

DWEC

TEMA 8



AJAX



ÍNDICE

8.1 Introducción

8.2 Los web workers.

8.3 Comunicación asíncrona. AJAX.

8.4 Comunicación API REST y
operaciones CRUD.

8.5 AJAX con el objeto XMLHttpRequest.

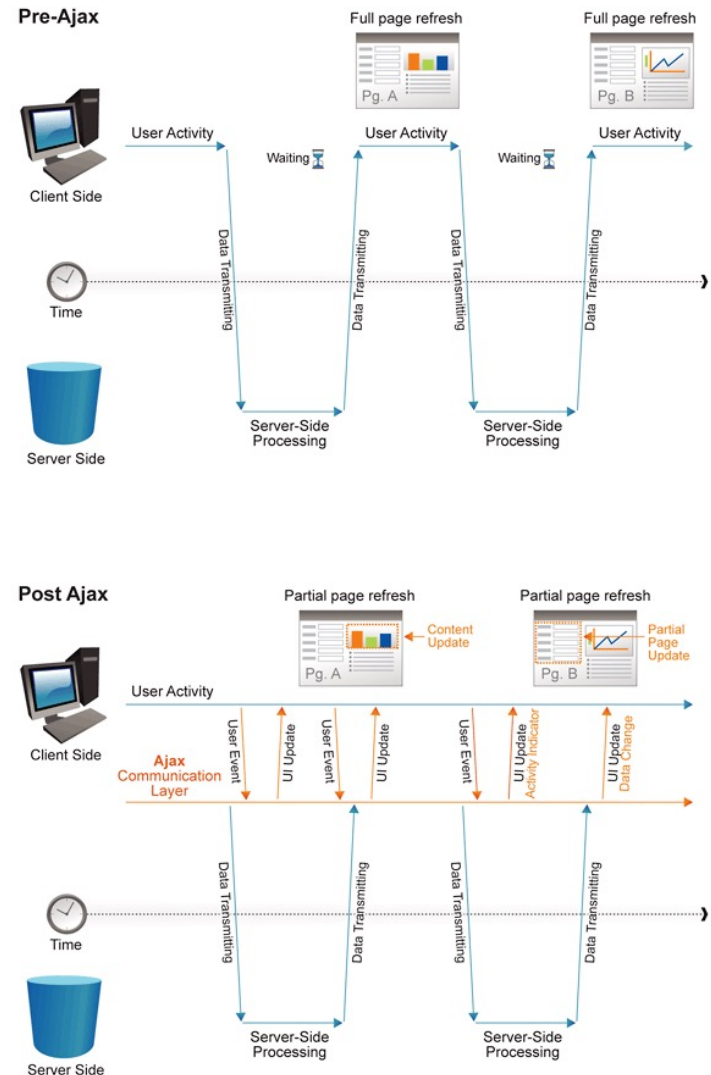
8.6 AJAX con JQuery.

8.7 Ajax con fetch.

8.1 INTRODUCCIÓN

La comunicación asíncrona es una de las cualidades que le da a JavaScript su máxima potencia. Con la creación de AJAX cambió la forma de comunicarse con los servidores. Ya no se necesitaba recargar las páginas para comunicarse con el servidor, solo se recargaban los elementos que cambiaban en la página.

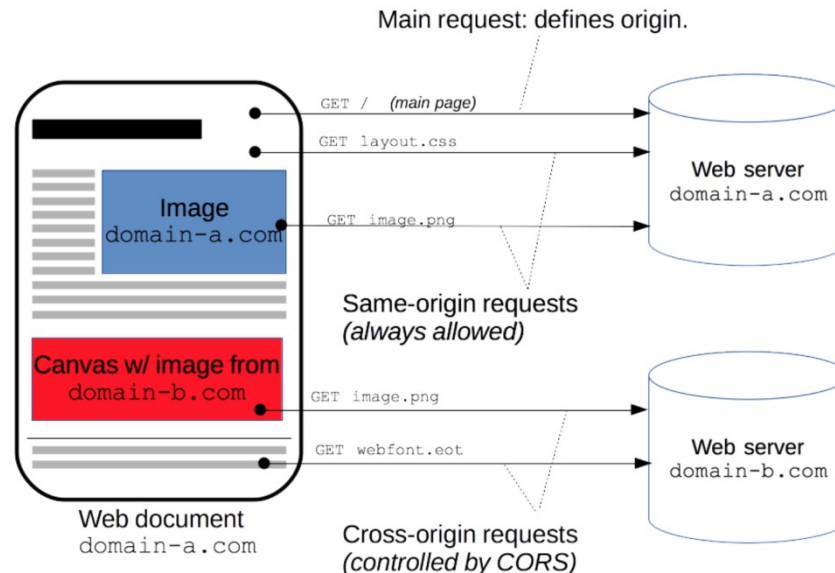
A demás no se deja en espera la usuario y se disminuye el rendimiento del servidor.



8.1 INTRODUCCIÓN

CORS

Un concepto nuevo que sale a partir del uso de AJAX es el CORS, el acrónimo de Cross Origin Resource Sharing, que es un mecanismo por el cual una página web pide permiso para poder acceder a recursos que se encuentran en otro **servidor o dominio distinto** al que pertenece el primero. Esta acción es muy común en el uso de APIs publicas.




8.1 INTRODUCCIÓN

CORS

Tenemos que tener en cuenta que los navegadores restringen este tipo de conexiones cruzadas por razones de seguridad. Tanto si utilizas XMLHttpRequest o Fetch te vas a encontrar con ese problema, salvo que utilices encabezados CORS en las peticiones o actives un plugin CORS.

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>


Resultados de la búsqueda

**CORS Everywhere**

24.757 usuarios

A firefox addon allowing the user to enable CORS everywhere by altering http responses. Report issues to the repository, with enough information to reproduce the problem: <https://github.com/spenibus/cors-everywhere-firefox-addon/issues>

★★★★★ spenibus

**Allow CORS: Access-Control-Allow-Origin**

11.978 usuarios

Easily add (Access-Control-Allow-Origin: *) rule to the response header.

★★★★★ Muyor

8.2 LOS WEB WORKERS

Un Web worker es un proceso que se ejecuta en segundo plano, de tal manera que la página no se vea penalizada ni ralentizada por tener un proceso ejecutándose de forma permanente.

Se pueden utilizar web workers para analizar los patrones de navegación o para realizar llamadas asíncronas a un servicio de forma transparente al usuario.

Para ejecutar web workers el navegador debe de soportar HTML 5.

<https://www.youtube.com/watch?v=zhfsqqZMLUY>

8.2 LOS WEB WORKERS

Los Web Workers se ejecutan en un hilo aislado. Como resultado, el código que ejecutan debe estar contenido en un archivo separado. Pero antes de hacer eso, lo primero que debe hacer es crear un nuevo objeto Worker en su página principal. El constructor toma el nombre del script de trabajo:

```
var worker = new Worker('task.js');
```

Si el archivo especificado existe, el navegador generará un nuevo hilo de trabajo, que se descarga de forma asincrónica. El trabajador no comenzará hasta que el archivo se haya descargado y ejecutado por completo. Si la ruta a su trabajador devuelve un 404, el trabajador fallará silenciosamente.

8.2 LOS WEB WORKERS

Después de crear el trabajador, inícialo llamando al método `postMessage ()`:

```
worker.postMessage(); // Start the worker.
```

Comunicarse con un workers con el paso de mensajes:

La comunicación entre un work y su página principal se realiza mediante un modelo de evento y el método `postMessage ()`.

Puede aceptar una cadena o un objeto JSON como su único argumento. Las últimas versiones de los navegadores modernos admiten el paso de un objeto JSON.

8.2 LOS WEB WORKERS

Ejemplo del uso de una cadena para pasar 'Hola mundo' a un trabajador en doWork.js. El trabajador simplemente devuelve el mensaje que se le pasa.

Script principal:

```
var worker = new Worker('doWork.js');

worker.addEventListener('message', function(e) {
  console.log('Worker said: ', e.data);
}, false);

worker.postMessage('Hello World'); // Send data to our worker.
```

doWork.js (el worker):

```
self.addEventListener('message', function(e) {
  self.postMessage(e.data);
}, false);
```

Cuando se llama a `postMessage ()` desde la página principal, nuestro worker maneja ese mensaje definiendo un controlador `onmessage` para el evento del mensaje. Se puede acceder a la carga útil del mensaje (en este caso, 'Hola mundo') en `Event.data`.

8.2 LOS WEB WORKERS

Otro Ejemplo : Script Principal

```
<button onclick="sayHI()">Say HI</button>
<button onclick="unknownCmd()">Send unknown command</button>
<button onclick="stop()">Stop worker</button>
<output id="result"></output>

<script>
  function sayHI() {
    worker.postMessage({cmd: 'start', msg: 'Hi'});
  }

  function stop() {
    // worker.terminate() from this script would also stop the worker.
    worker.postMessage({cmd: 'stop', msg: 'Bye'});
  }

  function unknownCmd() {
    worker.postMessage({cmd: 'foobard', msg: '???'});
  }

  var worker = new Worker('doWork2.js');

  worker.addEventListener('message', function(e) {
    document.getElementById('result').textContent = e.data;
  }, false);
</script>
```

doWork.js

```
self.addEventListener('message', function(e) {
  var data = e.data;
  switch (data.cmd) {
    case 'start':
      self.postMessage('WORKER STARTED: ' + data.msg);
      break;
    case 'stop':
      self.postMessage('WORKER STOPPED: ' + data.msg +
        '. (buttons will no longer work)');
      self.close(); // Terminates the worker.
      break;
    default:
      self.postMessage('Unknown command: ' + data.msg);
  }
}, false);
```

8.3 COMUNICACIÓN ASÍNCRONA. AJAX.

AJAX es el acrónimo de Asynchronous JavaScript and XML. Con esta técnica, se pueden crear páginas dinámicas de forma rápida que consuman de una forma asíncrona los datos externos a la página.

Esta técnica permite que la página se actualice sin tener que enviar información otra vez al servidor (y esperar al que el servidor la responda).

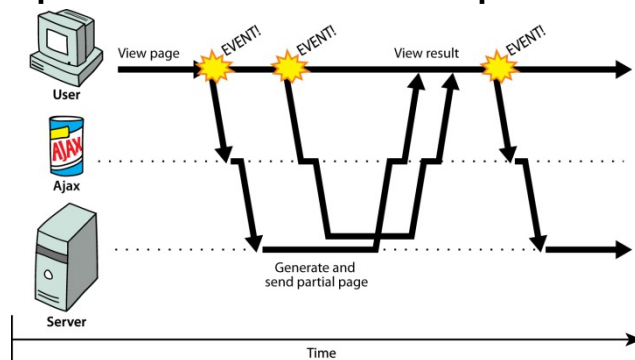
Como únicamente se intercambian datos, solo se envía lo que se necesita (no toda la página)



8.3 COMUNICACIÓN ASÍNCRONA. AJAX.

La programación multihilo tiene un funcionamiento parecido a AJAX. Por lo que cuando se ejecuta una sentencia AJAX, el programa generará un hilo de programación nuevo el cual se ejecutará en paralelo a nuestro hilo de programación principal.

Este concepto lo tenemos que tener en cuenta cuando estamos programando, ya que no sabemos cuando recibiremos la respuesta de nuestra petición ajax.



8.4 COMUNICACIÓN API REST Y OPERACIONES CRUD.

Las comunicaciones que realizamos con la tecnología AJAX necesitan que el servidor funcione como un servicio API (**A**pplication **P**rogramming Interfaces, que en español significa *interfaz de programación de aplicaciones*) RESTful. Características:

- Los servicios REST transmiten la información en JSON o XML y no HTML .
- Hacen uso explícito de los métodos HTTP (el CRUD).
- Necesitan de un "endpoint" para realizar la petición al servidor API. Es decir necesitan de una URI para intercambiar los datos. En entorno servidor se conocen como rutas (routes).

8.4 COMUNICACIÓN API REST Y OPERACIONES CRUD.

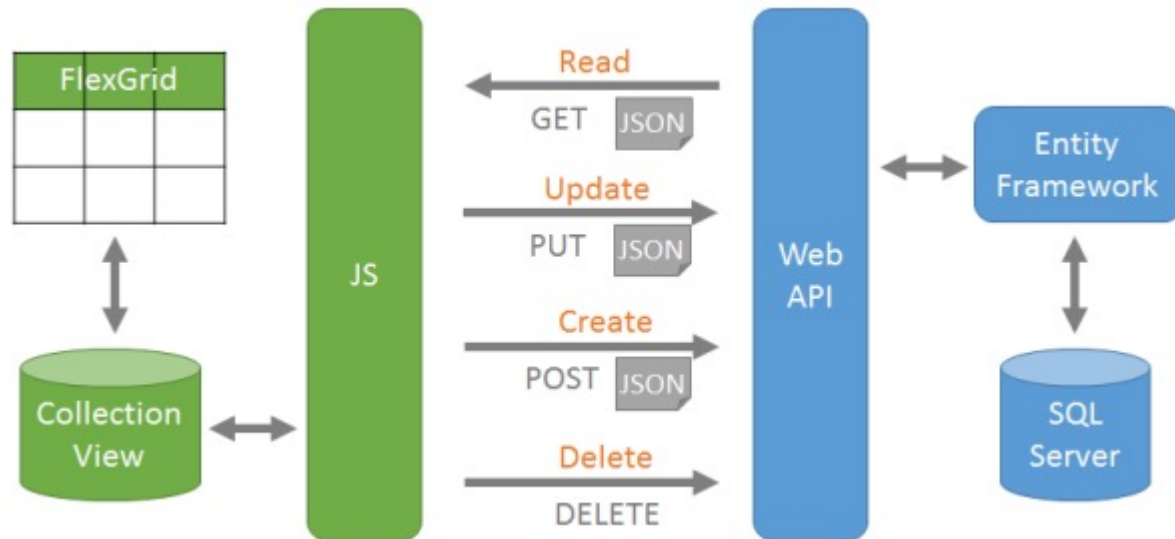
Las ventajas de utilizar las API son:

- Se envían respuestas mas concretas a las peticiones del front-end. Ya que el servidor solo envía la información que se requiere.
- El servidor tiene menos carga de trabajo.
- Se puede utilizar la estructura de la API para poder intercomunicar con otros clientes: Navegadores Web, Aplicaciones móviles, Aplicaciones escritorios u otras API's.

8.4 COMUNICACIÓN API REST Y OPERACIONES CRUD.

CRUD

Cada vez que se envíe o reciba una petición, esta tendrá asociado un método HTTP y el API traducirá ese método a peticiones a la base de datos. Tenemos esta correlación:



8.4 COMUNICACIÓN API REST Y OPERACIONES CRUD.

CRUD: HTTP GET

GET se utiliza solo para **recuperar información** o recursos.

- No se le envía nada en el cuerpo, pero si que se puede modificar la URI para especificar la petición.
- La respuesta:
 - Si el recurso se encuentra en el servidor, devuelve el código de respuesta HTTP 200 (OK). Y en el cuerpo de la respuesta tenemos los datos en formato JSON o XML. Aunque el servidor puede devolver en la respuesta algún mensaje de error.
 - Si el recurso NO se encuentra en el servidor, se devuelve el código HTTP 404.
 - Si la solicitud GET no está formada correctamente, el servidor nos informará con el código HTTP 400 (BAD REQUEST).

8.4 COMUNICACIÓN API REST Y OPERACIONES CRUD.

CRUD: HTTP POST

Se utiliza el POST para crear **nuevos recursos** .

- Se informará mediante el método POST y en el cuerpo se introducirá los datos a crear mediante un JSON.
- Si se ha creado un recurso en el servidor, la respuesta debe de ser el código de HTTP 201 (Creado) o 200 i el cuerpo de la respuesta contendrá información donde se describe el estado de la solicitud.
- La acción realizada por el método POST no tiene porqué derivar en un recurso que pueda ser identificado por una URI. En este caso, el código de respuesta será HTTP 204 (No Content) que será el estado de respuesta adecuado.

8.4 COMUNICACIÓN API REST Y OPERACIONES CRUD.

CRUD: HTTP PUT

Utilizamos PUT principalmente para **actualizar** un recurso existente (si el recurso no existe, entonces la API puede decidir crear un recurso nuevo o no).

- Si se ha creado un recurso nuevo por la API PUT, el servidor de origen debe informar al cliente mediante la respuesta del código HTTP 201 (Creada) y si se modifica un recurso existente, los 200 (OK) o los 204 (Sin contenido).
- La diferencia entre las API de POST y PUT se puede observar en los URI de solicitud. La solicitud POST se informa en el URI del recurso que se va a crear, mientras que la solicitud PUT se informa en el URI del recurso que se va a actualizar.

8.4 COMUNICACIÓN API REST Y OPERACIONES CRUD.

CRUD: HTTP DELETE

DELETE se utiliza para eliminar recursos (identificados por el URI de la solicitud).

- Una respuesta exitosa de las solicitudes DELETE debería ser el código de respuesta HTTP 200 (Ok) si la respuesta incluye una entidad que describe el estado, 202 (Aceptado).

8.5 AJAX CON EL OBJETO XMLHTTPRequest.

AJAX esta basado en el objeto XMLHttpRequest para intercambiar datos con el servidor de forma asíncrona. Existen otros métodos como JQuery, el API Fetch que está integrada en la mayoría de navegadores o el api AXIOS que necesita instalarse igual que JQuery. Encara que el API Fetch es mucho más sencillo de utilizar al igual que JQuery debemos de conocer el funcionamiento de este obeejto.

8.5 AJAX CON EL OBJETO XMLHttpRequest.

XMLHttpRequest tiene los siguientes atributos:

Atributo	Descripción
readyState	Devuelve el estado del objeto como sigue: <ul style="list-style-type: none">● 0 = sin inicializar● 1 = abierto● 2 = cabeceras recibidas● 3 = cargando● 4 = completado.
responseBody	Devuelve la respuesta como un array de bytes.
responseText	Devuelve la respuesta como una cadena.
responseXML	Devuelve la respuesta como XML. Esta propiedad devuelve un objeto documento XML, que puede ser examinado usando las propiedades y métodos del árbol del Document Object Model .
status	Devuelve el estado como un número (p. ej. 404 para "Not Found" y 200 para "OK").
statusText	Devuelve el estado como una cadena (p. ej. "Not Found" o "OK").

8.5 AJAX CON EL OBJETO XMLHTTPREQUEST.

XMLHttpRequest se pueden llamar los siguientes métodos:

Método	Descripción
abort()	Cancela la petición en curso.
getAllResponseHeaders()	Devuelve el conjunto de cabeceras HTTP como una cadena.
getResponseHeader (nombreCabecera)	Devuelve el valor de la cabecera HTTP especificada.
open (método, URL, [asíncrono [nombreUsuario [clave]])	<p>Especifica el método, URL y otros atributos opcionales de una petición.</p> <p>El parámetro de método puede tomar los valores "GET", "POST", o "PUT" ("GET" y "POST" son dos formas para solicitar datos, con "GET" los parámetros de la petición se codifican en la URL y con "POST" en las cabeceras de HTTP).</p> <p>El parámetro URL puede ser una URL relativa o completa.</p> <p>El parámetro <code>asíncrono</code> especifica si la petición será gestionada asíncronamente o no. Un valor <code>true</code> indica que el proceso del script</p>

8.5 AJAX CON EL OBJETO XMLHttpRequest.

XMLHttpRequest se pueden llamar los siguientes métodos:

	<p>continúa después del método <code>send()</code>, sin esperar a la respuesta, y <code>false</code> indica que el script se detiene hasta que se complete la operación, tras lo cual se reanuda la ejecución.</p> <p>En el caso asíncrono se especifican manejadores de eventos, que se ejecutan ante cada cambio de estado y permiten tratar los resultados de la consulta una vez que se reciben, o bien gestionar eventuales errores.</p>
<code>send([datos])</code>	Envía la petición.
<code>setRequestHeader</code> (etiqueta, valor)	Añade un par etiqueta/valor a la cabecera HTTP a enviar.

8.5 AJAX CON EL OBJETO XMLHttpRequest.

XMLHttpRequest también tiene los siguientes eventos:

Propiedad	Descripción
onreadystatechange	Evento que se dispara con cada cambio de estado.
onabort	Evento que se dispara al abortar la operación.
onload	Evento que se dispara al completar la carga.
onloadstart	Evento que se dispara al iniciar la carga.
onprogress	Evento que se dispara periódicamente con información de estado.

8.5 AJAX CON EL OBJETO XMLHTTPREQUEST.

Pasos a realizar para hacer una llamada AJAX:

```
function muestraSugerencia(str) {  
    if (str.length == 0) {  
        document.getElementById("txtSugerencia").innerHTML = "";  
        return;  
    } else {  
        var xmlhttp = new XMLHttpRequest();  
        xmlhttp.onreadystatechange = function() {  
            if (this.readyState == 4 && this.status == 200) {  
                document.getElementById("txtSugerencia").innerHTML = this.  
                    responseText;  
            }  
        };  
        xmlhttp.open("GET", "http://localhost/Proyectos/JC/getSugerencia.  
            php?param=" + str, true);
```