.

\s **Cualquier espacio en blanco**. El símbolo \s indica que tiene que haber un carácter individual de espacio en blanco: espacios, tabulaciones, saltos de página o saltos de línea.

\S **Carácter que no sea blanco**. El símbolo \S indica que tiene que haber cualquier carácter individual que no sea un espacio en blanco.

\t **Tabulación**. El símbolo \t indica que tiene que haber cualquier tabulación.

\w **Carácter alfanumérico**. El símbolo \w indica que puede haber cualquier carácter alfanumérico, incluido el de subrayado.

\W **Carácter que no sea alfanumérico**. El símbolo \W indica que puede haber cualquier carácter que no sea alfanumérico.

3.2. Validar un formulario con expresiones regulares

Con la combinación de estas expresiones podemos abordar infinidad de patrones para validar datos en la mayoría de los formularios. Combinando cada una de las condiciones obtenemos patrones como pueden ser, una dirección de correo electrónico, un teléfono, un código postal, un DNI, etc. A continuación, vamos a ver cómo realizar algunas de las configuraciones de expresiones regulares más utilizadas para validar datos de un formulario.

Validar una dirección de correo electrónico.

En el caso de que el usuario de la aplicación web introduzca una dirección de correo electrónico, es casi imprescindible que esta sea correcta. Utilizando expresiones regulares vamos a comprobar que la estructura de la dirección sea correcta. En primer lugar, comprobaremos que comienza por una cadena de texto, que sigue por una @ y que termina por otra cadena de texto un punto y otra cadena de texto. De esta forma aseguraremos que, al menos, la estructura de la dirección es correcta. Combinando las funciones con las expresiones regulares, obtenemos un código que valida una dirección de correo electrónico.

La llamada en la estructura principal del form sigue siendo la misma que en los casos anteriores y recogemos en el evento onsubmit la respuesta de la llamada a la función que valida, en nuestro caso la dirección de correo electrónico. El código correspondiente a la función JavaScript que valida un correo electrónico sería el siguiente:

```
function validaEmail()
{
  valor = document.getElementById("campo").value;
  if(!(/\w+([-+.´]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)/.test (valor)) )
  {
    return false;
  }
  return true;
}
```

Validar un DNI.

Un dato muy habitual en los formularios es el documento nacional de identidad. A la hora de que el usuario interactúe con la aplicación web, es necesario comprobar que el DNI que introduce es correcto. No podemos asegurar que el DNI sea el de la persona que se está identificando, pero sí podemos asegurar que el DNI que introduce el usuario es correcto. Para validad un DNI introducido podemos utilizar el siguiente código.

```
function validaDNI()
{
  valor = document.getElementById("dni").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'];
  if( !(/^\d{8}[A-Z]$/.test(valor)) )
  {
    return false;
  }
  if(valor.charAt(8) != letras[(valor.substring(0, 8))%23])
  {
    return false;
  }
  return true;
}
```

En el código anterior validamos un DNI en varios pasos. Primero recogemos en la variable valor el campo del DNI que ha introducido el usuario. Generamos un array con las letras válidas para el DNI. En la primera condición comprobamos a través de expresiones regulares que el patrón del campo introducido tiene una longitud de 8 números y una letra. Si no es así devuelve falso la función.

En la última condición realizamos la división entre 23 y tomamos el resto. Comprobamos que la posición del array letras, perteneciente al resto de la operación, se corresponde con el valor de la letra del DNI introducido. En caso de que sea igual devuelve verdadero (al final de la función). Si no es igual la letra del valor introducido a la calculada, entra en la condición y devuelve falso.

Validar un número de teléfono.

Otro dato que es importante que sea correcto para interactuar en una aplicación web con el usuario son los números de teléfono. Al igual que en casos anteriores, no podemos comprobar que el numero pertenezca al usuarios que lo introduce, pero si podemos asegurar que el número de teléfono tiene un formato correcto. Las expresiones regulares son un recurso muy útil para validar un teléfono. A continuación vamos a ver cómo validaremos un número de teléfono:

```
function valídaTelefono()
{
  valor = document.getElementById("telefono").value;
  var expresionregular = /^\d{9}$/;
  if( ! (expresionregular.test(valor) ) )
  {
    return false;
  }
  return true;
}
```