

## Manejo de métodos asíncronos en JavaScript

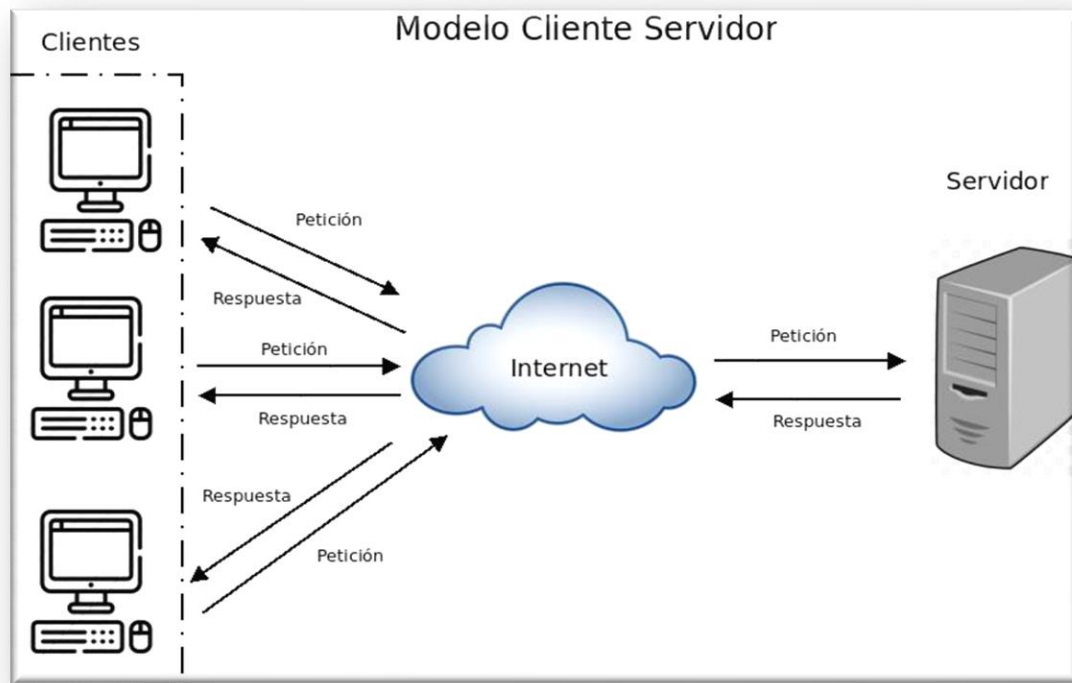
Hasta el momento hemos visto como enviar formularios y hacer intercambio de datos entre páginas mediante los métodos **GET y POST**, estos métodos involucran que se recargue la página. En esta guía los aprendices conocerán y aplicarán tecnologías que permitirán realizar **llamados asíncronos** al servidor para el intercambio de datos lo cual permite la creación de aplicaciones interactivas, dinámicas y atractivas

### AJAX

AJAX son las siglas de **Asynchronous JavaScript And XML**. No es un lenguaje de programación sino un conjunto de tecnologías que nos permiten hacer páginas de internet más interactivas.

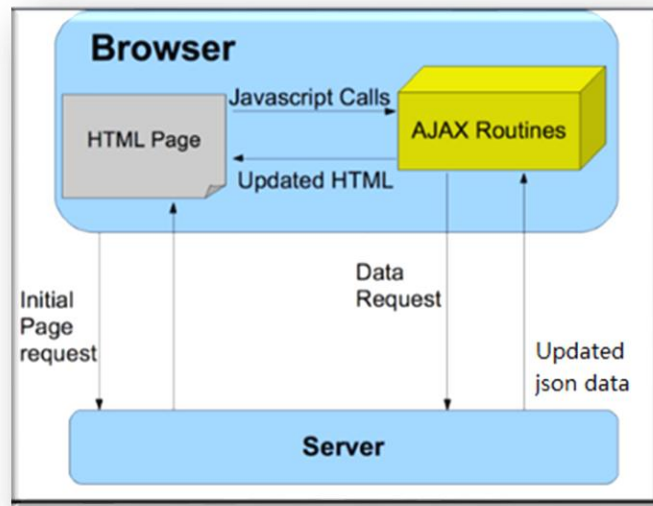
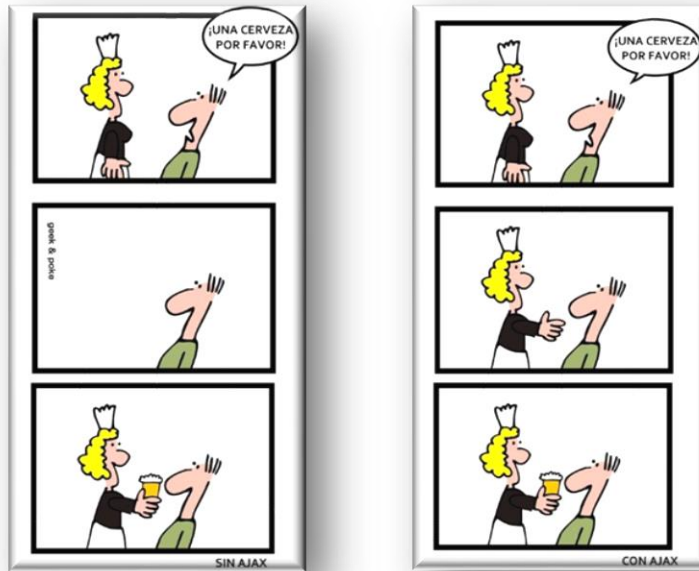
En la web, las peticiones se hacen normalmente en cada evento de un cliente, por ejemplo, un clic en un enlace o un inicio de sesión; Tradicionalmente cada petición al servidor requiere recargar, refrescar o actualizar la página para ver la nueva información, esto puede ser normal si se va a otra página interna del sitio, pero no es eficiente cuando lo que se quiere es solo cambiar un fragmento de la página, como un chat, un mapa, o mostrar alguna información contextual sobre una compra.

La característica fundamental de AJAX es permitir actualizar parte de una página con información que se encuentra en el servidor sin tener que refrescar completamente la página. De modo similar podemos enviar información al servidor. JavaScript es el encargado de comunicarse con el servidor enviando y recibiendo datos desde la página web, en el servidor la solicitud es procesada y se envía una respuesta que es interpretada de nuevo por JavaScript en la página web.



Cabe destacar que Ajax no es un lenguaje de programación, sino una técnica para crear mejores y más aplicaciones web interactivas.

Ventajas	Desventajas
<ol style="list-style-type: none"> <li>1. Utiliza tecnologías ya existentes.</li> <li>2. Soportada por la mayoría de los navegadores modernos.</li> <li>3. Interactividad. El usuario no tiene que esperar hasta que lleguen los datos del servidor.</li> <li>4. Portabilidad</li> <li>5. Mayor velocidad, esto debido que no hay que retornar toda la página nuevamente.</li> <li>6. La página se asemeja a una aplicación de escritorio.</li> </ol>	<ol style="list-style-type: none"> <li>1. Se pierde el concepto de volver a la página anterior.</li> <li>2. Si se guarda en favoritos no necesariamente al visitar nuevamente el sitio se ubique dónde nos encontrábamos al grabarla.</li> <li>3. La existencia de páginas con AJAX y otras sin esta tecnología hace que el usuario se desoriente.</li> <li>4. No funciona si el usuario tiene desactivado el JavaScript en su navegador.</li> <li>5. Dependiendo de la carga del servidor podemos experimentar tiempos tardíos de respuesta que desconciertan al visitante.</li> </ol>

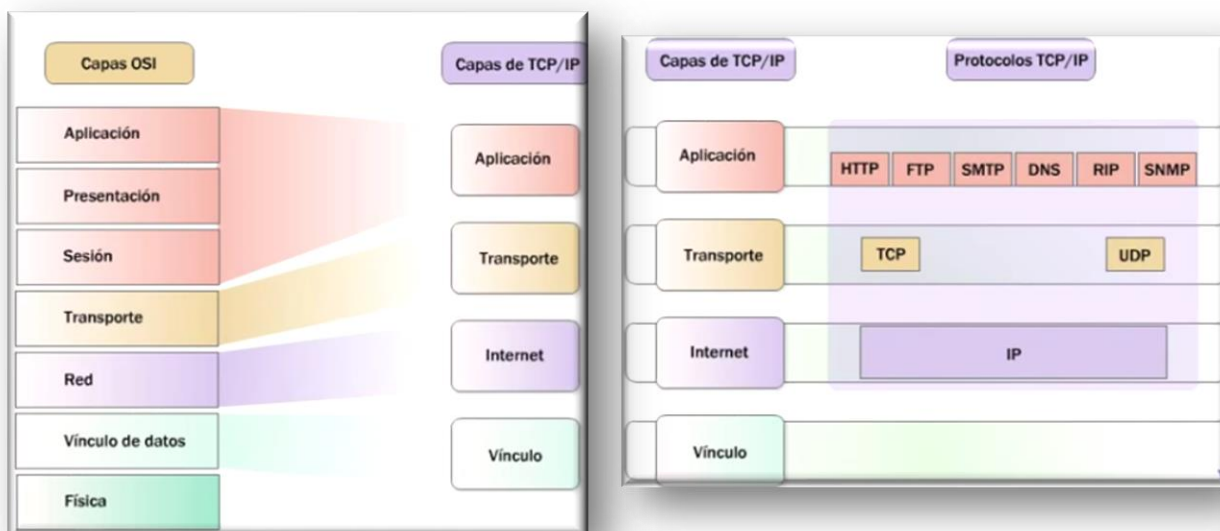


<https://oscarleuro.com/javascript/guia-de-ajax-en-espanol/>

## PROTOCOLO HTTP

Hypertext Transfer Protocol (HTTP) (o Protocolo de Transferencia de Hipertexto) es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML. Fue diseñado para la comunicación entre los navegadores y servidores web, aunque puede ser utilizado para otros propósitos también. Sigue el clásico modelo cliente-servidor, en el que un cliente establece una conexión, realizando una petición a un servidor y espera una respuesta de este.

La web funciona bajo el protocolo HTTP que es el conjunto de reglas que deben seguir cliente / servidor para comunicarse efectivamente. Las peticiones HTTP no tienen información de estado, por eso se manda desde “frontend” una información de sesión (cookies), y un proxy permite crear cachés y balancear las cargas.



<b>Aplicación</b>	Proporciona la interfaz para que las aplicaciones tengan acceso a la red.
<b>Presentación</b>	Permite a los clientes usar una sintaxis común.
<b>Sesión</b>	Asegura una comunicación fluida entre clientes.
<b>Transporte</b>	Proporciona el tipo adecuado de conexión.
<b>Red</b>	Agrega la dirección y garantiza la entrega en conexiones confiables.
<b>Vínculo de datos</b>	Prepara los paquetes para su transmisión.
<b>Física</b>	Envía los paquetes al hardware de red.

<https://developer.mozilla.org/es/docs/Web/HTTP>

[https://developer.mozilla.org/es/docs/Web/HTTP/Basics\\_of\\_HTTP](https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP)

#### Métodos de petición HTTP

HTTP define una serie predefinida de métodos de petición (algunas veces referido como "verbos") que pueden utilizarse. El protocolo tiene flexibilidad para ir añadiendo nuevos métodos y para así añadir nuevas funcionalidades. El número de métodos de petición se ha ido aumentando según se avanzaba en las versiones. Cada método indica la acción que desea que se efectúe sobre el recurso

identificado. Lo que este recurso representa depende de la aplicación del servidor. Por ejemplo, el recurso puede corresponderse con un archivo que reside en el servidor.

#### *GET*

El método GET solicita una representación del recurso especificado. Las solicitudes que usan GET solo deben recuperar datos y no deben tener ningún otro efecto. (Esto también es cierto para algunos otros métodos HTTP.)

#### *POST*

Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.

#### *PUT*

Sube, carga o realiza un upload de un recurso especificado (archivo o fichero) y es un camino más eficiente ya que POST utiliza un mensaje multiparte y el mensaje es decodificado por el servidor. En contraste, el método PUT permite escribir un archivo en una conexión socket establecida con el servidor. La desventaja del método PUT es que los servidores de alojamiento compartido no lo tienen habilitado.

#### *DELETE*

Borra el recurso especificado.

## JSON

Aunque con Ajax se puede solicitar cualquier tipo de recurso web, el intercambio de datos entre la página web y el servidor ha sido realizado tradicionalmente, como el propio nombre indica, en formato XML (eXtensible Markup Language), un lenguaje de marcas que permite definir una gramática específica y, por tanto, permite el intercambio de información estructurada y legible.

```
XML
<profesores>
  <profesor>
    <nombre>Pepe</nombre><apellido>Perez</apellido>
  </profesor>
  <profesor>
    <nombre>Ana</nombre><apellido>Sanchez</apellido>
  </profesor>
</profesores>
```

```
JSON
{ "profesores": [
  { "nombre": "Pepe", "apellido": "Perez" },
  { "nombre": "Ana", "apellido": "Sanchez" }
]}
```

Posteriormente nació JSON como una alternativa a XML, más ligero y con una notación más simple, y se convirtió en el formato más utilizado para el intercambio de datos cuándo se trabaja con Ajax, dejando obsoleto el formato XML.

JSON (JavaScript Object Notation) es un formato para el intercambio de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído por cualquier lenguaje de programación. Por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

El formato JSON tiene la siguiente notación:

```
{key : value, key2 : value2, key3 : value3,...}
```

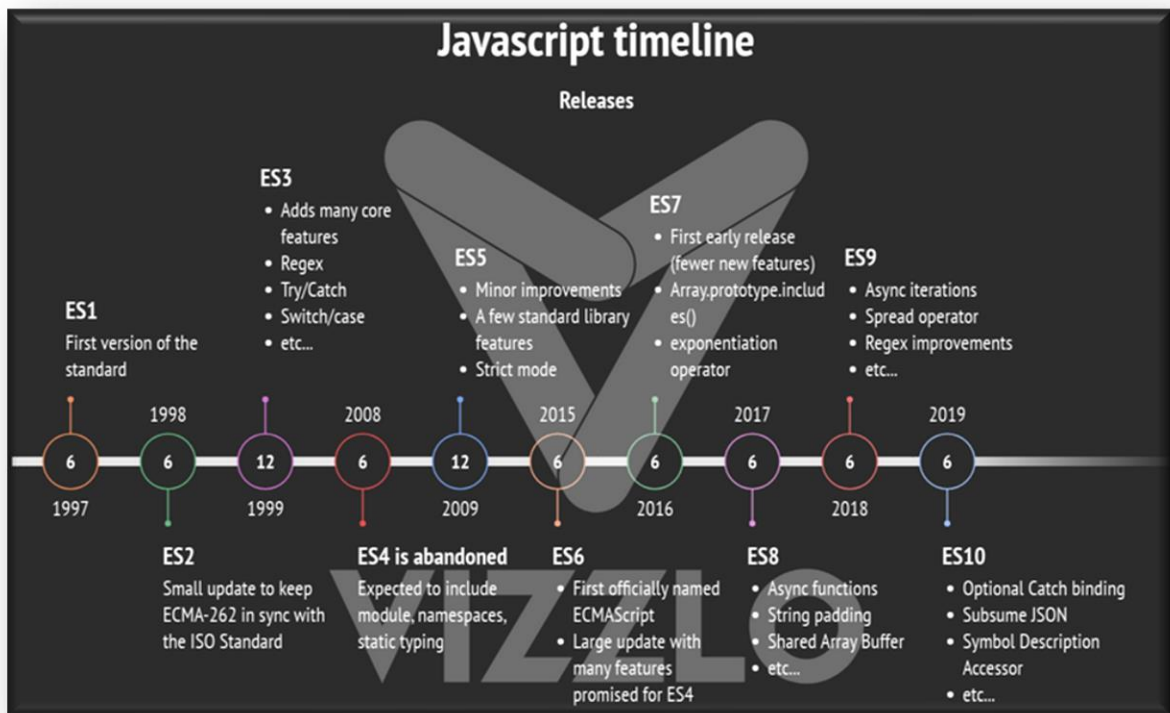
Y también puede ser serializado y multidimensional, por ejemplo:

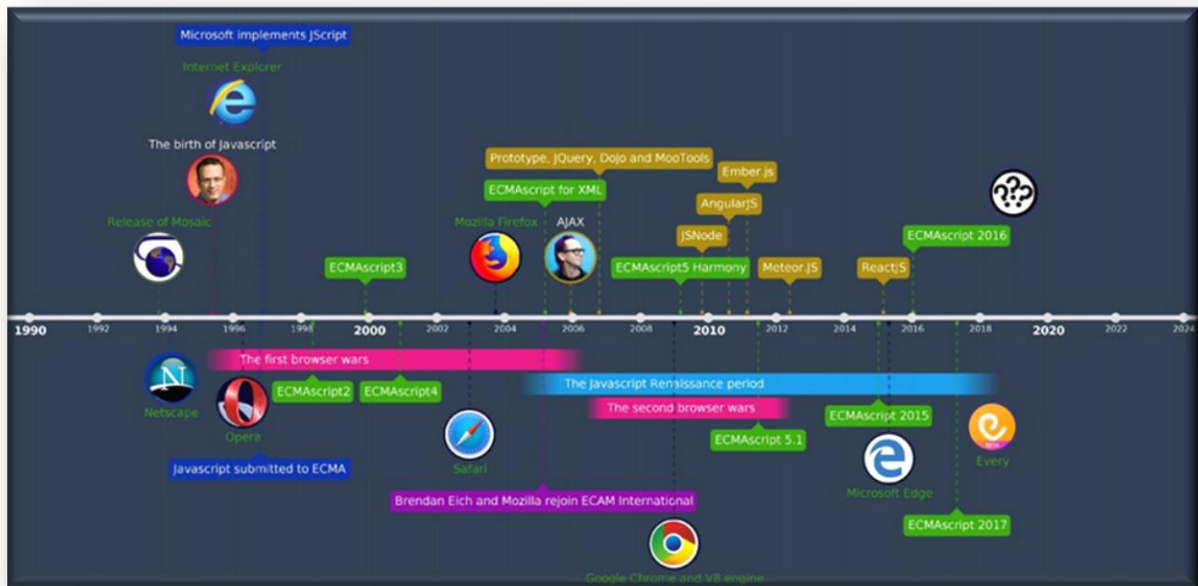
```
[
  {key : value, key2 : value2, key3 : value3,
    key : { key : value, key2 : value2, key3 : value3}
  },
  {key : value, key2 : value2, key3 : value3,...}
]
```

En el siguiente enlace se puede consultar más información sobre las especificaciones del formato JSON: <http://www.json.org/>

## ECMAScript

ECMAScript específicamente es el estándar que a partir del año 2015 a la actualidad se encarga de regir como debe ser interpretado y funcionar el lenguaje **JavaScript**, siendo este (JS – JavaScript) interpretado y procesado por multitud de plataformas, entre las que se encuentran los navegadores web, **NodeJS** u otros ambientes como el desarrollo de aplicaciones para los distintos sistemas operativos que actualmente existen en el mercado. Los responsables de dichos navegadores y JavaScript deben encargarse de interpretar el lenguaje tal como lo fija ECMAScript.





<https://www.syncfusion.com/blogs/post/top-6-javascript-es12-features-you-should-use.aspx>

## JavaScript Asíncrono

Podemos hacer que nuestro código se ejecute de forma asíncrona, y como en el siguiente ejemplo primero se muestre: “Paso 1”, luego “Paso 3” y por último “Paso 2”, pasa lo mismo con una petición AJAX , donde el código puede saltar líneas y si el resultado de la petición es para dar continuidad y si esta no se ha completado afectará el programa por completo, generando un error y el programa terminaría, con esta acción no se tiene la certeza de saber en qué momento va a terminar la ejecución o petición de un proceso, para poder enviar otro proceso que depende de él.

```
console.log('Paso 1')
setTimeout(function(){console.log("Paso 2")}, 1000)
console.log('Paso 3');
```


Para el correcto funcionamiento del siguiente código, necesitamos una forma de decirle al lenguaje de programación: “espera a que la función anterior termine”, es aquí donde se utilizan las funciones “**callbacks**” siendo estas, funciones que se ejecutan en otras. Esto se hace con el fin de estar seguros de que van a ejecutarse cuando termine la acción anterior; sin embargo, al ser funciones dentro de funciones, la anidación de “**callbacks**” puede volverse **inmantenible**, convirtiéndose en lo que se conoce como “**callback hell**”, siendo muy difícil de controlar



```

let b
console.log('a')
setTimeout(() => b = 10, 1000)
console.log(b)
//a
// undefined

```



```

1  function hell(win) {
2    // for listener purpose
3    return function() {
4      loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5        loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6          loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7            loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8              loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9                loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                     async.eachSeries(SCRIPTS, function(src, callback) {
14                       loadScript(win, BASE_URL+src, callback);
15                     });
16                   });
17                 });
18               });
19             });
20           });
21         });
22       });
23     });
24   });
25 }
26

```

## XMLHttpRequest

**XMLHttpRequest** es un objeto JavaScript que fue diseñado por Microsoft y adoptado por Mozilla, Apple y Google. Actualmente es un estándar de la W3C. Proporciona una forma fácil de obtener información de una URL sin tener que recargar la página completa. Una página web puede actualizar sólo una parte de la página sin interrumpir lo que el usuario está haciendo. XMLHttpRequest es ampliamente usado en la programación AJAX.

```
const xhr = new XMLHttpRequest();  
// Configurar la petición  
xhr.open('GET',url,true)  
// Que hacer con la petición  
xhr.addEventListener('load', hacerAlgo)  
// Realizar la petición  
xhr.send();
```

<https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest>

[https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest/Using XMLHttpRequest](https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest)

Ejemplos de llamados asíncronos con XMLHttpRequest

```

function ajax() {
    let data = new FormData();
    data.append("num1", 5);
    data.append("num2", 2);

    const xhr = new XMLHttpRequest();
    const url = 'test.php';

    xhr.onreadystatechange = function () {
        if (this.readyState == 4) {
            if (this.status == 200) {
                console.log(this.responseText);
            } else {
                console.log("Error cargando la página\n");
            }
        }
    };

    xhr.open("POST", url);
    xhr.send(data);
}

document.getElementById("btn_aceptar").addEventListener("click", ajax);

```

```

document.getElementById("btn_enviar").addEventListener("click", ajax);

function ajax(event) {
    event.preventDefault();
    var formElement = document.getElementById("myForm");

    const xhr = new XMLHttpRequest();
    const url = 'test.php';

    xhr.addEventListener("load", resultado);
    xhr.open("POST", url);
    xhr.send(new FormData(formElement));
}

function resultado() {
    if (this.readyState == 4) {
        if (this.status == 200) {
            console.log(this.responseText);
        } else {
            console.log("Error cargando la página\n");
        }
    }
}

```

## Promesas

Una promesa es un objeto que representa la terminación o el fracaso de una operación asíncrona. es un objeto devuelto al cuál se adjuntan funciones callback, en lugar de pasar callbacks a una función.

las “promesas” permiten esperar a que se cumpla o se rechace la acción o “promesa”. Las promesas son acciones que van a pasar a futuro, si se cumple o no, las acciones continúan, es lo que pasa con Javascript, sin saber el resultado de la “promesa”, la página no se bloquea y los demás procesos siguen ejecutándose.

Las promesas se crean usando un constructor llamado **Promise** y pasándole una función que recibe dos parámetros, **resolve** y **reject**, que nos permiten indicarle a esta que se resolvió o se rechazó.

```
let promesa = () => {  
  return new Promise((resolve, reject) => {  
    resolve(hacerAlgo) // resuelve la promesa  
    reject(hacerOtraCosa) // rechaza la promesa  
  })  
}  
  
// Ejecutamos la promesa  
// .then, cuando se resuelve la promesa  
// .catch, cuando se rechaza la promesa  
  
promesa()  
  .then(funcion)  
  .catch(otraFuncion)
```

[https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Usar\\_promesas](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Usar_promesas)

## Ejemplo

```
const promise = new Promise((resolve, reject) => {
  const number = Math.floor(Math.random() * 10);

  setTimeout(() => {
    console.log(number);
    if (number > 5)
      resolve(number)
    else
      reject(new Error('Menor a 5'))
  }, 1000);
});

promise
  .then(number => console.log(number))
  .catch(error => console.error(error));
```

## Fetch

Es una API que nos permite hacer peticiones sin usar XHR, donde “Fetch”, realiza las promesas maximizando las líneas de código, haciéndolo más corto y mejor aún, es compatible con todos los navegadores.

Proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, tales como peticiones y respuestas. También provee un método global [fetch\(\)](#) que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red.

```
fetch(url)
  .then(response => response.json())
  .then(response => { hacerAlgo() })
```

[https://developer.mozilla.org/es/docs/Web/API/Fetch\\_API/Utilizando\\_Fetch](https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch)

<https://dev.to/codewithahsan/8-techniques-to-write-cleaner-javascript-code-369e>

<https://www.30secondsofcode.org/js/p/1>

<https://htmlcheatsheet.com/js/>

<https://devhints.io/react>

## Ejemplos

### *Envío de datos por POST y respuesta en texto plano*

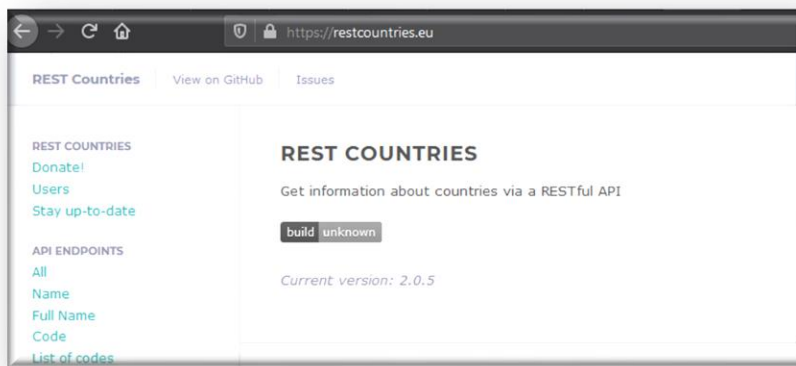
```
let myForm = document.getElementById("myForm");
myForm.addEventListener("submit", ajax);

function ajax(event){
  event.preventDefault(); // No permitir el envío del formulario
  const url = 'test.php';
  fetch(url, {
    method : 'POST',
    body : new FormData(myForm)
  })
  .then(resp => {
    if(resp.ok)
      return resp.text()
    else
      throw new Error("Error en la llamada");
  })
  .then(data => {
    document.getElementById("contenido").innerHTML = data
  })
  .catch(error => console.error(error))
}
```

### *Consumo de API de países con su respectiva información*

Para el siguiente ejemplo, realizaremos el consumo de la API pública del siguiente enlace:

<https://restcountries.eu/>



```
document.querySelector('#lista_paises').addEventListener('change', traerInfoPais);
let baseUrl = 'https://restcountries.eu/rest/v2/';
traerPaises();

function traerPaises() {
  fetch(baseUrl + 'all')
    .then(resp => resp.json())
    .then(data => {
      for (let item of data) {
        document.getElementById("lista_paises").innerHTML += `
        <option value="${item.alpha2Code}">${item.name}</option>
        `;
      }
    })
}

function traerInfoPais() {
  fetch(baseUrl + 'name/' + this.value)
    .then(resp => {
      if (resp.ok)
        return resp.json();
      else
        throw new Error("Error en la llamada");
    })
    .then(data => {
      document.getElementById("contenido").innerHTML += `
      <tr>
        <th scope="row">${data[0].alpha2Code}</th>
        <td>${data[0].capital}</td>
        <td>${data[0].region}</td>
        <td>${data[0].population}</td>
        <td></td>
      </tr>
      `;
    })
    .catch(error => console.error(error));
}
```

### Ejercicios

1. Consultar el API de <https://coinmarketcap.com> y realizar un consumo a través de PHP y mostrar los datos consumidos de manera agradable al usuario, según las indicaciones del instructor
2. Consultar las API dispuesta en <https://www.datos.gov.co/>, seleccionar un API y realizar un ejemplo con los datos que se puedan consumir, según indicaciones del instructor.

Ejemplos:

- <https://dev.socrata.com/foundry/www.datos.gov.co/xdk5-pm3f>
- <https://dev.socrata.com/foundry/www.datos.gov.co/32sa-8pi3>

### Referencias

- <https://oscarleuro.com/javascript/guia-de-ajax-en-espanol/>
- [https://developer.mozilla.org/es/docs/Web/HTTP/Basics\\_of\\_HTTP](https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP)
-