

Uso de PHP a modo de prueba. Seguir los pasos:

1. Crear una carpeta llamada **EjemplosPHP**
2. Descargar y descomprimir la versión actual php para windows de <https://windows.php.net/download/>
3. Por ejemplo: <https://windows.php.net/downloads/releases/php-8.0.12-Win32-vs16-x64.zip> y descomprimir en **EjemplosPHP\PHP**
4. Abrir con VSCode el directorio **EjemplosPHP**
5. Abrir un terminal y ejecutar **.\php\php -S localhost:8080**

Ahora el servidor está operativo y podemos añadir páginas html y php:

```
C:\tmp\EjemplosPHP>  
C:\tmp\EjemplosPHP>.\php\php -S localhost:8080  
[Thu Nov 4 20:35:59 2021] PHP 8.0.12 Development Server (http://localhost:8080) started
```

Añadir el fichero hola.php y probar el resultado:

hola.php

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>Document</title>
7 </head>
8 <body>
9 <h1>Prueba PHP</h1>
10 <?php
11     echo "¡¡Hola mundo!! desde php";
12     phpinfo();
13 ?>
```

localhost:8080/hola.php

Prueba PHP

¡¡Hola mundo!! desde php

PHP Version 8.0.12

System	Windows
Build Date	Oct 19 20

Añadir dos ficheros .php para probar formularios:

pruebaGET.php

```
1  <?php
2      echo "  Prueba GET  ";
3      echo "  Nombre: " . $_GET["nombre"];
4      echo "  Edad: " . $_GET["edad"];
5  ?>
```

pruebaPOST.php

```
1  <?php
2      echo "  Prueba POST  ";
3      echo "  Nombre: " . $_POST["nombre"];
4      echo "  Edad: " . $_POST["edad"];
5  ?>
```

- Diferencias entre GET y POST:
 - GET envía la información en la URL con el formato:
<http://www.site.com/page.html?var1=value1&var2=value2&var3=value3>
 - POST envía la información en las cabeceras.
 - GET está limitado al tamaño de la URL.
 - POST no está limitado y permite enviar ficheros adjuntos.
 - GET se suele utilizar para recuperar información.
 - POST se suele utilizar para enviar datos o modificarlos.

Ejemplo: index0.html. Enviar datos de formularios:

```
<form action="pruebaGET.php" method="get">
  <fieldset>
    <legend>Prueba con GET</legend>
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre">
    <label for="edad">Edad:</label>
    <input type="text" id="edad" name="edad">
    <input type="submit" value="Enviar">
  </fieldset>
</form>
<br>
<form action="pruebaPOST.php" method="post">
  <fieldset>
    <legend>Prueba con POST</legend>
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre">
    <label for="edad">Edad:</label>
    <input type="text" id="edad" name="edad">
    <input type="submit" value="Enviar">
  </fieldset>
</form>
```

- Probar el resultado:

Prueba formularios PHP

Prueba con GET

Nombre: Edad:

Prueba con GET

Nombre: Edad:

← → ↻ ⓘ localhost:8080/pruebaGET.php?nombre=Pepe&edad=34

Prueba GET Nombre: Pepe Edad: 34

← → ↻ ⓘ localhost:8080/pruebaPOST.php

Prueba POST Nombre: Paco Edad: 43

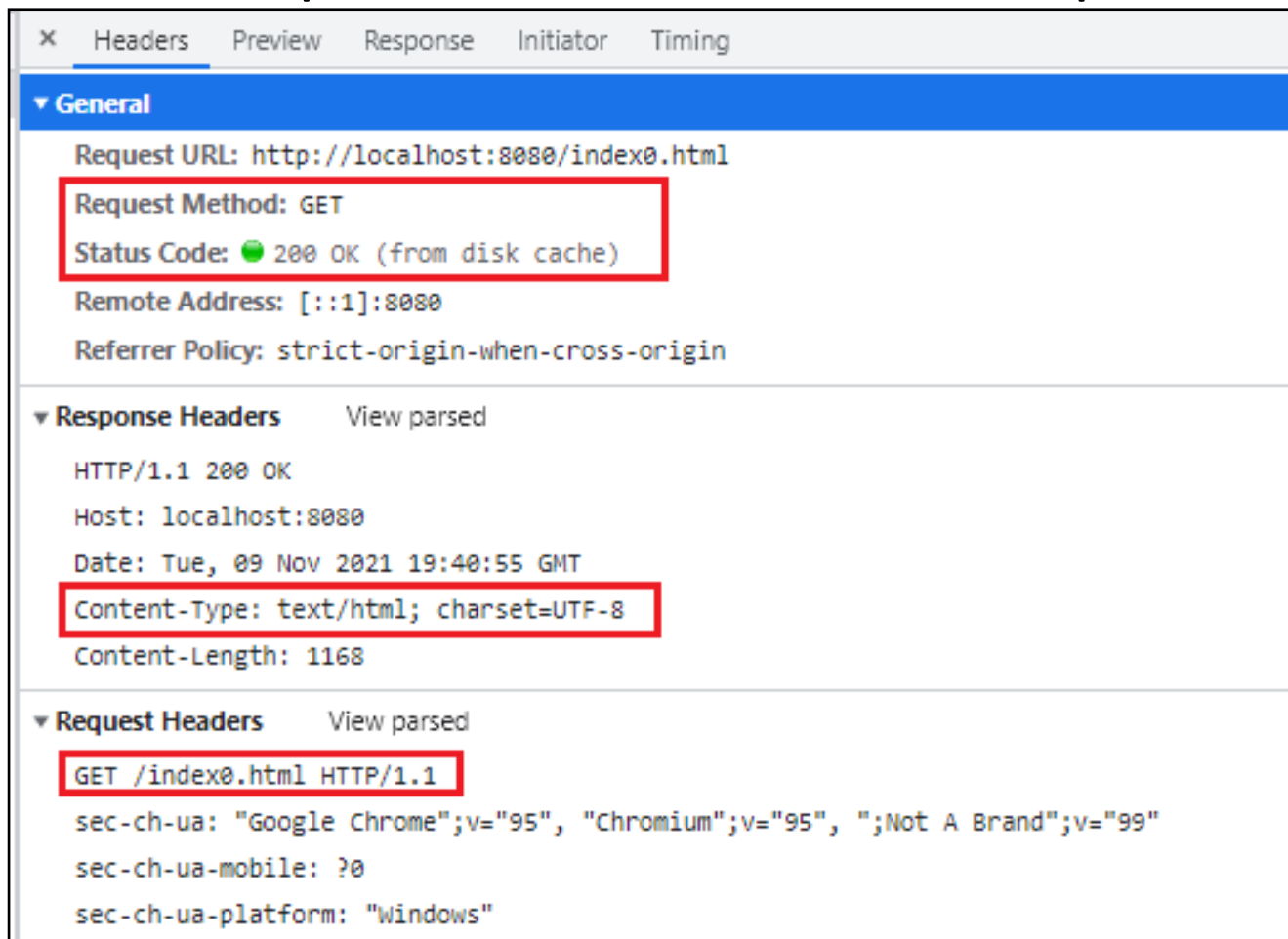
- Desde el inspector también se puede comprobar el tipo de petición (GET/POST).

Media Types o tipos MIME

- Cuando se transfiere una solicitud HTTP, tanto el cliente como el servidor, deben conocer los tipos de datos que van a procesar.
- Los **Media Types** (antes conocidos como tipos MIME o Multipurpose Internet Mail Extensions) son identificadores que definen el tipo de datos a transmitir en un correo electrónico , una conexión HTTP en Internet, etc.
- La propiedad **Content-Type** de las cabeceras HTTP permiten establecer los Media Types. Ej: “text/plain”, “text/javascript”, “image/jpeg” o “audio/mpeg”.
- Cuando se solicita una página web, el servidor responde con un mensaje en el que hay dos partes: las cabeceras y el propio cuerpo del mensaje.
- Dentro de las cabeceras tenemos:
 - **Response Headers:** son cabeceras que contiene el resultado obtenido (Content-Type, fecha y hora, host, etc).
 - Request Headers:** son las cabeceras que contienen lo que se ha pedido (Content-Type, host, URL, método de petición empleado GET/POST, cookies, etc).

Media Types o tipos MIME

- Por ejemplo si abrimos el navegador con <http://localhost:8080/index0.html> se generará una petición al servidor y se obtendrá una respuesta que podemos ver en la pestaña “Network” del inspector:

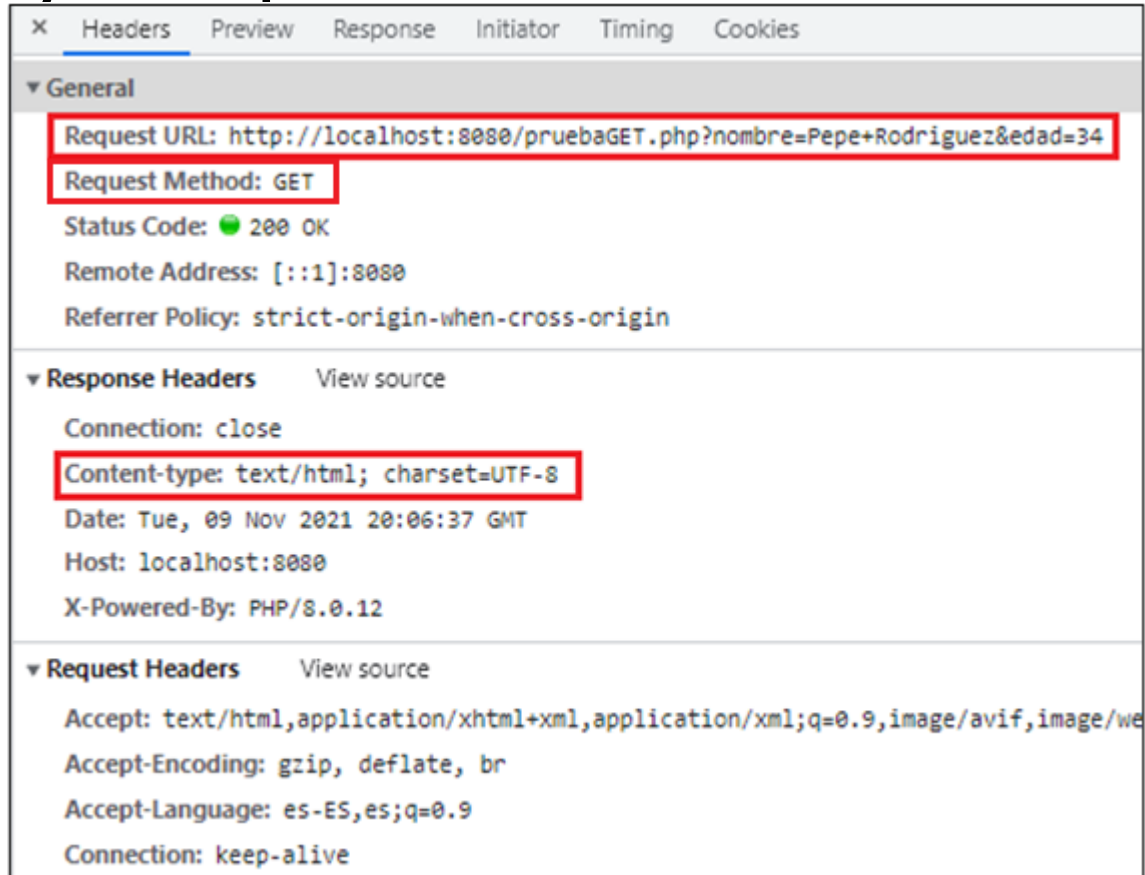


Observar:

- Se solicitó la página con el método GET.
- Se obtuvo una respuesta correcta (Status Code: 200)
- El Content-Type de la respuesta del servidor es **text/html**
- En cabecera de petición (Request Header) no aparece la propiedad Content-Type (por defecto para una petición de tipo GET no se establece este valor).

Media Types o tipos MIME

- Ahora podemos enviar los formularios (mediante GET o POST) y comprobar el resultado del servidor con el inspector:



The screenshot shows the Chrome DevTools Network tab with a GET request selected. The 'General' tab is active, displaying the following information:

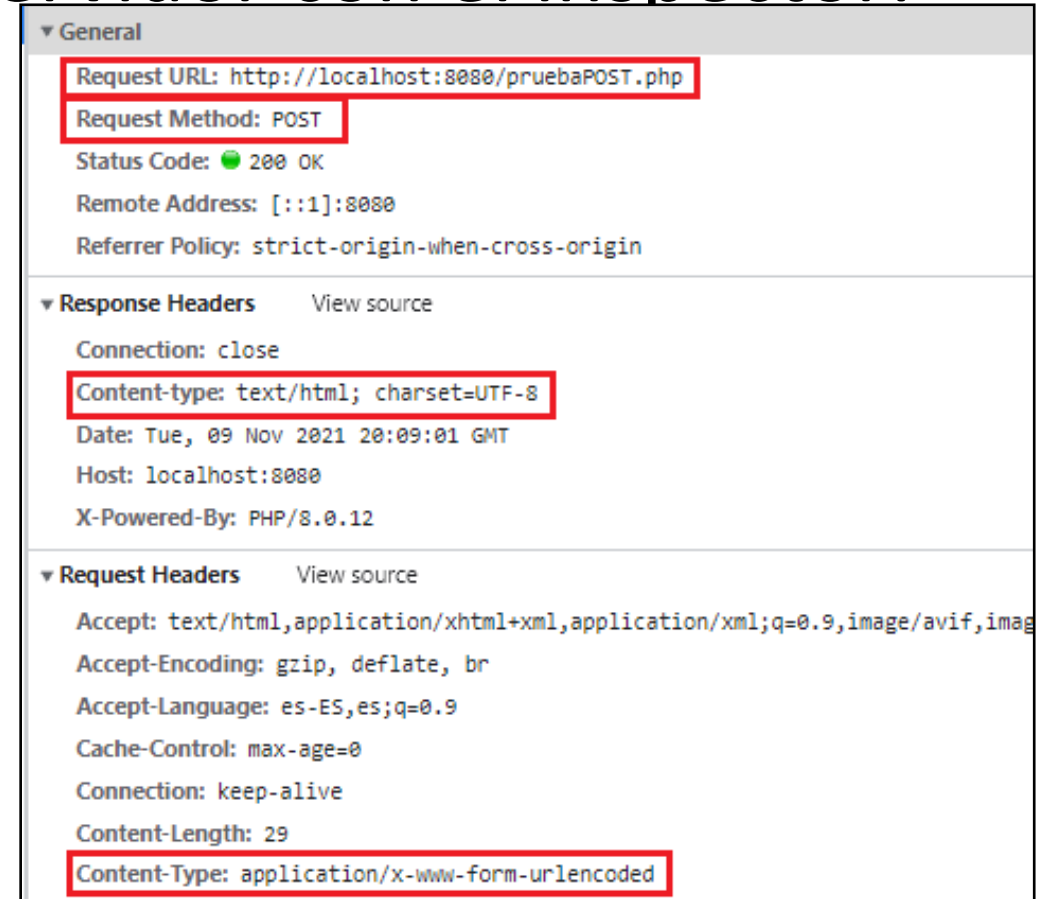
- Request URL:** `http://localhost:8080/pruebaGET.php?nombre=Pepe+Rodriguez&edad=34`
- Request Method:** `GET`
- Status Code:** `200 OK`
- Remote Address:** `::1:8080`
- Referrer Policy:** `strict-origin-when-cross-origin`

The 'Response Headers' section shows:

- Content-type:** `text/html; charset=UTF-8`
- Date:** `Tue, 09 Nov 2021 20:06:37 GMT`
- Host:** `localhost:8080`
- X-Powered-By:** `PHP/8.0.12`

The 'Request Headers' section shows:

- Accept:** `text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp`
- Accept-Encoding:** `gzip, deflate, br`
- Accept-Language:** `es-ES,es;q=0.9`
- Connection:** `keep-alive`



The screenshot shows the Chrome DevTools Network tab with a POST request selected. The 'General' tab is active, displaying the following information:

- Request URL:** `http://localhost:8080/pruebaPOST.php`
- Request Method:** `POST`
- Status Code:** `200 OK`
- Remote Address:** `::1:8080`
- Referrer Policy:** `strict-origin-when-cross-origin`

The 'Response Headers' section shows:

- Content-type:** `text/html; charset=UTF-8`
- Date:** `Tue, 09 Nov 2021 20:09:01 GMT`
- Host:** `localhost:8080`
- X-Powered-By:** `PHP/8.0.12`

The 'Request Headers' section shows:

- Content-Type:** `application/x-www-form-urlencoded`

En los dos casos la cabecera de respuesta tiene un Content-Type="text/html". (es el tipo de datos que manda el servidor).

En el caso de GET en las cabeceras de petición (Request Headers) no está establecido el Content-Type.

En el caso de POST el Content-Type de la Request Header es "application/x-www-form-urlencoded" (es la opción por defecto para POST).

Content-Type en formularios

- Cuando se envía un formulario mediante GET no se establece el Content-Type porque no hay cuerpo del mensaje HTTP. No hay datos adicionales. Todo se envía en la propia URL.
- Cuando se envía un formulario mediante POST se debe codificar la información que se manda en el cuerpo del mensaje con alguno los siguientes métodos:
 - 1.- **application/x-www-form-urlencoded** (valor por defecto) : es similar a la querystring en una petición GET pero envía los datos en el cuerpo del mensaje en lugar de utilizar la URL.
 - 2.- **multipart/form-data** : es más complejo pero es más eficiente con datos pesados y es necesario para enviar ficheros o datos binarios (ej: se debe utilizar cuando hay un `<input type="file">`)
 - 3.- **text/plain** : no se utiliza.
- Para establecer el Content-Type se pueden utilizar varios métodos:
 - En un formulario convencional indicar el atributo “**enctype**” en la etiqueta `<form>`. Ej:

```
<form action="pruebaPOST.php" method="post" enctype="multipart/form-data">
  <fieldset>
    <legend>Prueba con POST</legend>
    <label for="nombre">Nombre:</label>
```
 - En AJAX, utilizando el método `setRequestHeader()` sobre el objeto `XMLHttpRequest` . Ej:
 - `xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');`

Ejemplo: index1.html. Enviar formulario desde javascript.

```
<form action="pruebaGET.php" method="get">
  <fieldset>
    <legend>Prueba con GET</legend>
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre">
    <label for="edad">Edad:</label>
    <input type="text" id="edad" name="edad">
    <input type="submit" value="Enviar">
  </fieldset>
</form>
<script>
  const formulario=document.forms[0];
  formulario.addEventListener("submit",function(e) {
    e.preventDefault();
    // .... validaciones .....
    alert("Se enviará el formulario");
    e.target.submit();
  });
</script>
```

- El envío de los datos del formulario se controla desde javascript.
- En el evento “submit” se pueden hacer validaciones de datos.
- El formulario se envía con el método “**submit()**”
- Es válido para GET y para POST.

Ejemplos: index2.html y index3.html. location.href

- A partir de un string se puede construir una URL y utilizar **location.href** para hacer una petición GET sin utilizar formularios:

```
<h1>Prueba formularios PHP javascript</h1>
<script>
  let url="http://localhost:8080/pruebaGET.php?nombre=Pepe&edad=20";
  alert ("Se enviarán los datos...")
  location.href = url;
</script>
```

← No es válido para POST

- Para evitar problemas con los caracteres no permitidos en la URL se debe utilizar **encodeURIComponent()** para cada campo:

```
<h1>Prueba formularios PHP javascript</h1>
<script>
  let nombre=encodeURIComponent("Pepe García");
  let edad=encodeURIComponent("20");
  let url="http://localhost:8080/pruebaGET.php?nombre="+nombre+"&edad="+edad;
  alert ("Se enviarán los datos...");
  location.href = url;
</script>
```

Los datos se codifican para evitar caracteres no ASCII especiales en la URL

Pepe%20Garc%C3%ADa
20

Ejemplos: index4.html y index5.html. URLSearchParams.

- La clase **URLSearchParams** permite crear un conjunto de parámetros que se pueden añadir a la petición GET:

```
const parametros=new URLSearchParams('nombre=Pepe García&edad=10');
for (let parametro of parametros) {
    console.log(parametro);
}
alert("Se enviarán los datos...");
location.href="http://localhost:8080/pruebaGET.php?" + parametros.toString();
```

▶ (2) ['nombre', 'Pepe García']
▶ (2) ['edad', '10']

No es necesario
codificar con
encodeURIComponent
ya lo hace
URLSearchParams

- URLSearchParams** tiene más métodos como: append, get, set, has, etc.

```
const parametros=new URLSearchParams();
parametros.append("nombre","Pepe García");
parametros.append("edad",20);
parametros.append("nacionalidad","española");
console.log(parametros.toString());
console.log(parametros.get("nacionalidad"));
parametros.set("nacionalidad","francesa");
console.log(parametros.get("nacionalidad"));
parametros.delete("nacionalidad");
console.log(parametros.has("nacionalidad"));
```

nombre=Pepe+Garc%C3%ADa&edad=20&nacionalidad=espa%C3%B1ola
española
francesa
false

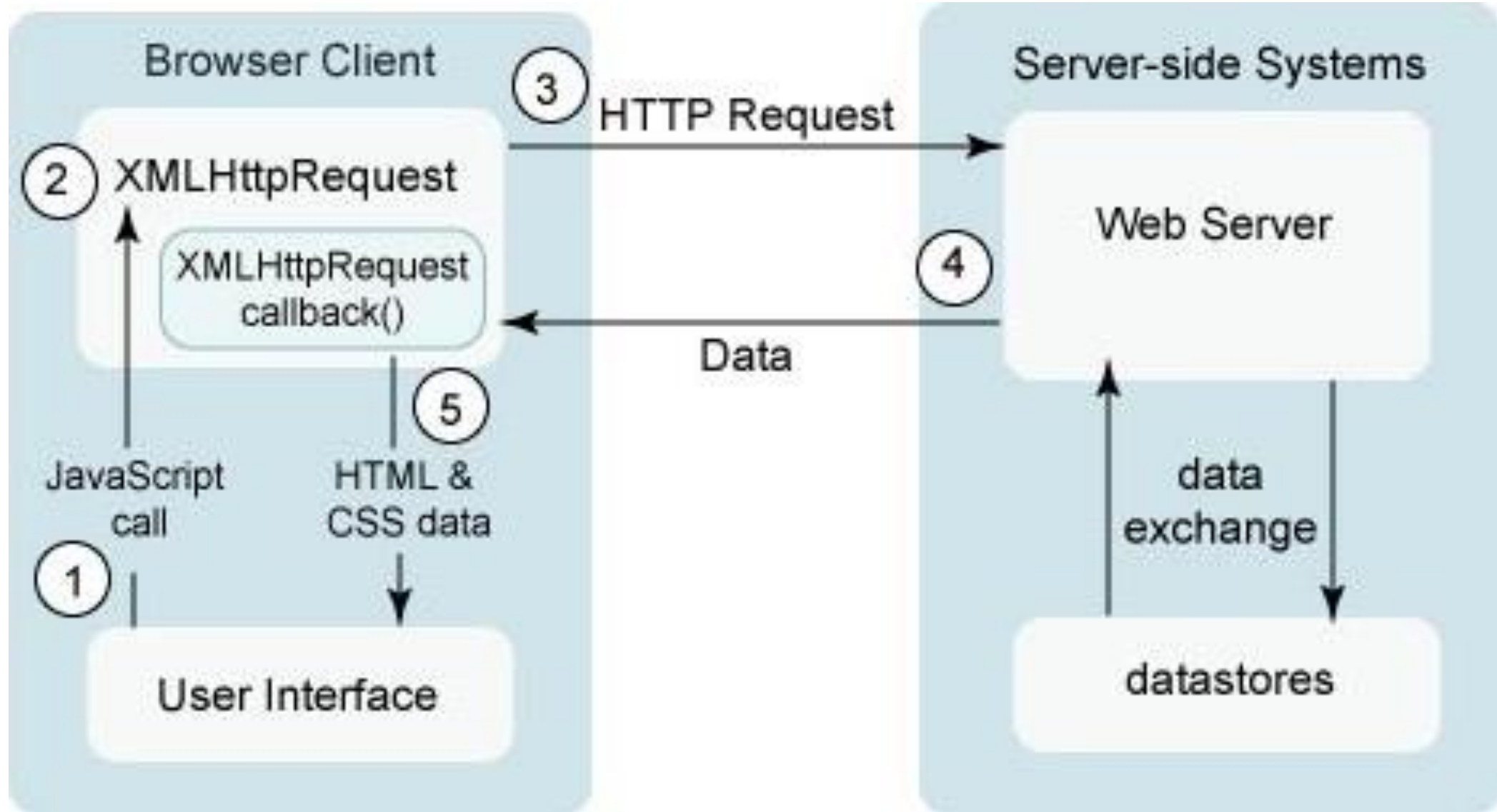
Además el atributo "href" de un enlace (<a href...) contiene una propiedad denominada "searchParams" que es de tipo URLSearchParams

AJAX

- Ajax es acrónimo de **A**synchronous **J**avaScript **A**nd **X**ML
- No se trata de un lenguaje ni de un protocolo concreto sino más bien de una tecnología utilizada en el desarrollo web.
- El objeto central utilizado por AJAX es **XMLHttpRequest (XHR)**.
- La petición y respuesta al servidor se realiza mediante HTTP.
- Gracias a AJAX es posible enviar y recibir información al servidor sin recargar la página.
- El formato de los datos enviados y recibidos puede ser: XML, HTML, JSON, texto plano, etc...
- AJAX es por naturaleza asíncrono: el cliente no tiene porqué esperar de manera síncrona la respuesta del servidor.

AJAX

- Fases en la comunicación cliente-servidor con AJAX:



Ejemplos: ajax1.html. Enviar GET con AJAX

- Hasta ahora hemos visto como enviar datos de formularios con una URL o con URLSearchParams produciendo siempre la recarga de la página.
- AJAX permite enviar la petición con los datos que se deseen sin recargar la página, obteniendo la respuesta asíncrona:

```
<h1>Prueba formularios PHP javascript</h1>
<h2 id="datos"></h2>
<script>
  const parametros = new URLSearchParams('nombre=Pepe García&edad=10');
  const url = "http://localhost:8080/pruebaGET.php?" + parametros.toString();
  let xhr = new XMLHttpRequest();
  xhr.addEventListener("load", function (e) {
    if (xhr.status === 200) {
      document.querySelector("#datos").innerHTML = xhr.responseText;
    } else {
      alert(`Error: ${xhr.status} ${xhr.statusText}`)
    }
  });
  xhr.open("GET", url);
  xhr.send();
</script>
```

Pasos a seguir:

- 1.- Crear el objeto **XMLHttpRequest**.
- 2.- Capturar el evento **load** que se produce cuando se devuelven los datos.
- 3.- Configurar la petición con el método **open()**.
- 4.- Enviar la petición con el método **send()**.

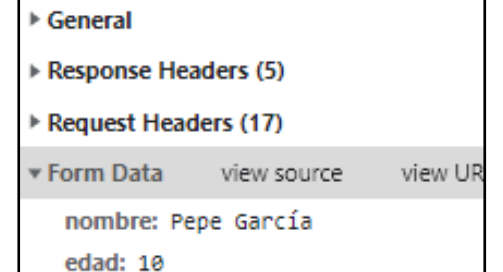
El Status HTTP 200 significa que todo es correcto.

Ejemplos: ajax2.html. Enviar POST con AJAX

- El mismo ejemplo pero utilizando POST también es posible.
- Basta con omitir los parámetros en la URL y pasar al método send() el objeto URLSearchParams con los parámetros deseados.

```
<h2 id="datos"></h2>
<script>
  const parametros = new URLSearchParams('nombre=Pepe García&edad=10');
  const url = "http://localhost:8080/pruebaPOST.php";
  let xhr = new XMLHttpRequest();
  xhr.addEventListener("load", function (e) {
    if (xhr.status === 200) {
      document.querySelector("#datos").innerHTML = xhr.responseText;
    } else {
      alert(`Error: ${xhr.status} ${xhr.statusText}`)
    }
  });
  xhr.open("POST", url);
  xhr.send(parametros);
</script>
```


En este caso es una petición POST con Content-Type= **application/x-www-form-urlencoded** que es el valor por defecto. Los parámetros se pueden ver con el inspector en el apartado "Form Data" de las cabeceras.



Uso de FormData.


- **FormData** permite crear un conjunto de pares clave-valor que representan los campos de un formulario.
- Se puede crear un FormData de forma individual o a partir de un formulario.
- Admite diferentes métodos como: `append()`, `delete()`, `get()`, `set()`, etc.
- Los objetos de tipo FormData se pueden pasar como parámetro en el método `send()` de XMLHttpRequest para enviar los campos del formulario.

```
xhr.open("POST", "http://localhost:8080/pruebaPOST.php");  
xhr.send(new FormData(document.forms[0]));
```



Crea un FormData a partir del primer formulario existente en el documento.

```
let formulario=new FormData();  
formulario.append("nombre","Pepe García");  
formulario.append("edad",34);  
xhr.open("POST", "http://localhost:8080/pruebaPOST.php");  
xhr.send(formulario);
```



Crea un FormData y añade manualmente los campos necesarios.

Uso de FormData. Ajax3.html

Enviar FormData

```
let xhr = new XMLHttpRequest();
xhr.addEventListener("load", function (e) {
    if (xhr.status === 200) {
        document.querySelector("#datos").innerHTML = xhr.responseText;
    } else {
        alert(`Error: ${xhr.status} ${xhr.statusText}`)
    }
});

document.forms[0].addEventListener("submit",function(e) {
    e.preventDefault();
    // Enviar datos del formulario:
    xhr.open("POST", "http://localhost:8080/pruebaPOST.php");
    xhr.send(new FormData(document.forms[0]));
    // Crear un nuevo FormData y enviarlo:
    let formulario=new FormData();
    formulario.append("nombre","Pepe García");
    formulario.append("edad",43);
    xhr.open("POST", "http://localhost:8080/pruebaPOST.php");
    xhr.send(formulario);
});
```

- En el ejemplo se ven los dos casos:
 - Crear un FormData a partir de un formulario
 - Crear un FormData individual
- También es posible crear un FormData a partir de un formulario y añadir o eliminar elementos con `append()` y `delete()`

En ambos casos es una petición POST con Content-Type= **multipart/form-data**

Enviar un fichero. Fichero1.html y fichero.php

- Para subir un fichero al servidor con un formulario convencional es necesario establecer los atributos **method="post"** y **enctype="multipart/form.data"** . Ej:

```
<form method="post" enctype="multipart/form-data" action="http://localhost:8080/fichero.php" >
    <input type="file" name="mifichero" id="mifichero">
    <input type="submit" value="Enviar">
</form>
```

- El php que recoge el fichero podría ser:

```
<?php
$fichero_subido = 'C:\\tmp\\' . basename($_FILES['mifichero']['name']);
if (move_uploaded_file($_FILES['mifichero']['tmp_name'], $fichero_subido)) {
    echo "<h1>Se ha subido el fichero ".$fichero_subido."</h1>";
} else {
    echo "<h1>¡Error!</h1>";
}
// A tener en cuenta:
// mifichero: es el nombre de la variable en el formulario
// tmp_name: es el nombre temporal que asigna php al fichero subido
// name: es el nombre original del fichero en el cliente
?>
```

Se puede comprobar con el inspector que la petición es de tipo POST y que **Content-Type** es **multipart/form-data**

El fichero subido se almacenará en el directorio C:\tmp

También se podría haber controlado el evento "submit" desde javascript.

Enviar un fichero desde Ajax. Fichero2.html

```
<form>
  <input type="file" name="mifichero" id="mifichero">
  <input type="submit" value="Enviar">
</form>
<div id="datos">
<script>
  const xhr = new XMLHttpRequest();
  xhr.addEventListener('load', function (e) {
    if (xhr.status === 200) {
      document.querySelector('#datos').innerHTML = xhr.responseText;
    } else {
      alert(`Error: ${xhr.status} ${xhr.statusText}`);
    }
  });
  document.forms[0].addEventListener('submit', (e) => {
    e.preventDefault();
    xhr.open('POST', 'http://localhost:8080/fichero.php');
    xhr.send(new FormData(document.forms[0]));
  });
</script>
```

En el formulario obviamos utilizar los atributos “method” y “action” porque la petición la hacemos con AJAX

En este caso, por defecto **Content-Type** es **multipart/form-data** incluso si no se utiliza el atributo “enctype” del formulario.

En este caso no se debe utilizar el método `setRequestHeader()` de `XMLHttpRequest` porque `send()` detecta automáticamente el tipo de dato a enviar.

Enviar un fichero desde Ajax. Fichero3.html

```
<form>
  <input type="file" name="mifichero" id="mifichero">
  <input type="submit" value="Enviar">
</form>
<div id="datos">
<script>
  let xhr=new XMLHttpRequest();
  xhr.addEventListener("load", function (e) {
    if (xhr.status === 200) {
      document.querySelector("#datos").innerHTML = xhr.responseText;
    } else {
      alert(`Error: ${xhr.status} ${xhr.statusText}`)
    }
  });
  document.forms[0].addEventListener("submit",(e)=>{
    e.preventDefault();
    xhr.open("POST","http://localhost:8080/fichero.php");
    const formulario =new FormData();
    formulario.append("mifichero",document.querySelector("#mifichero").files[0]);
    xhr.send(formulario);
  });
</script>
```

Este ejemplo es igual que el anterior pero para crear el FormData únicamente seleccionamos el campo **input type="file"**

Mediante un campo input con type="file" se pueden seleccionar varios ficheros. Para elegir el primero de ellos utilizamos **.files[0]**

Ejercicio2: Subir imágenes.

- Se dispone del siguiente script php (**RecibirFichero.php**) que permite recibir una imagen enviada desde un formulario y guardar la imagen en el directorio actual. El script devuelve el nombre del fichero .jpg o “error” en su caso.

Tema05 > EJ2 > RecibirFichero.php

```
1  <?
2  $fichero_subido='.\'.basename($_FILES['mifichero']['name']);
3  if (move_uploaded_file($_FILES['mifichero']['tmp_name'],$fichero_subido)) {
4      echo basename($fichero_subido);
5  } else {
6      echo "error";
7  }
8  ?>
```

- Se desea codificar un fichero denominado **EnviarFicheros.html** que contenga el html y código javascript para realizar las siguientes tareas:
 - Seleccionar un fichero .jpg y enviarlo mediante ajax al servidor.
 - Obtener el resultado y crear dinámicamente un nuevo elemento “IMG” que permita visualizar la imagen que se ha subido al servidor.
- Como consecuencia, cada vez que se sube una nueva imagen, se añade a la página web dinámicamente. Todo ello sin realizar la recarga de la página web actual.

