Tipos de páginas web

Una sencilla clasificación de los tipos de páginas web podría ser esta:
Páginas estáticas
Páginas dinámicas

Páginas estáticas

Diremos que una página es estática cuando sus contenidos **no** pueden ser modificados –ni desde el *servidor* que la aloja (ordenador remoto) ni desde el *cliente* (navegador)– mediante ninguna intervención del usuario ni tampoco a través de ningún programa.

Páginas dinámicas

Llamaremos dinámicas a las páginas cuyos contenidos **sí** pueden ser modificados –de forma automática o mediante la intervención de un usuario– bien sea desde el *cliente* y/o desde el *servidor*.

Para que esas modificaciones puedan producirse es necesario que *algo o alguien* especifique: *qué*, *cómo*, *cuándo*, *dónde* y *de qué forma* deben realizarse, y que exista otro *algo o alguien* capaz de acceder, interpretar y ejecutar tales instrucciones en el momento preciso.

Igual que ocurre en la vida cotidiana, las *especificaciones* y las *instrucciones* requieren: un **lenguaje** para definirlas; un **soporte** para almacenarlas y un **intérprete** capaz de *eje- cutarlas*.

Somos capaces de entender unas instrucciones escritas en castellano pero si estuvieran escritas en *búlgaro* las cosas seguramente serían bastante distintas, y, por supuesto, *a un búlgar@* le pasaría justamente lo contrario.

Igual ocurre con los programas *intérpretes* de los lenguajes de script. Ellos también requieren órdenes escritas en su *propio idioma*.

Scripts

Se llama *script* a un conjunto de *instrucciones* escritas en un *lenguaje* determinado que van **incrustadas** dentro de una *página WEB* de modo que su *intérprete* pueda acceder a ellas en el momento en el que se requiera su *ejecución*.

Cuando se **incrustan** *scripts* en

Un ejemplo de página estática

Cualquier usuario que acceda a ésta -ya sea en *modo local*, o a través de un *servidor remoto*– visualizará siempre la misma fecha: *11 de febrero de 2009*.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
Hoy es 4-5-2009 y son las 14:23:57 horas
</BODY>
</HTML>
```

ejemplo1.html

Una par de páginas dinámicas

Si pulsas en el enlace del primero de estos dos ejemplos verás que la fecha que aparece en la página es la *fecha actual* de tu sistema, y además, cada vez que pulses el botón *Actualizar* de tu navegador podrás comprobar que *se actualiza* la hora.

Una *intervención* del usuario *modifica los contenidos*.

```
<HTML>
<HEAD>
<script language="JavaScript">
var son= new Date();
var fecha=son.getDate()+" - "+(son.getMonth()+1)+" - "+son.getFullYear();
var hora=son.getHours()+":"+son.getMinutes()+":"+son.getSeconds();
document.write('Hoy es '+fecha+' y son las '+hora+' horas');
</script>
</HEAD>
<BODY>
</BODY>
</HTML>
```

ejemplo2.html

En este otro ejemplo la modificación de los contenidos *no requiere intervención alguna* por parte del usuario. Cada *5 segundos* (fíjate donde dice var frecuencia=5000). *Cinco mil* es el período de actualización, expresado en *milisegundos*) se **rescribirán** de forma automática *la fecha y la hora*. Tenemos un *cronómetro* automático.

```
<HTML>
<HEAD>
<script language="JavaScript">
var reloj=0;
var frecuencia=5000;
             function actualiza(){
var son= new Date();
var fecha=son.getDate()+" - "+(son.getMonth()+1)+" - "+son.getFullYear();
var hora=son.getHours()+":"+son.getMinutes()+":"+son.getSeconds();
var escribe='Hoy es '+fecha+' y son las '+hora+' horas';
var situa=document.getElementById('capa0');
situa.innerHTML=escribe;
reloj=setTimeout("actualiza()",frecuencia);
                                  }
</script>
</HEAD>
<BODY onLoad="actualiza()";>
```

una *página WEB* empiezan a *convivir* en un mismo docu- mento informaciones destina- das a distintos *intérpretes*.

Por una parte, el *código HTML* que ha de ser *interpretado* por el *navegador*, y por la otra, los *scripts* que han de ser *ejecuta- dos* –dependiendo del lenguaje en el que hayan sido escritos– por **su** *intérprete* correspon- diente.

La manera de diferenciar los contenidos es delimitar los scripts *marcando* su comienzo con una etiqueta de *apertura* **<script>** y señalando el final con una etiqueta de *cierre* **</script>**.

Lo que no está contenido entre esas etiquetas se considerará *código HTML*.

La posibilidad de **insertar** en un mismo documento *scripts* desarrollados en distintos *len- guajes* obliga a especificar cuál se ha utilizado en cada caso, para que en el momento en el que vayan a ser *ejecutados* se invoque el *intérprete* adecuado.

Para ello, dentro de la propia etiqueta de apertura (*<script>*) se inserta *una referencia* al tipo de *lenguaje* con esta sintaxis:

language="nombre"

Por ejemplo:

<script language="PHP">
.....
..... instrucciones ..
.....
</script>

indicaría que las instrucciones están escritas con la sintaxis de **PHP**.

Por el contrario, en este otro supuesto:

<script language="JavaScript">
.....
..... instrucciones ..
.....
</script>

estaríamos señalando que en las instrucciones contenidas en el *script* se ha utilizado sintaxis de **JavaScript**.

Para el caso concreto de **PHP**, existe una *sintaxis alternativa*, mucho más cómoda y que es la que se usa habitualmente.

Es la siguiente:

<?
.....
.....instrucciones..
.....
?>

```
<div class="capa0">
</div>
</BODY>
</HTML>
```

ejemplo3.html

Ejercicio nº 1

Abre tu *block de notas* y escribe el código fuente del ejemplo nº 3 prestando **especial atención** a la trascripción de las **mayúsculas y las minúsculas** (*JavaScript*, igual que **PHP**, diferencia entre unas y otras) y también a las **comillas** y a los **punto y coma** que aparecen al final de cada línea.
Guarda el documento con el nombre **ejercicio1.html**, luego abrélo con el *navegador* y comprueba el funcionamiento del cronómetro.
Una vez que hayas comprobado que funciona, prueba a sustituir el **5000** por *otros valores numéricos* y comprueba como se modifica la *frecuencia* del *cronómetro*.

Servidores y clientes

Es frecuente observar *en la calle* que son muchas las personas que cuando se refieren a **los servidores** lo hacen con sí se tratara de máquinas *complejísimas*, *misteriosas*, *lejanas* y *enormes* que, bajo esa aureola de *cripticismo*, parecen totalmente distintas al ordenador que usamos habitualmente. *¡Nada más lejos de la realidad!*

Intentaremos aclarar algunos conceptos con ejemplos cotidianos. Pensemos en esos *ordenadores remotos* (también llamados **host**) como si se tratara de uno esos sitios desde los que se sirven *comidas a domicilio*.

Quizá lo primero en lo que se te ocurra pensar sea en una *pizza*, no porque desconozcas que también *es posible comprar otras cosas* sino por la popularidad de ese tipo de *servicio*. Algo similar ocurre con los **host**. La frecuencia con la que accedemos a ellos en demanda de *páginas web* hace que tendamos a identificarlos con ellas, pero... también los **host** ofrecen –o pueden ofrecer– más *servicios*. Sigamos con las *comidas a domicilio*.

Cada una de esas empresas puede **atender peticiones** de **uno solo** o de **varios** servicios distintos (*pizzas*, *helados*, o *platos regionales*, por citar algunos ejemplos), pero la oferta de *cada uno de esos servicios* **requiere** una infraestructura *adecuada a cada caso*. La oferta de *pizzas* exigirá disponer de un *horno*, y la de *helados* necesitará de una *instalación frigorífica*.

Pues bien, algo muy similar ocurre con los **host**. También éstos pueden ofrecer *uno* o *varios* servicios (páginas web, correo electrónico, transferencias FTP, noticias, etcétera) y también es necesario que cada servicio disponga de su propia infraestructura, que en este caso sería **un programa distinto** (*software de servidor*) para cada uno de ellos.

Como puedes ver, **no basta** con hablar de **servidores**. Es necesario especificar también *qué es lo que sirven*, para lo cual habría que decir: **servidor de páginas web**, **servidor de correo**, etcétera y tener presente que -aunque *convivan* en el mismo **host**– cada uno de ellos requiere su propio software y su propia configuración.

Resumiendo, cuando en lenguaje coloquial hablamos de un *servidor* estamos aludiendo un **host** (ordenador remoto) –el tamaño y la lejanía carecen de importancia– provisto de **programas** (*software de servidor*) que, *cuando está accesible* (conectado a Internet) y con el *software activo* (servidor en funcionamiento) > es capaz de *atender* **peticiones** y *devolver* a los **clientes** los *documentos solicitados*, o un *mensaje de error*, en el caso de que *no estuvieran disponibles*.

Veamos un ejemplo de como se desarrolla ese proceso de **petición-respuesta**.

Para *leer* el correo electrónico necesitas disponer de un **programa** –supongamos que es *Outlook Express*– **instalado en tu ordenador** y hacer, a través de él, una **petición** a un ordenador remoto (**host**). Si quisieras visualizar páginas web tendrías que utilizar un programa distinto –Internet Explorer, por ejemplo– capaz de realizar esta otra tarea.

Al programa que utilizamos para realizar cada **petición** le llamaremos **cliente**.

¿Qué es una petición?

Una **petición** es un conjunto de datos que un **cliente** (recuerda que el cliente siempre es uno de los

<? hará la misma función que *<script language="PHP">* y **?>** será equivalente a *</script>*.

Lenguajes

Hay múltiples posibilidades en cuanto a lenguajes de *script*. Pero antes de hacer mención a algunos de ellos es conveniente hacer una clasificación previa.

Los lenguajes de *script* pueden clasificarse en dos tipos:

- Del lado del cliente
- Del lado del servidor

En la columna derecha hemos comentado algunos conceptos sobre *servidores* y *clientes* que pueden serte útiles a la hora de analizar las diferencias entre estos dos tipos de lenguaje.

Lenguajes del lado del cliente

Diremos que un lenguaje es *del lado del cliente* cuando el *intérprete* que ha de *ejecutar* sus *scripts* es **accesible** desde éste –el *cliente*– sin que sea necesario hacer ninguna *peti- ción* al *servidor*.

Seguramente te ha ocurrido alguna vez que al intentar acceder a una *página web* ha aparecido un mensaje diciendo que *la correcta visualización de la página requiere un **plug-in** determinado*, y que, a la vez, se te haya ofrecido la posibilidad de *descargarlo* en ese momento.

Eso ocurre porque cuando el *navegador* –que en el caso de las *páginas web* es el *cliente*– trata de *interpretar* la página, encuentra **incrustado** en ella *algo* (un fichero de sonido, una animación *Flash*, etcétera) que –de forma muy similar a lo que ocurre con los *scripts*– requiere un *intérprete adecuado* del que no dispone en ese momento.

Cuando los *scripts* contenidos en un documento son de este tipo, el *servidor* lo *entrega al cliente* si efectuar ningún tipo de modificación.

Lenguajes del lado del servidor

Un lenguaje es *del lado del servidor* cuando la ejecución de sus *scripts* se efectúa, por *instancia* de este –el *servidor*–, **antes** de *dar respuesta a la petición*, de manera que el *cliente* no recibe el documento original sino el resultante de esa interpretación previa.

Cuando se usan estos tipos de lenguaje el *cliente* recibe un documento en el que cada *script* contenido en el original habrá sido

programas instalados en tu ordenador) envía a través de Internet solicitando una respuesta determinada por parte de un ordenador remoto.

¿Qué contendría esa petición?

Cada tipo de **petición** tendrá contenidos distintos. Por ejemplo, cuando se trata de *leer mensajes de correo*, la petición realizada por el **cliente** (*Outlook Express*) contendría, entre otros, muchos de los datos de la configuración de la cuenta, tales como: el **protocolo** (forma de comunicación) – en el caso del correo lo habitual sería el protocolo **POP** (**P**ost **O**ffice **P**rotocol)–, el nombre de **host** donde está alojado el *buzón* (servidor POP ó servidor de correo entrante), el nombre de la cuenta, la contraseña de acceso, y algunas otras informaciones relativas a la gestión de esa cuenta tales como si deben conservarse o no los mensajes en el servidor, etcétera.

¿Qué ocurre con esa petición?

Cualquier **petición** pasa en primera instancia por un *servidor de nombres de dominio* (Domain Name Server) **DNS**, una especie de guía telefónica que contiene los nombres de los servidores y las direcciones IP a través de las cuales están conectados a Internet. Podría decirnos –los datos son ficticios– que **olmo.cnice.mecd.es** es el nombre de un **host** que está conectado a Internet a través de la dirección IP **111.112.113.114**

Una vez *resuelta* esa petición por el *servidor* **DNS** (*direccionamiento* de la *petición* a la IP correspondiente) se comprobará si esa IP está activa (si efectivamente hay un ordenador conectado a través de ella) y, en caso de estarlo, se determinará si ese ordenador al que estamos accediendo *es capaz* de **atender la petición**.

¿Qué tiene que ocurrir para que pueda atenderse una petición?

Es necesario que el **ordenador remoto** tenga **instalado y funcionando** el **software de servidor adecuado al protocolo de nuestra petición**. Ello quiere decir -siguiendo con el ejemplo– que el ordenador remoto debe tener instalado y funcionando un software específico de **servidor de correo** capaz de interpretar el *protocolo POP3* especificado en la petición.

> **¡Cuidado!**
>
> El **ordenador remoto** debe tener **instalado y funcionando** el **software adecuado a cada tipo de petición (servicio) que deba atender**.
> No basta con decir **servidor**, es preciso conocer los *servicios* que presta y es factible que un mismo ordenador preste -simultáneamente- varios *servicios*, siempre que tenga *instalado* y *activo* el software específico para cada uno de esos *servicios*.

Cuando el *ordenador remoto* **acepta la petición** el *software de servidor* y/o las **aplicaciones del lado del servidor** (software instalado en el ordenador remoto y vinculado con el software de servidor) **resuelven la petición** (*comprobar que el nombre de la cuenta y la contraseña son correctas*, *comprobar si existen mensajes*, *borrarlos del buzón si así lo especifica la petición*, etc.) y **devuelven al cliente** (recuerda que el *cliente* era nuestro Outlook Express) la información requerida.

Solo falta que una vez recibida la **respuesta** *Outlook Express* (**cliente**) **interprete** la información recibida y nos permita visualizar o imprimir el contenido de los mensajes *descargados* del servidor.

Servidor y cliente en una misma máquina

Hasta ahora -al referirnos a servidores y clientes– hemos hecho alusión a dos *máquinas*: nuestro propio ordenador (*ordenador local*) en el que estarían instaladas las aplicaciones **cliente** y un *ordenador remoto* en el que se alojarían las aplicaciones de **servidor**. Eso es lo más *habitual*, pero *no es la única posibilidad*.

Dado que **servidor** y **cliente** son únicamente **aplicaciones** es perfectamente posible que ambas *convivan* dentro de la misma *máquina*.

La diferencia sustancial sería que ahora no es necesario el servidor de DNS para buscar la dirección IP. Utilizaríamos una IP (habitualmente la **127.0.0.1**) **reservada** para estos casos –preestablecida en la configuración del servidor– y a través de ella se canalizarían las *peticiones* a *nuestro propio servidor*. Ya hablaremos más adelante de esta IP.

Esquemas de diferentes *peticiones* de páginas WEB

sustituido por los resultados de su ejecución.

Esto es algo a tener muy en cuenta, porque, en este caso, los usuarios **no tendrán** la posibilidad de **visualizar el código fuente**, mientras que cuando se trata de *lenguajes del lado del cliente* siempre es posible visualizar los *scripts*, bien sea de forma directa –mirando el código fuente de la página recibida– o *leyendo* el contenido de ficheros externos –vinculados a ella– que son bastante fáciles de encontrar en la *caché* del navegador.

La utilización de este tipo de *scripts* requiere que el *intérpre- te* del lenguaje sea *accesible* –esté *del lado*– desde el propio *servidor*.

¿Cómo resuelve sus *dudas* el servidor?

Dado que en unos casos el servidor debe *entregar* el **documento original** –páginas estáticas o páginas dinámicas en las que se usan *lenguajes* del *lado del cliente*– mientras que en otros casos –páginas dinámicas usando lenguajes del *lado del servidor*– tiene que devolver el **resultado** de la ejecución de los *scripts*, es razonable que te preguntes: ¿cómo *sabe* el servidor lo que debe hacer en cada caso?

La respuesta es simple. Eso hay que *decírselo*. Y se le dice de una forma bastante simple. Se indica al poner la extensión al documento.

Si en la *petición* se alude a un documento con extensión **.htm** o **.html** el servidor entenderá que esa página no requiere la intervención previa de ningún *intérprete* de *su lado* y entre- gará la página *tal cual*.

Si en esa petición se aludiera a una extensión distinta –**.php**, por ejemplo– el servidor enten- dería que *antes de servir la página* debe *leerla* y requerir al intérprete de PHP que ejecute los scripts desarrollados en ese lenguaje (en caso de que los contuviera) y devolvería al cliente el **documento que resultara** de las eventuales ejecuciones de tales scripts.

Algunos lenguajes con *nombre y apellidos*

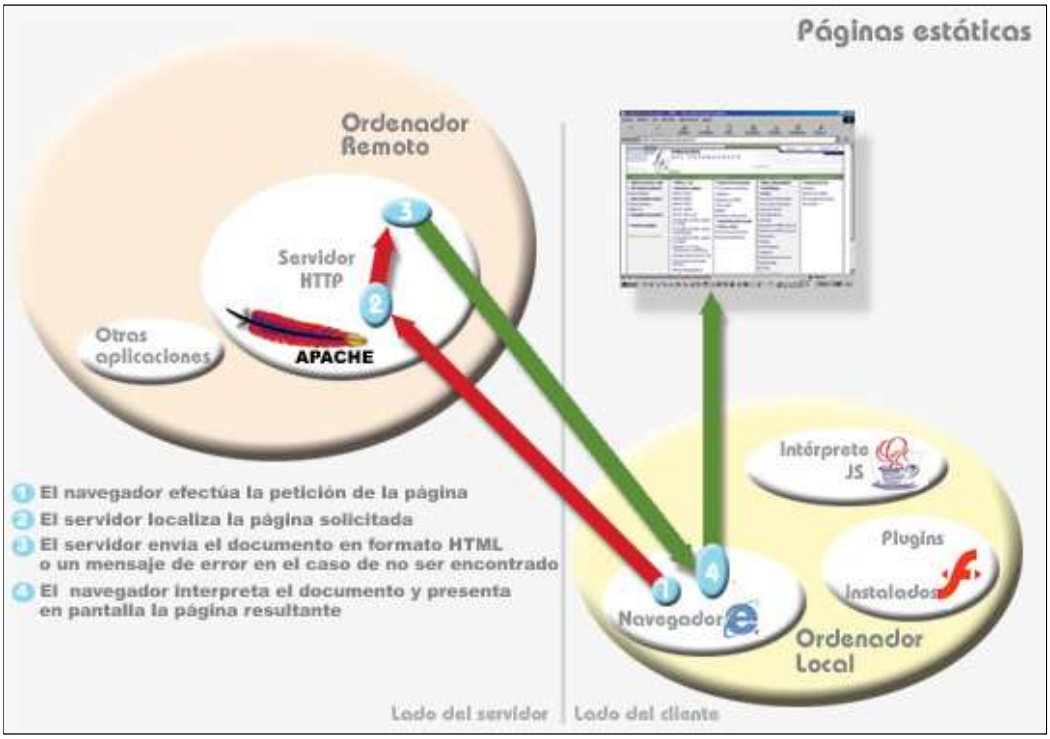
Sin pretender hacer una enu- meración exhaustiva, los *len- guajes* de *script* más populares son los siguientes:

Del lado del cliente

- DHTML
- JavaScript
- VBScript

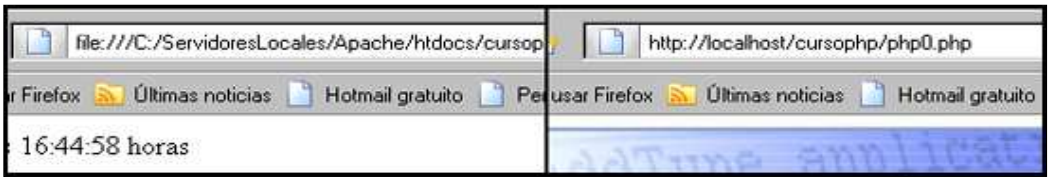
Intentaremos resumir de forma esquemática los procesos de algunos de los diferentes tipos de *peticiones* de páginas WEB.

Aquí tenemos la más sencilla de ellas:



Si observas con detenimiento el esquema de la parte superior es posible que encuentres algo *que no te cuadre*... porque en el esquema *hay un servidor* que *parece imprescindible* para atender las peticiones y sin embargo tú –**sin tener instalado ningún servidor**– eres capaz de visualizar tus propias páginas web sin más hacer un *doble click* sobre su icono.

Eso es cierto, pero fíjate en las dos direcciones que aparecen en esta otra imagen.



La de la izquierda –consecuencia de haber hecho doble click sobre el icono del documento– contiene como dirección una *ruta* (el *path* que conduce hasta el documento) mientras que en la de la derecha aparece el sintagma **http** al principio de la dirección.

En el primer caso no hemos hecho ninguna *petición de página web* sino que hemos abierto un documento cuya extensión (html) está asociada con Internet Explorer en nuestra configuración de Windows. El proceso ha sido exactamente el mismo que si hubiéramos hecho doble click sobre el icono de un documento con extensión **txt**, con la única salvedad de que en este último caso se habría abierto el *block de notas* (por la asociación de extensiones y aplicaciones en la configuración de Windows).

En el segundo caso las cosas son distintas. Se incluye el *sintagma* **http** – acrónimo de **H**iper**T**ext **T**ransfer **P**rotocol– para indicar que ese es el **protocolo** que debe ser utilizado y que será preciso que el servidor que reciba la **petición** *sea capaz de interpretarlo*. Por eso a los servidores que alojan páginas web se les suele llamar **servidores HTTP** y se les requiere que *soporten* este protocolo.

Siguiendo con los esquemas, he aquí el correspondiente a una petición de página en la que hay *incrustados scripts* escritos en *lenguaje del lado del cliente*:

DHTML no es exactamente un lenguaje de programación. Se trata más bien de una serie de capacidades que se han ido añadiendo a los navegadores *modernos* mediante las cuales las páginas pueden contener *hojas de estilo* y/o organizarse en *capas* susceptibles de ser redimensionadas, modificadas, desplazadas y/o ocultadas.

JavaScript es uno de los lenguajes más populares. Cada navegador incluye su propio intérprete y es frecuente que los resultados de visualización sean *algo* distintos según el navegador y la versión que se utilice.

Parece ser que las versiones más recientes de los distintos navegadores se aproximan a un estándar –*ECMA Script-262*– que ha sido desarrollado por la **ECMA** (Asociación Europea de Normalización de Sistemas de Información y Comunicación), lo que hace suponer que en un futuro muy próximo todos los navegadores se ajustarán a esa especificación y que, con ello, las páginas web ya se visualizarán de forma idéntica en todos ellos.

VBScript es un lenguaje de *script* derivado de *VisualBasic* y diseñado específicamente para los navegadores de *Microsoft*.

Del lado del servidor

Los más populares de este tipo son:
- PHP
- ASP
- Perl
- JSP

Cada uno de ellos tiene sus propias peculiaridades. Pero dado que aquí tratamos sobre **PHP** quizá sea conveniente –a modo de recordatorio– hacer algunas precisiones sobre los requisitos imprescindibles para trabajar con este lenguaje.

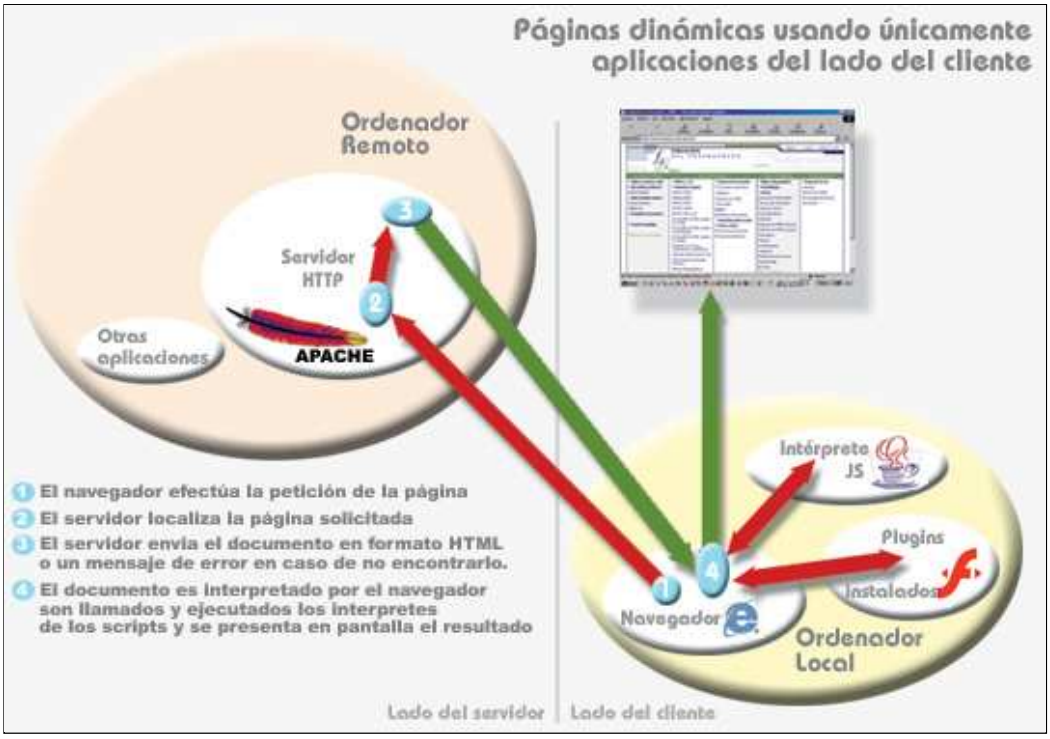
Requisitos para el uso del lenguaje PHP

De acuerdo a lo comentado en los párrafos anteriores y en los esquemas que tenemos a la derecha, el uso del lenguaje PHP requiere tener *instalado* y *configurado*:

- Un **software de servidor** –configurado para interactuar con el intérprete de **PHP**– que soporte el protocolo HTTP y que en nuestro caso será el denominado servidor **Apache**.

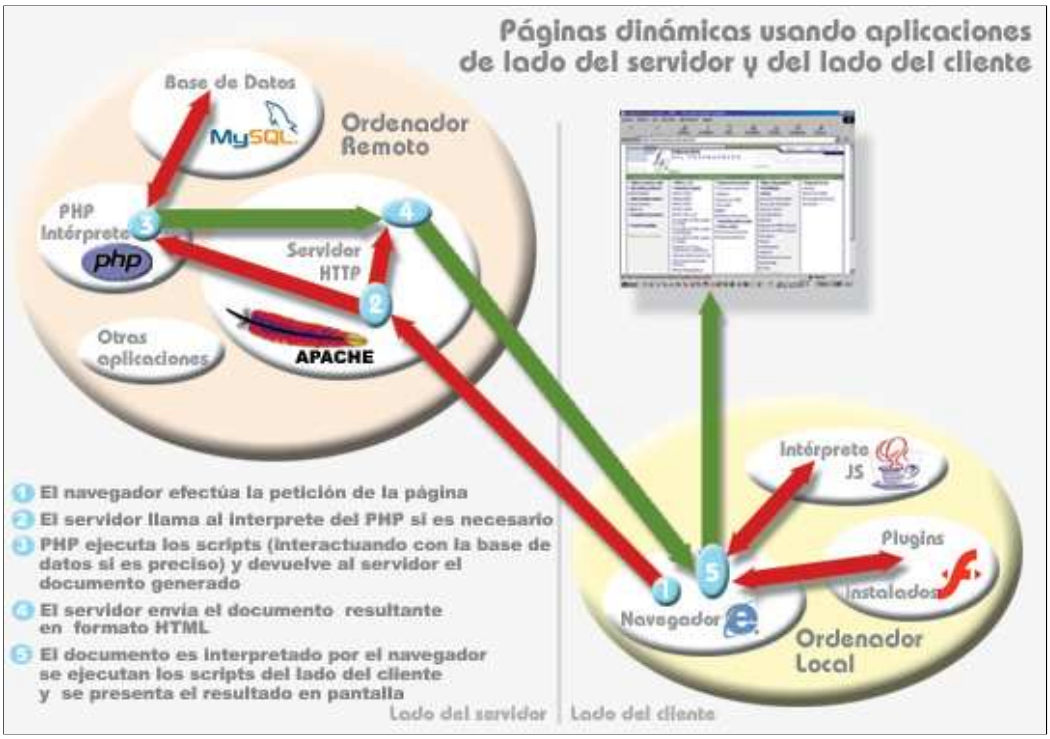
- El intérprete de **PHP**.

- Un software de servidor de **bases de datos** capaz de ser gestionado mediante funciones propias de **PHP**.



Como puedes observar no requiere *nada distinto* a lo del supuesto anterior. La diferencia sería que en este caso se *harían llamadas* al *intérprete de JavaScript* –incluido en los navegadores, tal como comentamos al margen– y/o a eventuales *plugins* necesarios para interpretar otros tipos de *script*.

Y por último, el esquema más complejo: un ejemplo de *convivencia* en un mismo documento de varios *scripts* y varios tipos de lenguaje.



Aquí ya es preciso que, además de un servidor capaz de soportar el protocolo **HTTP**, esté instalado –del lado del servidor– un intérprete PHP, un **servidor de bases de datos MySQL** y que, además, estén configurados de modo que puedan interactuar entre ellos.

El lenguaje **PHP** dispone de funciones que le permiten acceder a muy diversos tipos de *servidores de bases de datos* pudiendo: **crear**, **consultar**, **borrar** y **modificar** tanto bases de datos como tablas y registros de las mismas.

Nosotros vamos a utilizar **MySQL**, unos de los gestores más potentes y populares que existen en este momento.

Utilizaremos el servidor de bases de datos conocido como **MySQL**.

En las páginas siguientes trataremos sobre la forma de realizar los procesos de instalación y configuración de estas aplicaciones.

Diferentes servicios de *hosting*

Cuando empezamos a trabajar con páginas dinámicas elaboradas mediante scripts de **PHP** (lenguaje del lado del servidor) suele surgirnos las necesidad –o simplemente el deseo– de *publicarlas* en algún *espacio de alojamiento* (**hosting**), sea *gratuito* o *de pago*.

Si queremos publicar páginas en las que utilicemos **PHP** y bases de datos **MySQL** habremos de buscar un **hosting** que, aparte de espacio de alojamiento, nos ofrezca *estos dos servicios*. Además, antes de elegir uno deberíamos informarnos sobre *la funcionalidad* que nos ofrece, ya que es importante conocer no sólo las versiones de PHP y MySQL de que dispone sino también *las restricciones que puedan existir para su uso* (bastante frecuentes y por razones de seguridad en la mayoría de los casos).

Si decides publicar tus páginas no te precipites en la elección de un **hosting**. A lo largo de este curso te iremos dando algunas pautas que te permitirán hacer una elección acorde con tus necesidades.

Anterior Índice Siguiente