Programación con JavaScript II

AJAX

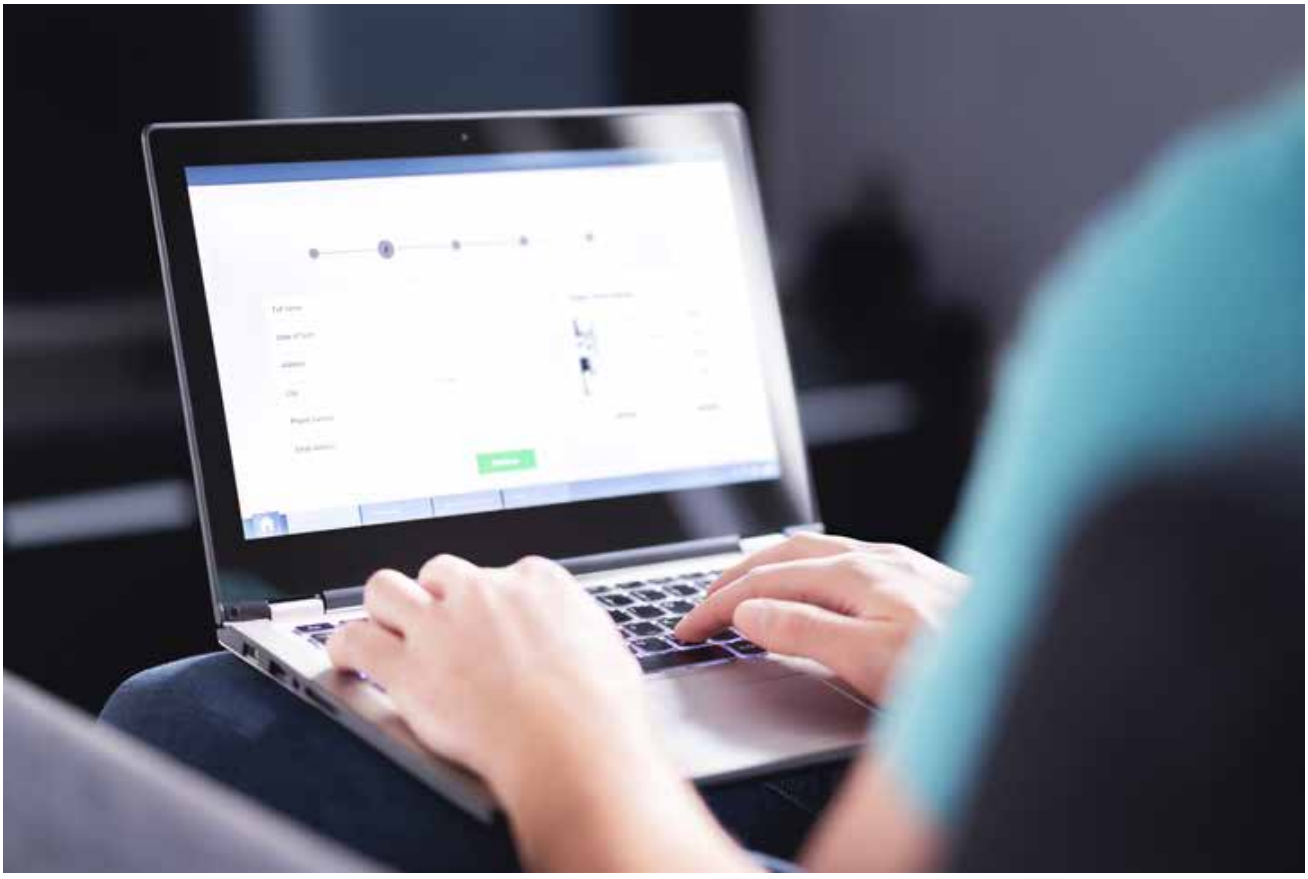
Introducción

Seguramente cuando navegas por Internet te habrás dado cuenta de que hay páginas que, al interactuar con sus diferentes módulos o herramientas, responden de manera inmediata, sin largos tiempos de espera para recargar el contenido.

Un ejemplo de esto son las páginas en las que consultas el correo electrónico como Yahoo!, Outlook y Gmail.

La mayoría de estas páginas son construidas con una plataforma cliente – servidor, en la que se garantiza que la experiencia del usuario con la GUI (Graphic User Interface), o interfaz gráfica del usuario, sea muy ágil, dando la sensación de estar interactuando con una aplicación de escritorio.

En este tema aprenderás a cambiar entre las secciones de un sitio web, lo que asegura una respuesta casi inmediata del servidor, utilizando solicitudes asíncronas de JavaScript, también conocidas como AJAX.



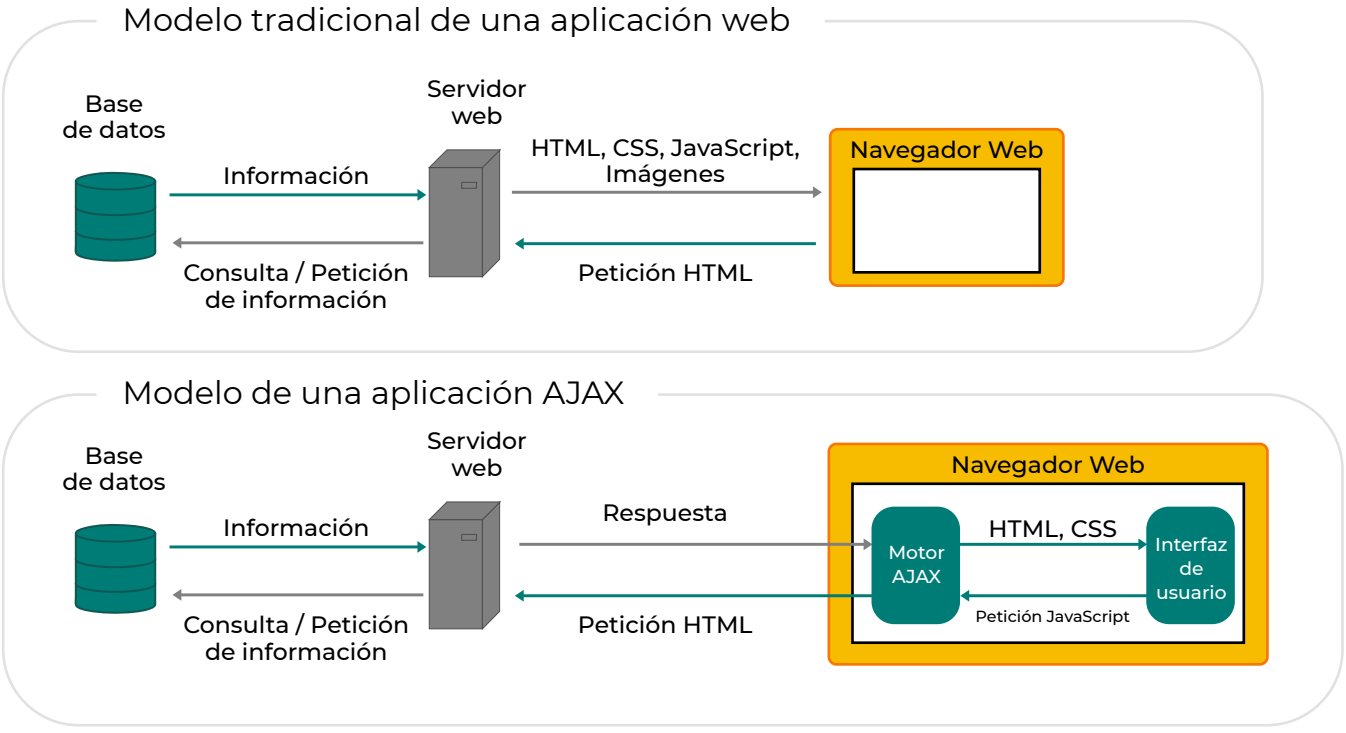
Explicación

Introducción a AJAX

Según MDN (2022), AJAX, o *Asynchronous* JavaScript and XML, se puede traducir como JavaScript Asíncrono + XML. Por su parte, Bustos (2022) lo define como "un conjunto de técnicas de desarrollo web que permiten que las aplicaciones web funcionen de forma asíncrona, procesando cualquier solicitud al servidor en segundo plano".

El tipo de comunicación asíncrona que AJAX realiza con diferentes servicios (HTML, CSS, DOM, JavaScript, XML y XSLT) puede decirse que es tras bambalinas. Su ventaja es que posibilita la generación de aplicaciones web que se comunican con diferentes tecnologías sin la necesidad de hacer una recarga completa de la página. Lo anterior agiliza mucho el tiempo de respuesta, ya que no tiene que hacer peticiones directamente al servidor y el usuario puede seguir interactuando con la página web mientras se procesan sus peticiones, tal y como lo hacen las aplicaciones de escritorio. Solo se actualizan los elementos que cambiaron, lo que finalmente da una sensación de que la página sigue respondiendo sin congelarse, porque no interfieren con los elementos visuales existentes.

Considera, por ejemplo, interactuar con una base de datos que contenga estadísticas que permitan realizar cálculos sobre los vuelos de diferentes aerolíneas y con esto ofrecer una página que optimice el tiempo de traslado entre un punto origen hasta un punto destino (obtener información), reservar vuelos (modificar o añadir) o cancelar reservaciones (borrar información). Esta página podría construirse con tecnología AJAX y ofrecer una muy buena experiencia de usuario comparada con otra que usa el modelo tradicional. La siguiente imagen ilustra la comparativa entre el modelo tradicional y el modelo que utiliza la tecnología AJAX:



Explicación

En la imagen anterior puedes observar la diferencia entre el modelo de aplicaciones web tradicional y el modelo AJAX. Mientras que en el modelo tradicional las peticiones web obligan a la página a actualizarse por completo, el modelo AJAX interactúa con el motor AJAX y la interfase del usuario antes de realizar una solicitud al servidor y esta, a su vez, a la base de datos.

Algunas de estas herramientas de interacción de servicios, como las peticiones al servidor, se modernizaron en la versión de JavaScript ES6, haciéndose más eficientes y simples, sin la necesidad de usar JQuery. Actualmente muchas aplicaciones están construidas sobre esta plataforma, entre ellas, Google Maps, Gmail y YouTube.

Los elementos que debes considerar para usar esta arquitectura son los siguientes:

- XMLHttpRequest object (para intercambiar datos de manera asíncrona con el servidor).
- JavaScript/DOM (para mostrar o interactuar con la información).
- CSS (para mantener una estética al mostrar los datos).
- XML (para convertir datos en diferentes formatos).

Objeto XMLHttpRequest

Al estar trabajando con AJAX, encontrarás que uno de los objetos principales para comunicarse de manera asíncrona con el servidor es el objeto **XMLHttpRequest.** Este objeto permite usar un archivo XML y transformarlo a diferentes formatos.

La sintaxis que debes utilizar es la siguiente:

variable = new XMLHttpRequest();

```
var xhttp = new XMLHttpRequest();
```

Un ejemplo bastante sencillo es en el que se utiliza un objeto XMLHttpRequest para actualizar una página al hacer clic con un botón. Observa que la página solo actualizará el contenido. Para usar el siguiente ejemplo, considera que los archivos deben estar cargados en un servidor web.

Al revisar el siguiente código, encontrarás secciones que están relacionadas con la creación del objeto **XMLHttpRequest** que pretende sustituir la estructura de “demostración” encapsulada en los tags html .

Explicación

```
<!DOCTYPE html>
<html>
<body>

<div id="demostracion">
<h1>Objeto XMLHttpRequest</h1>
<button type="button" onclick="CargaDocumento()">Cambia el contenido</button>
</div>

<script>
function CargaDocumento() {
 // Genera una instancia del objeto XMLHttpRequest
  var xhttp = new XMLHttpRequest();

  // Utiliza la propiedad onreadystatechange para asegurar el estado de la peticion
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demostracion").innerHTML =
      this.responseText;
    }
  };
  //Especifica las características de la solicitud
  // Método GET, url=archivo, async=true
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>

</body>
```

Contenido del archivo ajax_info.txt:

```
<h1>AJAX</h1>
<p>AJAX es una técnica para acceder a servidores Web desde una página
Web.</p>
<p>Estas líneas de Código se encuentran en el archive Ajax_info.txt</p>
<p>La actualización de la información se realiza de manera asíncrona.</p>
```

Explicación

```
<!DOCTYPE html>
<html>
<body>

<div id="demostracion">
<h1>Objeto XMLHttpRequest</h1>
<button type="button" onclick="CargaDocumento()">Cambia el
contenido</button>
</div>

<script>
function CargaDocumento() {
 // Genera una instancia del objeto XMLHttpRequest
  var xhttp = new XMLHttpRequest();

  // Utiliza la propiedad onreadystatechange para asegurar el
estado de la peticion
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demostracion").innerHTML =
      this.responseText;
    }
  };
  //Especifica las características de la solicitud:
  // Método GET, url=archivo, async=true
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>

</body>
</html>
```

Objeto XMLHttpRequest

Cambia el contenido

Al hacer clic sobre el botón “Cambia el contenido”, AJAX cambiará la sección para desplegar la información del servidor, llamando la función “**CargaDocumento**”, que, a su vez, obtendrá el contenido del archivo ajax_info.txt.

AJAX

AJAX es una técnica para acceder a servidores Web desde una página Web.

Estas líneas de Código se encuentran en el archive Ajax_info.txt

La actualización de la información se realiza de manera asíncrona.

Explicación

La imagen anterior muestra el resultado al hacer clic sobre el botón **<Cambia el contenido>.**

Algunos de los métodos más útiles del objeto **XMLHttpRequest** son los siguientes (W3Schools, s.f.):

Método	Descripción
open(method,url,async,user,psw)	Especifica las características de la solicitud: **method:** puede ser GET o POST. **url:** especifica la ubicación del archivo. **async:** true/false. **user:** credencial del usuario (opcional). **psw:** clave del usuario (opcional).
new XMLHttpRequest	Crea una instancia del objeto XMLHttpRequest.
send()	Envía solicitudes GET del servidor.
send(string)	Envía solicitudes POST del servidor.
abort()	Cancela la petición realizada.
getAllResponseHeaders()	Regresa el valor del encabezado.

Las propiedades del objeto **XMLHttpRequest** son las siguientes:

Propiedad	Descripción
onreadystatechange	Define una función cuando hay un cambio en la propiedad readyState.
readyState	Conserva el estatus del objeto XMLHttpRequest: 0: Petición no iniciada. 1: Conexión con el servidor establecida. 2: Petición recibida. 3: Petición procesada. 4: Petición lista.
responseText	Regresa los datos como un string.
responseXML	Regresa los datos en formato XML.
status	Regresa el número de estado: 200: OK 403: “Forbidden” 404: “Not found” 503: “Bad Gateway”
statustext	Regresa el estatus de la respuesta a una petición en forma de texto: 0: UNSENT 1: OPENED 3: LOADING OK 4: DONE OK

Explicación

Consideraciones

Según W3Schools (s.f.), se deben tomar en cuenta las siguientes consideraciones al utilizar AJAX con XMLHttpRequest:

- Al enviar una petición al servidor, existen dos métodos de respuesta: GET y POST.
- El método GET es más simple y rápido que el método POST en la mayoría de los casos.
- No se recomienda usar la propiedad “async”=*fals*, pues el código JavaScript no dejará de ejecutarse hasta que el servidor esté disponible, lo que provocaría que el servidor tenga que administrar las peticiones no resueltas, causando lentitud en la respuesta.

Uso de archivos XML con AJAX

En el siguiente ejemplo podrás observar cómo hacer uso de llamadas a un servidor para obtener datos tomados de una estructura XML.

Observa que solicita el archivo catalog.xml, cuya estructura se muestra más abajo. Este archivo se encuentra alojado en el servidor. La función llamada “myFunction” obtiene los datos cuyas etiquetas correspondan a las direcciones “Address” de los clientes. Finalmente, se crea la tabla HTML para acomodar los datos obtenidos del archivo XML.

Código EjemploXML.html

Explicación

```
<!DOCTYPE html>
<html>
<style>
table,th,td {
  border : 1px solid black;
  border-collapse: collapse;
}
th,td {
  padding: 5px;
}
</style>
<body>

<h2>Objeto XMLHttpRequest</h2>
<h3>Peticiones de datos XML</h3>
<button type="button" onclick="CargaDocumento()">Obten direcciones</button>
<br><br>
<table id="demo"></table>

<script>
function CargaDocumento() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    myFunction(this);
  }
  xhttp.open("GET", "catalog.xml");
  xhttp.send();
}
function myFunction(xml) {
  const xmlDoc = xml.responseXML;
  const x = xmlDoc.getElementsByTagName("Address");
  let table="<tr><th>Nombre</th><th>Calle</th><th>Ciudad</th><th>Estado</th></tr>";
  for (let i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("Name")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("Street")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("City")[0].childNodes[0].nodeValue +
    "</td><td>"+
    x[i].getElementsByTagName("State")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```

Explicación

Al ejecutar el código EjemploXML.html, se mostrará un texto **“Objeto XMLHttpRequest”** y **“Peticiones de datos XML”** con un botón **“Obtén direcciones”**, que ejecutará la función **“CargaDocumento”** al presionarlo.

```
<script>
function CargaDocumento() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    myFunction(this);
  }
  xhttp.open("GET", "catalog.xml");
  xhttp.send();
}
function myFunction(xml) {
  const xmlDoc = xml.responseXML;
  const x = xmlDoc.getElementsByTagName("Address");
  let table="<tr><th>Nombre</th><th>Calle</th><th>Ciudad</th><th>Estado</th></tr>";
  for (let i = 0; i <x.length; i++) {
    table += "<tr><td>" +
    x[i].getElementsByTagName("Name")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("Street")[0].childNodes[0].nodeValue +
    "</td><td>" +
    x[i].getElementsByTagName("City")[0].childNodes[0].nodeValue +
    "</td><td>"+
    x[i].getElementsByTagName("State")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>
```

Objeto XMLHttpRequest

Peticiones de datos XML

Obten direcciones

En la imagen anterior se muestra el código a la izquierda y la ejecución de este a la derecha. Al presionar el botón, la petición se ejecuta en segundo plano, creando la tabla HTML con el contenido del archivo.

Objeto XMLHttpRequest

Peticiones de datos XML

Obten direcciones

Nombre	Calle	Ciudad	Estado
Ellen Adams	123 Maple Street	Mill Valley	CA
Tai Yee	8 Oak Avenue	Old Town	PA
John Doe	123 Rodeo Street	Dallas	TX
Tira Taylor	9 Av	Sprinfield	Az

Explicación

Analicemos la forma de ponerlo en práctica. Esta es una implementación de una función basada en promesa que suma dos números: a y b:

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
    <Address Type="Shipping">
    <Name>John Doe</Name>
    <Street>123 Rodeo Street</Street>
    <City>Dallas</City>
    <State>TX</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tira Taylor</Name>
    <Street>9 Av</Street>
    <City>Sprinfield</City>
    <State>Az</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
</PurchaseOrder>
```

Explicación

API Fetch

Según MDN (2022), los navegadores más actualizados utilizan **API** Fetch en lugar del objeto **XMLHttpRequest**, ya que su interfase les permite realizar peticiones HTTP a servidores web de una manera más eficaz con un consumo de recursos mínimos y más potente. Este comando ya no requiere el objeto **XMLHttpRequest** para generar peticiones HTTP al servidor sin tener que recargar la página completa.

Fetch es parte de ECMAScript6 (ES6). Puede utilizarse en la mayoría de los navegadores modernos como Chrome, Edge, Firefox, Safari y Opera, mientras que ya no es soportado por Internet Explorer versión 11 o versiones anteriores. No se requieren librerías al usar el API FETCH, puesto que utilizan código del mismo navegador.

Su sintaxis es la siguiente:

```
fetch(file)
  .then(function() {
  // gestionar la respuesta
})
.catch(function() {
  // gestionar el error
}
```

El parámetro **file** consiste en el nombre del recurso a recuperar y su valor de retorno es una “promesa”, es decir, son objetos con métodos then() y catch().

En el siguiente código puedes observar la estructura fetch y el uso del método then(). Nota que es una estructura más esbelta para usar en la sustitución.

```
<!DOCTYPE html>
<html>
<body>

<h1>API Fetch de JavaScript</h1>
<h2>Metodo then()</h2>
<div id="demo">
<p>Texto a sustituir por el archivo fetch_info.</p>
<button type="button" onclick="CargaDocumento()">Cambia el contenido</button>
</div>

<script>
function CargaDocumento() {
    let file = "fetch_info.txt"
    fetch (file)
    .then(x => x.text())
    .then(y => document.getElementById("demo").innerHTML = y);
}
</script>

</body>
</html>
```

Explicación

Resultado al ejecutar el código:

```
<h1>API Fetch de JavaScript</h1>
<h2>Metodo then()</h2>
<div id="demo">
<p>Texto a sustituir por el archivo fetch_info.</p>
<button type="button" onclick="CargaDocumento()">Cambia el contenido</button>
</div>

<script>
function CargaDocumento() {
    let file = "fetch_info.txt"
    fetch (file)
    .then(x => x.text())
    .then(y => document.getElementById("demo").innerHTML = y);
}
</script>
```

La imagen anterior muestra el código ejecutado desde un servidor web. Al hacer clic sobre el botón, se desplegará el contenido del archivo **“fetch_info.txt”**, sustituyendo el texto y el botón inicial sin recargar la página.

La imagen anterior muestra el resultado después de hacer clic en el botón “Cambia el contenido”.

Contenido del archivo “**fetch_info.txt**”:

```
<p>La interfase API FETCH permite que el navegador realice peticiones HTTP a servidores Web.</p>
<p>Fetch realiza exactamente lo mismo de un objeto XMLHttpRequest de una manera más sencilla.</p>
```

Cierre

AJAX es una técnica que rompe con el paradigma de la interacción cliente-servidor tradicional, en la que se esperaba que cada petición de datos a un servidor provocara la recarga de la página, lo que consecuentemente generaba un retardo en el tiempo de respuesta, ya que el código HTML tenía que limpiarse y reconstruirse en cada petición.

Con AJAX, es posible interactuar con los datos del servidor y obtener resultados de manera inmediata.

En este tema has aprendido a usar el objeto **XMLHttpRequest** y el API Fetch, ambos tienen el mismo objetivo, pero pueden ser utilizados en diferentes escenarios.

Referencias bibliográficas

- Bustos, G. (2022). *¿Qué es AJAX y cómo funciona?* Recuperado de https://www.hostinger.mx/tutoriales/que-es-ajax
- MDN. (2022). *AJAX.* Recuperado de https://developer.mozilla.org/es/docs/Web/Guide/AJAX
- W3Schools. (s.f.). *AJAX - Send a Request To a Server.* Recuperado de https://www.w3schools.com/xml/ajax_xmlhttprequest_send.asp
- MDN. (2022). *Fetch API.* Recuperado de https://developer.mozilla.org/es/docs/Web/API/Fetch_API

Para saber más

Lecturas
Para conocer más acerca de sobre el **AJAX y API Fetch**, te sugerimos leer lo siguiente:

- MDN. (2022). *Fetch API*. Recuperado de https://developer.mozilla.org/es/docs/Web/API/Fetch_API

- Firthous. (2020). *The Ultimate JavaScript Fetch API Cheatsheet*. Recuperado de https://javascript.plainenglish.io/the-ultimate-javascript-fetch-api-cheatsheet-e60b98c8cdbe

Videos
Para conocer más acerca de **AJAX y API Fetch,** te sugerimos revisar lo siguiente:

- Vida MRR – Programación Web. (2018, 21 de noviembre). *Diferencia entre Fetch vs AJAX* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=V_acKBZ1ZPU

  ProCode TV. (2018, 26 de noviembre). *Fetch API - #JavaScript - Consumir API Publica con JavaScript* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=cjrt3NilcNw

- Leonidas Esteban. (2019, 30 de agosto). *Fetch API en JavaScript (GET, POST, PUT, DELETE) | programación asíncrona* [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=aKPcs-EIzZI

Checkpoints

Asegúrate de:

- Reconocer las ventajas de usar AJAX para actualizar una página sin necesidad de recargarla a través del objeto **XMLHttpRequest.**
- Comprender la estructura de la **API** Fetch en peticiones a un servidor web en navegadores modernos.
- Identificar la diferencia entre usar un objeto **XMLHttpRequest** y la **API Fetch**.

Requerimientos técnicos

- Usar Visual Studio Code (https://code.visualstudio.com/).
- Acceso a una plataforma con servidor web (Live Server).

Prework

- Deberás reconocer las secciones HTML y JavaScript en una estructura de código.
- Deberás reconocer las estructuras del código de este tema y probarlas.
- Es necesario que tengas acceso a un servidor web para trabajar con el código AJAX, por ejemplo, Live Server.
- Para usar Live Server como una extensión de Visual Studio Code, ingresa a la siguiente liga: https://marketplace.visualstudio.com/