Curso HTML, CSS y JavaScript

Junio de 2017

#### Autor: Fran Linde Blázquez

# Tema 6: Trabajando con API’s

ÍNDICE

[Tema 6: Trabajando con API’s 1](#_Toc485978925)

[1. Introducción 3](#_Toc485978926)

[2. Accediendo a una API 4](#_Toc485978927)

[2.1 Peticiones AJAX con XMLHttpRequest 4](#_Toc485978928)

[2.2 Headers 5](#_Toc485978929)

[2.3 Body 6](#_Toc485978930)

[3. CORS 8](#_Toc485978931)

## 1. Introducción

El paradigma de la web actual ha cambiado mucho respecto a lo que encontrábamos años atrás. Es muy común encontrar que las páginas no contienen toda la información que necesitan para pintarse, si no que deben obtenerla mediante peticiones a un servidor realizadas mediante JavaScript.

A la interfaz que nos da un acceso limitado a la base de datos de un servicio web, evitando que se conozca o acceda al propio código fuente de la aplicación original se le conoce comúnmente como API (Aplication Programming Interface).

## 2. Accediendo a una API

### 2.1 Peticiones AJAX con XMLHttpRequest

Para realizar una petición a una API mediante JavaScript debemos usar crear un nuevo objeto de tipo XMLHttpRequest.

Veamos un ejemplo:

var httpRequest = new XMLHttpRequest();

De esta manera hemos creado el objeto, ahora para trabajar con él debemos indicarle algunos parámetros:

function makeRequest() {

httpRequest.onreadystatechange = handleResponse;

httpRequest.open('GET', 'https://ironhack-characters.herokuapp.com/characters');

httpRequest.send();

};

makeRequest();

Mediante esta función estamos indicando que cuando el estado de la petición cambie ejecute la función **handleResponse**, y que la petición debe ser de tipo GET y estamos indicando la URL de la API a la que debemos llamar.

Veamos un ejemplo de la función que maneja el cambio de estado en la petición:

function handleResponse(response) {

if (httpRequest.readyState === 4) {

if (httpRequest.status === 200) {

showCharacters(JSON.parse(httpRequest.responseText))

} else {

alert("There was an error");

}

}

}

### 2.2 Headers

En general una dirección web comienza con http://... Donde http significa “HyperText Transfer Protocol”. Este es un protocolo que usa todo internet para las comunicaciones desde los años 90.

Con este protocolo, la carga de una página web se basa en peticiones HTTP que envía el navegador al servidor y en respuestas HTTP que envía el servidor al navegador. Al cargar una página web se producen decenas de peticiones y respuestas HTTP cada una de las cuales lleva cabeceras (headers). Las cabeceras llevan información necesaria para la comunicación y pueden incluir diferentes aspectos como: tipo de navegador que realiza la petición, dirección de la página solicitada, juego de caracteres utilizado, etc.

Podemos encontrar dos tipos de cabeceras:

* De petición (request): Son las cabeceras que se envían cuando se realiza una llamada HTTP. En nuestro caso serán las cabeceras que mandemos desde nuestra aplicación a la API.

Para setear una header en la request haremos lo siguiente:

objetoXMLHttpRequest.setRequestHeader("nombre de la cabecera", "valor");

* De respuesta (response): Una respuesta del servidor consta de una línea de status que contiene alguna información básica sobre la respuesta del servidor, de diversas líneas que constituyen los headers y finalmente una línea en blanco seguida de la respuesta del servidor, normalmente un código HTML, XML o JSON extenso.

Para acceder a una cabecera de respuesta deberemos hacer lo siguiente:

var myHeader = objetoXMLHttpRequest.getResponseHeader("nombre de la cabecera");

### 2.3 Body

El body contiene el grueso de la información que se transfiere durante la petición. Del mismo que sucede con los headers, podemos encontrar dos tipos de body:

* De petición (request): Contiene los datos que enviamos a la API, por ejemplo, para realizar la creación de una cuenta de un usuario, debemos mandar los datos de este.

Ejemplo de petición seteando un body en el envío:

// Creamos el objeto

var xhr = new XMLHttpRequest();

xhr.open("POST", '/users', true);

// Seteamos las cabeceras necesarias

xhr.setRequestHeader("Content-type", "application/json");

// Seteamos la función que queremos que se ejecute al finalizar la petición

xhr.onreadystatechange = function() {

if (xhr.readyState == XMLHttpRequest.DONE && xhr.status == 200) {

// la petición ha terminado

// podemos manejar aquí el resultado

}

}

// Seteamos el body

xhr.send("{name: 'Fran Linde'}");

* De respuesta (response): Una respuesta del servidor deberá constar de un body. Para la mayoría de nuestros accesos a una API el body contendrá un JSON.

Veamos un ejemplo de cómo acceder al body de respuesta:

// Seteamos la url a la que hacer la petición

var url = 'somePage.html';

function load(url, callback) {

// Creamos el obbjeto

var xhr = new XMLHttpRequest();

// Seteamos función que maneja la respuesta

xhr.onreadystatechange = function() {

if (xhr.readyState === 4) {

// accedemos a la respuesta

console.log(xhr.response);

}

}

xhr.open('GET', url, true);

xhr.send('');

}

## 3. CORS

Un recurso hace una solicitud HTTP de origen cruzado (CORS) cuando solicita otro recurso de un dominio distinto al que pertenece. Por ejemplo, una página HTML localizada en <http://domain-a.com> hace una solicitud <img src a http://domain-b.com/image.jpg. Actualmente muchas páginas en la web cargan recursos como hojas de estilos CSS, imágenes y scripts de dominios separados.

Por razones de seguridad, los exploradores restringen las solicitudes HTTP de origen cruzado iniciadas dentro de un script. Por ejemplo, XMLHttpRequest sigue la política de mismo-origen. Por lo que, una aplicación usando XMLHttpRequest solo puede hacer solicitudes HTTP a su propio dominio. Para mejorar las aplicaciones web, los desarrolladores pidieron a los proveedores de navegadores que permitieran a XMLHttpRequest realizar solicitudes de dominio cruzado.

Para permitir las peticiones de orígenes cruzados existe una cabecera especial que debe ser seteada por el servidor en la repuesta: “Access-Control-Allow-Origin”. Esta cabecera deberá contener la URL de origen de la petición o al menos una regla que permita dicha URL.

No es raro encontrar peticiones cuya cabecera es:

Access-Control-Allow-Origin: \*

Esto está indicando al navegador que puede hacer peticiones a esta URL sea cual sea la URL, de origen. Por motivos de seguridad no es muy correcto. Lo más apropiado es que solo se permitan las URL’s que hayamos identificado que van a realizar peticiones JavaScript.