Desarrollo web en entorno servidor

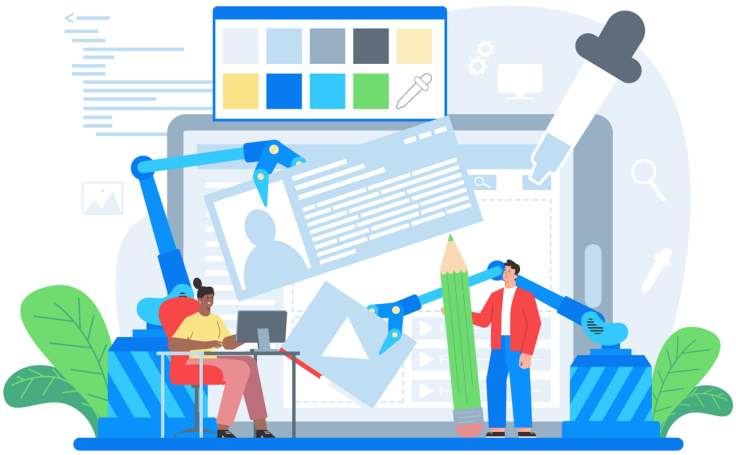
Unidad 9: Generación dinámica de páginas web interactivas

ÍNDICE

INTRODUCCIÓN

En esta unidad didáctica nos pondremos en la piel de un **desarrollador *frontend*** que es aquel que crea interfaces y **scripts del lado del cliente**.

En este módulo ya habíamos creado páginas web con formularios para que un usuario pudiese interactuar con nuestras aplicaciones, pero hasta ahora habíamos puesto toda la lógica en el lado del servidor. Esto es algo natural, ya que es propio de este módulo.



Ahora vamos un paso más allá y miraremos como traspasar esta lógica al lado del cliente. Esto nos puede suponer beneficios en cuanto a **ahorro de recursos en el servidor**. También nos abrirá la mente a un paradigma de programación quizás nuevo para nosotros.

Tanto si ponemos la lógica en el cliente o en el servidor, lo más importante es entender qué hacemos y por qué lo hacemos, y escoger la mejor estrategia en cada situación.

OBJETIVOS / CAPACIDADES

En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:

- ✔ **Utilizar herramientas y lenguajes específicos, cumpliendo las especificaciones, para desarrollar e integrar componentes software en el entorno del servidor web.**
- ✔ **Programar y realizar actividades para gestionar el mantenimiento de los recursos informáticos.**

PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

Enunciado:

Construyendo una web moderna.

En esta actividad de evaluación queremos hacer un script que a partir de las coordenadas de una ciudad mostremos imágenes de la misma.

Para ello, utilizaremos la API de MediaWiki. Concretamente el recurso para mostrar fotos a partir de coordenadas lo encontramos en este código. Las coordenadas latitud 40.4160254 y longitud 3.7105986 corresponden cerca del centro de Madrid.

Crea una aplicación con un formulario que pida la latitud y la longitud de unas coordenadas. A partir de esa información haz la petición al recurso de MediaWiki para obtener las fotos y muéstralas de la forma que te parezca más atractiva.

Puedes usar:

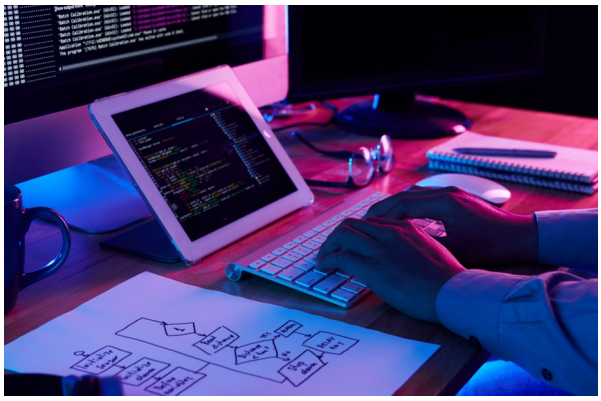
- Sólo frontend, con peticiones AJAX.

- Sólo backend, con el método de PHP file_get_contents([URL]).
- Una combinación de frontend y backend.

1. PROCESAMIENTO EN EL SERVIDOR Y EN EL CLIENTE

A pesar que el **objetivo principal de este módulo** es todo aquello que se encuentra en el **lado del servidor**, es obvio que es muy difícil, y quizás contraproducente, dar la espalda a todo aquello relacionado al cliente.

Es cierto que esta diferenciación entre cliente y servidor nos ayuda a separar responsabilidades y tareas de producción, y es lo que se conoce como el **frontend** y el **backend**.



- El **frontend**, o lado del cliente, es la parte de una aplicación que interactúa con los usuarios. Básicamente es todo lo que vemos en la pantalla cuando accedemos a un sitio web o aplicación: tipos de letra, colores, adaptación para distintas pantallas, los efectos del ratón, teclado, movimientos, desplazamientos, efectos visuales… y otros elementos que permiten navegar dentro de una página web. Este conjunto crea la **experiencia del usuario**.

  Como hemos dicho, el desarrollador frontend se encarga de la experiencia del usuario, es decir, en el momento en el que este entra a una página web, debe ser capaz de **navegar** por ella, por lo que el usuario verá una interface sencilla de usar, atractiva y funcional.

*Mientras que el **frontend** es la capa de programación ejecutada en el navegador del usuario, el **backend** procesa la información que alimentará el frontend de datos.*

Es la capa de **acceso a los datos**, ya sea de un software o de un dispositivo en general, es la lógica tecnológica que hace que una página web funcione, lo que queda oculto a ojos del visitante.

- El **backend** de una aplicación, determina qué tan bien se ejecutará la aplicación y qué experiencia, positiva o negativa, obtendrá el usuario de su uso.

  Trabajar en este apartado supone algo totalmente diferente al frontend, ya que exige el dominio de otros términos de programación, lenguajes que requieren una lógica, ya que esta área es también la encargada de **optimizar recursos**, de la **seguridad** de un sitio y otros factores.

Te recomendamos visualizar el vídeo "¿Qué es backend y frontend?" que encontrarás en recursos para ampliar.

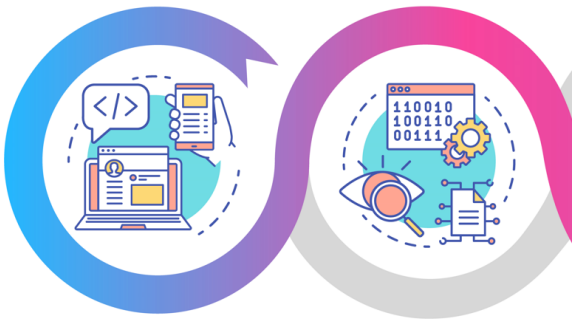
2. LIBRERÍAS Y TECNOLOGÍAS RELACIONADAS

El **frontend y el backend** están separados por varias razones. En primer lugar, son interfaces completamente diferentes que utilizan sus propios frameworks, lenguajes, librerías y mecanismos de desarrollo.

DESTACADO

*Cada profesional del desarrollo de software se especializa en un apartado. Pero dependiendo del tamaño del proyecto o de la empresa muchos desarrolladores adoptan el rol de **full-stack developers**, es decir, tanto se encargan del frontend como del backend.*

Los desarrolladores del frontend y del backend tienen **cajas de herramientas (toolkits)** únicos, y estas herramientas incluyen diferentes **elementos**:



- **Frameworks**: proporcionan una base, como una plantilla, para que los desarrolladores puedan crear rápidamente sitios y aplicaciones web.

FRAMEWORKS BACKEND	FRAMEWORKS Y BIBLIOTECAS FRONTEND
Django, Spring Boot, Laravel, Symfony, Rails, ExpressJS, ASP .Net, CakePHP, Phoenix, etc.	Angular, React, Vue, jQuery, etc.

- **Lenguajes**: los lenguajes de programación permiten a los desarrolladores escribir secuencias de comandos, instrucciones y archivos de sitio que finalmente son ejecutados por un ordenador.

LENGUAJES BACKEND	LENGUAJES FRONTEND
PHP, Java, Python, C++, Ruby, etc.	HTML, CSS, JavaScript, Dart.

- **Bibliotecas o librerías**: en el lado del frontend, los desarrolladores pueden elegir entre frameworks y bibliotecas. A veces la línea es borrosa entre lo que constituye una biblioteca y un framework, pero, en general, un framework de frontend es una plantilla de archivos, lenguajes y herramientas para construir y escalar rápidamente la parte frontal de una aplicación web o sitio web.
- **Bases de datos**: las bases de datos suelen ser gestionadas únicamente por los desarrolladores de backend. El desarrollo full-stack también maneja las bases de datos, pero un trabajador del frontend sólo interactúa con las bases de datos para asegurarse de que la interfaz de usuario produce los resultados correctos.

3. GENERACIÓN DINÁMICA DE PÁGINAS INTERACTIVAS

Crear **páginas interactivas de forma dinámica** ya lo hemos hecho, mínimo, desde la unidad didáctica 5. Hasta ahora el servidor recibía una petición y generaba contenido dinámico. Que la propia web fuese más o menos interactiva ya depende de la programación del lado del cliente con, por ejemplo, JavaScript. El escenario que veremos ahora es probar a hacer una petición a un recurso de un tercero **desde el lado del cliente** y **desde el lado del servidor**. Ambas soluciones conseguirán el mismo resultado. La diferencia radicará en cómo consultaremos un volumen muy grande de datos.

Cuando hagamos la petición desde el servidor bloquearemos la ejecución hasta que no consigamos todos los datos. Desde el cliente, y gracias a técnicas como AJAX, la aplicación no quedará bloqueada, brindando así más interactividad al usuario si fuese necesario.

Si lo hacemos desde el **lado del servidor**:

Prueba el siguiente script de PHP

```
<?php
$page = 1;
do {
                                                $data             =
json_decode(file_get_contents('https://api.themoviedb.org/3/search
/movie?
api_key=cbd87797a471967b37c6e72ea06b2593&query=mat&page=1'),
true);
    $page++;
    var_dump($data);
} while($page <= $data["total_pages"]);


?>
```

Básicamente lo que hace es consultar la API de TMDB (The Movie Data Base) todas las películas que contengan la palabra "mat". Concretamente nos devolverá unos 4450 resultados paginados de 20 en 20. Esto es mucha información y un mínimo de 223 peticiones a la API.

Mientras se vayan obteniendo poco a poco los resultados el usuario no podrá interactuar con nada más que pueda haber en la página.

En cambio, si hacemos la petición **desde el lado del cliente** utilizando la librería jQuery y la técnica de peticiones asíncronas AJAX, los resultados se cargarán igual, pero como el resto de la página ya estará cargada, el usuario podrá seguir trabajando mientras se van mostrando los resultados de la consulta.

Copia el siguiente script de JavaScript y ejecútalo

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <meta   name="viewport"   content="width=device-width,   initial-
scale=1.0">
    <script src="https://code.jquery.com/jquery-3.2.1.js"></script>
    <title>Document</title>
</head>
<body>
    <script>
        $(document).ready(function() {
                        loadData('https://api.themoviedb.org/3/search/movie?
api_key=cbd87797a471967b37c6e72ea06b2593&query=mat&page=1');
        });
```

```
        function loadData(url) {
            $.ajax({
                type: "GET",
                url: url,
                dataType: "JSON",
                success: function(data) {

                    if (data.page < data.total_pages) {
                        let nextPage = parseInt(data.page);
                          let newURL = url.replace(/page=\d*/g, "page=" +
(++nextPage));

                        document.write(JSON.stringify(data));
                        loadData(newURL);
                    }
                }
            });
        }
    </script>
</body>
</html>
```

Compara ambos resultados. Verás que es igual. Pero fíjate en el spinner de carga en tu navegador. El script en JavaScript (lado del cliente) casi ni aparecerá, en cambio el script en PHP (lado del servidor) estará dando vueltas hasta que consiga mostrar todos los resultados.

Vídeo: backend vs frontend (I)

Visualiza el siguiente vídeo en el que pondrás en práctica alguno de los recursos desde backend y frontend.

Cuestionario

Lee el enunciado e indica la opción correcta:

El frontend:

a. Se encarga del acceso a datos y de la lógica de negocio.

b. Se encarga de la interfaz de usuario y de la lógica de la parte del cliente.

c. Se encarga tanto de las competencias del lado del cliente como del servidor.

Lee el enunciado e indica la opción correcta:

El backend:

a. Se encarga tanto de las competencias del lado del cliente como del servidor.

b. Se encarga del acceso a datos y de la lógica de negocio.

c. Se encarga de la interfaz de usuario y de la lógica de la parte del cliente.

Lee el enunciado e indica la opción correcta:

Decimos que un desarrollador es full-stack developer cuando:

a. Sabe JavaScript y PHP.

b. Conoce, a fondo, tanto tecnologías del frontend como del backend.

c. Cuando ha llegado al nivel 100 de conocimientos.

4. CONTROLES CON VERIFICACIÓN DE INFORMACIÓN EN EL CLIENTE

Prácticamente no hay nada que se pueda hacer desde el lado del **servidor** que no se pueda hacer desde el lado del **cliente**, y viceversa. Entonces, ¿cómo elegimos qué debe ir a un sitio y qué debe ir a otro? Es una decisión que no debe tomarse a la ligera y que implica muchos **factores**.

Ejemplo:

*Imaginemos que tenemos un **formulario** con un campo para que el usuario escriba su DNI. Como sabemos, el DNI tiene 8 dígitos y una letra. Si el usuario escribe bien su DNI podemos operar con él, pero en caso contrario deberemos **avisarle que hay un error** y que debe volver a introducirlo.*

En relación con este ejemplo: ¿de quién será la **responsabilidad**?, ¿del cliente o del servidor? Pues lo cierto es que ambos pueden hacer esta comprobación, pero con leves **matices** entre ellos:

- **Cliente**.

  Si la hace el **cliente**, cargamos de trabajo el ordenador del usuario, pero nos ahorramos peticiones improductivas al servidor.

- **Servidor**.

  Si la hace el **servidor**, aliviamos la carga de trabajo del cliente, pero corremos el riesgo de saturar a peticiones el servidor para una comprobación tan simple.

*En el siguiente subapartado, veremos un escenario de ejemplo para trabajar con los **scripts** pertenecientes a **index.html** y a **main.php**.*

4.1. Pruebas de scripts cliente y servidor

El escenario planteado en el apartado anterior es el que probaremos ahora. En este caso, tendremos un formulario en el archivo **index.html** (cliente) donde recogeremos el valor del DNI, el cual lo mandaremos a **main.php** (servidor). Además, primeramente cargaremos a **main.php** con la tarea de comprobar que el DNI esté correctamente escrito. Veamos a continuación los **scripts** pertenecientes tanto a **index.html** como a **main.php**:

➔ **Script perteneciente a index.html**.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <meta   name="viewport"   content="width=device-width,   initial-
scale=1.0">
    <title>DNI</title>
</head>
<body>
    <form action="main.php" method="POST">
        <input type="text" name="dni" id="dni">
        <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

➔ **Script perteneciente a main.php**.

```
<?php

if (!isset($_POST['dni']) || empty($_POST['dni'])) {
    echo "Debes escribir un DNI.</br>";
} else {
    $dni = $_POST['dni'];
    $letra = substr($dni, -1);
    $numeros = substr($dni, 0, -1);
     if (substr("TRWAGMYFPDXBNJZSQVHLCKE", $numeros%23, 1) == $letra &&
strlen($letra) == 1 && strlen ($numeros) == 8) {
        echo "DNI correcto.</br>";
    } else {
        echo "El formato del DNI no es correcto.</br>";
    }
}

echo "Pulsa <a href='index.html'>aquí</a> para volver al formulario";

?>
```

Si probamos la ejecución de ambos **scripts** vemos que realiza bien la tarea de comprobar que el DNI esté bien escrito. Pero por cada intento realizado hemos mandado una **petición al servidor**. Probemos en este caso que sea **index.html** quien, gracias a un pequeño script en JavaScript, haga la tarea de **comprobar el formato del DNI**:

Código: script perteneciente a index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
          <meta  name="viewport"  content="width=device-width,  initial-
scale=1.0">
    <title>DNI</title>
</head>
<body>
    <form action="main.php" method="POST" onsubmit="return checkDNI()">
        <input type="text" name="dni" id="dni">
        <input type="submit" value="Enviar">
    </form>

    <script>
        function checkDNI() {
            let dni = document.getElementById('dni').value;
            let numero, letra;
            var letras = 'TRWAGMYFPDXBNJZSQVHLCKET';
            var expresion_regular_dni = /^\d{8}[a-zA-Z]$/;

            if(expresion_regular_dni.test (dni) == true) {
                numero = dni.substr(0,dni.length-1);
                letra = dni.substr(dni.length-1,1);
                numero = numero % 23;
                letras=letras.substring(numero,numero+1);
                if (letras!=letra.toUpperCase()) {
                            alert('DNI  erroneo,  la  letra  del  NIF  no  se
corresponde');
                } else {
                    alert('DNI correcto');
                    return true;
                }
            } else {
                alert('DNI erroneo, formato no válido');
            }

            return false;
        }
    </script>
```

```
</body>
</html>
```

Ahora podemos, si queremos, no hacer la comprobación del DNI en el servidor, ya que se comprueba en el **cliente**. A pesar de ello, es recomendable siempre **validar aquellos datos** que nos llegan al servidor por motivos de seguridad.

Vídeo: backend vs frontend (II)

Visualiza el siguiente vídeo en el que continuarás poniendo en práctica alguno de los recursos desde backend y frontend.

Actividad de aprendizaje 1: validación de un formulario

Actividad en parejas.

El equipo de ciberseguridad de la empresa nos alerta que recibimos intentos de ataques de inyección. Estos pueden ser de SQL o de XSS. Aunque nosotros no seamos responsables del tratamiento de estos ataques, sí que se nos pide que validemos todos los formularios propios, es decir, cualquier campo de texto que se envía al servidor.

Uno de los miembros creará un formulario web con los campos nombre y apellidos. Deberá mandar estos datos a un script en el servidor, creado por el otro miembro del equipo, pero previamente deberá comprobar que los nombres no estén vacíos.

El otro miembro deberá escribir el código del script. Se supone que recibirá los campos nombre y apellidos, pero deberá verificar que se trate de texto y no, por ejemplo, de números.

5. OBTENCIÓN REMOTA DE INFORMACIÓN

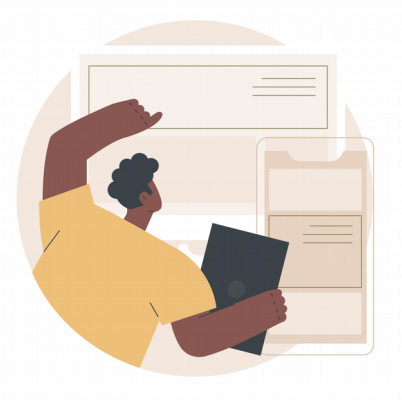
En el punto 3 hemos sido capaces de obtener información de un recurso remoto gracias a una técnica llamada **AJAX**.

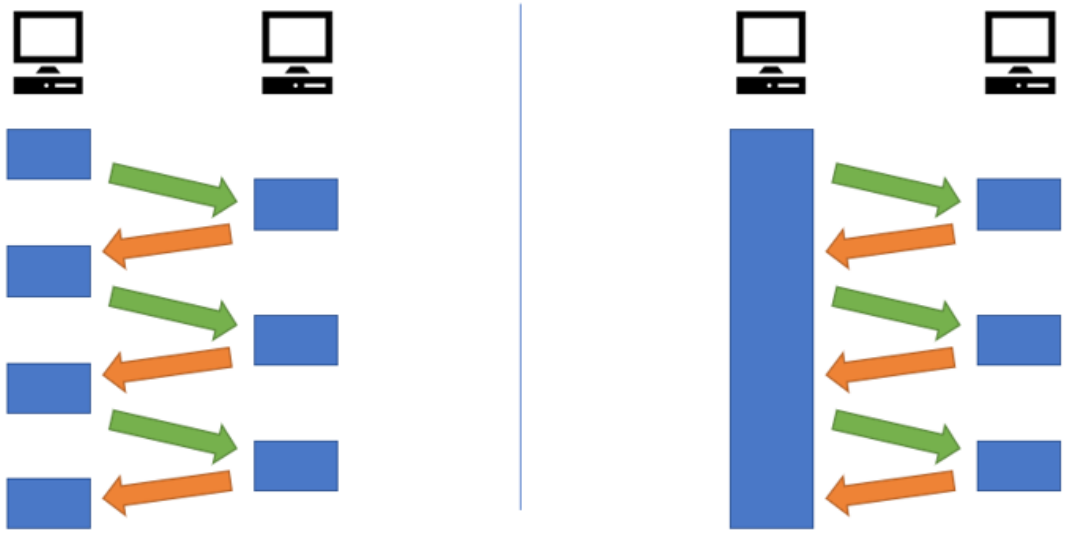
DESTACADO

AJAX *son las siglas de Asynchronous JavaScript and XML, es decir, ejecutar JavaScript y XML de forma asíncrona.*

➔ **¿Qué significa trabajar de forma asíncrona?**

Fíjate en la siguiente imagen. En la parte de la izquierda, un cliente hace una **petición a un servidor** (flecha verde) y, hasta que este no acaba de procesarla y responder (flecha naranja), el cliente se queda **esperando**.





Comparación de un modelo síncrono con un modelo asíncrono.

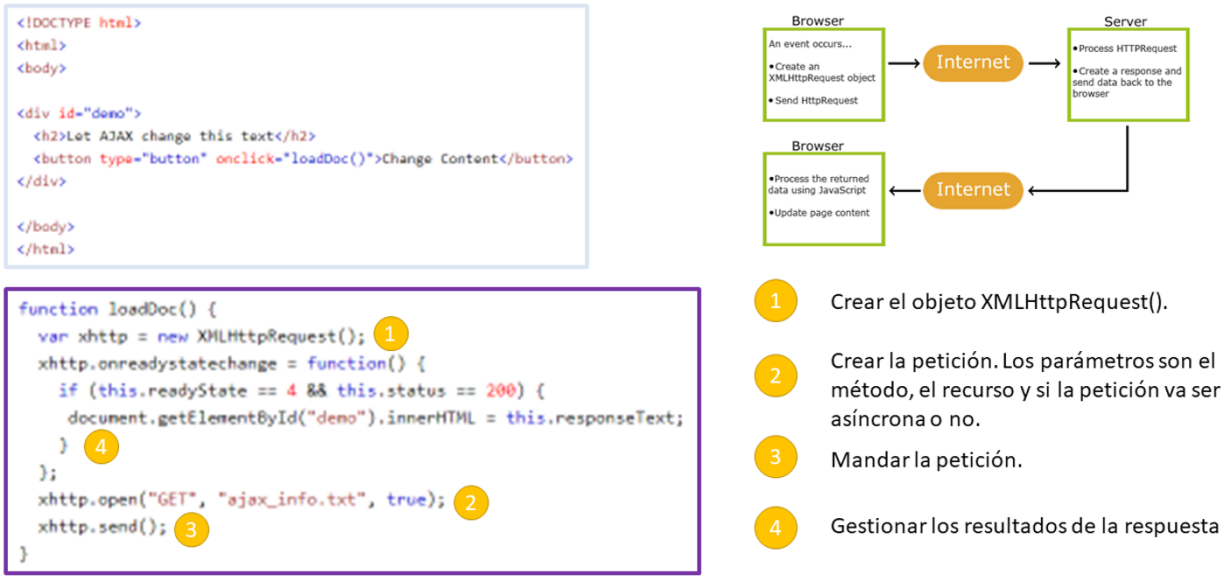
En cambio, en la imagen de la derecha, el cliente **pide unos datos al servidor**, pero su flujo de trabajo no se interrumpe. Cuando el servidor manda los datos, el cliente los incorpora sin más.

➔ **¿Qué permite AJAX?** AJAX permite que las páginas web se actualicen de forma **asíncrona** mediante el intercambio de datos con un servidor web en segundo plano. Esto significa que es posible actualizar partes de una página web sin recargar toda la página.

AJAX sencillamente usa una **combinación** de:

- ✔ Un objeto XMLHttpRequest integrado en el navegador (para solicitar datos de un servidor web).
- ✔ JavaScript y HTML DOM (para mostrar o usar los datos).

➔ El **workflow** de una petición con AJAX se resume en la siguiente imagen:



Workflow de una petición AJAX.

6. MODIFICACIÓN DE LA ESTRUCTURA DE LA PÁGINA WEB

En este punto, pondremos en práctica la **teoría** sobre AJAX vista en el punto anterior. Crearemos una página web con un ***iframe*** donde cargaremos un juego y, al lado, un **botón**. Al pulsar este botón haremos una **petición AJAX** para recuperar información de la película Matrix sobre la API de IMDB. Con la **respuesta del servidor**, modificaremos la parte derecha de la página.

➔ **¿Para que necesitamos el minijuego?** Sencillamente, para demostrar que la petición funciona en segundo plano y que la ejecución del juego no se interrumpirá, aunque carguemos datos y actualicemos parte de la página web.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
          <meta  name="viewport"  content="width=device-width,  initial-
scale=1.0">
    <title>AJAX</title>
</head>
<body>
            <iframe  src="https://js13kgames.com/games/onoff/index.html"
frameborder="0"  style="width:50%;  top:50%;  transform:translateY(50%);
float: left;"></iframe>
    <div style="width: 50%; float: left">
        <button id="btn">Get info</button>
        <div id="info"></div>
    </div>
    <script>
                document.getElementById('btn').addEventListener('click',
function() {
            var xhttp = new XMLHttpRequest();
            xhttp.onreadystatechange = function() {
                if (this.readyState == 4 && this.status == 200) {
                            document.getElementById("info").innerHTML  =
this.responseText;
                }
            };
              xhttp.open("GET", "https://api.themoviedb.org/3/movie/603?
api_key=cbd87797a471967b37c6e72ea06b2593", true);
            xhttp.send();
        }, false);
    </script>
</body>
</html>
```

➔ En la siguiente imagen vemos el **resultado**: hemos cargado los datos de la película Matrix en la parte derecha de la pantalla mientras no se ha interrumpido para nada la ejecución del juego.

En este caso, la **respuesta del servidor** ha sido inmediata porque los datos a cargar eran pocos, pero de todas formas se ha visto que el workflow del cliente, el juego, no se ha visto interrumpido.

Actividad de aprendizaje 2: peticiones asíncronas

El departamento de marketing de nuestra empresa quiere ilustrar la página web corporativa con imágenes de sitios emblemáticos de las ciudades desde donde los visitantes acceden a nuestra web.

Nosotros conocemos que la API de Wikimedia, a partir de unas coordenadas, nos devuelve imágenes de estas coordenadas, en el caso que existan.

Crea una página web estática con un campo de texto y un botón. El usuario debe escribir el nombre de una ciudad y cuando pulse el botón se tiene que consultar la API de Wikipedia para ver la información de dicha ciudad.

La URL contra la que hacer la petición es esta, donde Madrid es la ciudad de ejemplo.

Haz la petición AJAX tal y como lo hemos trabajado en la teoría.

Cuestionario

Lee el enunciado e indica la opción correcta:

Laravel es:

a. Una librería del frontend.

b. Un framework del backend.

c. Un framework de Java.

Lee el enunciado e indica la opción correcta:

Para escribir lógica al lado del cliente usamos:

a. Laravel.

b. JavaScript.

c. PHP.

Lee el enunciado e indica la opción correcta:

AJAX significa:

a. Asynchronous JavaScript and XML.

b. Synchronous JavaScript and XML.

c. Asynchronous JavaScript Assembled Xerox.

RESUMEN

En esta unidad didáctica hemos aprendido la diferencia entre **frontend** y **backend**, ya no solo de forma teórica sino practicando el mismo **script** desde el lado del cliente como del lado del servidor.

También hemos visto la **validación de formularios** y hemos podido decidir cuándo nos interesará más implementarla en el lado del cliente o en el lado del servidor, o quizás a ambos lados.

Esta decisión quizás no dependerá de nosotros en un **entorno real**, que que dependerá de los compañeros que levanten las **arquitecturas** o dependerá de los compañeros de administración y de dónde prefieren invertir el dinero: ancho de banda, potencia del servidor, etc.

Nosotros, como buenos **desarrolladores**, debemos de tener las herramientas necesarias para ser versátiles e implementar soluciones con criterio pero también consensuadas con nuestro equipo.

Finalmente hemos practicado peticiones asíncronas con AJAX. **AJAX** es una técnica 100% del lado del cliente que hace peticiones contra recursos que están en el servidor.

RECURSOS PARA AMPLIAR

PÁGINAS WEB

- ¿Qué es backend y frontend?: https://www.youtube.com/watch?v=50RbVujPPGs [Consulta noviembre 2022].
- API de MediaWiki: https://www.mediawiki.org/wiki/API:Main_page [Consulta noviembre 2022].
- Recurso para mostrar fotos a partir de coordenadas: https://commons.wikimedia.org/w/api.php?action=query&format=json&prop=imageinfo&generator=geosearch&iiprop=url&ggscoord=40.4160254|-3.7105986&ggsradius=500&ggsnamespace=6&ggsprimary=all [Consulta noviembre 2022].
- Tutorial de AJAX en PHP de w3schools: https://www.w3schools.com/php/php_ajax_intro.asp [Consulta noviembre 2022].
- URL contra la que hacer la petición de la actividad 2: https://es.wikipedia.org/w/api.php?action=query&list=search&srprop=snippet&format=json&origin=*&utf8=&srsearch=madrid [Consulta noviembre 2022].

BIBLIOGRAFÍA

PÁGINAS WEB

- Minijuego JavaScript: *https://js13kgames.com/games/onoff/index.html* [Consulta noviembre 2022].
- Referencia oficial de PHP: *https://www.php.net/* [Consulta noviembre 2022].

GLOSARIO

- **AJAX**: siglas en inglés de Asynchronous JavaScript and XML.
- **API**: siglas en inglés de Application Programming Interface.
- **Backend**: en desarrollo del software es la capa que se encarga de la lógica de negocio y del acceso a datos.
- **Frontend**: en desarrollo del software es la capa que se encarga de la presentación de datos y scripts del lado del cliente.
- **XML**: siglas en inglés de eXtensible Markup Language.