educación IT
CODE YOUR FUTURE

# Bootcamp Java Developer

## Fase 2 - Java Web Developer

Módulo 19

educación IT
CODE YOUR FUTURE

# Introducción a AJAX

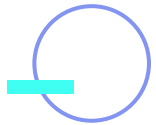
# AJAX

## ¿Qué es y para qué sirve?

**JavaScript Asíncrono y XML (AJAX)** no es una tecnología por sí misma, es un término que describe un **nuevo modo de utilizar conjuntamente varias tecnologías existentes**.

Esto incluye: HTML o XHTML, CSS, JavaScript, DOM, XML, XSLT, y el objeto **`XMLHttpRequest`**.

Cuando estas tecnologías se combinan en un **modelo AJAX**, es posible lograr **aplicaciones web capaces de actualizarse continuamente sin tener que volver a cargar la página completa**. Esto crea aplicaciones más rápidas y con mejor respuesta a las acciones del usuario.



# API XMLHttpRequest

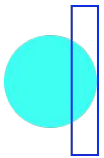
Es un objeto JavaScript que fue diseñado por Microsoft y adoptado por Mozilla, Apple y Google. Actualmente, es un estándar de la W3C.

Proporciona una **forma fácil de obtener información de una URL sin tener que recargar la página completa**.

A pesar de su nombre, **`XMLHttpRequest` puede ser usado para recibir cualquier tipo de dato, no sólo XML,** y admite otros formatos además de HTTP (incluyendo File y FTP).

Para crear una instancia de **`XMLHttpRequest`**, debes hacer lo siguiente:

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



Las versiones viejas de Internet Explorer pueden tener una implementación anterior en su JScript por lo que si queremos **tener compatibilidad** para esos navegadores deberíamos hacer:

```
var xhr ;
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
    xhr = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
    xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

El **objeto que se obtiene** a partir de la inicialización de la API XHR contiene propiedades y métodos que nos permiten establecer **comunicación asincrónica con otra máquina a través del protocolo HTTP**.

# ReadyState

Como podemos observar, si mostramos el objeto que acabamos de crear en la consola, notamos que todas sus propiedades se encuentran vacías en **`null`, `undefined`, `0` (cero) o “”(string vacío)**. Esto es porque aún no lo configuramos para que pueda realmente enviar la petición.

De todas las propiedades que se muestran, la que sí nos está dando información es la propiedad **`XHR.readyState`**, que **indica el estado** en el que se encuentra una solicitud y puede tomar los valores que veremos en la tabla de la siguiente diapositiva.

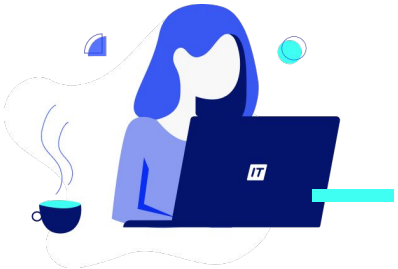
## Valores posibles

| Estado | Descripción |
|---|---|
| `0 - UNSENT` | Objeto Inicializado. |
| `1 - OPENED` | Objeto configurado. |
| `2 - HEADERS_RECEIVED` | El objeto ya se envió y el status de los headers del servidor ya volvieron. |
| `3 - LOADING` | Descargando. La propiedad `responseText` mantiene datos parciales. |
| `4 - DONE` | La operación se completó no necesariamente exitosa. |

# Open

**Permite configurar una solicitud saliente ya inicializada o reconfigurar una que ya se encontraba configurada.**

```
XMLHttpRequest.open(String metodo,
String url[, Boolean async])
```



Recibe como parámetros obligatorios el método HTTP por el cual vamos a enviar la solicitud, y la URL a la que la vamos a enviar. Como último parámetro opcional recibe un booleano que determina si la solicitud va a ser sincrónica o asincrónica.

Configurar un objeto XHR no implica que ya lo hayamos enviado, sino que está listo para poder enviarse a futuro.

# ReadyStateChange

Todos los objetos que extiendan de la API XHR tienen este evento que se dispara cada vez que la propiedad **`readyState`** cambia. Con él podemos tener control total sobre cada uno de los estados del pedido:

```
var xhr = new XMLHttpRequest()
xhr.addEventListener("readystatechange",function(){
    console.log(xhr.readyState)
})
xhr.open("GET","url.com")
```

Como podemos observar, aparece en consola el mensaje “1” ya que, en la primera línea de código, teníamos el estado 0(cero). Pero, luego de la asignación, configuramos el evento, por lo tanto, cambió su estado a 1(uno).

Eventualmente, nos vamos a enterar cuando pasen otras cosas durante el transcurso del pedido. El mismo evento va a ejecutar su *callback* cada vez que el objeto XHR cambie de estado.

Este es un buen lugar para realizar comprobaciones de peso y tipo de recurso solicitado, en el caso de que se haya solicitado alguno. Por ejemplo, un archivo de texto del que conocemos su peso.

Podríamos consultar qué tamaño tiene antes de siquiera descargarlo o consultar de qué tipo de archivo estamos hablando. Estas y más informaciónes o, mejor dicho, meta-información está disponible para nosotros a partir del estado 2(dos) en donde ya tenemos los *headers* de la respuesta del servidor.

Asumimos, por un momento, que alguno de los *headers* no nos gusta, no eran lo que estábamos esperando a cambio. En tal caso, estamos a tiempo de abortar la solicitud con el método **`abort()`**. Se debe tener en cuenta que **este método automáticamente reinicia el objeto XHR y perderemos toda configuración hecha hasta el momento.**

# Load

Este evento se dispara cuando la propiedad **`readyState`** es igual a 4. Hay que recordar nuevamente que esto no implica que el pedido haya sido exitoso por lo que aún tenemos que realizar una mínima comprobación para saber su **`status`**.

Este es un buen momento para revisar la respuesta que obtuvimos ya que pasamos por todos los estados, entonces deberíamos tener el contenido del recurso que fuimos a pedir.

```
var xhr = new XMLHttpRequest()
xhr.open("GET","miArchivo.ext")
xhr.addEventListener("load",function(){
    if (xhr.status == 200) {
        //...
    }
})
```

# Response

La propiedad **`response`** de un objeto XHR se va a completar cuando el pedido haya sido despachado y una vez que se haya descargado la información necesaria, que puede estar codificada en varios formatos: **`JSON`, `Document`, `Blob`, `ArrayBuffer` y `DOMString` dependiendo de cómo configuremos la propiedad `responseType`.**

# Send

Este método permite **enviar el pedido** una vez que ya lo hayamos configurado. También, enviar parámetros en su interior como argumentos siempre y cuando la solicitud se haya realizado por el método **POST**.

```
var xhr = new XMLHttpRequest()
 xhr.addEventListener("readystatechange",()=>{
       console.log(xhr.readyState)
   })
   xhr.addEventListener("load",function(){
       if (xhr.status == 200) {
           console.log(xhr.response)
       }
   })
   xhr.open("GET","url.com")
   xhr.send()
```

# ¡Sigamos trabajando!