UT06 Introducción a Ajax

Para este ejercicio es necesario disponer de un servidor web al que enviar las peticiones HTTP. Vosotros ya sabéis bastante de servidores, y si queréis podéis utilizar aquello que consideréis, pero si preferís una solución rápida que os permita centraros en la parte del cliente, tenemos esta opción:

* [json-server](https://github.com/typicode/json-server). Es un paquete de Node que nos permite crear un pequeño servidor local al que podemos enviar peticiones, y del que retornará la información en formato JSON que necesitamos.

Hay que descargar e instalar *Node* y *npm* en vuestro ordenador.

sudo apt install nodejs

sudo apt install npm

Una vez terminada la instalación, abre la terminal de comandos de tu ordenador y comprueba que Node y npm se han instalado correctamente, utilizando el comando node -v y npm -v. Si todo ha ido bien, deberás ver la versión que ha sido instalada.

Ahora instala *json-server*, el paquete de Node que te permitirá crear el servidor local que responda a tus peticiones HTTP. El comando para instalarlo es el siguiente:

npm install -g json-server

Una vez instalado, crea una carpeta con el archivo JSON que contendrá la información a la que queremos acceder desde el cliente. Yo he creado la siguiente ruta de directorios:

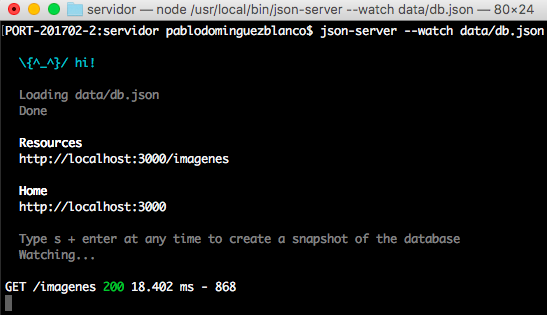
/home/alumno/Escritorio/Manu/GitHub/DWEC/UT06/IntroAjaxservidor/data/db.json.

El contenido en formato JSON puede ser el que quieras, pero yo he creado uno con unas cuantas informaciones sobre distintas imágenes ficticias, los puedes encontrar aquí.

Ahora, sitúate dentro de la carpeta servidor y ejecuta el siguiente comando:

json-server --watch data/db.json .

Con esto, habrás iniciado tu servidor local, al que puedes acceder a través de http://localhost:3000 . Tu servidor está listo para recibir peticiones HTTP, y enviar una respuesta con la información requerida:



Enviar una petición al servidor

Crea el documento curso.html y añade el siguiente contenido:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Envío de peticiones Ajax</title>

</head>

<body>

<div id="imagenes"></div>

<script src="../js/curso.js"></script>

</body>

</html>

Crea también el documento curso.js con el contenido :

// Creación de la petición HTTP

var req = new XMLHttpRequest();

// Petición HTTP GET síncrona hacia el archivo fotos.json del servidor

req.open("GET", "http://localhost:3000/imagenes", false);

// Envío de la petición

req.send(null);

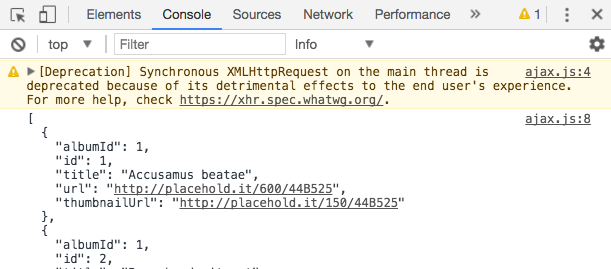
// Impresión por la consola de la respuesta recibida desde el servidor

console.log(req.responseText);

Abre la página curso.html en el navegador y mira el resultado mostrado por consola. Se muestra una lista de elementos que contienen información sobre diferentes imágenes: id, título, dirección de la imagen, etc.

Se muestra también un mensaje que advierte que las peticiones síncronas están obsoletas. En seguida veremos por qué.

A través de JavaScript hemos conseguido obtener datos provenientes de un servidor. Veamos en detalle cómo se ha realizado la petición HTTP al servidor.



El objeto XMLHttpRequest

El código JavaScript que acabas de ver contiene tres líneas de código que no habíamos visto nunca antes. El objeto **XMLHttpRequest** permite crear peticiones HTTP en JavaScript. Con él es posible conectarse a un servidor para intercambiar información.

// Creación de la petición HTTP

var req = new XMLHttpRequest();

Una vez se ha creado el objeto XMLHttpRequest podemos empezar a pedir datos. El primer paso es llamar al método open para iniciar el objeto.

// Petición HTTP GET síncrona hacia el archivo fotos.json del servidor

req.open("GET", "http://localhost:3000/imagenes", false);

Este método acepta tres argumentos (método http, url, async):

* El primer argumento es el método http, es una cadena de texto que contiene el tipo de petición a realizar (GET, POST, DELETE,...).
* El segundo argumento es la dirección url donde se encuentra el servidor al que vamos a realizar la petición.
* El tercer argumento es un booleano que indica si queremos que la petición sea síncrona o asíncrona.

Una petición síncrona detiene la ejecución del resto de código JavaScript hasta que se obtiene la respuesta del servidor. Esto puede afectar al funcionamiento de la web, así pues, la mayoría de las veces se hace uso de peticiones asíncronas.

En las peticiones asíncronas el navegador continúa ejecutando la aplicación mientras que el objeto XMLHttpRequest espera la respuesta del servidor.

// Envío de la petición

req.send(null);

// Impresión por la consola de la respuesta recibida desde el servidor

console.log(req.responseText);

En tercer lugar, se envía la petición al servidor a través del método send . Este método acepta como argumento una cadena de caracteres con información enviada al servidor (para peticiones POST o PUT). En el caso de una petición GET, en la que no se envía nada al servidor, se pasa el parámetro null.

La propiedad responseText contiene la respuesta devuelta por el servidor en forma de texto.

En resumen, este código envía una petición GET para obtener el archivo JSON que contiene información de una lista de imágenes. La petición se realiza de forma síncrona.

Peticiones asíncronas

Para asegurarnos de que la experiencia del usuario sea la mejor posible, es recomendable hacer uso de peticiones asíncronas que no bloqueen la ejecución del código JavaScript mientras se espera la respuesta del servidor.

¿Cómo podemos saber que la respuesta del servidor ha sido recibida? Pues haciendo uso de una técnica que ya conocemos bien: los eventos. Gracias a ellos, el programa sabrá que la respuesta ha sido recibida y ejecutará el bloque de código adecuado.

Realicemos una nueva petición pero esta vez, de forma asíncrona.

// Creación de la petición HTTP

var req = new XMLHttpRequest();

// Petición HTTP GET asíncrona si el tercer parámetro es "true" o no se especifica

req.open("GET", "http://localhost:3000/imagenes", true);

// Gestor del evento que indica el final de la petición (la respuesta se ha recibido)

req.addEventListener("load", function() {

// Muestra la respuesta recibida

console.log(req.responseText);

});

// Envío de la petición

req.send(null);

Un evento de tipo load indica el final de la petición, el evento se activa cuando se ha obtenido una respuesta del servidor. Con este código, el resultado obtenido es el mismo, pero la respuesta se obtiene de forma asíncrona.

Gestión de errores

Todo intercambio entre dos programas puede dar lugar a errores: la dirección url del servidor es incorrecta, el servidor no está disponible, la conexión a internet no funciona bien, etc. La gestión de los errores es una problemática compleja. En primer lugar, vamos a añadir el código necesario para detectar si se ha producido un error, y lo mostraremos por consola.

// Creación de la petición HTTP

var req = new XMLHttpRequest();

// Petición HTTP GET asíncrona si el tercer parámetro es "true" o no se especifica

req.open("GET", "http://localhost:3000/imagenes", true);

// Gestor del evento que indica el final de la petición (la respuesta se ha recibido)

req.addEventListener("load", function() {

// La petición ha tenido éxito

if (req.status >= 200 && req.status < 400) {

console.log(req.responseText);

} else {

// Se muestran informaciones sobre el problema ocasionado durante el tratamiento de la petición

console.error(req.status + " " + req.statusText);

}

});

// Gestor del evento que indica que la petición no ha podido llegar al servidor

req.addEventListener("error", function(){

console.error("Error de red"); // Error de conexión

});

// Envío de la petición

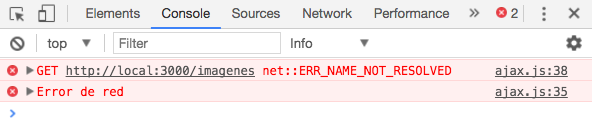
req.send(null);

La petición al servidor puede tener éxito, pero es posible también que se produzca algún error en la respuesta como por ejemplo, que no se encuentre la información solicitada. Para ello, se verifica que la propiedad status de la petición contenga los valores que aseguran que la respuesta ha tenido éxito.

Se distinguen dos casos de error principales:

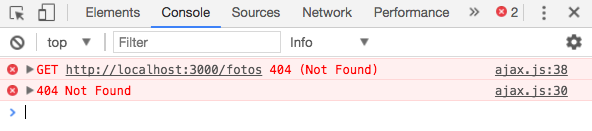
* La petición no ha alcanzado el servidor (nombre del servidor incorrecto, error de conexión, etc.). Estos errores activan el evento de tipo error sobre la petición. El gestor asociado muestra el mensaje "Error de red" en la consola.
* La petición ha recibido una respuesta del servidor, pero el procesamiento de la información ha fallado (problema interno del servidor, recurso no encontrado, etc.). Es el retorno de la petición, a través de la propiedad status la que indica el resultado. Un código mayor o igual que 200 y menor que 400 marca que la respuesta ha tenido éxito.

Veamos la gestión que hemos hecho de los errores. En primer lugar modifica la URL del servidor, por ejemplo, cambiando el nombre localhost por sólo local .



Al recargar la página en el navegador obtenemos un error de red: la petición no ha llegado al servidor ya que no existe.

Probemos ahora a modificar la URL del servidor, volviendo a escribir bien localhost pero cambiando la ruta hacia el recurso que queremos pedir. En este caso cambiaremos /imagenes por /fotos .



Obtendremos como resultado un error 404, ya que no se puede encontrar la información solicitada.

Escribir una función AJAX genérica

Si tu programa necesita realizar varias peticiones HTTP hay que evitarla duplicación de código, haciendo una función que se encargue de esta tarea. Las únicas variaciones de la función serán la URL del servidor al que enviar la petición, y la acción a realizar una vez obtenida la respuesta.

function ajaxGet(url, callback) {

var req = new XMLHttpRequest();

req.open("GET", url, true);

req.addEventListener("load", function() {

if (req.status >= 200 && req.status < 400) {

// Llamada ala función callback pasándole la respuesta

callback(req.responseText);

} else {

console.error(req.status + " " + req.statusText);

}

});

req.addEventListener("error", function(){

console.error("Error de red");

});

req.send(null);

}

La función ajaxGet permite ejecutar una petición HTTP asíncrona.

Se dice que un lenguaje de programación soporta funciones de primera clase si es posible pasar funciones como argumentos de otras funciones, usar funciones como valor de retorno de otras funciones, usar funciones anónimas y asignarlas a variables.

JavaScript soporta las funciones de primera clase, y por tanto, se permite que sean pasadas como parámetros, como podemos ver en nuestra función ajaxGet.

El término callback utilizado como nombre del segundo parámetro es normalmente empleado para nombrar una función que se pasa como parámetro de otra, y que será ejecutada posteriormente en respuesta a un cierto evento.

El código para hacer una petición Ajax es ahora mucho más simple: sólo necesitaremos llamar a la función ajaxGet pasándole le URL del servidor y la función (callback) que queramos ejecutar cuando obtengamos una respuesta válida.

function mostrar(respuesta) {

console.log(respuesta);

}

ajaxGet("http://localhost:3000/imagenes", mostrar);

En este caso definimos la función callback mostrar y la pasamos como parámetro de la función ajaxGet.

En lugar de definir la función mostrar para luego pasarla como parámetro, podemos crear sobre la marcha la función en el momento de pasarla como parámetro. Esto se hace de la siguiente forma:

ajaxGet("http://localhost:3000/imagenes", function(respuesta)

console.log(respuesta);

});

En principio, sólo vamos a llamar a la función mostrar como parámetro de ajaxGet , por lo tanto, para qué queremos definirla y darle un nombre si podemos pasar una función anónima directamente como parámetro.

Esta forma de pasar funciones anónimas como un parámetro lo has visto anteriormente, por ejemplo, en el método forEach .

Utilizar la función Ajax en varios ficheros

La creación de la función ajaxGet nos permite evitar la duplicación de código con cada llamada Ajax. Hay una problema más por resolver: ¿qué ocurre si varios ficheros JavaScript necesitan efectuar peticiones HTTP?

La solución de copiar y pegar la función en cada archivo que necesite usarla no es una solución aceptable. Ya sabes que nunca hay que repetir código. Para evitar esto, podemos definir la función en otro archivo que esté disponible para el resto de archivos de JavaScript.

Crea un nuevo archivo ajax.js e incluye en él la función ajaxGet , suprimiéndola de curso.js. Finalmente, modifica el archivo curso.html para incluir el archivo ajax.js.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Envío de peticiones Ajax</title>

</head>

<body>

<div id="imagenes"></div>

<script src="../js/ajax.js"></script>

<script src="../js/curso.js"></script>

</body>

</html>

El fichero ajax.js debe ser incluido siempre antes del resto de archivos JavaScript que utilicen la función.

La razón es que el navegador lee el archivo html de arriba a abajo, por lo tanto, si se intenta ejecutar la función ajaxGet antes de que se haya cargado el archivo que contiene la definición de la función, ocurrirá un error.

Analizar y convertir al formato JSON

Hasta ahora hemos imprimido por consola la información obtenida en formato JSON, pero la gracia de transmitir JSON es que podemos fácilmente transformarlo en objetos de JavaScript válidos, y viceversa.

ajaxGet("http://localhost:3000/imagenes", function(respuesta) {

// Transformación de formato JSON a JavaScript

var imagenes = JSON.parse(respuesta);

console.log(imagenes);

});

A través de la función JSON.parse podemos transformar una cadena de caracteres con formato JSON en un objeto de JavaScript. Por otro lado, la función JSON.stringify permite obtener el resultado inverso: transforma un objeto JavaScript en una cadena de texto con formato JSON.

En este caso pasamos una función callback que transforma a JavaScript la respuesta en formato JSON del servidor. Como puedes ver en la siguiente imagen, ahora la información se ha transformado en un array que contiene objetos con información de diferentes imágenes.



Finalmente, vamos a extraer la información obtenida del servidor y mostraremos las imágenes en miniatura en el cuerpo de nuestro documento html, no por consola.

ajaxGet("http://localhost:3000/imagenes", function(respuesta) {

// Transformación de formato JSON a JavaScript

var imagenes = JSON.parse(respuesta);

imagenes.forEach(function(elemento) {

var imgElt = document.createElement("img");

imgElt.src = elemento.thumbnailUrl;

document.getElementById("imagenes").appendChild(imgElt);

});

});

En este caso, la función callback crea un elemento <img> por cada elemento del array, y le añade a la propiedad src la dirección de la imagen en miniatura thumbnailUrl . Para concluir, añadimos cada una de las imágenes al elemento con id="imagenes" de curso.html.



En resumen

* La ejecución de peticiones HTTP necesita de un servidor que envíe los datos necesarios.
* json-server es un servidor ligero que nos permite hacer peticiones HTTP de datos en formato JSON fácilmente, sin necesidad de montar por nuestra cuenta un servidor clásico con herramientas más complejas.
* El objeto JavaScript XMLHttpRequest permite crear una petición HTTP. El método open configura la petición. El método send envía la petición al servidor.
* Una petición HTTP síncrona bloquea el código JavaScript hasta que obtiene una respuesta.
* Las peticiones asíncronas continúan la ejecución del resto de código JavaScript hasta que el evento load alerta de la respuesta del servidor. Es preferible utilizar siempre peticiones asíncronas.
* La gestion de errores es necesaria cuando se realizan llamadas Ajax. Puede obtenerse un error al enviar la petición al servidor debido a algún tipo de problema de red. También es posible obtener una respuesta a la petición, pero que la respuesta no sea satisfactoria.
* La función JSON.parse permiten transformar información en formato JSON en un objeto de JavaScript.
* La función JSON.stringify permite transformar un objeto de JavaScript en formato JSON.

Este ejercicio está sacado de :

<https://openclassrooms.com/en/courses/4387551-crea-paginas-web-interactivas-con-javascript/4476366-envia-peticiones-ajax-al-servidor>