# DWC

# (Desarrollo Web en entorno cliente)

# Índice

1.- Formularios ..... 1
1.1.- Estructura básica de un formulario HTML ..... 1
2.- Interacción con formularios usando JavaScript ..... 3
2.1 Acceso a los elementos del formulario ..... 3
2.2 Gestión de eventos en formularios ..... 4
2.3 Validación de formularios con JavaScript ..... 6
2.3.1. Expresiones regulares (regex) ..... 6
3.- Controles de formulario ..... 9
3.1.- Cuadro de texto y textarea ..... 9
3.2.- Radiobutton ..... 9
3.3.- Checkbox ..... 10
3.4.- Select ..... 11
3.4.1.- Acceso a cualquier elemento del Select ..... 11
3.4.2.-Añadir y borrar elementos en un Select ..... 13

# 1.- Formularios

Los formularios HTML son elementos fundamentales en la creación de páginas web interactivas. Permiten a los usuarios introducir datos, realizar selecciones y enviar información al servidor para su procesamiento. Sin embargo, los formularios básicos HTML pueden resultar limitados en cuanto a la funcionalidad y la experiencia del usuario. Es aquí donde JavaScript entra en juego, proporcionando herramientas para mejorar la interacción con los formularios y añadir características avanzadas.

Los formularios HTML son secciones de una página web que contienen campos donde los usuarios pueden introducir información. Estos campos pueden ser de diferentes tipos, como campos de texto, casillas de verificación, botones de opción, listas desplegables...

Los datos introducidos por el usuario se envían al servidor a través de un proceso llamado "envío del formulario".

Los formularios se utilizan para una amplia variedad de propósitos en sitios web, incluyendo:

- **Registro de usuarios:** Recopilar información personal de los usuarios para crear cuentas.
- **Contacto:** Permitir a los usuarios enviar mensajes al personal del sitio web.
- **Búsqueda:** Facilitar a los usuarios encontrar información específica en el sitio web.
- **Compras en línea:** Procesar pedidos de productos y pagos de clientes.
- **Comentarios:** Permitir a los usuarios dejar sus opiniones sobre productos o servicios.

## 1.1.- Estructura básica de un formulario HTML

Un formulario HTML básico se define utilizando la etiqueta **<form>.** Dentro de esta etiqueta, **se incluyen los diferentes elementos del formulario**, como campos de entrada, botones y etiquetas.

La estructura básica de un formulario HTML es la siguiente:

```
<form name="registro">

    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre">

    <label for="correo">Correo electrónico:</label>
    <input type="email" id="correo" name="correo">

    <button type="submit">Enviar</button>
</form>
```

En este ejemplo, tenemos un formulario simple con dos campos de entrada: uno para el nombre del usuario y otro para su correo electrónico. También hay un botón de envío que, cuando se hace clic, enviará los datos del formulario al servidor.

**Los formularios HTML pueden contener una variedad de elementos**, cada uno con un propósito específico. Algunos de los elementos más comunes son:

- **Campos de entrada (<input>):** Permiten a los usuarios introducir datos de texto, números, fechas, contraseñas, etc.
- **Casillas de verificación (<input type="checkbox">):** Permiten a los usuarios seleccionar múltiples opciones.
- **Botones de opción (<input type="radio">):** Permiten a los usuarios seleccionar una única opción de un grupo.
- **Menús desplegables (<select>):** Permiten a los usuarios seleccionar una opción de una lista.
- **Botones (<button>):** Inician una acción cuando se hace clic, como enviar el formulario o restablecer los campos.
- **Etiquetas (<label>):** Proporcionan texto descriptivo para los campos de entrada.

**Los elementos del formulario pueden tener una serie de atributos** que definen su comportamiento y apariencia. Algunos de los atributos más importantes son:

- **id:** Un identificador único para el elemento.

- **name:** El nombre del elemento, utilizado para enviar los datos del formulario al servidor.
- **type:** El tipo de elemento, como text, email, checkbox, radio, etc.
- **value:** El valor inicial del elemento.
- **required:** Indica si el campo es obligatorio.
- **placeholder:** Texto de marcador que se muestra dentro del campo cuando está vacío.

**Los formularios HTML son elementos esenciales para la interacción con los usuarios en sitios web. JavaScript proporciona herramientas para mejorar la funcionalidad y la experiencia del usuario con los formularios, permitiendo la validación de datos, la creación dinámica de formularios, la integración con servicios web y mucho más.**

## 2. Interacción con formularios usando JavaScript

JavaScript permite interactuar con los formularios HTML de forma dinámica y agregar funcionalidades que no son posibles con HTML puro. Esto mejora la experiencia del usuario, haciendo que los formularios sean más intuitivos y eficientes.

## 2.1 Acceso a los elementos del formulario

Para interactuar con los elementos de un formulario en JavaScript, primero hay que acceder a ellos.

Esto se puede hacer utilizando diferentes métodos, como:

- **document.getElementById(id):** Obtiene un elemento por su ID.
- **document.querySelector(selector):** Obtiene un elemento que coincida con un selector CSS.
- **document.forms[nombre]:** Obtiene un formulario por su nombre.
- **form.elements[nombre]:** Obtiene un elemento dentro de un formulario por su nombre.

Un ejemplo de acceso podría ser:

```
<script>
    const nombreInput = document.getElementById('nombre');
    const emailInput = document.querySelector('input[name="correo"]');
    const formulario = document.forms['registro'];
    const botonEnviar = formulario.elements['enviar'];
</script>
```

Una vez que se tenemos acceso a un elemento del formulario, podemos obtener su valor actual o modificarlo. Para obtener el valor, se utiliza la propiedad **value**. Para modificarlo, se asigna un nuevo valor a la propiedad value.

## 2.2 Gestión de eventos en formularios

Los eventos son acciones que ocurren en un elemento o en la página web. En el contexto de los formularios, algunos eventos comunes son:

- **submit:** Se dispara cuando se envía el formulario.
- **change:** Se dispara cuando se cambia el valor de un campo.
- **focus:** Se dispara cuando un campo recibe el foco.
- **blur:** Se dispara cuando un campo pierde el foco.

Para gestionar eventos en formularios, se utilizan los manejadores de eventos. Estos manejadores son funciones que se ejecutan cuando se produce el evento correspondiente.
Se pueden asociar a los elementos del formulario utilizando diferentes métodos, como:

- **on<event>:** agrega un manejador DOM
- **addEventListener:** Agrega un manejador de eventos a un elemento.
- **removeEventListener:** Elimina un manejador de eventos de un elemento.

```
nombreInput.addEventListener('change', function() {
                    const nombre = nombreInput.value;
                    console.log('El nombre ha cambiado a:', nombre);
                    });
```

En este caso estamos comprobando el valor escrito en el campo nombre, el evento se lanzará cuando cambie el valor del nombre, el valor cambiará cuando escribamos en él un valor nuevo y salgamos del campo

También podíamos haber utilizado el evento onchange con el atributo del campo nombre o asignándolo con el DOM.

El evento más importante de los formularios es el submit, ya que es el encargado de mandar la información. Lo más habitual en el manejador de este evento es un esquema como el siguiente:

```
formulario.addEventListener('submit', function(event) {
     event.preventDefault(); // Evita el envío del formulario por defecto

                             // Validar los datos del formulario

                             // Si son válidos, enviar el formulario
                    });
```

Lo más habitual es que el evento submit se haga lo siguiente:

- inicialmente cancele el envio
- realizar las validaciones oportunas
- si los datos no son correctos mostrar mensajes de error
- si los datos son correctos mandar los datos

En HTML puro, la etiqueta form tiene dos atributos que indican cómo será el comportamiento del envio de datos, estos atributos son:

- **action**: indica a que url o mail se mandará el formulario
- **mehod**: indica el método en el que se mandara el formulario, por defecto POST

Hoy en día lo más habitual es mandar los datos del formulario mediante AJAX, esto lo veremos en la comunicación asíncrona.

## 2.3 Validación de formularios con JavaScript

La validación de formularios es un proceso que consiste en verificar que los datos introducidos por el usuario sean correctos y completos. JavaScript proporciona herramientas para realizar validaciones personalizadas en los formularios.

Podemos realizar las validaciones comprobando que los campos contienen la información requerida utilizando los elementos de programación que aportan javascript para los objetos de tipo string o number.

Para validaciones más complejas podemos utilizar expresiones regulares, para ello utilizaremos **Regex**

### 2.3.1.- Expresiones regulares (regex)

Las expresiones regulares, también conocidas como regex o regexp, son patrones que se utilizan para buscar y manipular cadenas de texto. Son herramientas que se emplean en una amplia variedad de aplicaciones, como:

- **Validación de datos:** Verificar si una cadena de texto cumple con un formato específico, como un correo electrónico o un número de teléfono.
- **Búsqueda y extracción de información:** Encontrar texto coincidente con un patrón determinado dentro de un documento o base de datos.
- **Procesamiento de texto:** Reemplazar texto, eliminar partes no deseadas o transformar cadenas de texto de una forma a otra.

**Componentes básicos de una expresión regular:**

- **Caracteres literales:** Representan a sí mismos. Por ejemplo, "a" coincide con la letra "a".
- **Metacaracteres:** Tienen un significado especial dentro de la expresión. Algunos ejemplos comunes son:
    - . (punto): Coincide con cualquier carácter.
    - * (asterisco): Coincide con cero o más ocurrencias del patrón anterior.
    - + (signo más): Coincide con una o más ocurrencias del patrón anterior.

  - ? (signo de interrogación): Coincide con cero o una ocurrencia del patrón anterior.
  - [] (corchetes): Define un conjunto de caracteres. Por ejemplo, [aeiou] coincide con cualquier vocal.
  - ^ (circunflejo): Coincide con el inicio de la cadena de texto.
  - $ (signo de dólar): Coincide con el final de la cadena de texto.
- **Cuantificadores:** Indican cuántas veces debe coincidir el patrón anterior. Algunos ejemplos comunes son:
  - {n}: Coincide con exactamente n ocurrencias del patrón anterior.
  - {n,m}: Coincide con un rango de n a m ocurrencias del patrón anterior.
  - {n,}: Coincide con al menos n ocurrencias del patrón anterior.

Para usar una expresión regular, se necesita un motor de expresiones regulares que la interprete y la ejecute. Los motores de expresiones regulares están integrados en la mayoría de los lenguajes de programación y herramientas de procesamiento de texto.

En javaScript lo que deberemos hacer es **construir la expresión regular y aplicarle el método test pasándole como parámetro el valor que queremos validar**

Supongamos que queremos validar la dirección de correo electrónico de nuestro formulario. Podemos utilizar la siguiente expresión regular:

```
function validarEmail(email) {
        const regex =
/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-
9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-
zA-Z]{2,}))$/;
        return regex.test(email);
}

emailInput.addEventListener('blur', function() {
                    const email = emailInput.value;
                    if (!validarEmail(email)) {
                        alert('El correo electrónico no es válido.');
                    }
        });
```

Este patrón de expresión regular coincide con el formato básico de una dirección de correo electrónico, incluidos los caracteres alfanuméricos, puntos, guiones, signos más y guiones bajos en el nombre de usuario, seguidos de un @ símbolo, seguido de un nombre de dominio.

La gestión de formularios se adapta a las necesidades con las que fueron creados, pero lo más habitual en la gestión de un formulario es lo siguiente:

- A cada campo de un formulario añadirle, generalmente debajo, un div para mostrar si hay error en él, inicialmente todos los div de error estarán ocultos.
- En el manejador submit del formulario añadir las validaciones oportunas para cada campo.
- Si hay un error en la validación de un campo, mostrar el div de error asociado con el texto que indique el error
- No mandar los datos hasta que todos los campos sean válidos.

Al formulario que hemos utilizado con los campos de nombre y email, y el botón de submit, crear las siguientes validaciones:

- nombre y email no deben estar en blanco
- nombre debe de tener al menos 5 caracteres
- mail debe de tener formato de email valido
- al pulsar sobre el botón submit se realizarán las validaciones y se mostrarán los errores indicados
- si los datos de nombre y mail son válidos se mostrará un alert indicando que los datos se han enviado

# 3.- Controles de formulario

La mayoría de técnicas JavaScript relacionadas con los formularios requieren leer y/o modificar el valor de los campos del formulario. Vamos a ver las características más importantes de los controles más habituales en los formularios

## 3.1.- 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">
<textarea id="parrafo"></textarea>
<script>
    const valorText= document.getElementById("texto").value;
    const valorTextArea = document.getElementById("parrafo").value;
</script>
```

Hay veces en las que queremos que un cuadro de texto no se vea para ello deberemos usar el type **hidden**

```
<input type="hidden" id="idCliente" name="idCliente" value="1">
```

En otros casos nos puede interesar que sea sólo de lectura.

```
<input type="text" id="pais" name="pais" value="Esp" readonly>
```

## 3.2.- 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="casado" name="estadoCivil" id="casado"/> casado
<input type="radio" value="soltero" name="estadoCivil" id="soltero"/> soltero
<input type="radio" value="divorciado" name="estadoCivil" id="divorciado"/>
divorciado
<input type="radio" value="viudo" name="estadoCivil" id="viudo"/> viudo
<input type="radio" value="otro" name="estadoCivil" id="otro"/> otro
```

El siguiente código permite determinar si cada radiobutton ha sido seleccionado.

```
let elementos = document.getElementsByName("estadoCivil");
Array.from(elementos).forEach(ec=> alert(ec.value + " es " + ec.checked))
```

Generalmente lo que tenemos que hacer es recorrer la colección de los radiobutton bajo el mismo nombre y comprobar cuál de ellos esta seleccionado mediante la propiedad **checked==true,** una vez que sabemos cuál es podemos acceder a su valor mediante la propiedad value de ese radiobutton.

Al ser un control con varias opciones mutuamente excluyentes, los radiobutton deben tener el mismo nombre

## 3.3.- 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" id="condiciones"/>
          He leído y acepto las condiciones
<input type="checkbox" value="privacidad"  id="privacidad"/>
          He leído la política de privacidad
<button onclick="verCondicionesPrivacidad()">Ver seleccion</button>
```

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

```
function verCondicionesPrivacidad(){
  let el = document.getElementById("condiciones");
  texto= " no se ha aceptado"
  if (el.checked){
   let texto=" se ha aceptado"
  }
  alert(el.value + texto);

  el = document.getElementById("privacidad");
  texto= " no se ha aceptado"
```

```
  if (el.checked){
   let texto=" se ha aceptado"
  }
  alert(el.value + texto);
    }
```

## 3.4.- Select

Las listas desplegables (<select>) son los elementos en los que más propiedades tenemos. Un select o lista, podemos verlo como un array de elementos (versión gráfica) en los que cada elemento (option) ocupa una posición y tiene dos propiedades (**text**, lo que se en la lista y **value**, el valor que tiene esa opción). Cuando seleccionamos una opcion nueva de la lista se ejecuta el evento **onChange** y el valor del select es la propiedad **value**.

Si se dispone de una lista desplegable como la siguiente:

```
<select id="lista">
    <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>
```

El valor de la opción seleccionada de la lista se puede obtener de la siguiente forma:

```
let lista=document.getElementById("lista")
alert ("Valor lista " + lista.value)
```

```
lista.addEventListener("change",()=>alert(lista.value));
```

## 3.4.1.- Acceso a cualquier elemento del Select

En general, lo que se requiere es obtener el valor del atributo value de la opción (<option>) seleccionada por el usuario y generalmente cuando se cambie el option seleccionado, es decir en el evento onChange como hemos comentado anteriormente.

En algunas ocasiones igual necesitamos trabajar con una lista y realizar alguna de estas operaciones:

- Saber cuál es el elemento seleccionado y la posición del elemento en la lista.
- Acceder a un elemento de la lista en una posición determinada.
- Modificar desde código el elemento seleccionado de la lista

Estas operaciones ya no son tan sencillas como obtener el valor que devuelve una lista, ya que 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. Si no hay ningún elemento seleccionado esta propiedad devuelve -1

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

// 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)
alert("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:

```
// Acceder a lista
let lista = document.getElementById("lista");

// Obtener el texto que muestra la opción seleccionada
let textoSeleccionado = lista.options[lista.selectedIndex].text;

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

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).

## 3.4.2.-Añadir y eliminar elementos en un Select

Trabajando con listas es muy habitual crear una lista de forma dinámica o tener una lista ya creada y poder añadir elementos de la lista.

Esto lo podemos hacer de una forma sencilla utilizando los métodos vistos en el DOM. La idea es crear un elemento option por cada uno de los elementos a añadir en la lista y para cada uno de ellos definir su propiedad value y text. Una vez realizado esto deberemos añadir el option como hijo del select donde queramos que se cree la nueva opción de la lista.

```
<select id="lista2">
</select>
<button onclick="anyadirElementoLista()">Añadir elemento</button>
```

```
function anyadirElementoLista(){
   let texto=prompt("Dime Text");
   let valor=prompt("Dime Value");

   mioption=document.createElement("option");
   mioption.value=valor;
   mioption.text=texto;

   lista2.appendChild(mioption);
}
```

El element select dispone de métodos propios para insertar y eliminar. Son adicionales a los métodos generales del DOM.

**La ventaja que ofrecen frente al DOM es que permiten indicar directamente la posición para añadir o eliminar un elemento option en el select**

**Para añadir elementos se utiliza el método add**. La sintaxis es la siguiente:

```
selectObject.add(option, index)
```

- **option:** el elemento option a añadir al select
- **index:** es opcional e indica la posición en la que se añade el elemento option, si no se indica se añade al final del select.

Este ejemplo añade un elemento nuevo a la lista en la posición 2 (tercer elemento pués comienza por 0)

```
function addLista(){
   let texto=prompt("Dime Text");
   let valor=prompt("Dime Value");
   mioption=document.createElement("option");
   mioption.value=valor;
   mioption.text=texto;

   lista.add(mioption,2)
}
```

**Para eliminar elementos se utiliza el método remove**. La sintaxis es la siguiente:

```
selectObject.remove(index)
```

- **index:** La posición del option a eliminar del select.

Elimina el elemento en el índice 2 de la lista, elimina el elemento que aparece en la posición 3 del desplegable

```
lista.remove(2)
```

Elimina el último elemento de la lista.

```
if (lista.length > 0) {
  lista.remove(lista.length-1);
}
```