Los **formularios son una vía de entrada a la base de datos** y, por ende, a la **zona más sensible** de la aplicación web. La información que recibiremos debe pasar un **exhaustivo control de calidad**: La edad será un número, no una dirección de correo; una foto no debe ser un archivo comprimido zip. Y así podríamos continuar con una larga lista de especificaciones que no perdonaremos al visitante ni en un solo acento.

No buscamos castigar, sino guiar*. Con mensajes claros, bien explicados, indicaremos la mejor forma de rellenar **cada campo con un formato adecuado**. Y hasta que no se cumplan los requisitos, ignoraremos cualquier información que nos llegue a la base de datos. Incluso evitaremos desde el frontend que puedan pulsar el botón para enviar. Somos los últimos responsables de todo lo que pueda suceder.

Validar desde el frontend (JavaScript) no evita que pueda llegar la información al servidor en el formato adecuado. Siempre debemos comprobar desde el backend y devolver un código 400 en caso contrario.

No obstante debes ser consciente de que un formulario pasará por 4 estados:

1. **Reposo**: El usuario aún no ha introducido información. Los campos están vacíos, vírgenes, esperando ser rellenos.
2. **Validación**: Se incluye los datos. Los campos deben cumplir unas directrices. En caso contrario se muestra al usuario como debe corregirlo.
3. **Envío**: Los campos cumple las necesidades descritas. Se envía al servidor o servicio (como un API).
4. **Respuesta del servidor**: Se informa al usuario de si ha llegado correctamente o ha fallado.

En esta lección nos enfocaremos en la validación con el objetivo de mejorar la experiencia del usuario. Veamos algunos ejemplos utilizados de forma recurrente.

Si quisieramos enviar el formulario de forma asíncrona puedes ojear la lección de AJAX.

Input o Textarea

Obligatorio

Tan sencillo como añadir el atributo `required` en un `input`.

Nombre: Enviar

```
<form>
  <label for="name">Nombre: </label>
  <input type="text" name="name" id="name" required autocomplete="off">
  <button>Enviar</button>
</form>
```

Para personalizar el mensaje, como quitarlo cuando esté relleno, deberemos jugar con los eventos `input` (escribe dentro del campo) como `invalid` (no se cumple la validación del campo) que usaremos para mostrar el mensaje deseado con `setCustomValidity()`.

```
const miInput = document.querySelector('input');


// Quita la validación mientras escribes
miInput.addEventListener('input', () => {
    // Quita el mensaje según escribes
    miInput.setCustomValidity('');
    // Comprueba si debe validarlo
    miInput.checkValidity();
});


// Muestra el mensaje de validación
miInput.addEventListener('invalid', () => {
    miInput.setCustomValidity('Si no es molesta... ¿me dices tu nombre?');
});
```

Es recomendable usar `autocomplete="off"` para evitar que el autocompletador de los navegadores tapen las validaciones.

Límite de carácteres

Incluimos el atributo `pattern` en el `input` a validar, con una secuencia de patrón regular.

```
<form>
    <label>
        <input type="text" id="marca" required pattern="^[a-zA-Z0-9]{4,}$" autocomplete="off">
    </label>
    <input type="submit" id="submit" value="Enviar">
</form>
```

Mientras que JavaScript sería igual al ejemplo anterior.

```
// Variables
const inputMarca = document.querySelector('#marca');
const mensajeErrorMarcaCorto = "Muy corta. Dame un nombre con 4 o mas caracteres.";

// Eventos
inputMarca.addEventListener('input', () => {
    // Quita el mensaje según escribes
    inputMarca.setCustomValidity('');
    // Comprueba si debe validarlo
    inputMarca.checkValidity();
});

inputMarca.addEventListener('invalid', () => {
    inputMarca.setCustomValidity(mensajeErrorMarcaCorto);
});
```

Carácteres válidos

En este caso comprobaremos 2 casos:

- Debe estar relleno.
- Los carácteres permitidos son alfanuméricos, incluyendo espacios.

Incluimos el atributo `pattern` en el `input` a validar, con una secuencia de patrón regular.

Nombre: Enviar

```
<form>
    <label for="name">Nombre: </label>
    <input type="text" name="name" id="name" required pattern="^[a-zA-Z0-9 ]*$" autocomplete="off">
    <button>Enviar</button>
</form>
```

```
const miInput = document.querySelector('input');


// Quita la validación mientras escribes
miInput.addEventListener('input', () => {
    // Quita el mensaje según escribes
    miInput.setCustomValidity('');
    // Comprueba si debe validarlo
    miInput.checkValidity();
});


// Muestra el mensaje de validación
miInput.addEventListener('invalid', () => {
    if(miInput.value === '') {
        // Campo vacío
        miInput.setCustomValidity('Si no es molesta... ¿me dices tu nombre?');
    } else {
        // Patrón
        miInput.setCustomValidity('Debes introducir carácteres alfanuméricos');
    }
});
```

Quitar espacios innecesarios

Para evitar que nos envíen campos con espacios al principio y al final de un texto podemos ayudarnos de la función `trim()`.

```
trim('   Time After time   ')


//'Time After time'
```

Si lo usamos con el evento cuando se pierdo el foto (`blur`) podremos corregirlo rápidamente.

Nombre: Enviar

```
// Quita los espacios al principio y al final
miInput.addEventListener('blur', () => {
    miInput.value = miInput.value.trim();
});
```

Email

Disponemos de un input de tipo `email` que nos evitará usar complejos patrones.

```
<input type="email" required>
<form>
    <label for="email">Email: </label>
    <input type="email" name="email" id="email" required autocomplete="off">
    <button>Enviar</button>
</form>
const miInput = document.querySelector('input');
// Quita la validación mientras escribes
miInput.addEventListener('input', () => {
    // Quita el mensaje según escribes
    miInput.setCustomValidity('');
    // Comprueba si debe validarlo
    miInput.checkValidity();
});
// Muestra el mensaje de validación
miInput.addEventListener('invalid', () => {
    miInput.setCustomValidity('Me parece que esto no es un E-mail');
});
```

Números

En esta ocasión usamos el tipo `number`.

```
<input type="number" required>
```

Teléfono: Enviar

```
<form>
    <label for="email">Email: </label>
    <input type="email" name="email" id="email" required autocomplete="off">
    <button>Enviar</button>
</form>
```

```
const miInput = document.querySelector('input');


// Quita la validación mientras escribes
miInput.addEventListener('input', () => {
    // Quita el mensaje según escribes
    miInput.setCustomValidity('');
    // Comprueba si debe validarlo
    miInput.checkValidity();
});


// Muestra el mensaje de validación
miInput.addEventListener('invalid', () => {
    miInput.setCustomValidity('No es un número');
});
```

Checkbox

Es bastante común **obligar a que un usuario acepte un checkbox** antes de dejarle continuar (Llámalo acuerdo u extorsión legal). Sea el caso que sea, no difiere mucho de lo visto en anterioridad. Haremos uso el atributo `required`.

 Acepto vender mi alma Enviar

```
<form>
    <label>
        <input type="checkbox" required autocomplete="off"> Acepto vender mi alma
    </label>
    <button>Enviar</button>
</form>
const miInput = document.querySelector('input');


// Quita la validación mientras escribes
miInput.addEventListener('input', () => {
    // Quita el mensaje según escribes
    miInput.setCustomValidity('');
    // Comprueba si debe validarlo
    miInput.checkValidity();
});


// Muestra el mensaje de validación
miInput.addEventListener('invalid', () => {
    miInput.setCustomValidity('Si no aceptas no puedes continuar');
});
```

File

Veamos un ejemplo clásico, **validar que un campo sea una imagen**.

```
<input type="file" id="foto">
```

En el siguiente ejemplo validaremos que el campo de archivo, o tipo `file`, cumpla los siguientes requisitos:

- Extensiones: jpg, jpeg y png.
- No supere los 2 Mb.

Mi foto: Enviar

```
const oneMegaBytesInBytes = 10 ** 6;

const pesoLimite = oneMegaBytesInBytes * 2; // 2 megabyte

const extensionesPermitidas =  ['jpg','jpeg','png'];

const miInput = document.querySelector('#foto');


function validarImagen () {
    // Resetea mensaje
    miInput.setCustomValidity('');


    // Destructuramos para obtener el nombre y el tamaño
    const { name: archivoNombre, size: archivoPeso } = this.files[0];


    // Obtenemos la extensión
    const fileExtension = archivoNombre.split(".").pop();
    // Validamos si tienes una extensión válida
    if (!extensionesPermitidas.includes(fileExtension)){
        miInput.setCustomValidity('Formato no válido, solo se admite jpg y png');
    }
    // Validamos el peso
    if(archivoPeso > pesoLimite) {
        miInput.setCustomValidity('Demasiado grande');
    }
}


miInput.addEventListener("input", validarImagen);
```

Envío

Si todos los campos son correctos se enviará, no obstante si existe la necesidad de enviarlo sin pulsar el botón haremos uso de la función `submit()` de DOM formulario

```
<form id="mi-formulario">
    <input type="text" value="">
</form>
const miFormulario = document.querySelector('#mi-formulario');


miFormulario.submit();
```

Ejemplo sin usar validaciones nativas

Partiendo de un código donde todos sus campos son de tipo texto, haremos validaciones usando únicamente las herramientas de JavaScript. Se le podría denominar la forma “vieja”.

```
<form id="formulario">
    <p>
        <label>
            Nombre
            <input type="text" id="nombre" value="" autocomplete="off">
        </label>
    </p>
    <p>
        <label>
            Edad
            <input id="edad" type="text" value="" autocomplete="off">
        </label>
    </p>
    <p>
        <label>
            Mensaje
            <textarea cols="30" id="mensaje" name="" rows="10" autocomplete="off"></textare
a>
        </label>
```

```
    </p>
    <p>
        <input type="submit" id="enviar" value="Enviar"/>
    </p>
</form>
<!-- Mensajes de error -->
<ul id="errores"></ul>
```

Vamos a validar sus campos sin depender de las validaciones automáticas que nos proporciona HTML.

```
// Variables
const formulario = document.querySelector('#formulario');
const nombre = document.querySelector('#nombre');
const edad = document.querySelector('#edad');
const mensaje = document.querySelector('#mensaje');
const enviar = document.querySelector('#enviar');
const errores = document.querySelector('#errores');
let mensajesErrores = [];

// Funciones
function validar (evento) {
    // Evitar que se envie el formulario
    evento.preventDefault();

    // Vacia los mensajesErrores antes de rellenarlo nuevamente
    mensajesErrores = [];

    // VALIDACIONES

    // Nombre es obligatorio

    if (nombre.value.trim().length === 0) {
        mensajesErrores = mensajesErrores.concat('Nombre es un campo obligatorio');
    }
```

```
    // Nombre carácteres válidos

    if (!/^[a-zA-Z0-9]*$/.exec(nombre.value.trim())) {
        mensajesErrores = mensajesErrores.concat('Nombre no tiene carácteres válidos');
    }
    // Edad debe ser números
    if (isNaN(edad.value.trim())) {
        mensajesErrores = mensajesErrores.concat('La edad debe ser un número');
    }
    // Comprobamos que el mensaje tiene un mínimo de 10 carácteres

    if (mensaje.value.trim().length < 10) {
        mensajesErrores = mensajesErrores.concat('Mensaje demasiado corto');
    }
    // ENVIAR O MOSTRAR MENSAJES
    if (mensajesErrores.length === 0) {
        // Enviamos el formulario si no hay errores
        formulario.submit();
    } else {
        // Muestro los errores
        errores.textContent = '';
        mensajesErrores.forEach(function (mensaje) {
            const miLi = document.createElement('li');
            miLi.textContent = mensaje;
            errores.appendChild(miLi);
        });
    }
}
// Eventos
formulario.addEventListener('submit', validar);

// Inicio
```

Cómo validar formularios en JavaScript

El código JavaScript básico necesario para incorporar la validación a un formulario es :

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

La función validacion() se puede escribir así:

```
function validacion() { if (primera comprobación o validación en algún campo del formulario) { // Si esta comprobación no se cumple... alert('[ERROR] El campo del formulario debe tener un valor de...'); return false; } else if (Segunda comprobación o validación en algún campo del form) { // Si esta comprobación no se cumple... alert('[ERROR] El campo del formulario debe tener un valor de...'); return false; } ... else if (Tercera comprobación o validación en algún campo del form) { // Si esta comprobación no se cumple... alert('[ERROR] El campo del formulario debe tener un valor de...'); return false; }//etc // Si esta comprobación se encuentra correcta se procederá a ejecutar el envío de formulario al servidor // La validación del formulario es correcta retornamos el valor true return true;}
```

Al igual que otros eventos como onclick, onkeypress o upkeypress, el evento onsubmit varía su comportamiento en función del valor que se devuelve.

Si el evento onsubmit devuelve el valor true, el formulario se enviará normalmente. En cambio, si el evento onsubmit devuelve false, el formulario no se envía y muestra una alerta con el error o la indicación de dónde la comprobación no se superó.

El punto clave consiste en comprobar todos los campos y cada uno de los elementos del formulario. Por consiguiente, como primer paso es definir el evento onsubmit del formulario como:

```
onsubmit="return validacion()"
```

En la función validacion() se comprueban todas las condiciones impuestas por la aplicación; principalmente, una por cada campo del formulario. Cuando no se cumple una comprobación, se retorna false y, por tanto, el formulario no se envía. Si se llega al final de la función, entonces todas las comprobaciones se han cumplido correctamente, por lo que se retorna true y el formulario se envía.
En el código del ejemplo anterior se muestran mensajes mediante la función alert() indicando el error producido. Las aplicaciones web con un alto grado de diseño muestran cada mensaje de error al lado del campo del formulario y también suelen mostrar un mensaje o un popup principal indicando que contiene errores. La mayoría de estos elementos gráficos son interpretados con hojas de estilo que se muestran dependiendo del estado del error o de la comprobación correcta.

Teniendo la estructura definida de la función validacion(), se debe añadir a esta función el código correspondiente a todos los campos que se capturan en el formulario. Estas son algunas de las validaciones más habituales de los campos.

Ejemplo de cómo validar una dirección de correo en un formulario con JavaScript

Tiene como objetivo obligar al usuario a introducir una dirección de correo con un formato válido. Por tanto, lo que se comprueba es que la dirección parezca válida. 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 mediante las expresiones regulares (regex). Hay que tener en cuenta que valida las más usuales, pero no las que tengan una configuración distinta.

Ejemplo de cómo validar un campo de texto obligatorio en un formulario con JavaScript

La condición en JavaScript para validar un campo de texto obligatorio se puede indicar como:

```
valor = document.getElementById("campo").value;if( valor == null || valor.length == 0 || /^\s+$/.test(valor) ) { return false;}
```

Para que esta comprobación resulte correcta y tenga un texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducidos sea mayor que cero y que no haya solamente espacios.

Null indica que no hay ningún valor, lo cual se indicará si la longitud del mismo es igual a cero.

(/^\s+$/.test(valor)) comprueba que el valor introducido no solo esté formado por espacios en blanco. Esta comprobación se basa en el uso de regex, nuevamente. Por lo tanto, solo necesitas copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión para que esta pueda ser funcional.

Ejemplo de cómo validar un campo de texto con valores numéricos en un formulario con JavaScript

La condición JavaScript para una respuesta que requiere valores numéricos consiste en:

```
valor = document.getElementById("campo").value;if( isNaN(valor) ) { return false;}
```

La función isNaN facilita la validación, pues tiene en cuenta los signos, decimales y demás.

Ejemplo de cómo validar que se ha seleccionado un elemento de lista en un formulario con JavaScript

Este ejemplo fuerza a que el usuario seleccione un elemento de una lista desplegable:

```
indice = document.getElementById("opc").selectedIndex;if( indice == null || indice == 0 ) { return false;}

<select id="opc" name="opc"> <option value="">---Seleccione---</option> <option value="1">Primer valor</option> <option value="2">Segundo valor</option> <option value="3">Tercer valor</option></select>
```

Realizando el uso de la propiedad selectedIndex, se comprueba si el índice del select seleccionado es válido y además es diferente de cero. La primera opción de la lista (Seleccione) es solo la instrucción, por lo que no se permite el valor del índice cero que nos proporciona esta propiedad selectedIndex.

Ejemplo de cómo validar una fecha en un formulario con JavaScript

Las fechas suelen ser los campos de formulario más complicados de validar por la cantidad de formas diferentes en las que se pueden introducir. Nuevamente, usaremos las expresiones regulares (regex) validar la fecha en formato dd/mm/aa.

```
valor = document.getElementById("campo").value;if ( !(/^(0[1-9]|1[0-2])\/(0[1-9]|1\d|2\d|3[01])\/(0[1-9]|1[1-9]|2[1-9])$/.test(valor)) { return false;}
```

Ejemplo de cómo validar un número de teléfono en un formulario con JavaScript

Este código indica que un número (como el de teléfono) tiene diez dígitos de forma consecutiva:

```
valor = document.getElementById("campo").value;if( !(/^\d{10}$/.test(valor)) ) { return false;}
```

Una vez más, la condición de JavaScript se basa en el uso de expresiones regulares (regex), que comprueban si el valor indicado es una sucesión de diez números consecutivos.

Estos son ejemplos de expresiones regulares que se pueden utilizar para otros formatos de número de teléfono:

FORMATO	REGEX
1010001000	/^\d{10}$/
10-1000-1000	/^\d{2}-\d{4}-\d{4}$/
+11 1010001000	/^\+\d{2,3}\s\d{10}$/

Ejemplo de cómo validar que un checkbox ha sido seleccionado en un formulario con JavaScript

Para validar un checkbox, puede añadirse este código:

```
elemento = document.getElementById("campo");if( !elemento.checked ) { return false;}
```

Para validar más de un checkbox:

```
form = document.getElementById("form");for(var i=0; i<form.elements.length; i++) { var elemento = form.elements[i]; if(elemento.type == "checkbox") { if(!elemento.checked) { return false; } }}
```

Como podemos observar, en estas validaciones el uso de las expresiones regulares (regex) es de lo más común. No dudes en contar con la asesoría de un experto o experta que pueda optimizar la validación de tus formularios y generar más expresiones conformes con los campos que necesitas.