TEMA 7 – FORMULARIOS EN JAVASCRIPT

En una aplicación web, los **formularios** son uno de los principales medios de **interacción** entre el **usuario** y el **servidor**. Es por ello por lo que la validación de estos formularios es posible llevarla a cabo tanto en el lado del cliente como en el lado del servidor, aunque ya sabemos que la validación en el servidor es esencial para proteger la aplicación de peticiones malintencionadas realizadas con herramientas como Postman o cURL.

JavaScript nos ofrece una **capa adicional** de control y mejora de la experiencia de usuario desde el lado del cliente, no solo permitiendo validar los datos antes de enviarlos, sino también autocompletar campos, descargar información en segundo plano, y procesar datos complejos como imágenes antes de que se transmitan.

Estos formularios **organizan la información de manera estructurada en el DOM**, permitiendo que JavaScript acceda y manipule estos datos de forma eficiente. Desde la llegada de AJAX, JavaScript ha adquirido un papel fundamental en las comunicaciones **asíncronas con los servidores**, permitiendo una experiencia más dinámica y fluida en aplicaciones web

Algo muy importante que se debe comprender es cómo se manejan formularios en las aplicaciones JS. Cuando una página web se carga, el navegador crea automáticamente un **array llamado forms** que contiene todos los formularios presentes en el documento. A través de document.forms, podemos acceder a este array y **manipular los formularios utilizando su posición en la página**. Sin embargo, este método tiene desventajas: si cambia el diseño de la página o se añaden nuevos formularios, la posición de cada formulario en el array puede alterarse, lo que rompería el código que depende del orden. Así que para evitar estos problemas, es mejor **acceder** a los formularios utilizando el **atributo name o el id**, que se mantienen constantes sin importar los cambios en el orden del HTML. Por ejemplo:

```html
<form name="miFormulario">
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre">
    <button type="button" onclick="enviarFormulario()">Enviar</button>
</form>

<script>
    function enviarFormulario() {
        let formulario = document.forms["miFormulario"];
        //otra opción sería:
        let formulario2 = document.miFormulario;

        let nombre = formulario.elements["nombre"].value;
        //otra opción sería:
        let nombre2 = formulario2.nombre.value;
        alert("Nombre ingresado: " + nombre);
    }
</script>
```

Obviamente, también podemos acceder a esos elementos especificando un id como hemos hecho hasta este momento, con la función getElementoById. Como veremos a continuación.

1. Atributos de los formularios:

Para interactuar y manipular los datos de un formulario, es necesario conocer los atributos de los elementos que nos podemos encontrar en un formulario. Estos atributos nos permiten modificar, visualizar y controlar el comportamiento de los campos y opciones que un usuario introduce o selecciona.

Por ejemplo, el contenido de los **campos de entrada** en un formulario se puede visualizar y modificar utilizando el atributo **value**. Otros elementos del formulario, como los botones de opción (**radio button**) y las casillas de verificación (**checkbox**), deben tener un **name** común y también utilizan los **atributos value y checked**. Para los elementos **select**, se utilizan los atributos **options y selectedIndex.**

- **Cuadros de texto y textarea:** Como ya se ha comentado anteriormente, se obtiene y se establece directamente sobre la propiedad value:

ATRIBUTOS DEL FORM

```html
<input type="text" id="texto">
<textarea id="parrafo"></textarea>
<script>
    let texto1 = document.getElementById("texto").value;
    let texto2 = document.getElementById("parrafo").value;
</script>
```

La diferencia entre ambos elementos depende principalmente de la cantidad de texto que permiten.

- **RadioButton:** Cuando tenemos un grupo de radiobuttons **no** se quiere obtener el valor del atributo **value**, sino que lo importante es conocer **cual de todos los radiobuttons se ha seleccionado.**
    - El atributo **checked** devuelve **true** para el radiobutton seleccionado y **false** en cualquier otro caso.


```
<h1>ATRIBUTOS DEL FORM</h1>
<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
<script>
    let elementos = document.getElementsByName("pregunta");
    for (let elemento of elementos) {
        console.log("Elemento: " + elemento.value + "\nSeleccionado: " + elemento.checked);
    }
</script>
```

- **Checkbox:** Estos elementos son muy **similares** a los radiobutton, salvo que se debe **comprobar cada checkbox de forma independiente del resto**, ya que se pueden seleccionar de forma independiente respecto a los demás. Por el contrario, los radiobuttons son mutuamente excluyentes y solo se puede seleccionar cada uno de ellos cada vez.


```
<body>
    <h1>ATRIBUTOS DEL FORM</h1>
    <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
    <script>
        let elemento = document.getElementById("condiciones");
        alert("Elemento: " + elemento.value + "\nSeleccionado: " + elemento.checked);

        elemento = document.getElementById("privacidad");
        alert("Elemento: " + elemento.value + "\nSeleccionado: " + elemento.checked);
    </script>
</body>
```

- **Select:** Para obtener el valor del atributo value que ha seleccionado el usuario, debemos usar las siguientes propiedades:
    - **Options**: Es un array creado automáticamente por el navegador para cada lista desplegable que contiene referencia a todas las opciones de esa lista. **document.getElementById("identificador").options[0].**
    - **SelectedIndex:** Cuando el usuario selecciona una opción el navegador actualiza automáticamente el valor de esa propiedad, que guarda el índice de la opción seleccionada. El índice hace referencia al array de options creado automáticamente por el navegador.

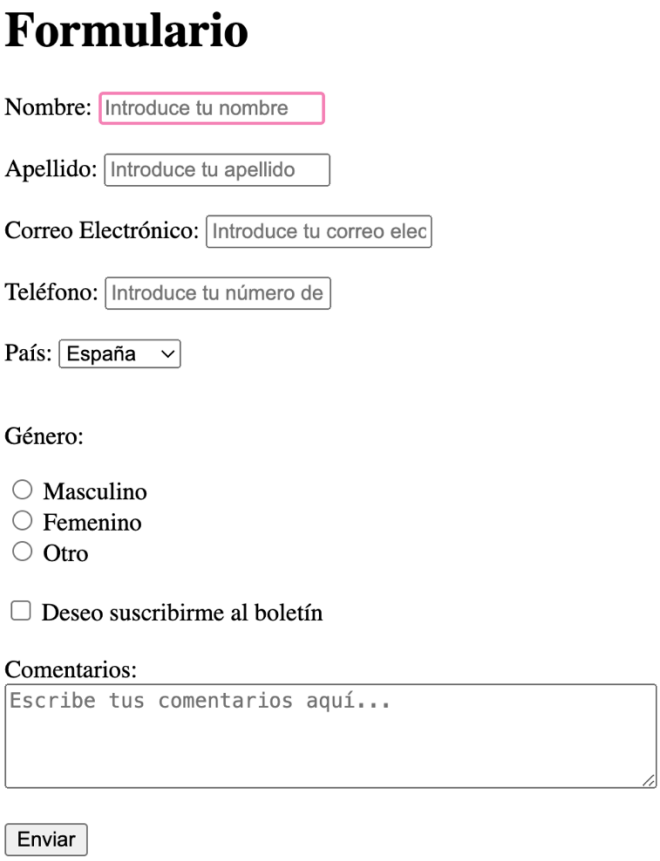
Aunque para obtener el valor del atributo value correspondiente a la opción seleccionado hay que realizar varios pasos. Se suele abreviar todos los pasos necesarios en una única instrucción:

```
let lista = document.getElementById("opciones");
let valorSeleccionado = lista.options[lista.selectedIndex].value;
let textoSeleccionado = lista.options[lista.selectedIndex].text;
```

2. Eventos más utilizados en formularios:

Los **eventos en formularios** son acciones que ocurren en respuesta a la interacción del usuario, como hacer clic, escribir en un campo, o seleccionar una opción. Estos eventos permiten que JavaScript detecte y reaccione a estos cambios, brindando la posibilidad de realizar validaciones en tiempo real, actualizar otros elementos de la página, o guiar al usuario en el proceso de llenado del formulario.

En base a este formulario se van a analizar los diferentes eventos que nos podemos encontrar en un formulario:



Los eventos que más se utilizan en el manejo de los formularios son:

- **Onclick:** Se produce cuando se **pincha** con el ratón sobre un elemento.

  Normalmente, se usa con cualquiera de los tipos de botones que permite definit HTML (“button”, “submit”, “image”).

```
generoRadios.forEach(radio => {
    radio.addEventListener("click", function () {
        console.log("Género seleccionado: " + radio.value);
    });
});

suscripcionCheckbox.addEventListener("click", function () {
    if (suscripcionCheckbox.checked) {
        console.log("El usuario desea suscribirse al boletín");
    } else {
        console.log("El usuario no desea suscribirse al boletín");
    }
});
```

- **Onchange:** Evento que se produce cuando el usuario **cambia el valor de un elemento** de texto (“text” o “textarea”), o cuando se **selecciona una opción en una lista desplegable (“select”).** Aunque el evento solo se produce si después de realizar el cambio, el usuario pasa al siguiente campo del formulario, es decir, cuando el elemento pierde el foco.

```
email.addEventListener("change", function () {
    console.log("El correo electrónico ha cambiado");
});

pais.addEventListener("change", function () {
    console.log("Se ha cambiado la selección de país a: " + pais.value);
});
```

- **Onfocus:** Se produce **cuando el usuario selecciona un elemento del formulario**.
  En programación, si tenemos un elemento seleccionado y se puede escribir en él, se dice que tiene el foco del programa. Si un cuadro de texto de un formulario tiene el foco, el usuario puede escribir directamente sin tener que pinchar previamente con el ratón en el interior del cuadro. Lo mismo sucede si tenemos una lista desplegable, el usuario puede seleccionar una opción directamente subiendo y bajando con las flechas del teclado. Al pulsar repetidamente la tecla TAB sobre una página web, los diferentes elementos (enlaces, imágenes, campos de formulario, etc.) van obteniendo el foco del navegador .

```
nombre.addEventListener("focus", function () {
    console.log("El campo Nombre ha recibido el foco");
});
```

  Algo que podemos realizar, es asignar un foco al primer elemento de forma automática:

```
// Comprobar si hay formularios en el documento
if (document.forms.length > 0) {
    // Recorre todos los elementos del primer formulario
    for (const campo of document.forms[0].elements) {
        // Comprueba si el tipo de campo no es "hidden"
        if (campo.type !== "hidden") {
            // Enfoca el primer campo visible y termina el bucle
            campo.focus();
            break;
        }
    }
}
```

Esto también se podría asignar directamente por el id del primer elemento, pero no sería de todo efectivo, ya que si añadimos un elemento antes que ese no se modificaría automáticamente.

- **Onblur:** Evento complementario de onfocus, se produce cuando el usuario ha **deseleccionado** un elemento por haber seleccionado otro. Es decir, cuando el elemento pierde el foco.

```
apellido.addEventListener("blur", () => {
    console.log("El campo Apellido ha perdido el foco");
});
```

Cuando trabajamos con formularios en aplicaciones web, uno de los aspectos más importantes es saber **cuándo y cómo usar los diferentes tipos de botones.** Dependiendo de la situación, puede ser necesario permitir que el formulario se envíe automáticamente o tener un control total sobre el envío, lo que es especialmente relevante en situaciones donde se necesita validación o cuando se quiere evitar un envío duplicado. Los tipos de botones que nos podemos encontrar son:

- **Submit:** Este tipo de botón es el más común para enviar formularios. **Al hacer clic en un botón submit, el formulario se envía automáticamente**, sin necesidad de escribir código adicional para enviarlo. Esto se suele utilizar cuando no se necesita realizar validaciones adicionales en JavaScript. Puede ser de tipo:
  - <button type="submit">
  - <input type="submit">.
- **Button:** Es un botón que genérico, que por sí solo, no realiza ninguna acción en el formulario. No envía el formulario a menos que se programe para hacerlo en javascript. **Es decir, no tiene asociada ninguna acción predeterminada, se debe especificar el comportamiento.**

  Este botón es útil cuando se necesita tener un control completo sobre el botón del formulario para hacer validaciones, deshabilitar el botón o realizar alguna acción adicional antes del envío.

3. Validaciones en los formularios:

Las validaciones en formularios son un paso crucial en el desarrollo de aplicaciones web, ya que permiten asegurarse de que los datos introducidos por el usuario cumplen con ciertos criterios antes de enviarlos al servidor. Este proceso es fundamental para garantizar la integridad, precisión y seguridad de la información que se recibe. Por ejemplo, se pueden validar campos como el correo electrónico para que tenga un formato correcto, verificar que una contraseña cumpla con ciertos requisitos de seguridad, o asegurarse de que un campo obligatorio no esté vacío.

Un problema común en formularios web es que el usuario haga clic dos veces en el botón de envío, ya sea porque la conexión es lenta o el servidor tarda en responder. Esto puede generar envíos duplicados, lo cual es problemático en aplicaciones que involucran transacciones, como pagos o registros. **Para evitar esto, una práctica recomendada es usar un botón de tipo button (o input type="button") en lugar de submit.** Luego, se controla el envío del formulario manualmente desde JavaScript, deshabilitando el botón inmediatamente después del primer clic y cambiando su texto para mostrar que el envío está en proceso. Así, se puede evitar envíos duplicados y se mejora la experiencia del usuario.

```html
<input type="button" value="Enviar" id="enviar" />
```

```javascript
enviar.addEventListener("click", manejarEnvio);

// Función para manejar el evento de envío
function manejarEnvio() {
    // Deshabilitar el botón
    enviar.disabled = true;
    // Cambiar el texto del botón
    enviar.value = "Enviando...";
    // Enviar el formulario manualmente
    document.forms.submit();
}
```

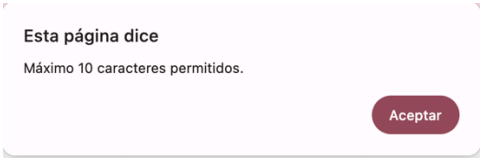
Otra opción sería utilizando el botón de submit, Algo que se puede tener en cuenta, es que existe un método que se llama **preventDefault()** que detiene el comportamiento predeterminado del botón submit, es decir, evita que el formulario se envíe automáticamente.

```
const formulario = document.getElementById("formulario");

formulario.addEventListener("submit", function(e) {
    // Detiene el envío automático del formulario
    e.preventDefault();

    const nombre = formulario.nombre.value;
    const email = formulario.email.value;

    if (nombre.trim() === "" || email.trim() === "") {
        alert("Por favor, completa todos los campos.");
    } else {
        formulario.submit();
    }
});
```

Ese parámetro e recoge la llamada submit, **es un parámetro que representa el evento que se está produciendo.** En este caso se trata del evento de envío de formulario.

Otra manera sencilla de validar un formulario, es por ejemplo limitar el número de caracteres que el usuario puede introducir en ciertos campos, como puede ser nombres o contraseñas. Un ejemplo sencillo puede ser el siguiente:

```
<body>
    <h2>Formulario</h2>
    <form id="formulario">
        <label for="nombre">Nombre (máximo 10 caracteres):</label>
        <input type="text" id="nombre" name="nombre">
    </form>
    <script>
        const campoNombre = document.getElementById("nombre");
        campoNombre.addEventListener("input", () => {
            if (campoNombre.value.length > 10) {
                alert("Máximo 10 caracteres permitidos.");
                campoNombre.value = campoNombre.value.slice(0, 10);
            }
        });
    </script>
</body>
```



Este ejemplo limita la cantidad de caracteres en el campo de nombre a 10 caracteres. Si el usuario intenta introducir más de 10, los caracteres adicionales son eliminados y se muestra un mensaje de alerta. Como ya sabemos, esto también se podría validar con las expresiones regulares vistas en el anterior tema:

```
<script>
    const campoNombre = document.getElementById("nombre");

    const regex = /^[a-zA-Z0-9]{0,10}$/;

    campoNombre.addEventListener("input", () => {
        if (!regex.test(campoNombre.value)) {
            alert("Máximo 10 caracteres permitidos.");
            campoNombre.value = campoNombre.value.slice(0, 10);
        }
    });
</script>
```

Otra forma de validar un formulario es crear un objeto en JavaScript que represente el formulario y almacene los criterios de validación. Luego, podemos definir métodos dentro del objeto para verificar si los datos cumplen con los requisitos.

```
<script>
    const boton = document.getElementById("enviar");
    const formulario = {
        username: document.getElementById("username"),
        password: document.getElementById("password"),

        esValido() {
            return this.validarUsername() && this.validarPassword();
        },

        validarUsername() {
            if (this.username.value.length < 5) {
                alert("El nombre de usuario debe tener al menos 5 caracteres.");
                this.username.focus();
                return false;
            }
            return true;
        },

        validarPassword() {
            if (this.password.value.length < 8) {
                alert("La contraseña debe tener al menos 8 caracteres.");
                this.password.focus();
                return false;
            }
            return true;
        }
    };

    boton.addEventListener("click", function(){
        if (formulario.esValido()) {
            alert("Formulario válido. Datos enviados.");
            // Aquí se enviaría el formulario
        }
    })
</script>
```

Otra opción de validación útil es marcar los campos obligatorios con un estilo visual si están vacíos. Esto se puede hacer fácilmente con JavaScript, aplicando una clase CSS que cambie el borde o el color de fondo del campo cuando el usuario no lo ha llenado.

```css
.campo-obligatorio {
    border: 2px solid red;
}
```

```
const enviar = document.getElementById("enviar");
function validarCampo() {
    const campoTelefono = document.getElementById("telefono");
    if (campoTelefono.value.trim() === "") {
        campoTelefono.classList.add("campo-obligatorio");
        alert("El campo de teléfono es obligatorio.");
        campoTelefono.focus();
    } else {
        campoTelefono.classList.remove("campo-obligatorio");
        alert("Formulario enviado correctamente.");
    }
}
enviar.addEventListener("click", validarCampoº)

</script>
```