

INTERCAMBIO DE INFORMACIÓN ENTRE PÁGINAS WEB CON PHP

OBJETIVOS

- Descubrir el problema de http como protocolo *sin estado*
- Asimilar los mecanismos de PHP para intercambios datos entre diferentes páginas
- Identificar los problemas de las cookies y las sesiones
- Enumerar los riesgos del uso de cookies y sesiones
- Asimilar el funcionamiento de las cookies u/j sesiones
- Almacenar datos en cookies y archivos de sesión
- Leer datos de cookies y sesiones
- Borrar cookies y datos de sesión

CONTENIDOS

- 5.1 LIMITACIONES DEL PROTOCOLO HTTP**
- 5.2 FORMAS DE GENERAR UN ESTADO O SESIÓN**
 - 5.2.1 USO DE LA DIRECCIÓN IP
 - 5.2.2 PASO DE PARÁMETROS MEDIANTE CADENA DE CONSULTA
 - 5.2.3 PASO DE PARÁMETROS MEDIANTE MÉTODO POST
 - 5.2.4 COOKIES
 - 5.2.5 SESIONES
 - 5.2.6 BASES DE DATOS
- 5.3 USO DE COOKIES DESDE PHP**
 - 5.3.1 FUNCIONAMIENTO DE LAS COOKIES
 - 5.3.2 ALMACENAMIENTO DE COOKIES DESDE PHP. SETCOOKIE
 - 5.3.3 ACCEDER A LOS DATOS DE LAS COOKIES
- 5.4 USO DE SESIONES EN PHP**
 - 5.4.1 VENTAJAS Y DESVENTAJAS
 - 5.4.2 FUNCIONAMIENTO
 - 5.4.3 INICIO DE SESIÓN
 - 5.4.4 USO DE LA SESIÓN PREVIAMENTE INICIADA
 - 5.4.5 OBTENER DATOS DE LA SESIÓN
 - 5.4.6 USAR VARIABLES DE SESIÓN
 - 5.4.7 BORRAR DATOS DE LA SESIÓN
 - 5.4.8 ELIMINAR LA SESIÓN ENTERA
- 5.5 PRÁCTICAS RESUELTA**
- 5.6 PRÁCTICAS PROPUESTAS**
- 5.7 RESUMEN DE LA UNIDAD**
- 5.8 TEST DE REPASO**

5.1 LIMITACIONES DEL PROTOCOLO HTTP

El protocolo que se utiliza para navegar por la web es **http**. Con lo cual la programación de aplicaciones web depende de este protocolo. http es un protocolo de petición (**request**) y respuesta (**response**); básicamente un cliente hace una petición de recurso y un servidor http responde a esa petición.

El problema está en que http es un **protocolo sin estado** (o sin memoria), cada transacción es independiente de la anterior.

Al principio, los desarrolladores de aplicaciones web no estaban preocupados por esta circunstancia ya que, esencialmente, los servidores web servían páginas estáticas que contenían información simple. Además esa información era la misma para cada usuario. Sin embargo en la actualidad, debido a que desde http se sirven todo tipo de servicios, necesitamos otorgar **memoria** a nuestros desarrollos.

El caso más evidente para entender por qué necesitamos memoria está en una aplicación web que nos pida usuario y contraseña para después mostrarnos nuestra página personal dentro de ese servicio. Esta portada puede ser nuestro buzón de mensajes, nuestra pared en una red social, nuestro carrito de la compra, etc. La cuestión es que esa zona personal debe recordar quiénes somos y para ello lo que hacemos en la pantalla de conexión (**login**) debe de generar un **estado**.

Luego tenemos un problema. Si el protocolo http no tiene estado y necesitamos ese estado, la única manera es enviar la información necesaria para generar el estado dentro del propio paquete http. Como veremos a continuación hay numerosas técnicas para hacerlo, el problema estriba en que alguien pueda capturar el envío del paquete y hacerse con esa información confidencial.

5.2 FORMAS DE GENERAR UN ESTADO O SESIÓN

5.2.1 USO DE LA DIRECCIÓN IP

En PHP disponemos del array **\$_SERVER** que proporciona mucha información interesante. Así, en ese array, **\$_SERVER["REMOTE_ADDR"]** nos permite obtener la IP del cliente. En el caso, por ejemplo, de acceder a una zona de seguridad puede ser un método de autentificación. Pero no es un método seguro, ya que el usuario/a puede acceder desde servidores proxy o utilizando otras tecnologías que no nos permitan identificar con seguridad su dirección IP.

5.2.2 PASO DE PARÁMETROS MEDIANTE CADENA DE CONSULTA

Toda URL puede tener una cadena de consulta. Por ejemplo, si escribimos en nuestro navegador la url: <https://www.google.es/#q=PHP> abriremos la página de Google en España y auto-

máticamente se nos mostrarán páginas que, según Google, se relacionan con PHP. Usado con habilidad es un método interesante para pasar datos de una página a otra.

En realidad, esta forma de trabajar se relaciona con el método GET de los formularios. Así podemos enviar parámetros usando direcciones URL como por ejemplo la siguiente: <http://miservidor.com/prueba.php?id=2309&precio=24> de modo que a la página **prueba.php** le pasamos los parámetros **id** (con valor **2309**) y **precio** (con valor **24**). Es un método muy utilizado, pero tiene la desventaja de que se lee perfectamente en la barra de direcciones lo que estamos enviando, por lo que no es válido para enviar información confidencial (a no ser que la cifremos).

ACTIVIDAD 5.1:

- Crea una página capaz de recoger un nombre mediante un parámetro GET de cadena de consulta. Así si la página se llama **consulta.php**, la podremos invocar con el texto: **consulta.php?nombre=jorge**.
- Consigue que la página escriba **Hola** seguida del nombre que le mandes en la cadena de consulta.

5.2.3 PASO DE PARÁMETROS MEDIANTE MÉTODO POST

Los formularios también permiten pasar información entre páginas. Si ese paso se hace por GET, tenemos la desventaja de que los datos se ven en la barra de dirección, al igual que hemos visto en el apartado anterior. Si usáramos POST, en lugar de GET, para pasar información entre páginas, los datos no se ven en la barra de dirección, sino que viajan en la cabecera del paquete http (véase Figura 3.6 en la página 150).

Muchas veces se usan controles de formulario ocultos para pasar información adicional sin que el usuario la perciba. El truco para ello es utilizar controles de formulario con el código HTML: **<input type="hidden">**, de modo que el usuario no los vea y cuando los datos visibles se envíen, también se estarán enviando los datos ocultos.

ACTIVIDAD 5.2:

- Prueba a crear un formulario que tenga un control visible y un control oculto. En el control visible pide por ejemplo el nombre al usuario. En el oculto almacena el texto “**No me ves**”.
- Haz una página que recoja estos datos vía POST y comprueba que recibes ambos valores mostrándolos por pantalla.

5.2.4 COOKIES

Se explica en detalle en esta misma unidad. La idea es que en el ordenador del usuario se almacena la información que necesitamos en forma de archivo de texto.

En realidad una cookie es un par nombre/valor (al igual que los parámetros GET o POST), de modo que a un determinado nombre o clave le asignamos un valor (que siempre es texto).

Lo malo es que el usuario puede bloquear las cookies o borrarlas y eso está fuera de nuestro control por lo que es una técnica que tiene también sus riesgos. Además hay técnicas para intentar obtener las cookies del usuario con la finalidad de conseguir información confidencial. Por este último tema, las cookies son motivo de controversia, de hecho, **estamos legalmente obligados a avisar al usuario de que estamos grabando cookies en su ordenador.**

5.2.5 SESIONES

En realidad todas las técnicas anteriores sirven para generar información de sesión. Entendiendo por sesión la actividad que tiene un usuario en su navegador desde que lo abre hasta que lo cierra.

Sin embargo cuando en PHP se habla de sesiones, se denomina así a la técnica, que se explica en detalle más adelante, por la que se almacenan datos de usuario, al estilo de las cookies, pero en el servidor. La técnica se basa en que al usuario se le asigna un **identificador de sesión**, por el cual relacionaremos su archivo de sesión en el servidor.

Ese número de sesión es el cuello de botella de la seguridad en esta técnica, obtenerlo por parte de terceros es una actividad habitual de aquellos que intentan un **robo de sesión (o hijacking)** sobre un usuario, para hacerse con su información confidencial.

5.2.6 BASES DE DATOS

El uso de esta técnica se explica en detalle en la siguiente unidad. Es el método habitual de almacenaje permanente de información. Se trata de que desde PHP accedemos a un sistema gestor de bases de datos encargado de almacenar la información que necesitamos.

Lo bueno es que toda la gestión y mantenimiento de los datos lo realiza un software especializado en ello, descargando a PHP de esa responsabilidad. Lo malo es que para datos de sesión, se convierte en una técnica difícil de mantener, por lo que en la práctica se combinan las sesiones y las bases de datos.

5.3 USO DE COOKIES DESDE PHP

5.3.1 FUNCIONAMIENTO DE LAS COOKIES

Son indudablemente la opción más habitual para almacenar información del usuario. Lo malo es lo ya comentado: el cliente de un sitio web puede prohibir o borrar las cookies almacenadas, con lo que no tenemos la seguridad de que esta técnica funcione en todos los clientes.

Las cookies son archivos de texto que almacenan información sobre el usuario y que se guardan en el propio ordenador del usuario. La información almacenada asocia un valor (de tipo string) a un nombre. Al igual que el paso por GET o POST, el uso de cookies es otra técnica de almacenaje de información mediante pares nombre/valor.

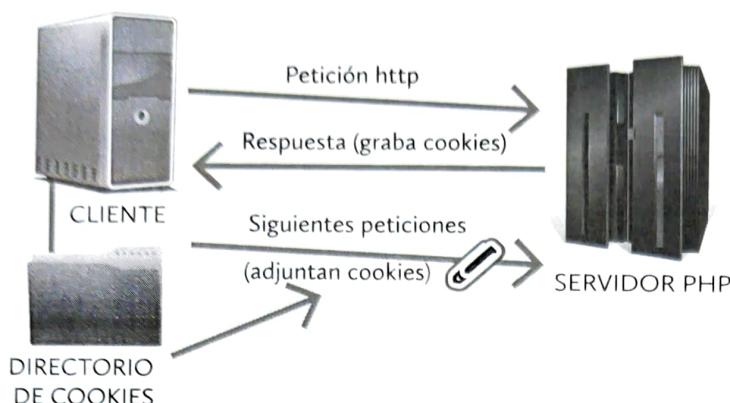


Figura 5.1: Esquema de funcionamiento de las cookies

Las cookies se pasan en la cabecera de la petición http por lo que su información viaja en el mismo paquete. En el momento que el servidor decide grabar los datos de la cookie, esta viaja con el paquete en todas las peticiones y respuestas de esa sesión.

La idea conceptual es la que refleja la Figura 5.1; en cuanto, desde una página PHP, grabamos datos mediante cookies, nuestras peticiones y respuestas llevan anexas los datos de las cookies que hemos almacenado y, por lo tanto, podremos utilizar en cualquier momento dichos datos.

La captura de pantalla muestra la sección 'Headers' de las herramientas de desarrollo de Google Chrome. Se detallan los siguientes encabezados:

- General:**
 - Remote Address: [::1]:80
 - Request URL: http://localhost/pruebas/otra.php
 - Request Method: GET
 - Status Code: 200 OK
- Response Headers:**
 - Connection: Keep-Alive
 - Content-Length: 245
 - Content-Type: text/html; charset=UTF-8
 - Date: Sun, 07 Jun 2015 15:57:26 GMT
 - Keep-Alive: timeout=5, max=98
 - Server: Apache/2.4.10 (Win32) OpenSSL/1.0.2i PHP/5.6.3
 - X-Powered-By: PHP/5.6.3
- Request Headers:**
 - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
 - Accept-Encoding: gzip, deflate, sdch
 - Accept-Language: es-ES,es;q=0.8,en;q=0.6
 - Connection: keep-alive
 - Cookie: nombre=Jorge; apellido1=Sánchez
 - Host: localhost
 - Referer: http://localhost/pruebas/pruebacookie.php
 - User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.81 Safari/537.36

Figura 5.2: Extracto de la cabecera http de un paquete que contiene cookies, visible a través de las herramientas de inspección de Google Chrome

5.3.2 ALMACENAMIENTO DE COOKIES DESDE PHP. SETCOOKIE

La función **setcookie** es la encargada de añadir (o borrar) una nueva cookie. Sintaxis:

```
setcookie(nombre [,valor [,expiración [,ruta [,dominio [,segura
[soloHttp]]]]]] )
```

Los parámetros son:

- **nombre.** Es el nombre que le damos a la cookie. Después podremos consultarle para poder saber su valor.
- **valor.** Valor que le damos a la cookie. Si no indicamos valor alguno, entonces estaremos borrando la cookie.
- **expiración.** Por defecto toma el valor cero, que significa que la cookie caduca en cuanto se cierre el navegador con el que se creó. Otra posibilidad es indicar una fecha en formato Unix. Por ejemplo, `time() + 600` indicaría que la cookie expira dentro de 10 minutos.
- **dominio.** Si no se indica nada toma como dominio en el que se aplica la cookie el dominio actual, pero se puede indicar otro.
- **segura.** Por defecto vale `false`, si indicamos `true`; entonces indica que la cookie solo se almacena si estamos comunicándonos mediante un protocolo seguro.
- **soloHttp.** Disponible desde la versión 5.2 de PHP. Es un valor booleano que indica que la cookie solo está disponible mediante el protocolo `http` y no, por ejemplo, desde `JavaScript`. La idea es paliar, en cierta forma, problemas de seguridad al hacer que la cookie no se pueda recoger en un ataque de tipo *Cross Side Scripting (XSS)*.

Ejemplos:

```
setcookie("visitas",1);
//graba la cookie visitas con valor 1 y que caduca con esta sesión
setcookie("usuario","andrei",time() + 60 * 60 * 24)
//graba la cookie con nombre usuario y valor andrei que
//caducará al día siguiente
```

5.3.3 ACCEDER A LOS DATOS DE LAS COOKIES

El array global `$_COOKIE` permite acceder a las cookies almacenadas. Para acceder al valor de una cookie concreta, basta indicar el nombre de la cookie como índice de este array. Por ejemplo, `$_COOKIE["visitas"]` devolvería el valor de la cookie con nombre *visitas* (suponiendo que esté definida dicha cookie). Ejemplo:

```
if(isset($_COOKIE["visitas"])){
    echo "esta es tu visita número".$_COOKIE["visitas"];
}
```

5.3.3.1 BORRAR COOKIES

Cuando se vuelve a utilizar la función `setcookie` indicando el nombre de una cookie ya existente y un nuevo valor, el nuevo valor sobrescribe el anterior valor que tuviera dicha cookie. Bajo la misma idea, si usamos `setcookie` indicando un nombre de cookie existente y el valor cero o `false`, entonces, se eliminará la cookie con ese nombre.

Ejemplo:

```
setcookie("visitas", false); //elimina la cookie de nombre visitas
```

Podemos también eliminar cookies más antiguas indicando una fecha de caducidad anterior a la actual, por ejemplo:

```
setcookie("visitas", false, time() - 60 * 60 * 24 * 7);
/* elimina la cookie de nombre visitas con fecha de caducidad de
hace al menos una semana */
```

5.3.3.2 ALMACENAR DATOS BINARIOS EN COOKIES

En las cookies no se pueden almacenar datos binarios. Las cookies solo admiten valores de tipo string. Es decir, no es válido el código:

```
setcookie("notas", array(9,7,6,5)); //inválido: datos binarios
```

Por ello, todos los programadores para almacenar datos binarios, como los arrays, en las cookies, crean mecanismos para convertirlos en texto. En general al proceso de convertir datos binarios en formato de texto, se lo conoce bajo el término de **serializar**; traducción excesivamente literal del inglés **serialize**.

Podemos idear nuestro propio mecanismo de *serialización*, pero en PHP hay una función que nos ayuda a automatizar este proceso. Se trata de la función **serialize**. A esta función se le pasa un dato binario y se convierte a un formato de texto equivalente. De ese modo, si queremos almacenar en una cookie llamada **notas** un array llamado **\$array1**, la instrucción sería:

```
setcookie("notas", serialize($array1));
```

El problema ahora es que cuando lo queramos leer, tenemos el problema contrario: es decir, convertir el texto en el binario correspondiente. Ese proceso es incluso más difícil; pero PHP lo facilita mediante la función contraria: **unserialize**. Esta función realiza el proceso inverso a **serialize**. Para extraer el array del ejemplo anterior escribiríamos este código:

```
$array1=unserialize($_COOKIE["notas"]);
```

5.4 USO DE SESIONES EN PHP

5.4.1 VENTAJAS Y DESVENTAJAS

PHP permite automatizar la gestión de sesiones. Las sesiones, comparadas con las cookies ofrecen estas ventajas:

- Pueden funcionar aunque el usuario/a desactive la posibilidad de cookies. Aunque hay que tener en cuenta que para ello debemos pasar el identificador de sesión mediante GET; es decir, el identificador de la sesión será visible en la barra de direcciones del

navegador. Además, por defecto PHP usa cookies para pasar el identificador de sesión, si no lo queremos pasar por cookies, debemos modificar varios parámetros del archivo de configuración de PHP.

- Almacenan más información que una cookie y, además, son capaces de almacenar datos binarios.
- Son más seguras puesto que los datos que se almacenan de la sesión lo hacen en el servidor y no en el cliente. En principio, la seguridad de los servidores es mayor que la de los ordenadores de los usuarios

Desventajas:

- Son susceptibles al ataque conocido como **secuestro de sesión** (*session hijacking*), mediante el cual un usuario se puede hacer con el identificador de sesión de otro usuario y hacerse pasar por él. El paso de ese identificador es el cuello de botella del sistema de sesiones.
- El identificador de sesión se almacena normalmente mediante cookies, por lo que si no nos damos cuenta, podríamos tener la misma dependencia con el usuario que en el caso de las cookies. Si el usuario las desactiva, no podremos utilizar sesiones.

5.4.2 FUNCIONAMIENTO

El manejo de sesiones se basa en asignar un identificador a cada sesión. PHP dispone de la posibilidad de calcular dicho número de sesión para que sea único en cada sesión de usuario. Tras lo cual podemos identificar de formas única a cada sesión. Ese identificador es el que se relacionará con los datos del usuario.

Los pasos son los siguientes:

- [1] Cuando deseamos generar una nueva sesión, se invoca a la función **session_start()**. Esta función genera un nuevo identificador de sesión (SESSION_ID).
- [2] Ese identificador de sesión se almacena en una cookie o bien se envía como parámetro GET dentro de cada petición del cliente. Que se utilice uno u otro método dependerá de la configuración de PHP.
- [3] Asociado a ese identificador de sesión, se crea un archivo en el servidor en el que se almacenarán físicamente los datos de la sesión.
- [4] Si queremos grabar un dato de sesión, debemos utilizar el array **\$_SESSION** al que le indicaremos el nombre y el valor (que puede ser binario) a almacenar.
- [5] En una página en la que deseemos recoger datos almacenados de la sesión actual, debemos invocar primero a **session_start()** y luego, mediante el array **\$_SESSION**, podremos leer, modificar o añadir datos de la sesión.
- [6] La sesión finaliza automáticamente cuando el usuario cierra el navegador.

Veamos el proceso con un ejemplo. Este código crea una página que inicia una sesión y graba en ella un parámetro llamado **nombre** asociado al valor **Jorge**:

```
<?php
    session_start();
    $_SESSION["nombre"] = "Jorge";
?>
<!doctype html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <p>Se ha iniciado la sesión</p>
    <a href="pagina2.php">Ir a la página 2</a>
</body>
</html>
```

En este código se crea una nueva sesión y se asigna en ella el nuevo valor **Jorge** asociado al identificador **nombre**. Además, se muestra el texto **Se ha iniciado la sesión** y un enlace a la página **pagina2.php**.

Supongamos que hacemos clic en el enlace y que el código de **pagina2.php** es este:

```
<?php
    session_start();
?>
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <?= "Hola ". $_SESSION["nombre"] ?>
</body>
</html>
```

Se mostraría en pantalla el texto **Hola Jorge**, resultado de acceder correctamente a los datos de la sesión.

Lo fundamental a recordar es el uso de la función **session_start** cada vez que utilicemos sesiones en PHP. Esta función gestiona el paso del identificador de sesión, además, retorna verdadero si la sesión se pudo realizar. Otro detalle es conseguir que esa instrucción sea la primera en el

código PHP, la razón es que es una instrucción que maneja cabeceras del protocolo http, y por ello, no tiene sentido colocarla en el cuerpo con el riesgo de que no se ejecute debidamente.

En realidad las sesiones son más sencillas de manejar que las cookies. Pero queda pendiente la cuestión sobre cómo se pasa el identificador de sesión de petición en petición. Las dos posibilidades son:

- **Mediante una cookie.** En este caso en cuanto se genera la sesión mediante la función `session_start`, se crea una cookie cuyo nombre es el nombre de la sesión y cuyo valor es el identificador de la sesión. En PHP es la opción por defecto. Como se ha comentado en el apartado dedicado a las cookies, éstas se envían en la cabecera del paquete http, por lo que se pueden interceptar en el caso de que un tercero tenga acceso a los datos que estamos enviando mediante un analizador de paquetes o *sniffer*. La solución más inmediata es que la comunicación esté cifrada de alguna manera, por ejemplo, con ayuda del protocolo https.

Utilizando el código anterior, el proceso de funcionamiento de la sesión sería el siguiente:

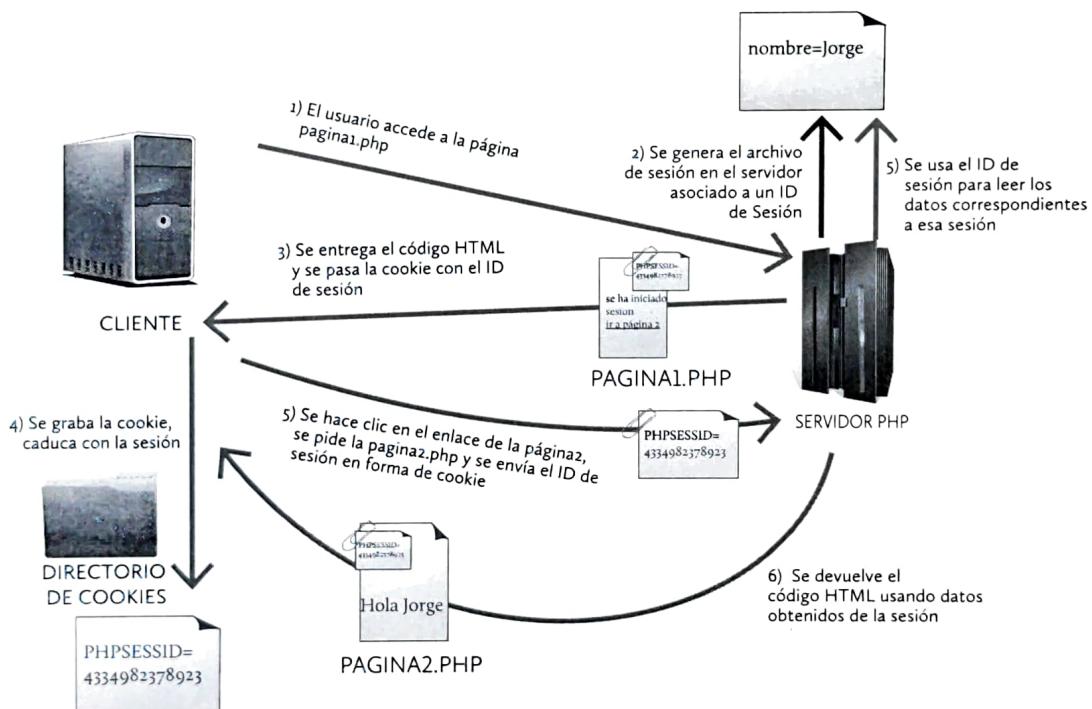


Figura 5.3: Funcionamiento de una sesión gestionada a través de una cookie

Para que esta opción sea la que funcione (si no fuera así) hay que modificar el archivo de configuración `php.ini` y colocar estos valores:

```
session.use_cookies 1
session.use_trans_sid 0
```

- **Mediante GET.** En este caso se pasa el identificador de la sesión acompañando a cada petición añadiendo una cadena de consulta GET a la URL. Es decir, que si nuestra sesión

tiene el nombre habitual de **PHPSESSID**, con el identificador de sesión **8FR45237A89** y estamos en la URL <http://prueba.com/pog1.php>, la sesión provoca una URL parecida a esta:

<http://prueba.com/pog1.php?PHPSESSID=8FR45237A89>

Para que sea esta la opción que está funcionando, deberemos modificar el archivo php.ini y hacer estos cambios:

```
session.use_cookies 0
session.use_trans_id 1
```

El esquema de funcionamiento de sesiones con identificador en la URL sería el siguiente:

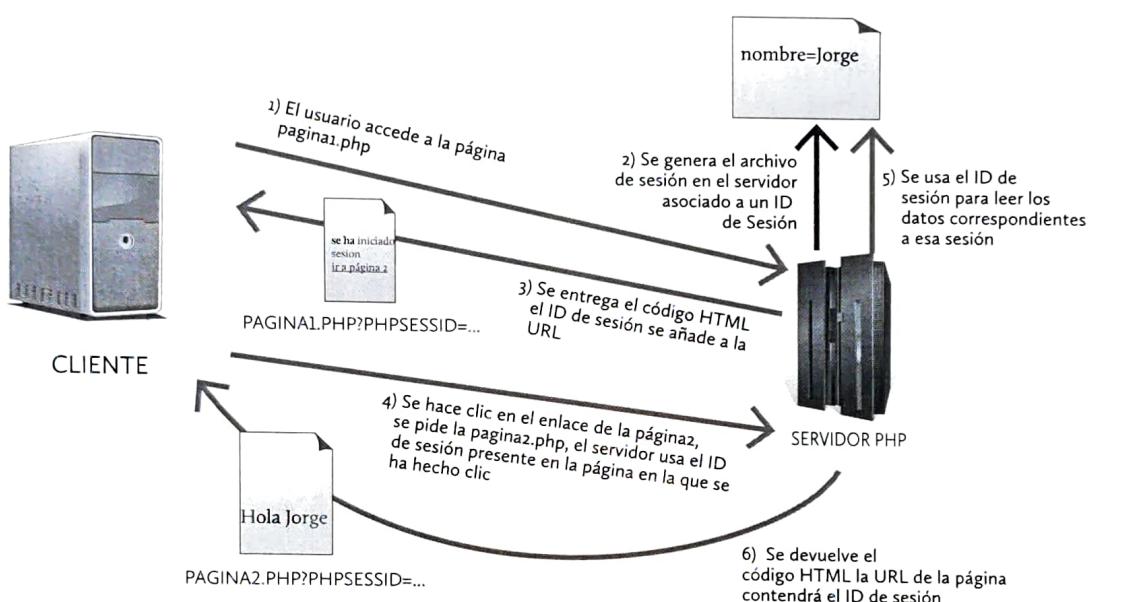


Figura 5.4: Funcionamiento de una sesión gestionada a través de una cookie

Normalmente, la sesión caduca en cuanto el usuario abandona el navegador. Se puede hacer (sobre todo si el identificador es una cookie) que dure más, pero no es lo recomendable. Las sesiones deben caducar cuando el usuario cierra la sesión, de otro modo hay más posibilidad de inseguridad. Es la forma habitual de usar sesiones.

Si necesitamos que los datos de la sesión perduren, lo lógico es usar cookies o bases de datos que son las técnicas de almacenamiento de información que están pensadas para esa finalidad.

5.4.3 INICIO DE SESIÓN

Una sesión comienza en la primera página PHP que invoque a la función **session_start**. Esta función no tiene argumentos, simplemente inicia la sesión si no ha sido iniciada, o bien permite el acceso a los datos de la sesión si ésta había sido iniciada anteriormente. Iniciar la sesión implica:

- [1] Crear un identificador aleatorio de sesión; salvo que a través de la función **session_id** (se explica más adelante) seamos nosotros los que generemos un identificador.

- [2] Crear en el servidor un archivo asociado a ese identificador, en el que se almacenarán los datos de la sesión.
- [3] Crear la variable superglobal `$_SESSION` que es un array (al estilo de `$_GET`, `$_POST` o `$_COOKIE`) en el que se almacenarán los datos de la sesión.

5.4.4 USO DE LA SESIÓN PREVIAMENTE INICIADA

Si la sesión ya había sido iniciada cuando se invocó a `sesión_start`, entonces, no se crea una nueva sesión sino que:

- Se sigue utilizando la sesión anterior
- Se permite el uso del array `$_SESSION` que contendrá los datos de la sesión, los cuales pueden ser modificados, leídos e incluso podremos añadir más datos.

RECOMENDACIÓN

- En resumen, siempre que deseemos trabajar con sesiones debemos utilizar la función `session_start`.
 - Como esta función manipula cabeceras http, debemos intentar que sea el primer código de la página PHP, antes incluso que la etiqueta HTML de cabecera, `!DOCTYPE`.
-

5.4.5 OBTENER DATOS DE LA SESIÓN

Dos funciones permiten obtener (y cambiar) los datos de la sesión:

- `session_id`. Si la invocamos sin parámetros, devuelve el identificador de sesión actual. Admite que le pasemos un identificador de sesión nosotros, con lo que ese se convierte en el identificador de la sesión actual. Poner números de sesión propios es peligroso, en cuanto a que si no tenemos un buen algoritmo de cálculo del identificador podríamos tener dos sesiones con el mismo identificador y, por lo tanto, indistinguibles y, lo que es peor, podríamos provocar que una sesión vea los datos de otra.
- `session_name`. Si la invocamos sin parámetros, devuelve el nombre de sesión actual (por defecto la sesión se llama `PHPSESSID`). Admite que le pasemos un nombre de sesión nosotros. Esto es útil para que no sea tan claro que estamos usando sesiones PHP ya que el nombre de sesión por defecto: `PHPSESSID`, es conocido por todos los programadores PHP, lo que facilita el robo de la sesión.

Otra opción para cambiar el nombre de los identificadores de sesión es utilizar la directiva `session.name` dentro del archivo de configuración de PHP (normalmente `php.ini`). A esta directiva se le puede asignar otro nombre para las sesiones.

5.4.6 USAR VARIABLES DE SESIÓN

Una variable de sesión permite asociar una clave a un valor de la sesión actual.

Para almacenar un dato de sesión basta usar esta sintaxis:

```
$_SESSION[“clave”]=valor
```

Para recoger un valor almacenado se utiliza el mismo array indicando la clave o nombre del valor que deseamos recoger. Por ejemplo:

```
$variable=$_SESSION[“clave”]
```

Como hemos comentado, es posible incluso almacenar datos binarios dentro de las variables de sesión, lo que convierte a las sesiones un método de trabajo más versátil que las cookies.

5.4.7 BORRAR DATOS DE LA SESIÓN

Si deseamos eliminar un dato de la sesión basta con usar la función PHP de eliminación de variables **unset**. De modo que si deseamos eliminar los datos de la sesión asociados al nombre o clave *visitas*, haremos lo siguiente:

```
unset($_SESSION[“visitas”])
```

Para eliminar todos los datos, hay que hacer esta secuencia:

```
unset($_SESSION);  
$_SESSION=array();//imprescindible, sino la sesión queda  
inutilizable
```

5.4.8 ELIMINAR LA SESIÓN ENTERA

Eliminar la sesión de forma absoluta, consiste en:

```
session_start(); //si no la hubiéramos invocado ya  
unset($_SESSION); //elimina el array, sino se queda gastando memoria  
session_destroy();//función de eliminación en sí de la sesión
```

Para almacenar un dato de sesión basta usar esta sintaxis:

```
$_SESSION[“clave”]=valor
```

Para recoger un valor almacenado se utiliza el mismo array indicando la clave o nombre del valor que deseamos recoger. Por ejemplo:

```
$variable=$_SESSION[“clave”]
```

Como hemos comentado, es posible incluso almacenar datos binarios dentro de las variables de sesión, lo que convierte a las sesiones un método de trabajo más versátil que las cookies.

5.4.7 BORRAR DATOS DE LA SESIÓN

Si deseamos eliminar un dato de la sesión basta con usar la función PHP de eliminación de variables **unset**. De modo que si deseamos eliminar los datos de la sesión asociados al nombre o clave **visitas**, haremos lo siguiente:

```
unset($_SESSION[“visitas”])
```

Para eliminar todos los datos, hay que hacer esta secuencia:

```
unset($_SESSION);  
$_SESSION=array(); //imprescindible, sino la sesión queda  
inutilizable
```

5.4.8 ELIMINAR LA SESIÓN ENTERA

Eliminar la sesión de forma absoluta, consiste en:

```
session_start(); //si no la hubiéramos invocado ya  
unset($_SESSION); //elimina el array, sino se queda gastando memoria  
session_destroy(); //función de eliminación en sí de la sesión
```