



# UD7: Comunicación asíncrona con AJAX

Antonio Sierra

Basado en el trabajo de Javier G. Pisano



Desarrollo Web en Entorno Cliente (DWEC)

Curso 2024-2025

# Índice

- ▶ Introducción a AJAX
- ▶ El objeto XMLHttpRequest
- ▶ Peticiones AJAX
- ▶ Recepción de datos
- ▶ Otros aspectos
- ▶ Geolocalización [HTML5]





# Introducción a AJAX

UD7: Comunicación asíncrona con AJAX



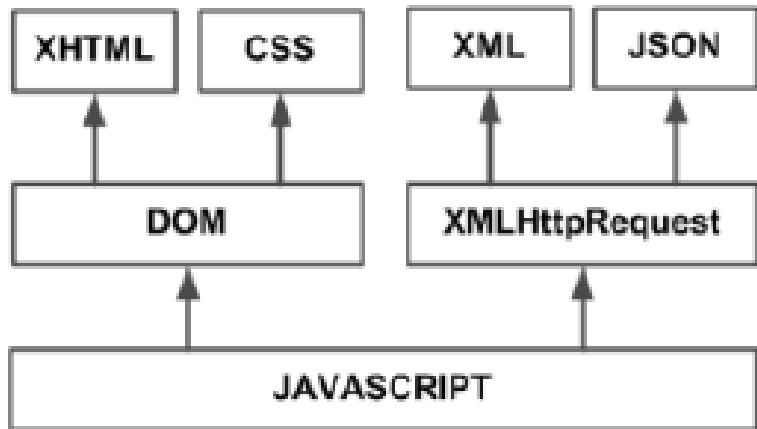
Desarrollo Web en Entorno Cliente (DWEC)

Curso 2024-2025

# AJAX

## (Asynchronous Javascript and XML)

- ▶ Es una tecnología que nos va a permitir comunicar cliente con servidor en un segundo plano (sin tener que esperar por la respuesta)



Será necesario tener un servidor Web al que haremos las peticiones



# Petición HTTP ‘clásica’

- ▶ El cliente realiza la solicitud al servidor
- ▶ El servidor ejecuta el proceso asociado
  - ▶ Procesa información
  - ▶ Consulta en la BBDD
  - ▶ Obtiene resultados
  - ▶ ...
- ▶ El servidor devuelve los resultados como un recurso HTML

Mientras el servidor trabaja, el cliente debe esperar por respuesta.



# Petición clásica vs AJAX

## SIN AJAX

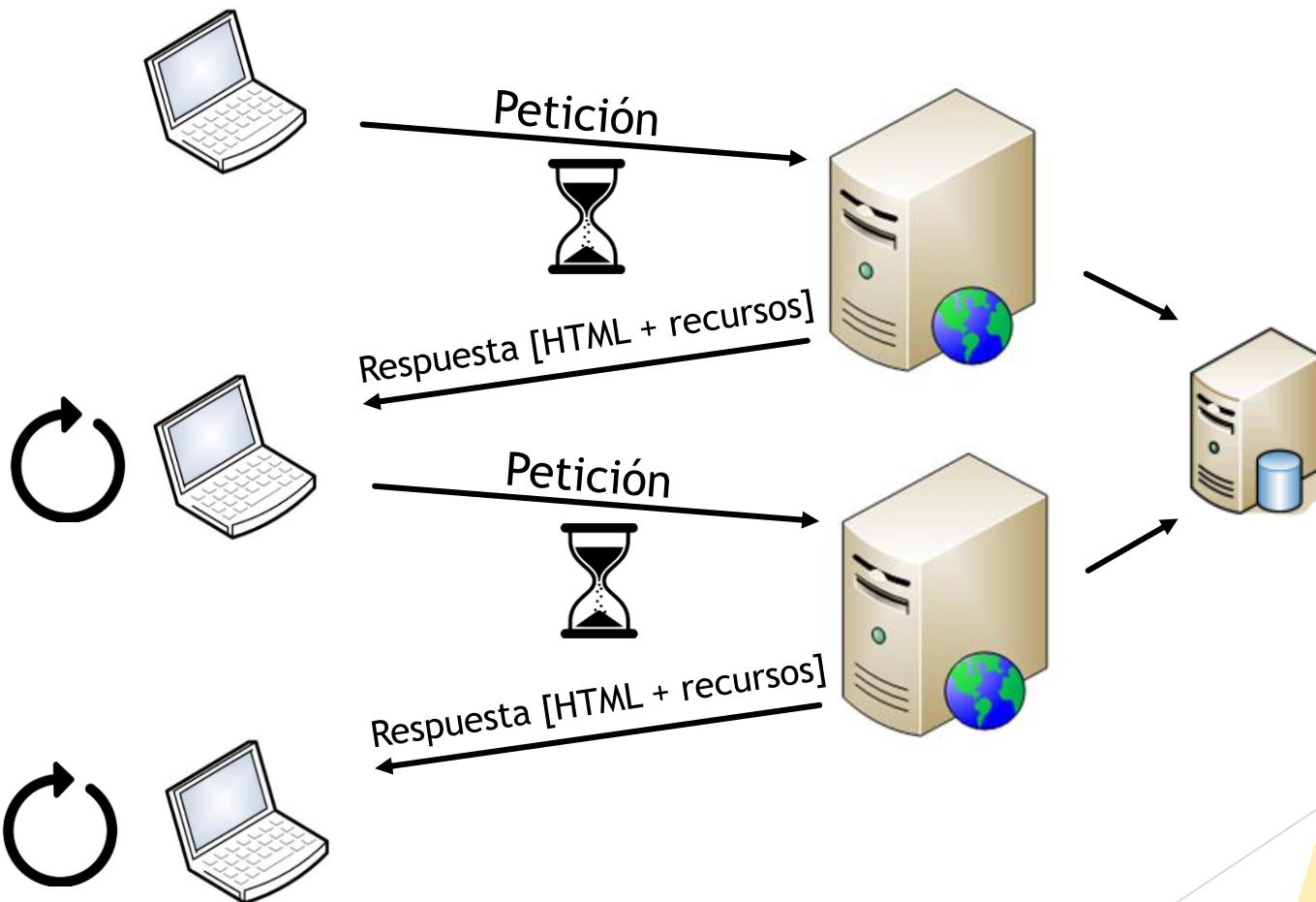
- ▶ El cliente hace solicitud HTTP al servidor
- ▶ El cliente espera hasta recibir la respuesta (cambio de página)
- ▶ El servidor realiza la acción
- ▶ El servidor devuelve la página al cliente (recarga)

## CON AJAX

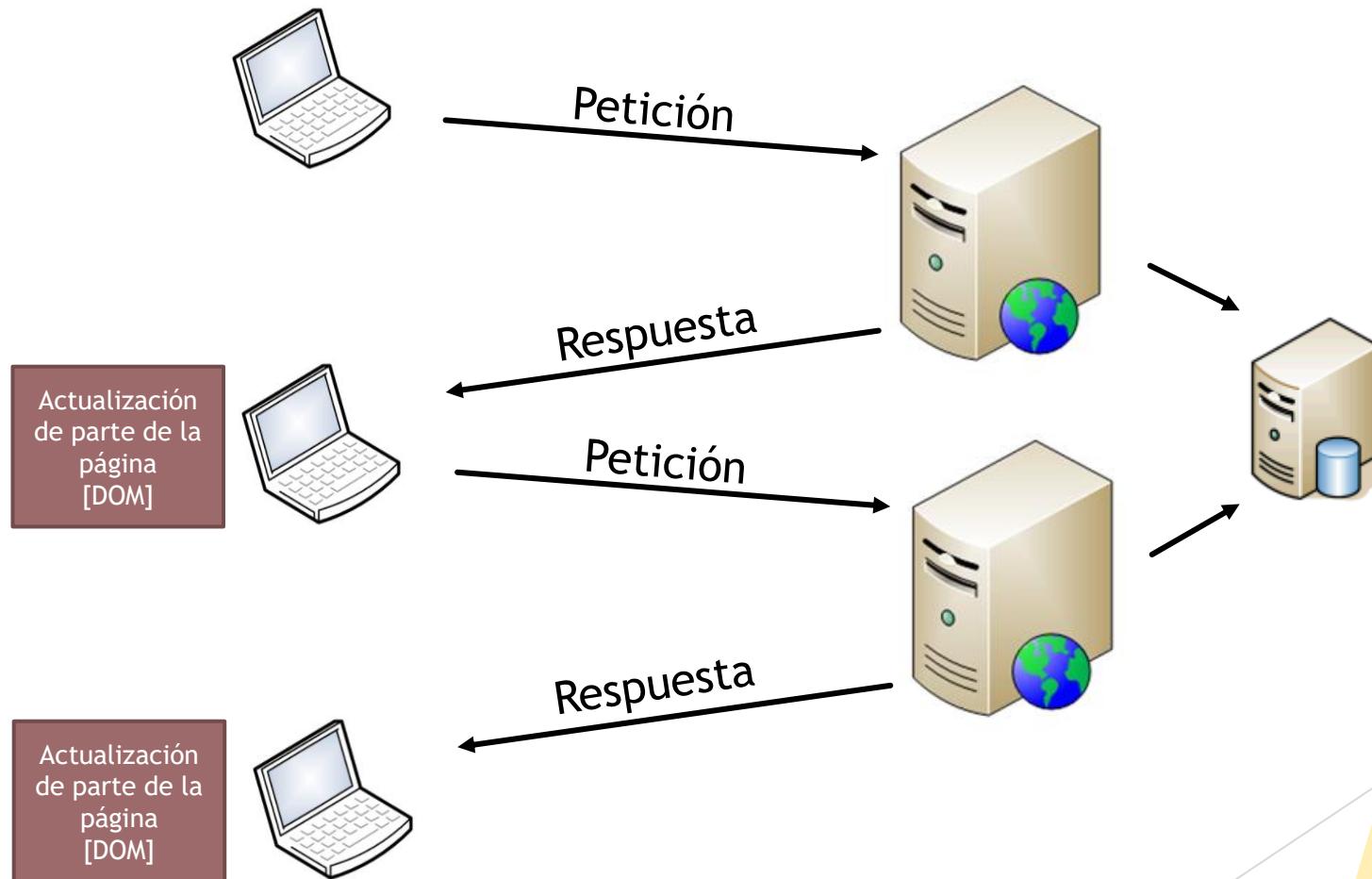
- ▶ El cliente hace solicitud HTTP al servidor
- ▶ El cliente sigue interaccionando con la página.
- ▶ El servidor realiza la acción
- ▶ Los resultados se muestran sin necesidad de recargar la página.



# Petición HTTP ‘clásica’



# Petición HTTP AJAX





# El objeto XMLHttpRequest

UD7: Comunicación asíncrona con AJAX



Desarrollo Web en Entorno Cliente (DWEC)

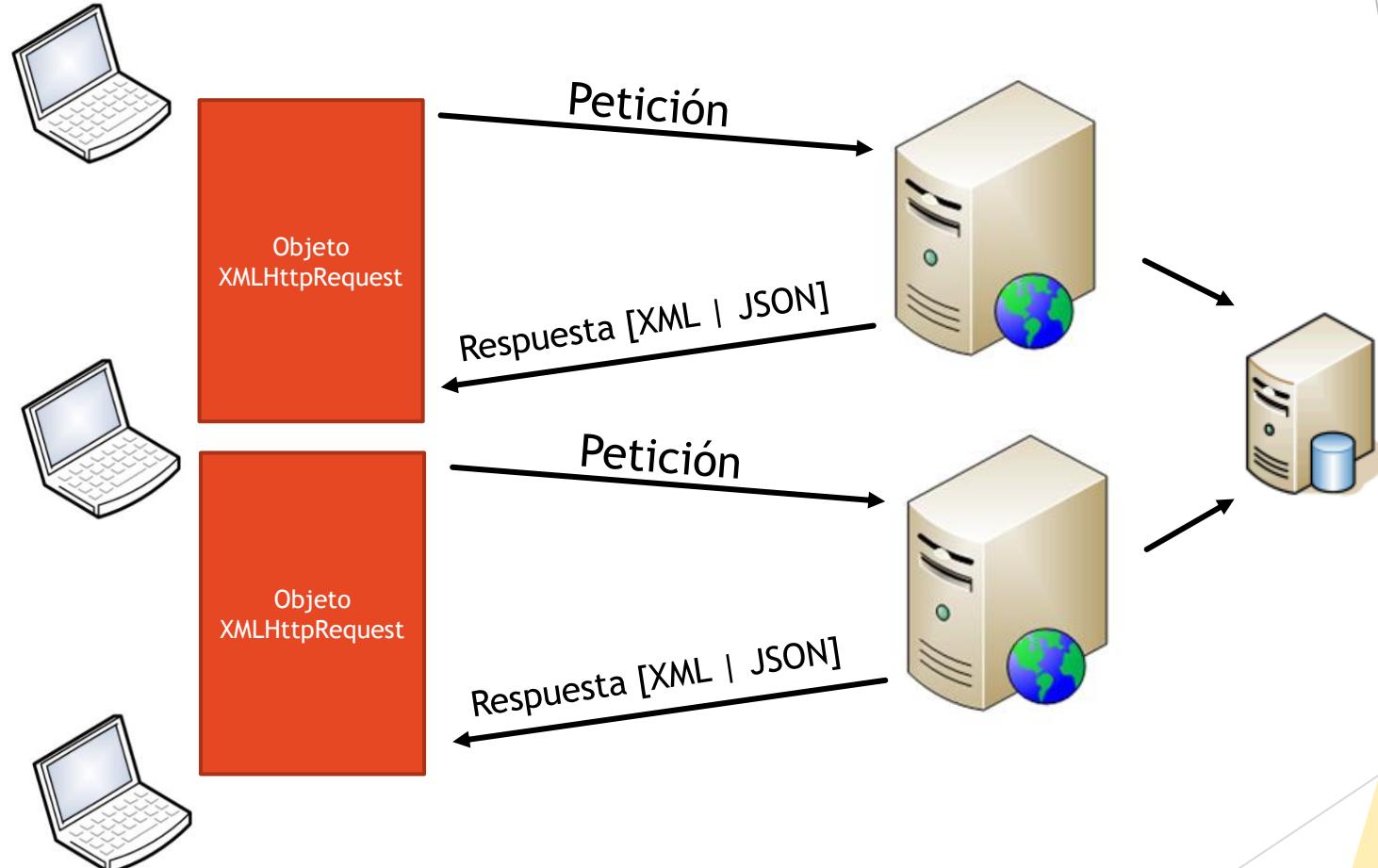
Curso 2024-2025

# Objeto XMLHttpRequest

- ▶ Objeto integrado en la especificación ECMAScript.
- ▶ Se utiliza para realizar y procesar las peticiones HTTP directamente al servidor Web
  - ▶ Es un “intermediario” entre el cliente y el servidor



# Petición HTTP AJAX



# Objeto XMLHttpRequest

## Características

- ▶ Permite hacer tanto llamadas **síncronas** (espera por respuesta) como **asíncronas** (no espero por respuesta), si bien lo habitual es lo segundo.
- ▶ Por **seguridad**, solo deja hacer llamadas a páginas que se encuentran hospedadas en el mismo dominio desde el cual se realiza la petición AJAX



# Objeto XMLHttpRequest

## Propiedades

<b>readyState</b>	Valor numérico que almacena el estado de la petición
<b>responseText</b>	Contenido de la respuesta del servidor en formato de texto
<b>responseXML</b>	Contenido de la respuesta del servidor en formato XML
<b>status</b>	<a href="#">Código de estado HTTP</a> devuelto por el servidor
<b>statusText</b>	<a href="#">Código de estado HTTP</a> devuelto por el servidor [cadena]

200: OK  
404: No encontrado  
500: Error de servidor

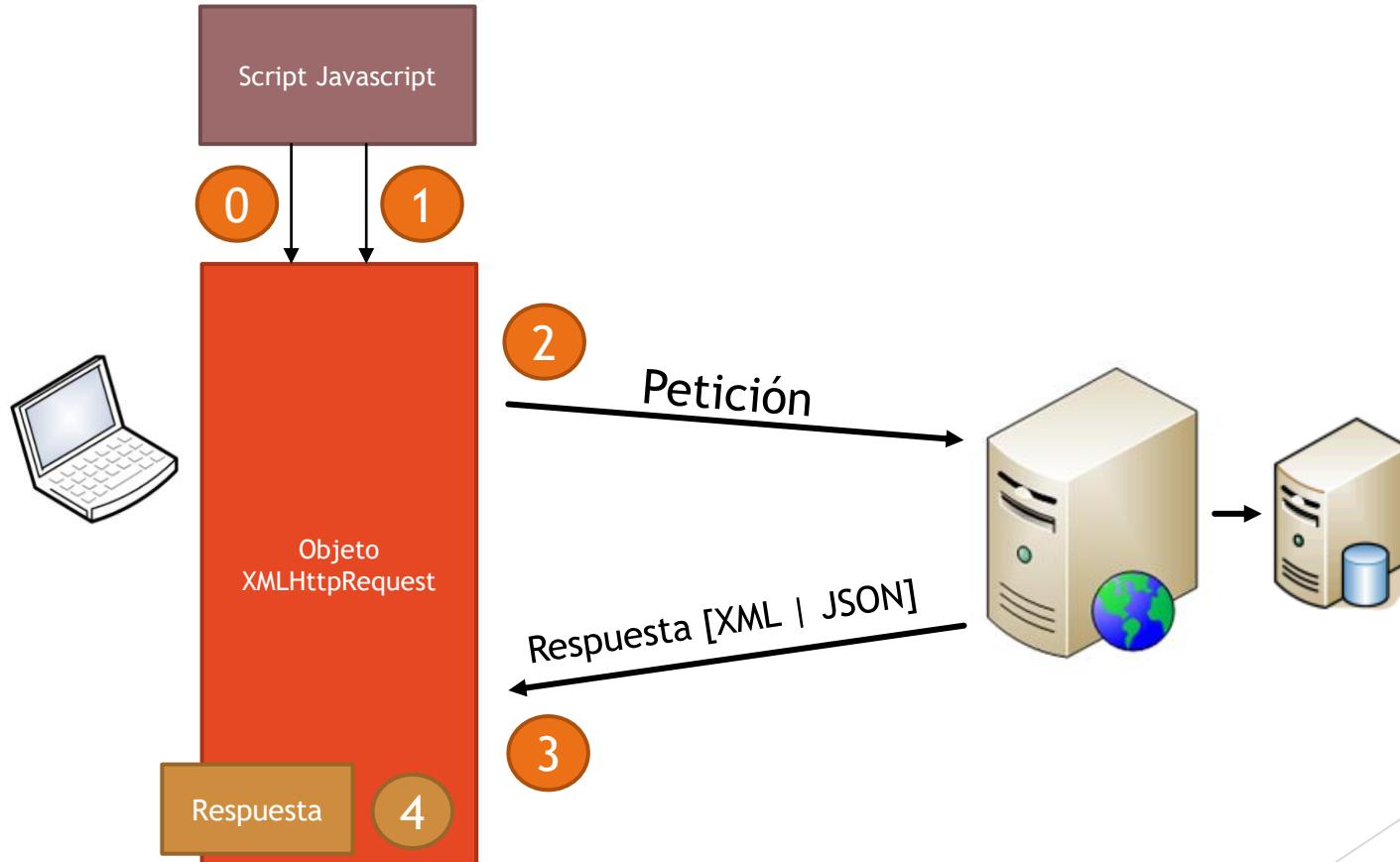


# Objeto XMLHttpRequest readyState

<b>0</b>	<b>No inicializado</b>	Objeto creado, no se ha invocado el método <b>open()</b>
<b>1</b>	<b>Cargando</b>	Objeto creado, no se ha invocado el método <b>send()</b>
<b>2</b>	<b>Cargado</b>	Se ha invocado el método <b>send()</b> pero el servidor aún no ha respondido).
<b>3</b>	<b>Interactivo</b>	Se han recibido algunos datos pero aún no podemos acceder a la respuesta).
<b>4</b>	<b>Completo</b>	<u>Se han recibido todos los datos de la respuesta del servidor</u>



# Valores de la propiedad readyState



# Objeto XMLHttpRequest

## Métodos

```
open(metodo,  
      url,  
      [asincrona],  
      [usuario],  
      [password])
```

Realiza la petición HTTP al servidor. Parámetros:

- **metodo** (cadena). Método HTTP utilizado para hacer la petición (**GET, POST**).
- **url** (cadena). URL del servidor al cual realizo la petición.
- **asincrona** (booleano) [opcional]. Booleano que indica si la petición es asíncrona (**true**) o síncrona (**false**). Por lo general nos interesa que las peticiones sean asíncronas.
- **usuario** (cadena) [opcional]. Credenciales si fueran necesarias.
- **password** (cadena) [opcional]. Credenciales si fueran necesarias.



# Objeto XMLHttpRequest

## Métodos

### **send(parametros)**

Indica los parámetros a usar en la petición POST.

**parametros** puede ser una cadena, un array de bytes o un XML.

Habitualmente una cadena de la forma  
**parametro1=valor1&parametro2:valor2...**

Si hacemos una petición GET o no queremos enviar parámetros, contiene el valor **null**

Si hacemos una petición GET y queremos enviar parámetros, irán codificados en la URL usando el formato:

**http://destino.php?parametro=valor1&parametro2=valor2**



# Objeto XMLHttpRequest

## Métodos

### **onreadystatechange**

En realidad no es un método, sino una propiedad a la que podemos asignar una función que se invocará bajo ciertas condiciones (*callback*).

Se invoca cada vez que se produce un cambio en el estado de la petición (cambia el valor de la propiedad **readyState**). Habitualmente nos interesará tratar la petición cuando se ha completado (estado 4).



# Objeto XMLHttpRequest

## Métodos

<b>abort()</b>	Detiene la petición actual
<b>getAllResponseHeaders()</b>	Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor
<b>getResponseHeader (cabecera)</b>	Devuelve una cadena de texto con el contenido de la cabecera solicitada en la cabecera <b>cabecera</b>
<b>setRequestHeader (cabecera, valor)</b>	Permite establecer cabeceras personalizadas en la petición HTTP. Se debe invocar el método <b>open()</b> previamente.
	Si enviamos métodos por POST, es obligatorio indicar el tipo de contenido que vamos a enviar, pues los datos POST forman parte de la cabecera

```
setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
```





# Peticiones AJAX

UD7: Comunicación asíncrona con AJAX



Desarrollo Web en Entorno Cliente (DWEC)

Curso 2020-2021

# Pasos en una petición AJAX

- ▶ Instanciar el objeto XMLHttpRequest
- ▶ Preparar la función de respuesta
  - ▶ Escribimos la función de callback que asignamos a la propiedad **onreadystatechange**
- ▶ Realizar la petición al servidor y/o enviamos parámetros
- ▶ Se ejecuta la función de respuesta
  - ▶ Comprobamos si se ha recibido respuesta (readyState es 4) y si es correcta (status es 200).
  - ▶ Una vez realizadas las comprobaciones, accedemos a la respuesta del servidor mediante **responseText** ó **responseXML**



# GET sin parámetros

## Ejemplo

JS

```
var miXHR=new XMLHttpRequest();  
  
miXHR.onreadystatechange=miXHRcambiaEstado;  
miXHR.open("GET","servidor/peticion_get.php");  
miXHR.send(null);  
  
function miXHRcambiaEstado(){  
    if((this.readyState==4) && (this.status==200)){  
        console.log(this.responseText);}}  
4
```



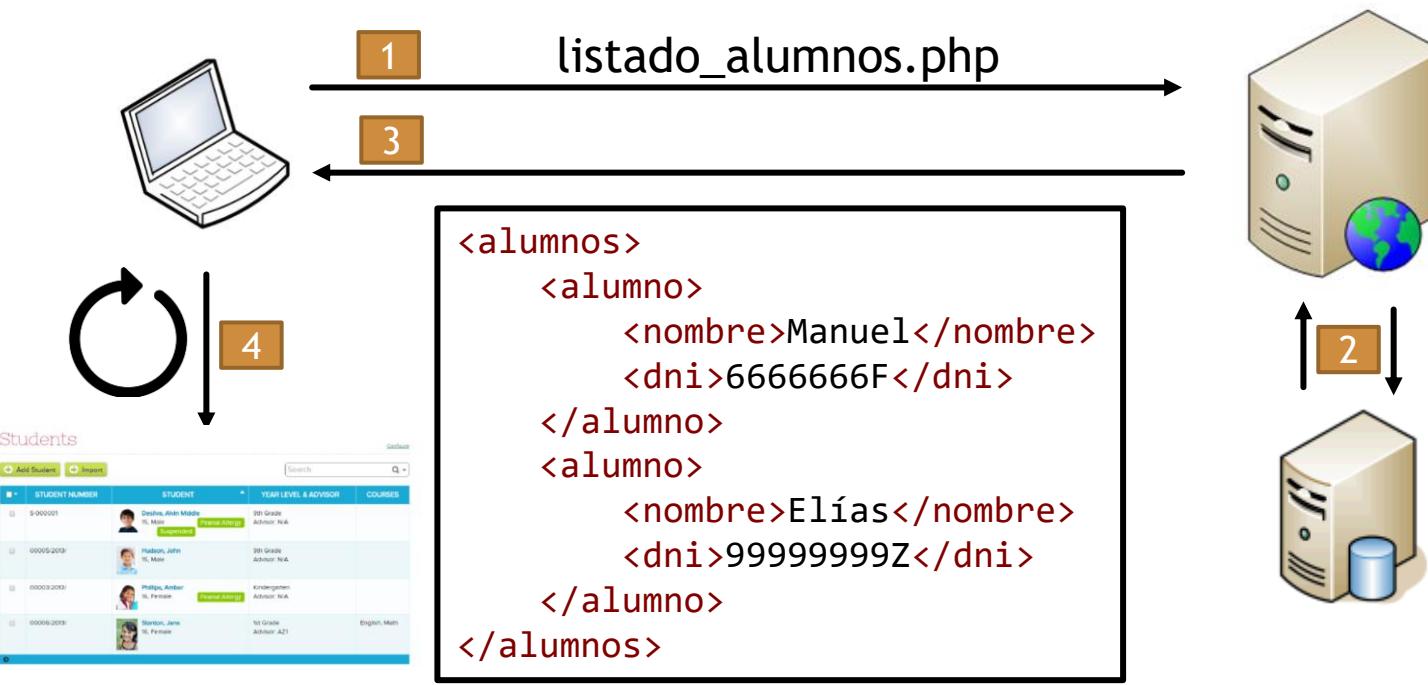
PHP

```
<?php  
    echo "Esta es mi respuesta";  
?>
```



# GET sin parámetros

## Caso de uso



# GET con parámetros

## Ejemplo

JS

```
var miXHR=new XMLHttpRequest();

miXHR.onreadystatechange=miXHRCambiaEstado;
miXHR.open("GET","servidor/peticion_get_parametros.php?nombre=Federico&apellidos=Gomez Perez");
miXHR.send(null);

function miXHRCambiaEstado(){
    if((this.readyState==4)&&(this.status==200)){
        console.log(this.responseText);}}
```



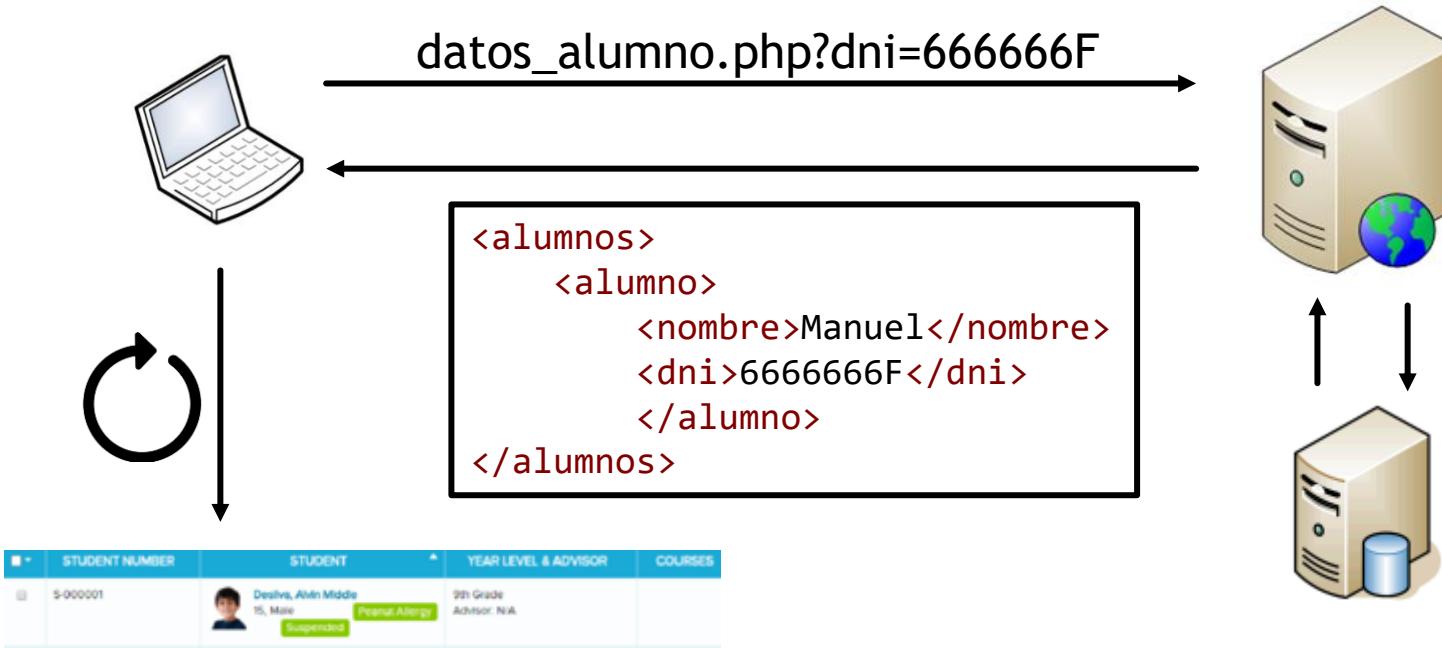
PHP

```
<?php
if((isset($_GET["nombre"]))&&(isset($_GET["apellidos"]))){
    echo "Hola ".$_GET["nombre"]." ".$_GET["apellidos"];
}
else{
    echo "Hola anonimo";
?>
```



# GET con parámetros

## Caso de uso



# POST

## Ejemplo

JS

```
var miXHR=new XMLHttpRequest();
miXHR.onreadystatechange=miXHRCambiarEstado;

miXHR.open("POST","servidor/peticion_post.php");
miXHR.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

var parametros="nombre=Federico&apellidos=Gomez Pérez";
miXHR.send(parametros);

function miXHRCambiarEstado(){
    if((this.readyState==4) && (this.status==200)){
        console.log(this.responseText);}}
```



PHP

```
<?php
    if((isset($_POST["nombre"]))&&(isset($_POST["apellidos"]))){
        echo "Hola ".$_POST["nombre"]." ".$_POST["apellidos"];
    }
    else{
        echo "Hola anonimo";
    }
?>
```



# GET con parámetros (Refactorizado)

## Ejemplo

```
var parametros={nombre:"Federico",apellidos:"Gomez Perez"};
realizaPeticionGET("servidor/peticion_get_parametros.php",parametros);

function realizaPeticionGET(url,parametros){
    var miXHR=new XMLHttpRequest();
    miXHR.onreadystatechange=miXHRCambiaEstado;

    if(parametros!==undefined){
        var claves=Object.keys(parametros);
        for (var i = 0; i < claves.length; i++) {
            var clave=claves[i];
            if(i==0)
                url+="?"+clave+"="+parametros[clave];
            else
                url+="&"+clave+"="+parametros[clave];
        }
        miXHR.open("GET",url);
        miXHR.send(null);

        function miXHRCambiaEstado(){
            if((this.readyState==4) && (this.status==200)){
                console.log(this.responseText);
            }
        }
    }
}
```



# POST con parámetros (Refactorizado)

## Ejemplo

```
var parametros={nombre:"Federico",apellidos:"Gomez Perez"};
realizaPeticionPOST("servidor/peticion_post.php",parametros);

function realizaPeticionPOST(url,parametros){
    var miXHR=new XMLHttpRequest();
    miXHR.onreadystatechange=miXHRCambiaEstado;

    var parametrosCadena=null;

    if(parametros!==undefined){
        parametrosCadena="";
        var claves=Object.keys(parametros);
        for (var i = 0; i < claves.length; i++) {
            var clave=claves[i];
            if(i==0)
                parametrosCadena+=clave+"="+parametros[clave];
            else
                parametrosCadena+="&"+clave+"="+parametros[clave];
        }
    }
    miXHR.open("POST",url);
    miXHR.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

    miXHR.send(parametrosCadena);

    function miXHRCambiaEstado(){
        if((this.readyState==4)&&(this.status==200)){
            console.log(this.responseText);
        }
    }
}
```



# Haciendo pruebas

- ▶ En ocasiones nos interesa comprobar los scripts PHP antes de comenzar a desarrollar las peticiones AJAX:
  - ▶ Comprobar tipo de parámetros esperados
  - ▶ Comprobar respuesta proporcionada
- ▶ Postman es una plataforma que nos permite realizar peticiones GET/POST y comprobar las respuestas.
  - ▶ Se puede instalar como una extensión más de Chrome aunque está deprecado y se recomienda el uso de las apps nativas. El funcionamiento es similar.



# Peticiones GET con Postman

- ▶ Introducimos la URL y escogemos como método de envío GET
  - ▶ Si fuera necesario indicamos los parámetros correspondientes
  - ▶ Consultamos la respuesta en la parte inferior

The screenshot shows the Postman application interface. At the top, there's a header with 'Launchpad' and 'GET Basic Auth'. Below it, a search bar says 'Basic Auth'. The main area has tabs for 'Params' (which is selected), 'Authorization', 'Headers (9)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. On the right, there are buttons for 'Comments (0)', 'Examples (1)', 'Send', and 'Save'. Under 'Params', there's a table for 'Query Params' with columns 'KEY', 'VALUE', and 'DESCRIPTION'. One row shows 'Key' and 'Value' but lacks a description. Below this, there are tabs for 'Body', 'Cookies', 'Headers (9)', and 'Test Results (1/2)'. The 'Body' tab is selected, showing a status of '200 OK', a time of '11ms', and a size of '26.2 KB'. A 'Save Response' button is also present. The 'Body' content is displayed as JSON code:

```
1 {
2     "codigo": "1000",
3     "apellidos": "Montes Barquín",
4     "nombre": "Andrés",
5     "fecha_nacimiento": "1996-11-15",
6     "curso": "ESO4",
7     "nota_media": "1"
8 },
9 {
10     "codigo": "1001",
11     "apellidos": "Sierra Minguez",
12     "nombre": "María",
13     "fecha_nacimiento": "1996-06-08".
14 }
```



# Peticiones GET con Postman

- ▶ Introducimos la URL y escogemos como método de envío POST
    - ▶ Los parámetros se indican en body. Pulsamos x-www.form-urlencoded e introducimos los mismos si los hay.

The screenshot shows the Postman application interface. At the top, there's a header bar with 'Launchpad' on the left, 'POST Basic Auth' in the center, and 'No Environment' on the right. Below the header, a navigation bar has 'Basic Auth' selected. On the far right of the navigation bar is a red 'Comments (0)' button. The main workspace shows a POST request to 'http://127.0.0.1:8080/ejemploJS/alumnos\_json.php'. To the right of the URL is a blue 'Send' button. Below the URL, tabs for 'Params', 'Authorization', 'Headers (10)', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible, with 'Params' being the active tab. Under 'Query Params', there's a table with one row containing 'Key' and 'Value' columns, both labeled 'Description'. At the bottom, tabs for 'Body', 'Cookies', and 'Headers (9)' are shown, along with 'Test Results (1/2)'. To the right of these tabs, status information is displayed: 'Status: 200 OK', 'Time: 11ms', 'Size: 26.2 KB', and a partially visible 'Save' button. Below the status is a toolbar with 'Pretty', 'Raw', 'Preview', 'Visualize BETA', and 'JSON' dropdown buttons, followed by a copy icon. The 'Body' section contains a JSON response with student details:

```
1
2
3     {
4         "codigo": "1000",
5         "apellidos": "Montes Barquín",
6         "nombre": "Andrés",
7         "fecha_nacimiento": "1996-11-15",
8         "curso": "ESO4",
9         "nota_media": "1"
10    },
```



# Recepción de datos

UD7: Comunicación asíncrona con AJAX



Desarrollo Web en Entorno Cliente (DWEC)

Curso 2020-2021

# Procesando respuestas

- ▶ Si bien las respuestas pueden ir en texto plano, lo habitual es que las respuestas del servidor vayan en uno de los siguientes formatos:
  - ▶ XML
    - ▶ Tecnología soportada por la mayor parte de lenguajes y madura (usada en un gran número de aplicaciones).
  - ▶ JSON
    - ▶ Más compacto y ligero
    - ▶ Más fácil de procesar en el navegador.



# Procesando respuestas con JSON

- ▶ Servidor
  - ▶ Establecemos la cabecera especificando que el tipo de contenido será JSON
    - ▶ `header("Content-Type: application/json");`
  - ▶ Desde PHP convertimos el dato que queremos enviar usando la función `json_encode(dato)`
    - ▶ `dato` puede ser de cualquier tipo salvo `resource`.
- ▶ Cliente
  - ▶ En `responseText` recibimos la cadena JSON
  - ▶ Podemos convertir la cadena a objeto con `JSON.parse()`



# Procesando respuestas con JSON

## Ejemplo

JS

```
var miXHR=new XMLHttpRequest();
miXHR.onreadystatechange=miXHRcambiaEstado;
miXHR.open("GET","servidor/peticion_get_respuesta_json.php")
;
miXHR.setRequestHeader("Content-type",
                      "application/x-www-form-urlencoded");
miXHR.send(null);

function miXHRcambiaEstado(){
  if((this.readyState==4) && (this.status==200)){
    respuesta=JSON.parse(this.responseText);
    console.log(respuesta);
  }
}
```



PHP

```
<?php
$cadena="Este es un ejemplo de cadena";
echo json_encode($cadena);
?>
```



# json\_encode()

## Ejemplos

\$dato	json_encode(\$dato)
"cadena" true 5	""cadena"" "true" "5"
array ("Manuel", '&Pepe&')	[ "Manuel", "&Pepe&" ]
Array ('color'=>'azul', 'talla'=>'L')	{"color": "azul", "talla": "L"}
class Dato{ public \$nombre; public \$edad; }	{"nombre": "J", "edad": 30}

Los arrays PHP no asociativos se devuelven como arrays  
Los arrays PHP asociativos se devuelven como objetos  
Los objetos PHP se devuelven como objetos



# Procesando respuestas con XML

- ▶ Servidor
  - ▶ Establecemos la cabecera especificando que el tipo de contenido será XML
    - ▶ `header("Content-Type: text/xml");`
  - ▶ Componemos la respuesta XML
    - ▶ Importante indicar en la primera línea la versión y codificación XML
      - ▶ `<?xml version=\"1.0\" encoding=\"utf-8\"?>`
- ▶ Cliente
  - ▶ En `responseXML` recibimos el documento XML.
    - ▶ Se trata de un objeto que podemos navegar usando los mismos métodos que usábamos para navegar por el DOM
    - ▶ Fundamentalmente usaremos el método `getElementsByTagName(etiqueta)`



# Procesando respuestas con XML

## Ejemplo

JS

```
var miXHR=new XMLHttpRequest();
miXHR.onreadystatechange=miXHRCambieEstado;
miXHR.open("GET","servidor/peticion_get_respuesta_xml.php");
miXHR.setRequestHeader("Content-type",
                      "application/x-www-form-urlencoded");
miXHR.send(null);

function miXHRCambieEstado(){
    if((this.readyState==4) && (this.status==200)){
        var respuesta=this.responseXML.getElementsByTagName("respuesta");
        var mensaje=respuesta[0].getElementsByTagName("mensaje");
        console.log(mensaje[0].firstChild.nodeValue);
    }
}
```



PHP

```
<?php
header("Content-Type: text/xml");
$respuestaXML=<?xml version="1.0\" encoding="utf-8"?>
<respuesta>
    <mensaje>Propicios días </mensaje>
</respuesta>;
echo $respuestaXML;
?>
```





# Otros aspectos

UD7: Comunicación asíncrona con AJAX



Desarrollo Web en Entorno Cliente (DWEC)

Curso 2020-2021

# Evitando el cacheado

- ▶ Un problema asociado a las peticiones AJAX puede ser la caché del navegador
  - ▶ Si repetimos las peticiones al mismo script y con los mismos parámetros, el navegador puede cachear el resultado
- ▶ Posibles soluciones:
  - ▶ Especificar en la cabecera HTTP (desde PHP) que no queremos cachear el resultado.
    - ▶ No es fiable al 100%
  - ▶ Añadir un parámetro aleatorio (llamado nocache) a todas las peticiones
    - ▶ Así cada petición varía en el valor de al menos uno de los parámetros.



# Procesando respuestas con XML

## Ejemplo

PHP

SOLUCIÓN 1

```
<?php  
    header('Cache-Control: no-cache, must-revalidate');  
    header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');  
...  
?>
```



JS

SOLUCIÓN 2

```
var parametros=  
    "parametro1=valor1&parametro2=valor2&nocache="  
    + Math.random();  
miXHR.send(parametros);
```



# Cancelando peticiones

- ▶ Aunque las peticiones se realizan de forma asíncrona en segundo plano, normalmente es necesario disponer de una respuesta rápida del servidor.
- ▶ La función `setTimeout()` se puede emplear a modo de temporizador
  - ▶ Si el servidor no responde y finaliza la cuenta atrás, se ejecuta una función encargada de abortar la petición, reintentarla, mostrar información al usuario, etc...



# Cancelando peticiones

## Ejemplo

```
var temporizadorCuentaAtras;
const TIEMPO=5000;

var miXHR=new XMLHttpRequest();
temporizadorCuentaAtras=setTimeout(funcionExpirada,TIEMPO,miXHR);

miXHR.onreadystatechange=miXHRcambiaEstado;
...
function miXHRcambiaEstado(){
    if((this.readyState==4) && (this.status==200)){
        clearTimeout(temporizadorCuentaAtras);
        console.log(this.responseText);
    }
}

function funcionExpirada(xhr){
    xhr.abort();
    console.error("ERROR de comunicación con el servidor");
}
```



# Codificación de caracteres

- ▶ Es posible que queramos enviar parámetros cuyos valores son “no estándar”.
- ▶ encodeURIComponent() reemplaza todos los caracteres que no podemos usar de manera directa por su representación hexadecimal.
  - ▶ Letras, números y ciertos caracteres (-\_.!|~\*'()) no se codifican.
- ▶ decodeURIComponent() realiza la conversión inversa.



# Seguridad

- ▶ Por cuestiones de seguridad, no es posible realizar peticiones AJAX:
  - ▶ A recursos ubicados en dominios distintos.
  - ▶ A recursos que usan protocolos distintos (http / https)
- ▶ Podemos establecer el valor de la propiedad `document.domain` para acceder desde un dominio a un subdominio.
- ▶ Por ejemplo:
  - ▶ `http://www.ejemplo.com/scripts/codigo1.js`
  - ▶ `http://scripts.ejemplo.com/codigo2.js`

Si desde ambos scripts se establece `document.domain="ejemplo.com"`  
los recursos de ambos scripts pueden interactuar entre sí



# Seguridad

- Si queremos acceder a un recurso ubicado en otro servidor tenemos dos opciones:
  - Añadir la cabecera Access-Control-Allow-Origin
    - No siempre es posible modificar las cabeceras.
  - Usar JSONP (JSON Padding)
    - Es una manera alternativa de tratar las respuestas que permite ejecutar código JS a medida





# Geolocalización [HTML5]

UD7: Comunicación asíncrona con AJAX



Desarrollo Web en Entorno Cliente (DWEC)

Curso 2020-2021

# Concepto

- ▶ Es un API sencillo que soportan la mayor parte de los navegadores actuales
- ▶ Para obtener la posición actual se basa en diversos métodos, ordenados de menor a mayor precisión:
  - ▶ IP
  - ▶ Redes GSM
    - ▶ Triangulación con las antenas de telefonía o puntos de acceso Wifi.
  - ▶ GPS
    - ▶ Basado en la posición con respecto a una serie de satélites.



# Comprobación de funcionalidad

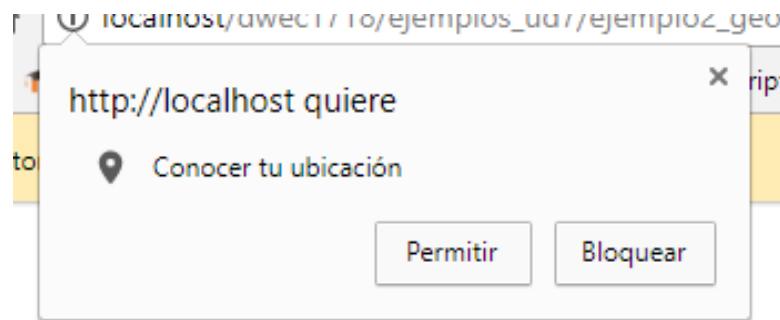
```
function compruebaGeolocalizacion(){
    if ("geolocation" in navigator)
        return true;
    else
        return false;
}
```

- ▶ También podemos usar la librería Modernizr que nos permite comprobar compatibilidad del navegador con muchas características



# Objeto navigator.geolocation

- ▶ Ofrece métodos para obtener la localización del usuario.
- ▶ Si es la primera vez que se solicita la localización al navegador, éste muestra un mensaje pidiendo permiso al usuario para compartir su localización.



# Objeto navigator.geolocation

## Métodos

**getCurrentPosition  
(funcionExito,  
[funcionError],  
[opciones])**

Obtiene la posición del usuario a través de una petición asíncrona.

Cuando se ha conseguido la posición se ejecuta la función de callback `funcionExito()`

Si se desea se puede proporcionar una función de error.

Por defecto intenta responder tan rápido como sea posible, usando métodos de baja precisión. Es útil si queremos una respuesta rápida sin importar su exactitud.

La función de éxito recibe como parámetro un objeto **position** que contiene como propiedad un objeto **coords** con las coordenadas (**latitude**, **longitude**) y la precisión en metros (**accuracy**)



# Objeto navigator.geolocation

## Métodos

<b>watchPosition (funcionExito, [funcionError], [opciones])</b>	Consulta cada cierto tiempo la posición del usuario, ejecutando la función de <i>callback</i> indicada únicamente si la posición ha cambiado desde la última consulta.  Devuelve un identificador único.
<b>clearWatch(id)</b>	Permite cancelar las consultas de posición iniciadas con <b>watchPosition()</b> . Recibe como parámetro el id obtenido mediante <b>watchPosition()</b>



# Objeto navigator.geolocation

## Métodos

- ▶ Tanto getCurrentPosition() como watchPosition() reciben 3 parámetros:
  - ▶ **funcionExito()** recibe como parámetro un objeto **position** que contiene como propiedad un objeto **coords** con las coordenadas (**latitude**, **longitude**) y la precisión en metros (**accuracy**).
  - ▶ **funcionError()** se ejecuta cuando el usuario deniega la petición de localización, o cuando el dispositivo pierde la localización. Recibe un argumento con dos propiedades (**code** y **message**), pudiendo ser el código 1 (**PERMISSION\_DENIED**), 2 (**POSITION\_UNAVAILABLE**) ó 3 (**TIMEOUT**).
  - ▶ **opciones** de configuración (aunque algunas pueden ser ignoradas por los navegadores):
    - ▶ **enableHighAccuracy**: booleano [por defecto false]
    - ▶ **timeout**: entero (milisegundos) [por defecto infinito]
    - ▶ **maximumAge**: entero (milisegundos) [por defecto 0]



# getCurrentPosition()

## Ejemplo

```
navigator.geolocation.getCurrentPosition(exito, error);

function exito(posicion){
    console.log("Tus coordenadas son "
    + posicion.coords.latitude, posicion.coords.longitude);
}

function error(p){
    console.error(
        "No se han podido obtener las coordenadas. Código “
        + p.code);
}
```



# watchPosition()

## Ejemplo

```
var id=navigator.geolocation.watchPosition(exito,error);
setTimeout(aborta,5000,id);

function exito(posicion){
    console.log("Tus coordenadas son "
    + posicion.coords.latitude,posicion.coords.longitude);
}

function error(p){
    console.error(
    "No se han podido obtener las coordenadas. Código “
    + p.code);
}

function aborta(id){
    navigator.geolocation.clearWatch(id);
}
```



# Opciones de configuración

## Ejemplo

```
var opciones={  
    enableHighAccuracy:true,  
    timeout:2000,  
    maximumAge:0  
};  
  
navigator.geolocation.getCurrentPosition(  
    exito,error,opciones);  
  
function exito(posicion){  
    console.log("Tus coordenadas son "  
    + posicion.coords.latitude,posicion.coords.longitude);  
}  
  
function error(p){  
    console.error(  
    "No se han podido obtener las coordenadas. Código "  
    + p.code);  
}
```



# Mostrando la localización en un mapa

- Vamos a utilizar la API de Google Maps para mostrar nuestra ubicación (obtenida mediante geolocalización) en un mapa.
  - Para obtener dicha API necesitaremos [obtener una clave gratuita](#).



# Mostrando la localización en un mapa

- ▶ Para mostrar el mapa usaremos 3 objetos:
  - ▶ `googleMaps.Map`
    - ▶ Representa el propio mapa.
    - ▶ Necesito indicarle la capa donde voy a mostrarlo y puedo pasarle opciones adicionales (como el zoom, el centro...)
  - ▶ `googleMaps.Marker`
    - ▶ Representa un marcador cuyas coordenadas le indico.
  - ▶ `googleMaps.InfoWindow`
    - ▶ Representa una ventana de información que puedo asociar a las coordenadas de un mapa o a un marcador

Calcularemos la coordenadas del centro del mapa y el marcador mediante geolocalización



# Mostrando la localización en un mapa - Ejemplo

## HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo 2</title>
    <meta charset="UTF-8">
    <script src="js/ejemplo_geolocalizacion.js" type="text/javascript">
      </script>
    <script src="https://maps.googleapis.com/maps/api/js?key=TU_API_KEY">
      </script>
    <link href="css/estilos.css" rel="stylesheet" type="text/css"/>
  </head>
  <body>
    <div id="mapa"></div>
  </body>
</html>
```

## CSS

```
#mapa {
  height: 300px;
  width: 400px;
  border: 2px solid black;
  margin: 20px;
}
```



# Mostrando la localización en un mapa - Ejemplo

```
window.addEventListener("load",cargaPagina);

function cargaPagina(){
    posiciona();
}

function posiciona(){
    navigator.geolocation.getCurrentPosition(
        cargaLocalizacion,errorLocalizacion);
}

function cargaLocalizacion(posicion){
    var localizacion = {
        lat: posicion.coords.latitude,
        lng: posicion.coords.longitude
    };
    var etiqueta="Mi ubicación";

    cargaMapa(localizacion,etiqueta);
}
```



# Mostrando la localización en un mapa - Ejemplo

```
function cargaMapa(centro,etiqueta){
    var opcionesMapa={zoom:10};
    var mapa = new google.maps.Map(
        document.getElementById('mapa'),opcionesMapa);
    mapa.setCenter(centro);
    var marcador =
        new google.maps.Marker({position:centro,map:mapa});
    marcador.addListener("click",function(){
        var informacion =
            new google.maps.InfoWindow({content: etiqueta});
        informacion.open(mapa,this);});}

function errorLocalizacion(){
    var localizacion = {lat: 40.7127837,
                        lng: -74.00594130000002};
    var etiqueta="New York, USA";
    cargaMapa(localizacion,etiqueta);};
```

