

# Aplicaciones web dinámicas: PHP y Javascript.

## Caso práctico

**Juan y Carlos** tienen definida la estructura de la aplicación web que deben desarrollar. Han llevado a cabo algunas pruebas de programación, y están contentos con los resultados obtenidos. Antes de seguir avanzando, deciden reunir al equipo de BK Programación para mostrarles los progresos realizados y el plan de desarrollo previsto.



Durante la presentación de la aplicación web, todo el equipo se muestra entusiasmado con el aspecto que está tomando el proyecto. **Ada**, la directora, les felicita por el trabajo que han llevado a cabo hasta el momento. **Ana**, que tiene experiencia como diseñadora gráfica, se ofrece a ayudarles con el aspecto visual del interfaz web. Pero de todas las opiniones, hay una que les llama la atención. **María**, que se encarga del mantenimiento de servidores y sitios web, les pregunta si han tenido en cuenta la posibilidad de **integrar en su aplicación algún tipo de código cliente**. Les comenta que muchas aplicaciones actuales lo utilizan dentro de su estructura para muy diversas funciones.

**Juan y Carlos** se miran, y saben que ambos están pensando lo mismo. Habrá que hacer un último esfuerzo e informarse sobre el tema. Si deciden que resulta interesante, aún están a tiempo de incorporarlo en su proyecto.



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

**Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.**

[Aviso Legal](#)

# 1.- Programación del cliente web.

## Caso práctico

El primer paso que deciden dar **Juan y Carlos**, siempre contando con el asesoramiento de **María**, es informarse sobre las tecnologías de programación web actuales. Conocen de oídas el lenguaje **JavaScript**, y saben cuáles son los fundamentos de la tecnología **AJAX**, pero no tienen claras las ventajas e inconvenientes que les puede suponer su incorporación al proyecto en el que están trabajando.



Lo más importante; si finalmente consideran que puede ser ventajoso introducir en su aplicación web algún tipo de programación que se ejecute en el navegador... ¿cómo se puede integrar ésta con la programación del servidor web? ¿Pueden coexistir, y aún más, integrarse ambos lenguajes de alguna forma?

Siguiendo su método de trabajo, deciden buscar información por separado y compartirla pasados unos días.

Cuando comenzaste con el presente módulo, uno de los primeros conceptos que aprendiste es a diferenciar entre la ejecución de código en el servidor web y la ejecución de código en el navegador o cliente web.

Todo lo que has aprendido hasta el momento se ha centrado en la ejecución de código en el servidor web utilizando el lenguaje **PHP**. La otra parte importante de una aplicación web, la programación de código que se ejecuta en el navegador, no forma parte de los contenidos de este módulo. Tiene su propio módulo dedicado, **Desarrollo Web en Entorno Cliente**.

Muchas de las aplicaciones web que existen en la actualidad tienen esos dos componentes: una parte de la aplicación, generalmente la que contiene la lógica de negocio, se ejecuta en el servidor; y otra parte de la aplicación, de menor peso, se ejecuta en el cliente. Existen incluso cierto tipo de aplicaciones web, como "*Google Docs*", en las que gran parte de las funcionalidades que ofrecen se implementan utilizando programación del cliente web.

En la presente unidad vas a aprender cómo integrar estos dos componentes de una misma aplicación web: el código **PHP** que se ejecutará en el servidor, con el código que se enviará al cliente para que éste lo ejecute.

La programación de guiones para su ejecución en un cliente web es similar a lo que ya conoces sobre **PHP**, salvo que en este caso el código completo del guión llega al navegador junto con las etiquetas **HTML**, y es éste el encargado de procesarlo.

Así como el código **PHP** se marcaba utilizando los delimitadores "`<?PHP ?>`", en **HTML** existe una etiqueta que se utiliza para integrar el código ejecutable por el navegador junto al resto de etiquetas. Se trata de la etiqueta `<script>`, que puede indicar tanto la localización del código

en un fichero externo, como simplemente delimitar unas líneas de código dentro del propio fichero HTML.

```
// Inclusión de código en el documento HTML
<script type="text/javascript">
// Código que ejecuta el navegador
</script>
// Inclusión de código en un fichero externo
<script type="text/javascript" src="codigo.js"></script>
```

## Autoevaluación

¿Qué etiqueta HTML se usa para marcar el código que ejecutará el navegador?

- ☐ **CDATA.**
- ☐ **script.**

CDATA no es una etiqueta HTML.

Efectivamente. La etiqueta `script` puede incluir por si misma el código que ejecutará el navegador, o indicar el fichero en que se encuentra.

## Solución

1. Incorrecto
2. Opción correcta

## 1.1.- Páginas web dinámicas.

La inclusión de código en páginas web para su ejecución por parte del navegador tiene ciertas limitaciones:

- ✓ Cuando ejecutas código **PHP** en un servidor, es normalmente el programador el que tiene el control sobre el entorno de ejecución. Al cliente llegan únicamente etiquetas en lenguaje **HTML** o **XHTML**. Sin embargo, cuando programas código para que se ejecute en un cliente web, no tienes siquiera la certeza de que el navegador del usuario soporte la ejecución del código que recibe. Existen ciertos sistemas, como dispositivos móviles o navegadores integrados en hardware específico, que no permiten la ejecución de código de cliente.
- ✓ El código que se ejecuta en el navegador está normalmente limitado a ser ejecutado en un entorno controlado, que no permite, por ejemplo, la lectura o escritura de ficheros en el ordenador del usuario. De esta forma se restringen los efectos negativos que pueda causar un guion y se favorece la confianza del usuario en este tipo de código.



[kreatikar](#) (Pixabay License)

Pese a estas limitaciones, la ejecución de código en el navegador encaja perfectamente con cierto tipo de tareas como:

- ✓ Comprobar y/o procesar los datos que introduce el usuario en los formularios, como paso previo a su envío al servidor web.
- ✓ Gestionar diferentes marcos y/o ventanas del navegador.
- ✓ Modificar de forma dinámica los elementos que componen la página web, ajustando sus propiedades o estilos en respuesta a la interacción del usuario.

El código **JavaScript** de una página se puede ejecutar en respuesta a eventos generados por el navegador. Por ejemplo, utilizando el evento **onsubmit** podemos llamar a una función **validar\_email** para validar una dirección de correo introducida por el usuario cuando se intenta enviar el formulario:

```
<form action="usuario.php" method="get" name="datos_usuario" id="formDatosUsuario">
  <input type="text" id="email" />
  <button type="submit">Enviar</button>
</form>
```

Para la función que realiza la validación básica de una dirección de **email** puedes utilizar, por ejemplo, el siguiente código:

```
document.addEventListener('DOMContentLoaded', function () {
  var form = document.getElementById('formDatosUsuario');
  form.addEventListener('submit', function(event) {
    if (!validar_email()) {
      event.preventDefault(); // Prevenir que el formulario se envíe si el email no es v
    }
  });
});
```

```
function validar_email() {  
    var valor = document.getElementById("email").value;  
    var pos_arroba = valor.indexOf("@");  
    var pos_punto = valor.lastIndexOf(".");  
    if (pos_arroba < 1 || pos_punto < pos
```

Las páginas web que se aprovechan de las capacidades de ejecución de código en el cliente para cambiar su apariencia, o su funcionamiento, se conocen como páginas web dinámicas.

Se llama **HTML dinámico (DHTML)** al conjunto de técnicas que emplean **HTML**, el modelo de objetos del documento web (**DOM**), hojas de estilo **CSS** y lenguaje ejecutado en el navegador para crear sitios webs dinámicos.



## 1.2.- El lenguaje JavaScript.

El lenguaje de guiones que se utiliza mayoritariamente hoy en día para la programación de clientes web es **JavaScript**. Su sintaxis está basada en la del lenguaje **C**, parecida a la que conocemos del lenguaje **PHP**. Aunque su utilización principal es incorporarlo a páginas web, también puedes encontrar **JavaScript** en otros lugares como en documentos **PDF**, o para definir la funcionalidad de extensiones de escritorio o de algunas aplicaciones widgets).



[The Oxygen Team, KDE](#)  
(LGPL)

Si bien, la gran mayoría de navegadores web soportan código en lenguaje **JavaScript**, debes tener en cuenta que:

- ✓ La ejecución de **JavaScript** en el navegador puede haber sido deshabilitada por el usuario.
- ✓ La implementación de **JavaScript** puede variar de un navegador a otro. Lo mismo sucede con el interface de programación que usa **JavaScript** para acceder a la estructura de las páginas web: el **DOM**. Por este motivo, es conveniente que verifiques la funcionalidad del código en diversos navegadores antes de publicarlo como parte de tu sitio web.

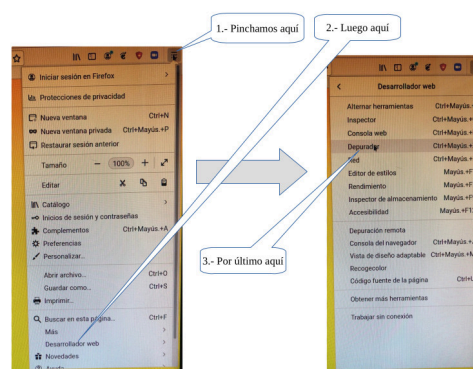
Se conoce como motor **JavaScript** a la parte del navegador encargada de interpretar y ejecutar el código **JavaScript** que forma parte de las páginas web. Los motores **JavaScript** que se incluyen en los navegadores han experimentado una importante mejora de rendimiento en los últimos tiempos. Existen pruebas específicas destinadas a medir la velocidad de ejecución de distintos motores **JavaScript**.

### Velocidad de ejecución de distintos motores JavaScript.

Aunque no vamos a aprender en esta unidad a programar en **JavaScript**, deberías saber cómo depurar el código que vamos a utilizar. Es conveniente que manejes un depurador para cada navegador que utilices.

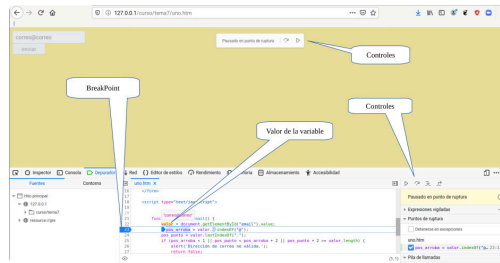
En versiones actuales de Firefox las herramientas de desarrollo ya vienen integradas en el ítem de menú "Desarrollador Web", para acceder a ellas solo hacemos lo siguiente:

Abriendo las herramientas de depuración



Captura de Pantalla Firefox (Elaboración propia)

Debug Funcionando



Captura de pantalla Firefox (Elaboración)

## Autoevaluación

¿Cuál es una de las principales desventajas de la programación del cliente web?

- ☐ Que no es posible asegurar que el navegador vaya a ejecutar el código.
- ☐ Que el navegador puede no ser capaz de mostrar correctamente la página al confundir el código con las etiquetas HTML /XHTML.

Efectivamente. La ejecución de código **JavaScript** puede estar deshabilitada por el usuario, o el navegador puede no soportar la ejecución de código.

Si el código está bien marcado, utilizando la etiqueta `<script>` y una sección `CDATA` comentada, no tendremos problema al mostrar la página independientemente del tipo de navegador que usemos.

## Solución

1. Opción correcta
2. Incorrecto

## 1.3.- Comunicación asíncrona con el servidor web: AJAX.

Una de las principales causas de la evolución de **JavaScript** es, sin duda, la tecnología **AJAX**. Como ya vimos en la primera unidad, el término **AJAX** hace referencia a la posibilidad de una página web de establecer una comunicación con un servidor web y recibir una respuesta sin necesidad de que el navegador recargue la página.

**AJAX** utiliza el objeto **XMLHttpRequest**, creado originariamente por **Microsoft** como parte de su librería **MSXML**, y que hoy en día se ha incorporado de forma nativa a todos los navegadores actuales.



[Gumme](#) (CC BY-SA)

Pese al nombre del objeto, y a que la letra X de las siglas AJAX hace referencia a **XML**, la información que se transmite de forma asíncrona entre el navegador y el servidor web no es necesario que se encuentre en formato XML.

Entre las tareas que puedes llevar a cabo gracias a AJAX están:

- ✓ Actualizar el contenido de una página web sin necesidad de recargarla.
- ✓ Pedir y recibir información desde un servidor web manteniendo la página cargada en el navegador.
- ✓ Enviar información de la página a un servidor web en segundo plano.

Dependiendo del navegador del usuario, y de si utiliza una versión antigua o moderna, tendrás que usar un método u otro para crear el objeto **XMLHttpRequest**:

```
// Distintas formas para crear el objeto
// XMLHttpRequest según el navegador
xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
xmlhttp = new XMLHttpRequest();
```

Un ejemplo de uso del objeto **XMLHttpRequest** sería:

```
// Crea un objeto XMLHttpRequest
let xhr = new XMLHttpRequest();
// Solicitud GET para la URL /article/.../load
xhr.open('GET', '/producto/info/3321');
// Envía la solicitud a la red
xhr.send();
// Función para tratar la respuesta
xhr.onload = function() {
```



```

if (xhr.status !== 200) { // analiza el estado HTTP de la respuesta
    alert(`Error ${xhr.status}: ${xhr.statusText}`); // ej. 404: No encontrado
} else { // muestra el resultado
    alert(`Hecho, obtenidos ${xhr.response.length} bytes`); // Respuesta del servidor
}
};

xhr.onprogress = function(event) {
    if (event.lengthComputable) {
        alert(`Recibidos ${event.loaded} de ${event.total} bytes`);
    } else {
        alert(`Recibidos ${event.loaded} bytes`); // sin Content-Length
    }
};

xhr.onerror = function() {
    alert("Solicitud fallida");
};

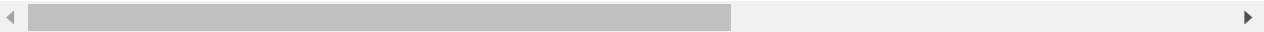
```

Afortunadamente, muchas de las librerías JavaScript de las que hablábamos antes soportan también AJAX, y utilizan un código adaptado según el navegador del usuario. Si utilizas una de estas librerías podrás ahorrarte muchos quebraderos de cabeza al programar. Por ejemplo, si utilizas jQuery podrías utilizar AJAX para enviar en segundo plano el email validado en el ejemplo anterior. Simplemente tendrías que incluir en el HTML la librería jQuery.

```

//Versión 3.6-.0 jQuery CDN jQuery
<script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+30JU5Exlc

```



Y tras el código de validación, debes ejecutar la función AJAX incluida en la librería jQuery:

```

// Realiza una solicitud GET usando jQuery
$.ajax({
    url: '/producto/info/3321', // URL de la solicitud
    method: 'GET', // Método HTTP
    success: function(data, status, xhr) {
        // Se llama en caso de éxito
        alert(`Hecho, obtenidos ${xhr.responseText.length} bytes`); // Respuesta del servidor
    },
    error: function(xhr, status, error) {
        // Se llama en caso de fallo en la solicitud
        alert(`Error ${xhr.status}: ${xhr.statusText}`); // ej. 404: No encontrado
    },
    progress: function(event) {
        // Manejo de la barra de progreso, si es necesario
        if (event.lengthComputable) {
            alert(`Recibidos ${event.loaded} de ${event.total} bytes`);
        } else {
            alert(`Recibidos ${event.loaded} bytes`); // sin Content-Length
        }
    }
});

```

```
});
```

```
// En jQuery, manejar errores de red se incluye en el callback 'error'
```



## 2.- PHP y JavaScript.

### Caso práctico

Pasados unos días, **Juan y Carlos** se vuelven a reunir y deciden que en el diseño de su aplicación existen muchos lugares en los que podrían aprovechar las capacidades que ofrece la programación del navegador web, concretamente el lenguaje **JavaScript**.



in embargo, ambos siguen sin tener claro cómo pueden integrar el código del cliente web en una aplicación programada en lenguaje **PHP**. Animados por **María**, que les orienta sobre el rumbo que deben tomar, preparan una serie de pruebas de programación que les puedan orientar sobre el tema.

Si sabes programar aplicaciones que se ejecuten en un servidor web (con un lenguaje como **PHP**) y en el navegador del usuario (con **JavaScript/jQuery**), tienes en tu mano las herramientas necesarias para construir aplicaciones web completas. Sin embargo, es necesario que antes de comenzar tengas claras las funcionalidades que soporta cada una de estas tecnologías de desarrollo, y cómo puedes hacer para utilizar ambas a la vez.

Llevándolo a los extremos, podrías hacer aplicaciones en **PHP** que utilicen programación del cliente simplemente para tareas sencillas como verificar campos en los formularios antes de enviarlos. O, por el contrario, sería también posible programar aplicaciones completas que se ejecuten en el navegador del usuario, dejando el lenguaje del servidor para proporcionar ciertas funcionalidades como almacenamiento en bases de datos.

Una alternativa no es necesariamente mejor que la otra. Es necesario analizar de forma independiente la lógica de cada aplicación, de manera que no se malgasten los recursos del servidor realizando tareas que podrían trasladarse al cliente web. En ocasiones también es necesario comprobar los tiempos de carga de las páginas y el tamaño de las mismas. Puede ser preferible utilizar **AJAX** para, por ejemplo, enviar nuevos registros al servidor, si el esfuerzo que invertimos redundaría en una interfaz de usuario más ágil y usable. En cualquier caso, la consistencia y robustez de la aplicación no debe verse afectada.

Si decides unir en una aplicación programación del cliente web con programación del servidor web, habrá ocasiones en que necesites comunicar ambos lenguajes. En el ejemplo anterior ya has comprobado cómo puedes hacer para pasar un valor o una variable **JavaScript** desde el navegador web a un guión en **PHP**: enviándolo como parámetro **POST** o **GET** en una petición de nueva página, bien sea al cargarla en el navegador o utilizando **AJAX** en una comunicación asíncrona.

En ambos casos deberás asegurarte de que las cadenas que insertes en las direcciones **URL** no incluyan caracteres inválidos. Para evitarlo, en **PHP** puedes usar la función `"urlencode"`, y en **JavaScript**: `"encodeURIComponent"`.

# Autoevaluación

¿Cómo es posible obtener en PHP el contenido de una variable **JavaScript**?

- ☐ En una petición de nueva página, como parámetro **POST** o **GET**.
- ☐ Enviando desde el servidor web una solicitud al navegador web.

Efectivamente. Además, ten en cuenta que cualquier contenido que forme parte de una URL debería procesarse previamente utilizando la función **JavaScript: "encodeURIComponent"**.

El servidor no puede enviar solicitudes al navegador; simplemente responde a las solicitudes de página que éste realiza.

## Solución

1. Opción correcta
2. Incorrecto

## 2.1.- Validación de los datos de un formulario.

Con lo que has visto hasta el momento, seguramente, ya te has hecho una idea de qué tareas son las que con más frecuencia hacen uso de **JavaScript**. Y si hay una que destaca sobre el resto es, sin duda, la validación de formularios **HTML**, que vamos a utilizar en los ejemplos que se desarrollan a lo largo de la presente unidad.

Como ya sabes, el mecanismo que utilizan las aplicaciones web para permitir al usuario la entrada de información es el formulario. Los datos que se introducen en un formulario web se envían al servidor utilizando bien el método **POST**, bien el método **GET**.

Muchos de los campos de los formularios tienen, normalmente, restricciones sobre el contenido que se debe rellenar. La validación de un formulario web consiste en comprobar que el contenido introducido en todos los componentes del mismo cumpla estas restricciones.

Si no utilizas código ejecutable en el navegador, la única forma de validar un formulario consiste en enviarlo al servidor web para comprobar si existen errores en los datos. En caso de que así sea, habrá que volver a enviar el formulario al navegador del usuario mostrando las advertencias oportunas.

Obviamente, si utilizas **JavaScript** en tus aplicaciones, la validación se puede realizar en el cliente web. De esta forma el proceso es mucho más rápido. No es necesario enviar la información al servidor hasta que se haya comprobado que no existen errores de validación.

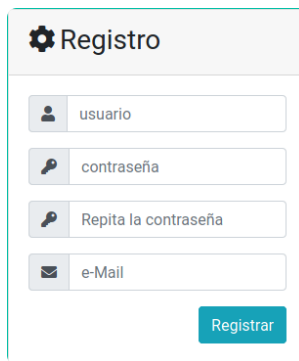
Sin embargo, aunque parecen claras las ventajas de la validación de formularios en el cliente web, hay ocasiones en las que ésta no es posible. Ya vimos que no siempre podrás asegurar que el navegador que utiliza el usuario tiene capacidad para ejecutar código **JavaScript**, o incluso puede suceder que se haya deshabilitado por motivos de seguridad.

En los casos que no sea posible asegurar la capacidad de ejecución de código de los clientes, la solución óptima sería utilizar un escenario dual: diseñar las aplicaciones suponiendo que es posible ejecutar **JavaScript** en los clientes, y al mismo tiempo prever la posibilidad de que no sea así. Por ejemplo, en el caso de la validación del formulario, podrías crear el código **JavaScript** de validación y, si en algún cliente este código no se puede ejecutar, preparar un código **PHP** similar que realice la validación en el servidor.

Por ejemplo, supongamos que queremos validar los datos que se introducen en el siguiente formulario web:



[Kreatikar](#) (Pixabay License)



The image shows a registration form titled "Registro" with a gear icon. It contains four input fields: "usuario" (with a person icon), "contraseña" (with a key icon), "Repita la contraseña" (with a key icon), and "e-Mail" (with an envelope icon). A blue "Registrar" button is located at the bottom right of the form.

Captura de pantalla Firefox  
(Elaboración propia.)

Si realizas la validación en PHP, podrías generar junto con la página web el texto con las advertencias de validación. Y si el formulario lo quieres validar también utilizando JavaScript, tendrás que crear los mismos textos o similares. Una posibilidad para no repetir el código que introduce esos textos en el cliente y en el servidor, es introducir los textos de validación en las etiquetas HTML de la página web, y utilizar estilos para mostrarlos, o no, según sea oportuno.

## 2.1.1.- Validación de los datos de un formulario en el cliente.

Existen muchas técnicas para la validación de formularios en el lado cliente. A continuación, se presentan las más utilizadas en navegadores modernos.

La opción más sencilla para validar los datos de un formulario en el cliente consiste usar el API de Validación de HTML5. En este caso, las restricciones de validación de los valores se definen de manera declarativa asociadas a los elementos de entrada del formulario, por ejemplo los elementos `input`, `select` y `textarea`.

Dichas restricciones se adjuntan como atributos adicionales. Por ejemplo:

- **required**: Especifica si es obligatorio rellenar un campo del formulario.
- **minlength** y **maxlength**: Especifican la longitud mínima y máxima permitida para un dato de tipo cadena.
- **min** y **max**: Especifican los valores mínimo y máximo de los valores de un dato numérico.
- **type**: Especifica si los datos deben ser un número, una dirección de correo o algún otro tipo de dato específico.
- **pattern**: Especifica una expresión regular que define un patrón que debe seguir el valor introducido en el formulario.

El formulario con los atributos asociados a las restricciones que validan los datos se muestra a continuación:

```
<form id="registro" name="registro" method="POST" action="?= $_SERVER['F
  <div class="input-group my-2">
    <span class="input-group-text"><i class="bi bi-person"></i></span>
    <input type="text" class="form-control" placeholder="usuar
      id="usuario" name="usuario" minlength="3" autofocus
  </div>
  <div class="input-group my-2">
    <span class="input-group-text"><i class="bi bi-key"></i></span>
    <input type="password" class="form-control" placeholder="cc
      id="password1" name="password1" minlength="5" requir
  </div>
  <div class="input-group my-2">
    <span class="input-group-text"><i class="bi bi-key"></i></span>
    <input type="password" class="form-control" placeholder="f
      id="password2" name="password2" minlength="5" requir
  </div>
  <div class="input-group my-2">
    <span class="input-group-text"><i class="bi bi-envelope"></i></span>
    <input type="email" class="form-control" placeholder="e-Ma:
  </div>
  <div class="text-end">
    <input type="submit" value="Registrar" class="btn btn-info"
  </div>
</form>
```

El proceso de validación integrado en HTML5 comprueba de manera automática que todas las restricciones definidas se cumplen para cada elemento del formulario. En base a dicha comprobación se establece la pseudoclase `:valid` o `:invalid` que determina la validación de cada campo. Si se detecta que algún campo no es válido se cancela el envío de los datos al servidor y se muestra un mensaje al usuario para que corrija el valor introducido en el campo inválido.

La mayoría de las restricciones pueden ser validadas en el navegador con este procedimiento. Sin embargo, hay ciertas situaciones que requieren codificar una lógica de validación un poco más compleja. Por ejemplo, en nuestro caso la comprobación de que las dos contraseñas introducidas sean iguales.

Para realizar dicha validación debemos hacer uso del API de validación de restricciones disponible en HTML5. Dicho API ofrece métodos javascript que permiten controlar la lógica de validación que tiene lugar en el navegador y aporta propiedades a los elementos de entrada para informar de su estado de validez.

A continuación, se listan algunas de las propiedades asociadas al API:

- ✔ **validationMessage**: Devuelve un mensaje que describe las restricciones de validación que el elemento no satisface.
- ✔ **validity**: Devuelve un objeto **validityState** que contiene varias propiedades que describen el estado de validez del elemento. Algunos valores posibles para el objeto SON **patternMismatch**, **tooLong**, **rangeOverflow**, **rangeUnderflow**, **typeMismatch**, **valid** y **valueMissing**.
- ✔ **willValidate**: Devuelve **true** si el elemento se valida cuando se envía el formulario.

Además se dispone de los siguientes métodos para los mismos elementos:

- ✔ **checkValidity()**: Devuelve **true** si el valor del elemento no tiene problemas de validez; **false** en caso contrario. Si el elemento no es válido, este método también activa un evento **invalid** en el elemento. Si se invoca en un formulario se activa el proceso de validación de todos los campos del mismo.
- ✔ **setCustomValidity(message)**: Añade un mensaje de error personalizado al elemento; si configuras un mensaje de error personalizado, el elemento se considera no válido y se muestra el error especificado. Si se invoca con la cadena vacía el estado del elemento se establece a válido.

Si vamos a utilizar el API de validación de restricciones para establecer mensajes de error personalizados debemos añadir el atributo **novalidate** al elemento HTML del formulario.

A continuación se muestra el archivo javascript **validar.js** que realiza la validación en el cliente, definiendo mensajes de error personalizados y comprobando la igualdad de las contraseñas:

```
document.addEventListener("DOMContentLoaded", function () {
    const formRegistro = document.getElementById('registro');
    // Añade el evento submit al formulario
    formRegistro.addEventListener("submit", (event) => {
        // Luego comprueba la validez general del formulario
        const allValid = formRegistro.checkValidity();
        if (!allValid) {
            event.preventDefault();
            event.stopImmediatePropagation();
        }
        const password1 = document.getElementById('password1');
        const password2 = document.getElementById('password2');
```



```

const passwordErrorBox = document.getElementById('password2Error');

// Limpia cualquier estado de error previo
passwordErrorBox.textContent = "";

// Verifica si las contraseñas coinciden
if (password1.value !== password2.value) {
    passwordErrorBox.textContent = "Los passwords introducidos deben de ser iguales";
    event.preventDefault(); // Previene el envío del formulario
    event.stopImmediatePropagation();
}
});

// Evento invalid y input para otros campos
const fields = Array.from(formRegistro.elements);
fields.forEach((field) => {
    const errorBox = document.getElementById(field.id + "Error");
    field.addEventListener("invalid", (event) => {
        let message = "";
        if (field.id === "usuario" && (field.validity.valueMissing || field.validity.tooShort))
            message = "El nombre debe estar formado por al menos 3 caracteres de palabra.";
        } else if (field.id === 'password2' && (field.validity.valueMissing || field.validity.minLength))
            message = "El password debe tener una minúscula, mayúscula, dígito y caracter especial";
        } else if (field.id === "email" && (field.validity.valueMissing || field.validity.typeMismatch))
            message = "El correo debe tener un formato correcto";
        }
        errorBox.textContent = message;
    });

    field.addEventListener("input", () => {
        // Limpia el mensaje de error personalizado
        errorBox.textContent = ""; // Limpia el texto del error
    });
});
});

```

## Para saber más

El API de validación de restricciones de HTML5 es muy potente y conviene conocer bien su funcionamiento para utilizarlo de la manera más adecuada. Consulta el artículo siguiente para conocerlo mejor

[API de validación de restricciones HTML5](#)

A continuación se muestra el código del script `index.php` completo:

```

<!DOCTYPE html>
<html lang="es">

```

```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!-- Bootstrap CDN -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
  <!-- Bootstrap Font Icon CSS -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.8.1/font/bootstrap-icons.css">
  <link rel="stylesheet" href="stylesheet.css">
  <title>Registro</title>
</head>
<body class="bg-info">
  <div class="container mt-5">
    <?php if (filter_has_var(INPUT_POST, 'enviar')): ?>
      <div class="alert alert-success" id="mensaje" role="alert">
        Registro realizado con éxito
      </div>
    <?php else: ?>
      <div class="d-flex justify-content-center h-100">
        <div class="card w-50">
          <div class="card-header">
            <h3><i class="bi bi-gear p-2"></i>Registro</h3>
          </div>
          <div class="card-body">
            <form id="registro" name="registro" method="POST" action="?= $_SEI
              <div class="input-group my-2">
                <span class="input-group-text"><i class="bi bi-person"></i>
                <input type="text" class="form-control" placeholder="usuario"
                  id="usuario" name="usuario" minlength="3" autofocus
              </div>
              <div class="text-danger" id="usuarioError"></div>
              <div class="input-group my-2">
                <span class="input-group-text"><i class="bi bi-key"></i></span>
                <input type="password" class="form-control" placeholder="contraseña"
                  id="password1" name="password1" required pattern="(?!
              </div>
              <div class="text-danger" id="password1Error"></div>
              <div class="input-group my-2">
                <span class="input-group-text"><i class="bi bi-key"></i></span>
                <input type="password" class="form-control" placeholder="confirmar contraseña"
                  id="password2" name="password2" required pattern="(?!
              </div>
              <div class="text-danger" id="password2Error"></div>
              <div class="input-group my-2">
                <span class="input-group-text"><i class="bi bi-envelope"></i>
                <input type="email" class="form-control" placeholder="e-Mail"
                  id="email" required
              </div>
              <div class="text-danger" id="emailError"></div>
              <div class="text-end">
                <input type="submit" value="Registrar" class="btn btn-info"
              </div>
            </form>
          </div>
        </div>
      </div>
      <script src="validar.js"></script>
    </div>
  <?php endif ?>
</div>

```

```

    </body>
</html>

```

Por último, presentamos un mecanismo de validación que integra Bootstrap y el API de validación de restricciones de HTML5. Su funcionamiento es el siguiente:

Bootstrap define estilos para elementos con las pseudoclases `:valid` e `:invalid`, de tal manera que cuando un elemento adquiere alguna de esas pseudoclases se le aplica el formato asociado a un valor válido o inválido. Estas reglas de estilos tienen como ámbito elementos formulario que tengan asociada la clase `was-validated`, de manera que es necesario añadir dicha clase al elemento usando JavaScript cuando se realiza el proceso de validación.

Además si se desea personalizar mensajes de error se pueden incluir elementos `div` con la clase `invalid-feedback` que contienen el error a mostrar. A continuación se muestra el código PHP y JavaScript correspondiente a este tipo de validación:

```

<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap CDN -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <!-- Bootstrap Font Icon CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.8.1/font/bootstrap-icons.css">
    <title>Registro</title>
  </head>
  <body class="bg-info">
    <div class="container mt-5">
      <?php if (isset($_POST['enviar'])): ?>
        <div class="alert alert-success" id="mensaje" role="alert">
          Registro realizado con éxito
        </div>
      <?php else: ?>
        <div class="d-flex justify-content-center h-100">
          <div class="card w-50">
            <div class="card-header">
              <h3><i class="bi bi-gear p-2"></i>Registro</h3>
            </div>
            <div class="card-body">
              <form id="registro" name="registro" method="POST" action="<?=$_SERVER['SCRIPT_NAME']">
                <div class="input-group my-2">
                  <span class="input-group-text"><i class="bi bi-person"></i></span>
                  <input type="text" class="form-control" placeholder="usuario"
                    id="usuario" name="usuario" minlength="3" autofocus>
                  <div class="invalid-feedback">
                    El nombre de estar formado por al menos 3 caracteres de
                  </div>
                </div>
                <div class="input-group my-2">
                  <span class="input-group-text"><i class="bi bi-key"></i></span>
                  <input type="password" class="form-control" placeholder="contraseña"
                    id="password1" name="password1" required pattern="[a-zA-Z0-9]{8,}">
                </div>
              </form>
            </div>
          </div>
        </div>
      </?php>
    </div>
  </body>
</html>

```

```

</div>
<div class="input-group my-2">
  <span class="input-group-text"><i class="bi bi-key"></i></span>
  <input type="password" class="form-control" placeholder="Password"
    id="password2" name="password2" required pattern="(?!.*[a-z])(?!.*[A-Z])(?!.*\d).*$">
  <div class="invalid-feedback">
    El password debe tener una minúscula, mayúscula, dígito y carácter especial.
  </div>
</div>
<div class="input-group my-2">
  <span class="input-group-text"><i class="bi bi-envelope"></i></span>
  <input type="email" class="form-control" placeholder="e-Mail"
    id="email" name="email" required>
  <div class="invalid-feedback">
    El correo debe tener un formato correcto
  </div>
</div>
<div class="text-end">
  <input type="submit" value="Registrar" class="btn btn-info">
</div>
</form>
</div>
</div>
<?php endif ?>
</div>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
<script src="validar.js"></script>
</body>
</html>

```

```

const formRegistro = document.getElementById('registro');
formRegistro.addEventListener('submit', validaForm);

```

```

function validaForm(e) {
  const form = e.target;
  const password1 = form.querySelector('#password1');
  const password2 = form.querySelector('#password2');
  const errorPassword2 = form.querySelector('#password2 ~ .invalid-feedback');

  // Limpiar estado de validación previo
  password2.classList.remove('is-invalid');

  const errPassword = password1.value !== password2.value;
  if (errPassword) {
    // Establece el mensaje de error y evita el envío del formulario
    password2.classList.add('is-invalid');
    errorPassword2.textContent = password2.validationMessage; // Asegúrate de que el mensaje de error sea el correcto
    errorPassword2.textContent = "Los passwords introducidos deben de ser iguales";
    e.preventDefault();
    e.stopImmediatePropagation();
  }
  if (!form.checkValidity()) {
    e.preventDefault();
    e.stopImmediatePropagation();
  }
}

```

```
form.classList.add('was-validated');  
}
```



## 2.1.2.- Validación de los datos de un formulario en el servidor.

Para realizar la validación del formulario en el servidor utilizando PHP, podemos programar nuestras propias funciones de validación que comprueban que los datos que se obtienen del formulario son correctos, como por ejemplo:

```
<?php
function validarNombre($nombre){
    return (strlen($nombre)>=4);
}
function validarEmail($email){
    return preg_match("/^[a-z0-9]+([_\\.-][a-z0-9]+)*@[a-z0-9]+(\\.[a-z0-9]+)*\\.\\.[a-z]{2,}$");
}
function validarPasswords($pass1, $pass2) {
    return ($pass1 == $pass2) && (strlen($pass1) > 5);
}
```

Fíjate en el uso de la función "`preg_match()`" y de expresiones regulares (regex) para validar la dirección de correo.

### [Función preg\\_match](#)

Ten en cuenta que las barras invertidas ("`\`") tienen un significado especial dentro de una cadena de PHP (sirven para escapar el siguiente carácter, por ejemplo por si queremos mostrar unas comillas); por ese motivo, la doble barra "`\\`" se convierte en una barra simple "`\`" cuando se interpreta la expresión regular.

## Debes conocer

En ocasiones es importante saber construir expresiones regulares en PHP para realizar comparación de cadenas de texto con patrones. Tienes más información en el siguiente enlace.

[Expresiones regulares en PHP.](#)

De manera alternativa, puedo usar las posibilidades de validación que me ofrecen funciones como `filter_input` y `filter_var` que me permiten crear filtros de validación de distintos tipos que comprueban si los datos son correctos::

```

$usuario = filter_input(INPUT_POST, 'usuario', FILTER_UNSAFE_RAW);
$errorUsuarioFormato = (filter_var($usuario, FILTER_VALIDATE_REGEXP, ["options" => [
    "regexp" => "/^[a-zA-ZáéíóúÁÉÍÓÚñÑüÜ'`^~]+(\s+[a-zA-ZáéíóúÁÉÍÓÚñÑüÜ'`^~]+){
$password1 = filter_input(INPUT_POST, 'password1', FILTER_UNSAFE_RAW);
$password2 = filter_input(INPUT_POST, 'password2', FILTER_UNSAFE_RAW);
$errorPasswordNoRepetido = ($password1 !== $password2);
$errorPasswordFormato = (filter_var($password1, FILTER_VALIDATE_REGEXP, ["options" => [
    "regexp" => "/(?!.*\d)(?!.*[a-z])(?!.*[A-Z])(?!.*[\W_]).{8,}/" ]]) === false
$email = filter_input(INPUT_POST, 'email', FILTER_UNSAFE_RAW);
$errorEmailFormato = (filter_var($email, FILTER_VALIDATE_REGEXP, ["options" => [
    "regexp" => "/^[a-z0-9]+([_\\-\\.][a-z0-9]+)*@[a-z0-9]+(\\-\\.?[a-z0-9]+)*\\.\\
$errorPassword = $errorPasswordFormato || $errorPasswordNoRepetido;
$error = $errorUsuarioFormato || $errorEmailFormato || $errorPassword;

```

A continuación se muestra cómo se usa [Bootstrap 5 Form Validation](#) para facilitar la validación del formulario y resaltar los mensajes de error que informan de la razón de los errores en los valores.

Fíjate que se utiliza el operador ternario de comparación en él la expresión "(expr1) ? (expr2) : (expr3)" evalúa a "expr2" si "expr1" se evalúa como "TRUE" y a "expr3" si "expr1" se evalúa como "FALSE".

Se asigna la clase "is-invalid" a aquellos elementos HTML <input ...> para los que se haya detectado un error en el valor asociado. Si un elemento HTML <input ...> tiene asignada la clase "is-invalid" se mostrará el contenido del elemento con la clase "invalid-feedback" con el que comparte grupo. A continuación se muestra un extracto de la validación del valor asociado al nombre de usuario:

```

<div class="input-group my-2">
    <span class="input-group-text"><i class="bi bi-person"></i></span>
    <input type="text" class="form-control" <?=$_isset($errorUsuario) ?
        id="usuario" name="usuario" value="<?=$_usuario ? ' ' ? :
    <div class="invalid-feedback">
        <?=$_RETRO_NOMBRE_FORMATO ?>
    </div>
</div>

```

**Registro**

Debe tener más de tres caracteres.

Deben tener más de 5 caracteres o ser iguales.

La dirección de email NO es válida.

El código del script completo se incluye a continuación:

```
<?php
define("RETRO_NOMBRE_FORMATO", "El nombre de estar formado por al menos 3 caracteres de palabra");
define("RETRO_EMAIL_FORMATO", "El correo debe tener un formato correcto");
define("RETRO_PASS_REPETIDO", "Los passwords introducidos deben de ser iguales");
define("RETRO_PASS_FORMATO", "El password debe tener una minúscula, mayúscula, dígito y carácter especial");

if (filter_has_var(INPUT_POST, 'enviar')) {
    $usuario = filter_input(INPUT_POST, 'usuario', FILTER_UNSAFE_RAW);
    $errorUsuarioFormato = (filter_var($usuario, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/^[a-zA-ZáéíóúÁÉÍÓÚñÑüÜ'`-]+(\s+[a-zA-ZáéíóúÁÉÍÓÚñÑüÜ'`-]+){1,3}$/"]))) === false;
    $password1 = filter_input(INPUT_POST, 'password1', FILTER_UNSAFE_RAW);
    $password2 = filter_input(INPUT_POST, 'password2', FILTER_UNSAFE_RAW);
    $errorPasswordNoRepetido = ($password1 !== $password2);
    $errorPasswordFormato = (filter_var($password1, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[W_]).{8,}/"]))) === false;
    $email = filter_input(INPUT_POST, 'email', FILTER_UNSAFE_RAW);
    $errorEmailFormato = (filter_var($email, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/^[a-z0-9]+([\-\.\-][a-z0-9]+)*@([a-z0-9]+([\-\.\-][a-z0-9]+)*)+\.[a-z]{2,4}$/"]))) === false;
    $errorPassword = $errorPasswordFormato || $errorPasswordNoRepetido;
    $error = $errorUsuarioFormato || $errorEmailFormato || $errorPassword;
}

?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap CDN -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <!-- Bootstrap Font Icon CSS -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.8.1/font/bootstrap-icons.css">
    <title>Registro</title>
</head>
<body class="bg-info">
    <div class="container mt-5">
        <?php if (isset($error) && !$error): ?>
            <div class="alert alert-success" id="mensaje" role="alert">
                Registro realizado con éxito
            </div>
        <?php endif ?>
        <div class="d-flex justify-content-center h-100">
            <div class="card w-50">
                <div class="card-header">
                    <h3><i class="bi bi-gear p-2"></i>Registro</h3>
                </div>
                <div class="card-body">
                    <form id="registro" name="registro" action="index.php" method="POST" novalidate="">
                        <div class="input-group my-2">
                            <span class="input-group-text"><i class="bi bi-person"></i></span>
                            <input type="text" class="form-control" <?php if (isset($errorUsuarioFormato)): ?>
                                <div class="text-danger"><small>El nombre de estar formado por al menos 3 caracteres de palabra</small>
                            </div>
                        </div>
                        <div class="input-group my-2">
                            <span class="input-group-text"><i class="bi bi-envelope"></i></span>
                            <input type="text" class="form-control" <?php if (isset($errorEmailFormato)): ?>
                                <div class="text-danger"><small>El correo debe tener un formato correcto</small>
                            </div>
                        </div>
                        <div class="input-group my-2">
                            <span class="input-group-text"><i class="bi bi-lock"></i></span>
                            <input type="password" class="form-control" <?php if (isset($errorPasswordFormato)): ?>
                                <div class="text-danger"><small>El password debe tener una minúscula, mayúscula, dígito y carácter especial</small>
                            </div>
                        </div>
                        <div class="input-group my-2">
                            <span class="input-group-text"><i class="bi bi-lock"></i></span>
                            <input type="password" class="form-control" <?php if (isset($errorPasswordNoRepetido)): ?>
                                <div class="text-danger"><small>Los passwords introducidos deben de ser iguales</small>
                            </div>
                        </div>
                        <div class="input-group my-2">
                            <span class="input-group-text"><i class="bi bi-check-circle"></i></span>
                            <input type="submit" value="Registrar" class="form-control">
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</body>
</html>
```



```

        id="usuario" name="usuario" value="<?= $usuario ?? ' ' ?>";
    <div class="invalid-feedback">
        <?= RETRO_NOMBRE_FORMATO ?>
    </div>
</div>
<div class="input-group my-2">
    <span class="input-group-text"><i class="bi bi-key"></i></span>
    <input type="password" class="form-control" placeholder="contraseña"
        value="<?= $password1 ?? ' ' ?>">
</div>
<div class="input-group my-2">
    <span class="input-group-text"><i class="bi bi-key"></i></span>
    <input type="password" class="form-control" <?= isset($errorPassword) ?>
        placeholder="Repita la contraseña" id="password2" name="password2">
    <div class="invalid-feedback">
        <?php if (isset($errorPasswordNoRepetido) && $errorPasswordNoRepetido) : ?>
            <?= RETRO_PASS_REPETIDO ?>
        <?php elseif (isset($errorPassword) && $errorPassword) : ?>
            <br><?= RETRO_PASS_FORMATO ?>
        <?php endif ?>
    </div>
</div>
<div class="input-group my-2">
    <span class="input-group-text"><i class="bi bi-envelope"></i></span>
    <input type="email" class="form-control" <?= isset($errorEmailFormato) ?>
        placeholder="e-Mail" name="email" id="email" value="<?= $email ?? ' ' ?>">
    <div class="invalid-feedback">
        <?= RETRO_EMAIL_FORMATO ?>
    </div>
</div>
<div class="text-end">
    <input type="submit" value="Registrar" class="btn btn-info" name="submit">
</div>
</form>
</div>
</div>
</div>
<script src = "https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js">
</body>
</html>

```

## Autoevaluación

¿Qué significa la siguiente expresión regular: `\.[a-z]{2,}`?

- ☐ Una barra seguida por un carácter cualquier, una letra entre la a y la z, y un número mayor o igual a 2.
- ☐ Un punto seguido de 2 o más letras minúsculas.

Revisa el contenido del enlace sobre expresiones regulares.

Efectivamente. El punto es un carácter especial, por lo que necesita ser escapado con la barra invertida. Los corchetes indican un carácter entre la a y la z (una letra minúscula). Y las llaves indican repetición, dos o más veces en este caso. Se trata de una expresión regular que podemos utilizar para validar los dominios de primer nivel (com, org, es, ...).

## Solución

1. Incorrecto
2. Opción correcta

## 2.2.- Utilización de AJAX con PHP.

### Caso práctico

En los últimos días **Juan** y **Carlos** han logrado grandes avances. Tienen claras las capacidades que les ofrece la programación del cliente web. Han profundizado en la tecnología **AJAX**. Y saben, incluso, de qué manera pueden integrar en una misma aplicación ambos tipos de programación: la programación del servidor web en lenguaje **PHP**, y la programación del cliente web en lenguaje **JavaScript**.



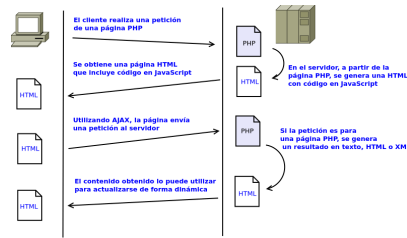
ólo les falta un detalle: las herramientas. Saben lo qué tienen que hacer, pero antes de ponerse manos a la obra, deben decidir de qué forma hacerlo. **María** les ha informado de que existen multitud de librerías y herramientas que se pueden utilizar para agilizar la programación de aplicaciones en lenguaje **JavaScript**. Su último paso será decidir qué librerías y herramientas utilizarán para facilitar la programación de la aplicación web, tomando como punto de partida que sería preferible utilizar un único lenguaje en todo el proyecto para evitar la fragmentación del código de la misma.

Como ya sabes, la tecnología **AJAX** se utiliza desde el cliente para permitir comunicaciones asíncronas con el servidor web, sin necesidad de recargar la página que se muestra en el navegador. Se basa en la utilización de código en lenguaje **JavaScript**. Por tanto, te estarás preguntando, ¿qué tiene que ver la tecnología **AJAX** con el lenguaje **PHP**?

Acabamos de ver cómo se pueden crear aplicaciones web que utilicen de forma simultánea la programación del cliente web (con **JavaScript**) y la programación del servidor web (con **PHP**). En el procedimiento seguido en el ejemplo anterior, ambas tecnologías coexistían en una misma página, pero su programación era independiente. El código **PHP** se ejecutaba en el servidor, y en la misma página se incluían también guiones en lenguaje **JavaScript** que se ejecutan en el navegador. En nuestro ejemplo solamente existía una relación: si el navegador permitía la ejecución de código **JavaScript**, se deshabilitaba el envío del formulario y la validación se realizaba en el cliente. En este caso no se llegaba a enviar la página al servidor y no se ejecutaba por tanto el código **PHP** de validación.

Si vas a usar aplicaciones que utilicen ambos lenguajes, es preferible tener un mecanismo mejor para integrarlos. Afortunadamente existen librerías para **PHP** que te permiten aprovechar las capacidades de **JavaScript** utilizando casi exclusivamente código en lenguaje **PHP**. Estas librerías definen una serie de objetos que puedes utilizar en el código de servidor, y que generan de forma automática código **JavaScript** en las páginas web que se envían al cliente.

La mayoría de estas librerías añaden a las aplicaciones web funcionalidades de la tecnología **AJAX**. Esto es: permiten crear páginas **PHP** que, tras ejecutarse en el servidor, producen páginas web que incorporan código **JavaScript** con funcionalidades **AJAX**. El mecanismo de funcionamiento lo puedes observar en el siguiente diagrama.



Editor DIA (Elaboración propia)

Muchas de estas librerías suelen apoyarse en librerías JavaScript como **jQuery** para la ejecución de código en el cliente. En Internet puedes encontrar información sobre librerías **PHP** con soporte para **AJAX**.

El uso de **AJAX** tiene considerables ventajas:

- Ahorro de ancho de banda.
- Mejor rendimiento a la hora de cargar los resultados de la página ya que no se recrea la página completa.
- Mejor experiencia de usuario ya que el funcionamiento está más cercano a una aplicación web y no a una secuencia de páginas web.

Por otro lado, también hay que tener en cuenta los siguientes inconvenientes:

- Complejidad de la tecnología.
- Necesidad de manejar el DOM mediante código en JavaScript lo que dificulta el trabajo de los motores de búsqueda.
- Imposibilidad de utilizar el botón de "volver atrás" ya que el funcionamiento no está basado en la carga de páginas web.

## 2.2.1.- Ajax (JavaScript puro)

---

AJAX se basa en el uso del objeto XMLHttpRequest de la librería de JavaScript.

Los métodos que soporta el objeto XMLHttpRequest son:

- **open**: Establece los parámetros de la petición que se realiza al servidor. Los parámetros necesarios son el método HTTP empleado y la URL destino.
- **send**: Realiza la petición HTTP al servidor.
- **abort**: Detiene la petición actual.
- **onreadystatechange**: Responsable de manejar los eventos que se producen. Se invoca cada vez que se produce un cambio en el estado de la petición HTTP. Normalmente es una referencia a una función JavaScript.
- **onload**: Se dispara solo cuando la petición ha completado exitosamente, eliminando la necesidad de verificar el estado de readyState y simplificando la comprobación del código de estado HTTP.
- **setRequestHeader**: Permite establecer cabeceras personalizadas en la petición HTTP. Se debe invocar el método open antes de setRequestHeader.
- **getResponseHeader**: Devuelve una cadena de texto con el contenido de la cabecera solicitada.
- **getAllResponseHeaders**: Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor.
- **onprogress**: Se dispara periódicamente cuando se están transmitiendo datos hacia o desde un servidor. Proporciona detalles sobre el progreso de la operación, lo que permite a los desarrolladores implementar, por ejemplo, una barra de progreso o cualquier otro indicador visual para informar al usuario sobre el avance de la operación.

Además el objeto XMLHttpRequest tiene las propiedades siguientes:

- **readyState**: estado de la conexión del objeto XMLHttpRequest:
  - 0: petición no inicializada
  - 1: conexión con el servidor establecida
  - 2: petición recibida
  - 3: procesando petición
  - 4: petición finalizada y respuesta lista
- **status**: estado del mensaje de respuesta HTTP. Algunos de los códigos más frecuentes son:
  - 200: "OK"
  - 403: "Forbidden"
  - 404: "Page not found"
  - ...
- **statusText**: devuelve el texto asociado al código de estado (p.e. "OK", "Not Found", etc)
- **response**: contenido del cuerpo de la respuesta. El tipo del contenido depende de valor del parámetro responseType de la petición.
- **responseText**: contenido de la respuesta.
- **responseXML**: contenido de la respuesta en XML.

La funcionalidad de validación del formulario de registro podría codificarse utilizando la tecnología AJAX con JavaScript puro de la siguiente forma, **validar.js**:

```
document.addEventListener("DOMContentLoaded", function () {  
    const registroForm = document.getElementById('registro');  
    registroForm.addEventListener("submit", validaForm);  
});
```

```

function validaForm(e) {
    event.preventDefault();
    const form = this;

    // Preparar datos del formulario para enviar
    const formData = new FormData(form);
    formData.append(form.enviar.name, form.enviar.value);

    const xhr = new XMLHttpRequest();
    xhr.responseType = 'json';

    // Configurar la solicitud
    xhr.open('POST', 'index.php', true);
    xhr.setRequestHeader('Accept', 'application/json'); // Esperamos JSON de vuelta
    xhr.send(formData);

    // Manejar la respuesta
    xhr.onload = function () {
        form.classList.remove('was-validated');
        const response = xhr.response;
        // Ajustar la validación en cada campo específico
        Array.from(form.elements).forEach(input => {
            if (response.errors[input.name]) {
                const feedback = input.nextElementSibling;
                feedback.textContent = response.errors[input.name];
                feedback.style.display = 'block';
                input.classList.add('is-invalid');
                input.classList.remove('is-valid');
            } else {
                input.classList.remove('is-invalid');
                input.classList.add('is-valid'); // Añadir is-valid si no hay errores
                if (input.nextElementSibling) {
                    feedback = input.nextElementSibling;
                    feedback.textContent = "";
                }
            }
        });
        if (response.success) {
            // alert('Registro completado con éxito.');
```

- Primero, inicializamos el objeto XMLHttpRequest. que es responsable de la llamada AJAX

- Se monitoriza la propiedad `readyState` según va cambiando de valor durante el ciclo de vida de la solicitud
- Cada vez que `readyState` cambie de valor se ejecuta la función que maneja el evento de cambio de estado
- Dentro de esa función se comprueba si la petición se ha procesado y la respuesta está lista (valor 4)
- Recuperamos el contenido json enviado por el servidor y lo examinamos para cambiar las clases de los elementos HTML que muestran los posibles errores al usuario.

## Para saber más

El proceso de envío y respuesta de un objeto `XMLHttpRequest` al servidor desde el cliente es muy importante para entender el funcionamiento de Ajax en JavaScript puro. Consulta el artículo sobre Ajax que se muestra a continuación.

[Guía básica sobre AJAX](#)

El código completo del script `index.php` se muestra a continuación:

```
<?php
define("RETRO_NOMBRE_FORMATO", "El nombre de estar formado por al menos 3 caracteres de palabra");
define("RETRO_EMAIL_FORMATO", "El correo debe tener un formato correcto");
define("RETRO_PASS_NO_REPETIDO", "Los passwords introducidos deben de ser iguales");
define("RETRO_PASS_FORMATO", "El password debe tener una minúscula, mayúscula, dígito y carácter especial");

if (!empty($_POST)) {
    $usuario = filter_input(INPUT_POST, 'usuario', FILTER_UNSAFE_RAW);
    $errorUsuarioFormato = (filter_var($usuario, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/^[a-zA-ZáéíóúÁÉÍÓÚñÑüÜ'`^\\-]+(\\s+[a-zA-ZáéíóúÁÉÍÓÚñÑüÜ'`^\\-]+)*$/"]))) === false;
    $password1 = filter_input(INPUT_POST, 'password1', FILTER_UNSAFE_RAW);
    $password2 = filter_input(INPUT_POST, 'password2', FILTER_UNSAFE_RAW);
    $errorPasswordNoRepetido = ($password1 !== $password2);
    $errorPasswordFormato = (filter_var($password1, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/(?!.*\\d)(?!.*[a-z])(?!.*[A-Z])(?!.*[\\W_]).{8,}/"]))) === false;
    $email = filter_input(INPUT_POST, 'email', FILTER_UNSAFE_RAW);
    $errorEmailFormato = (filter_var($email, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/^[a-z0-9]+([_\\-\\.][a-z0-9]+)*@[a-z0-9]+(\\-\\.)*[a-z0-9]+\\.([a-z0-9]+)*$/"]))) === false;
    $errors = [];
    if ($errorUsuarioFormato) {
        $errors['usuario'] = RETRO_NOMBRE_FORMATO;
    }
    if ($errorPasswordNoRepetido) {
        $errors['password2'] = RETRO_PASS_NO_REPETIDO;
    } else if ($errorPasswordFormato) {
        $errors['password2'] = RETRO_PASS_FORMATO;
    }
    if ($errorEmailFormato) {
        $errors['email'] = RETRO_EMAIL_FORMATO;
    }
}
```

```

if (filter_has_var(INPUT_POST, 'enviar')) {
    $response['success'] = empty($errors);
    $response['errors'] = $errors;
    header('Content-type: application/json');
    echo json_encode($response);
    die;
} else {
    $procesa = true;
}
}
?>

<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <!-- Bootstrap CDN -->
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
        <!-- Bootstrap Font Icon CSS -->
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.8.1/font/bootstrap-icons.css">
        <title>Registro</title>
    </head>
    <body class="bg-info">
        <div class="container mt-5">
            <?php if (isset($procesa)): ?>
                <div class="alert alert-success" id="mensaje" role="alert">
                    Registro realizado con éxito
                </div>
            <?php else: ?>
                <div class="d-flex justify-content-center h-100">
                    <div class="card w-50">
                        <div class="card-header">
                            <h3><i class="bi bi-gear p-2"></i>Registro</h3>
                        </div>
                        <div class="card-body">
                            <form id="registro" name="registro" action="index.php" method="POST">
                                <div class="input-group my-2">
                                    <span class="input-group-text"><i class="bi bi-person"></i></span>
                                    <input type="text" class="form-control" placeholder="usuario" id="usuario" name="usuario" value="<?= $usuario ?? ' ' ">
                                    <div class="invalid-feedback">
                                        </div>
                                </div>
                                <div class="input-group my-2">
                                    <span class="input-group-text"><i class="bi bi-key"></i></span></div>
                                    <input type="password" class="form-control" placeholder="contraseña" id="password1" name="password1" value="<?= $password1 ?? ' ' ">
                                </div>
                                <div class="input-group my-2">
                                    <span class="input-group-text"><i class="bi bi-key"></i></span></div>
                                    <input type="password" class="form-control" placeholder="Repita la contraseña" id="password2" name="password2" value="<?= $password2 ?? ' ' ">
                                </div>
                                <div class="input-group my-2">
                                    <span class="input-group-text"><i class="bi bi-envelope"></i></span>
                                </div>
                            </form>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>

```



```
<input type="email" class="form-control"
        placeholder="e-Mail" name="email" id="email" value='
<div class="invalid-feedback">
</div>
</div>
<div class="text-end">
    <input type="submit" value="Registrar" class="btn btn-info'
</div>
</form>
</div>
</div>
</div>
<?php endif ?>
</div>
<script src = "https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.mi
<script src="validar.js"></script>
</body>
</html>
```

## 2.2.2.- Ajax (jQuery)

---

jQuery es una librería JavaScript que simplifica la programación en JavaScript. jQuery permite reducir el número de líneas de código que requieren ciertas funcionalidades en JavaScript puro. La librería ofrece las siguientes funcionalidades:

- Manipulación de DOM cargado en el navegador
- Manipulación de los estilos CSS
- Gestión de eventos de elementos HTML
- Efectos y animaciones
- Implementación de comunicación entre el cliente y el servidor con Ajax

En esta sección vamos a estudiar las facilidades que nos ofrece jQuery para trabajar con AJAX. jQuery proporciona un número de métodos para solicitar información en formato texto, HTML, XML o JSON de un servidor remoto usando los métodos HTTP GET y POST.

El método de jQuery mas completo para implementar una petición `ajax` es:

```
$.ajax({name:value, name:value, ... })
```

Los parámetros especifican uno o más pares de valores de la petición AJAX. Los más importantes son:

- **type**: Tipo de petición (GET o POST).
- **url**: URL a donde enviar la petición. En nuestro caso el script PHP que procesa la petición.
- **data**: Especifica los datos enviados al servidor.
- **dataType**: Tipo de datos que se espera en la respuesta del servidor.
- **async**: Valor booleano que indica si la petición es síncrona o asíncrona. Por defecto es asíncrona.
- **context**: Especifica el valor de "this" en las callbacks asociadas al tratamiento de la respuesta.
- **beforeSend** (xhr): Función que se ejecuta antes de enviar la petición.
- **success** (result status, xhr): Función que se ejecuta cuando la petición ha sido procesada con éxito.
- **error** (xhr, status, error): Función que se ejecuta cuando la petición falla.
- **complete** (xhr, status): Función que se ejecuta cuando la petición se ha acabado (después de success y error).
- **username**: Especifica el usuario usado en el proceso de autenticación HTTP.
- **password**: Especifica una clave usada en el proceso de autenticación HTTP.
- **timeout**: El tiempo que se da en el cliente a la petición antes de darla por fallida.

La funcionalidad de validación del formulario de registro podría codificarse utilizando la tecnología AJAX con jQuery de la siguiente forma, `validar.js`:

```
$(document).ready(function () {  
    $('#registro').submit(validaForm);  
});  
function validaForm(e) {
```

```

e.preventDefault();
var form = this;

// Usar el método de jQuery para crear el objeto FormData
var data = $(this).serialize();
data += '&enviar=' + encodeURIComponent($('input[name="enviar"]').val());

// Usar jQuery.ajax() para enviar los datos
$.ajax({
    url: 'index.php',
    type: 'POST',
    data: data,
    dataType: 'json', // Esperamos JSON de vuelta
    success: function (response) {
        $(form).removeClass('was-validated');
        // Ajustar la validación en cada campo específico
        $.each(form.elements, function () {
            var input = this;
            var feedback = $(input).next('.invalid-feedback');
            if (response.errors[input.name]) {
                $(feedback).text(response.errors[input.name]);
                $(feedback).show();
                $(input).addClass('is-invalid').removeClass('is-valid');
            } else {
                $(input).removeClass('is-invalid').addClass('is-valid');
                $(feedback).text('').hide();
            }
        });
        if (response.success) {
            // Si quieres hacer algo al tener éxito, como redirigir o mostrar un mensaje
            // alert('Registro completado con éxito.');
```

También usamos JQuery para manipular el DOM y añadir o eliminar clases a distintos elementos para mostrar u ocultar los mensajes de error del formulario.

## Debes conocer

Para acceder al enlace CDN de descarga de la última versión de JQuery puede acceder a este [enlace](#).

El código completo del script `index.php` se muestra a continuación:

```

<?php
define("RETRO_NOMBRE_FORMATO", "El nombre de estar formado por al menos 3 caracteres de palabra
define("RETRO_EMAIL_FORMATO", "El correo debe tener un formato correcto");
define("RETRO_PASS_NO_REPETIDO", "Los passwords introducidos deben de ser iguales");
define("RETRO_PASS_FORMATO", "El password debe tener una minúscula, mayúscula, dígito y caracte

if (!empty($_POST)) {
    $usuario = filter_input(INPUT_POST, 'usuario', FILTER_UNSAFE_RAW);
    $errorUsuarioFormato = (filter_var($usuario, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/^[a-zA-ZáéíóúÁÉÍÓÚñÑüÜ'`~\-\-]+\s+[a-zA-ZáéíóúÁÉÍÓÚñÑüÜ'`~\-\-]+$/"]
    $password1 = filter_input(INPUT_POST, 'password1', FILTER_UNSAFE_RAW);
    $password2 = filter_input(INPUT_POST, 'password2', FILTER_UNSAFE_RAW);
    $errorPasswordNoRepetido = ($password1 !== $password2);
    $errorPasswordFormato = (filter_var($password1, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/(?!.*\d)(?!.*[a-z])(?!.*[A-Z])(?!.*[\W_]).{8,}/" ]]) === false
    $email = filter_input(INPUT_POST, 'email', FILTER_UNSAFE_RAW);
    $errorEmailFormato = (filter_var($email, FILTER_VALIDATE_REGEXP, ["options" => [
        "regexp" => "/^[a-z0-9]+([_\.\-][a-z0-9]+)*@[a-z0-9]+([\.\-][a-z0-9]+)*\.\w+$/" ]])
    $errors = [];
    if ($errorUsuarioFormato) {
        $errors['usuario'] = RETRO_NOMBRE_FORMATO;
    }
    if ($errorPasswordNoRepetido) {
        $errors['password2'] = RETRO_PASS_NO_REPETIDO;
    } else if ($errorPasswordFormato) {
        $errors['password2'] = RETRO_PASS_FORMATO;
    }
    if ($errorEmailFormato) {
        $errors['email'] = RETRO_EMAIL_FORMATO;
    }
    if (filter_has_var(INPUT_POST, 'enviar')) {
        $response['success'] = empty($errors);
        $response['errors'] = $errors;
        header('Content-type: application/json');
        echo json_encode($response);
        die;
    } else {
        $procesa = true;
    }
}
?>

```

```

<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <!-- Bootstrap CDN -->
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
        <!-- Bootstrap Font Icon CSS -->
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.8.1/font/bootstrap-icons.css">
        <title>Registro</title>
    </head>
    <body class="bg-info">
        <div class="container mt-5">
            <?php if (isset($procesa)): ?>
                <div class="alert alert-success" id="mensaje" role="alert">

```

```

        Registro realizado con éxito
    </div>
<?php else: ?>
    <div class="d-flex justify-content-center h-100">
        <div class="card w-50">
            <div class="card-header">
                <h3><i class="bi bi-gear p-2"></i>Registro</h3>
            </div>
            <div class="card-body">
                <form id="registro" name="registro" action="index.php" method="POST">
                    <div class="input-group my-2">
                        <span class="input-group-text"><i class="bi bi-person"></i></span>
                        <input type="text" class="form-control" placeholder="usuario"
                            id="usuario" name="usuario" value="<?= $usuario ?? ' ' ">
                        <div class="invalid-feedback">
                        </div>
                    </div>
                    <div class="input-group my-2">
                        <span class="input-group-text"><i class="bi bi-key"></i></span></div>
                        <input type="password" class="form-control" placeholder="contraseña"
                            value="<?= $password1 ?? ' ' ?>">
                    </div>
                    <div class="input-group my-2">
                        <span class="input-group-text"><i class="bi bi-key"></i></span></div>
                        <input type="password" class="form-control"
                            placeholder="Repita la contraseña" id="password2" name="password2">
                        <div class="invalid-feedback">
                        </div>
                    </div>
                    <div class="input-group my-2">
                        <span class="input-group-text"><i class="bi bi-envelope"></i></span></div>
                        <input type="email" class="form-control"
                            placeholder="e-Mail" name="email" id="email" value="<?= $email ?? ' ' ">
                        <div class="invalid-feedback">
                        </div>
                    </div>
                    <div class="text-end">
                        <input type="submit" value="Registrar" class="btn btn-info">
                    </div>
                </form>
            </div>
        </div>
    </div>
<?php endif ?>
</div>
<script src="https://code.jquery.com/jquery-3.6.0.min.js" integrity="sha256-/xUj+30JU5"
<script src = "https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
<script src="validar.js"></script>
</body>
</html>

```