COOKIES Y SESIONES

Índice

COOKIES Y SESIONES

1. INTRODUCCIÓN.

Como bien sabemos HTTP es un protocolo sin estado (stateless), eso provoca que no podamos saber de que página venimos. Cada vez que visitamos una página, un servidor es el encargado de responder nuestra petición. Cada solicitud que realizamos y la respuesta que recibimos, son totalmente independientes por naturaleza.

Por esto, hasta ahora, cada vez que queríamos pasar un dato de una página a otra, teníamos que usar hidden para enviarlo. Es más, en el caso de datos complejos, no sólo utilizamos hidden sino que además nos debemos encargar de serializarlos y deserializarlos.

Esta técnica en los albores de internet, podría ser útil, pero era costosa de mantener y engorrosa de programar. Por tanto, PHP como muchos otros lenguajes usa otros sistemas de envío/compartición de información entre páginas. Las cookies y sesiones.

2. COOKIES

Una cookie es un par clave/valor que tiene una fecha de caducidad y un dominio de validez que se guarda en el ordenador del usuario y se envía (a través de una cabecera HTTP especial) a cada página a la que pueda acceder.
Gracias a las cookies, puede identificarse con credenciales de seguridad un usuario que accede a una página, o guardar los datos para su posterior recuperación. Por ejemplo, podríamos guardar una cookie con un valor que indica la última página visualizada, para redirigir al usuario a la última página mostrada si se conecta de nuevo.
Características principales:

- Las Cookies son archivos que permiten guardar datos acerca de nuestras preferencias.
- Estos archivos se guardan en el **cliente,** de hecho es el navegador el encargado de almacenarlas.
- En un inicio, cuando recién se crearon, guardaban únicamente información bien puntual. Por ejemplo, en qué lenguaje queremos ver una página web determinada (suponiendo que esté disponible en varios idiomas).
- Para que la página pueda recordar nuestro idioma preferido, guarda una Cookie en nuestra computadora (esto lo gestiona el navegador web; por ejemplo Chrome, Firefox o Safari).
- De esta forma, cuando visitamos una página, esta puede leer las Cookies que tenemos registradas. Las Cookies pueden guardar cualquier dato que el servidor considere importante de recordar. Por ejemplo, la última fecha en que visitamos una página, los productos que hemos cargado a un carrito de compras, los enlaces a los que hicimos clic, etcétera.
- Si una página solicita la creación de una Cookie, entonces la Cookie se registra asociada a dicha página, y no puede ser consultada por una página distinta.
- Aunque en un inicio guardaban información bien puntual, con el paso del tiempo, los datos almacenados en Cookies fueron creciendo a fin de brindar una mejor experiencia de usuario (en base a las propias elecciones de cada usuario o en base a un análisis de su comportamiento en Internet).
- Debido a que las Cookies tienen un límite de tamaño, cuando se tiene mucha información asociada a cada usuario, lo que se hace es guardar la información en el servidor, y guardar en una Cookie únicamente un identificador (que le permita al servidor saber de quién se trata).
- Es importante tener en cuenta que, al visitar un sitio web, este puede contener fragmentos de otros sitios. Por ejemplo, muchos sitios presentan anuncios (que se sirven desde un servidor distinto). En este caso, tanto la página que visitamos, como el servidor que sirve los anuncios, pueden crear Cookies.

- Muchas veces una misma plataforma de anuncios es la que se encarga de proveer de anuncios a una gran cantidad de páginas. En esos casos, es posible que la plataforma haya desarrollado una estrategia para captar información de todas las páginas que muestran sus anuncios, a fin de mostrarnos anuncios en función a las páginas que más frecuentamos.
- ¿Las Cookies son malas? No exactamente. Todo depende de los creadores de la página: la información que almacenan en Cookies y el uso que se le da a esta información.

La creación de una cookie es una operación muy simple que se puede hacer en PHP utilizando la función **setcookie()**. Esta función tiene un número variable de parámetros. En orden:

- El nombre de la cookie.
- El valor de la cookie, que debe ser necesariamente un valor escalar (entero o una cadena, una matriz no se puede guardar directamente)
- Un número que indica la fecha de caducidad de la cookie. Si este número es 0 o no se especifica, la cookie durará hasta que el usuario cierra su navegador. En el caso de que el timestamp tenga una fecha anterior a la actual, se borrará la cookie.
- El path de validez de la cookie.
- El dominio de validez de la cookie.
- Un parámetro booleano que indica si las cookies se transmitirán sólo a través de una conexión segura HTTPS.

Dado que la función **setcookie()** genera un encabezado HTTP de manera explícita, es necesario en primer lugar que su uso no haya sido impreso (con echo, print o cualquier otro método de salida), de lo contrario se genera un error. Incluso una línea en blanco al principio del archivo antes de la etiqueta de apertura de PHP dará lugar a la generación de este error.
Algunos ejemplo de creación de una cookie:

```
//esta cookie dura una hora
setcookie('prueba_cookie', 'valor de la cookie',  time() + 3600);
//esta cookie dura hasta que el usuario cierre el navegador
setcookie('prueba_2', 'adios');
```

Para eliminar una cookie, debe utilizar la misma función utilizando los mismos parámetros utilizados en el proceso de la creación, pero con el uso de una fecha anterior a la actual.

```
setcookie ('prueba_cookie', ",time()-3600);
```

Una vez que se crea una cookie su valor será accesible a través de:
$ _COOKIE ['nombre_cookie']
en las siguientes páginas a la actual, en el supuesto de que la fecha de vencimiento no haya pasado y que cumpla con las restricciones del dominio y carpeta.
Es una buena práctica no crear demasiadas cookies, ya que los navegadores tienen un límite relacionado con un dominio específico y una carpeta específica. En caso de tener que mantener una gran cantidad de valores, lo mejor es guardarlos en una base de datos o un archivo.
Las cookies son fácilmente recuperables y de fácil lectura, por lo que es importante no guardar información privada o vital, excepto con el correspondiente cifrado.

2.1. Como crear una Cookie

Lo vamos a ver con un ejemplo muy simple

Cookie1.php

```
<html>
      <head>
            <title>Ejercicio</title>
            <meta charset = "UTF-8" />
```

```
	</head>
	<body>
		<form action="cookie1b.php" method="post" >
			Nombre: <input type="text" name="nombre" />
			<input type="submit" value="Enviar!!" />
		</form>
	</body>
</html>
```

Cookie1b.php

```
<?
$nombre = $_POST['nombre'];
setcookie('nombre', $nombre, time()+4800);
echo $_COOKIE['nombre'];
?>
```

2.2. Como leer una Cookie

Si te fijas en el ejemplo anterior, ya sabemos leer una cookie. Es tan simple como usar el array superglobal $_COOKIE ['nombre_cookie'] (¿te suena esta estructura? Nahhhhhhhhhh)
Así que seguimos con los ejemplos básicos.

```
<html>
	<head>
		<title>Ejercicio</title>
		<meta charset = "UTF-8" />
	</head>
	<body>
		<?php
			if (isset($_COOKIE['nombre'])) {
				echo "La cookie tiene el valor: ".$_COOKIE['nombre'];
			} else {
				echo "La cookie no ha podido ser encontrada!!";
			}
		?>
		<a href="cookie1d.php" >Salir del Sistema</a>
	</body>
</html>
```

Ejercicio Creación y lectura de una cookie
Crearemos un formulario que solicite la carga del nombre de usuario. Cuando se presione un botón creará una cookie para dicho usuario. Cada vez que ingresemos al formulario, este mostrará el último nombre de usuario ingresado.

2.3. Borrar una Cookie

Hay que tener claro una cuestión con respecto a las cookies: **se crean en el cliente** en su navegador. Entonces para borrarla deberíamos acceder al disco duro de la persona conectada y claro, estaría feo andar accediendo a discos que no son nuestros (intentar explicárselo google)
Por tanto en realidad no puedo **"borrar"** la cookie, al menos no puedo acceder al disco del usuario para borrarla. Pero lo que sí puedo hacer es decirle al navegador que la borre él. ¿Cómo? Muy simple, indicándole que dicha cookies esta caducada. (¿Quién no se ha comido una cookie caducada? El que dice una cookie dice un yogur... pero me desvío...)
Me repito la técnica es simple ponerle una fecha pasada a la cookie y automáticamente el navegador la borra.

Ejemplo.

```
<?php setcookie ('nombre', $_COOKIE['nombre'], time()-4800 );  ?>
<html>
	<head>
		<title>Ejercicio</title>
		<meta charset = "UTF-8" />
	</head>
	<body>

		<a href="cookie1c.php">Verificar</a>

	</body>
</html>
```

Ejercicio Escritura y Borrado de una Cookie

En nuestro nuevo ejercicio gestionaremos una página de noticias. En ella, el usuario podrá escoger qué tipo de titular desea que aparezca al visitarla, pudiendo ser: Noticia política, Noticia económica o Noticia deportiva.
Mediante tres objetos de tipo radio, podrá seleccionar qué titular debe mostrar el periódico. Almacenaremos en una cookie el tipo de titular que desea ver el usuario. La primera vez que visita el sitio deben aparecer los tres titulares, en visitas posteriores, solo los que haya seleccionado.
Crearemos también un hipervínculo a una tercer página que borrará la cookie creada.

2.4. Ejemplos de uso de cookies

Ejemplo 1. Detectar nuevas visitas

```
<?php
	if(isset($_COOKIE['visita'])){
		echo"Qué alegría verte de nuevo por aquí!!";
	} else {
		setcookie('visita', 'ok', time()+31536000);
		echo"Bienvenido a mi página por primera vez";
	}
?>
<html>
	<head>
		<title>Ejercicio</title>
		<meta charset = "UTF-8" />
	</head>
	<body>
	</body>
</html>
```

Ejemplo 2. Fecha ultima Visita

```
<?php
$fecha = date("d/m/Y | H:i:s");

setcookie("fecha", $fecha, time()+31536000);

if(isset($_COOKIE['fecha'])){
	echo "Hola de nuevo, tu última visita fue el ".$_COOKIE['fecha'];
} else {
	echo "Esta es tu primera visita a nuestra página";
}
?>
```

```
<html>
	<head>
		<title>Ejercicio</title>
		<meta charset = "UTF-8" />
	</head>
	<body>
	</body><?php
if(isset($_POST['color'])){
	$color = $_POST['color'];
	setcookie("color", $color, time()+8000000);
} else {
		if(isset($_COOKIE['color'])) {
			$color = $_COOKIE['color'];
		} else {
			$color = 'white';
		}
	}
?>
<html>
	<head>
		<title>Ejercicio</title>
		<meta charset = "UTF-8" />
	</head>
	<body
	<?php echo "style='background-color:$color'"; ?>>
		<form method="post" action="<?=$_SERVER['PHP_SELF'] ?>" >
			<label for="color">Escoge tu color de fondo</label>
			<select name="color">
				<option value="red">Rojo</option>
				<option value="blue">Azul</option>
				<option value="green">Verde</option>
				<option value="yellow">Amarillo</option>
				<option value="silver">Gris</option>
				<option value="black">Negro</option>
			</select>
			<input type="submit" value="Cambiar Color!" />
		</form>
	</body>
</html>
</html>
```

Ejemplo 3. Contador Visitas

```
<?php
if (isset($_COOKIE['contador'])){
	setcookie('contador', $_COOKIE['contador']+1, time()+365*24*60*60);
	echo "Número de visitas: ".$_COOKIE['contador'];
} else {
	setcookie('contador', 1, time()+365*24*60*60);
	echo "Bienvenido por primera vez a nuestra página";
}

?>
<html>
	<head>
		<title>Ejercicio</title>
		<meta charset = "UTF-8" />
	</head>
	<body>
	</body>
</html>
```

Ejemplo 4. Elegir color página

```
<?php
if(isset($_POST['color'])){
      $color = $_POST['color'];
      setcookie("color", $color, time()+8000000);
} else {
            if(isset($_COOKIE['color'])) {
                  $color = $_COOKIE['color'];
            } else {
                  $color = 'white';
            }
      }
?>
<html>
      <head>
            <title>Ejercicio</title>
            <meta charset = "UTF-8" />
      </head>
      <body <?php echo "style='background-color:$color'"; ?>>
            <form method="post" action="$_SERVER['PHP_SELF']" >
                  <label for="color">Escoge tu color de fondo</label>
                  <select name="color">
                        <option value="red">Rojo</option>
                        <option value="blue">Azul</option>
                        <option value="green">Verde</option>
                        <option value="yellow">Amarillo</option>
                        <option value="silver">Gris</option>
                        <option value="black">Negro</option>
                  </select>
                  <input type="submit" value="Cambiar Color!" />
            </form>
      </body>
</html>
```

Ejemplo 5. Cookie definida como un array

```
<?
$persona = array ("Pedro", "Pérez", "26", "Madrid", "abcde");
setcookie("micookie[nombre]", $persona[0], time()+3600);
setcookie("micookie[apellido]", $persona[1], time()+3600);
setcookie("micookie[edad]", $persona[2], time()+3600);
setcookie("micookie[ciudad]", $persona[3], time()+3600);
setcookie("micookie[password]", $persona[4], time()+3600);

echo "<br/>El nombre es: ".$_COOKIE['micookie']['nombre'];
echo "<br/>El apellido es: ".$_COOKIE['micookie']['apellido'];
echo "<br/>El edad es: ".$_COOKIE['micookie']['edad'];
echo "<br/>El ciudad es: ".$_COOKIE['micookie']['ciudad'];
echo "<br/>El password es: ".$_COOKIE['micookie']['password'];
?>
```

3. SESIONES

Las sesiones, en aplicaciones web realizadas con PHP y, en el desarrollo de aplicaciones web en general, nos sirven para almacenar información que se memorizará durante toda la visita de un usuario a un sitio web. Dicho de otra forma, un usuario puede ver varias páginas durante su paso por un sitio web y mediante el uso de sesiones, podemos almacenar variables que podremos acceder en cualquiera de esas páginas.

Digamos que las sesiones son una manera de guardar información, específica para cada usuario, durante toda su visita. Cada usuario que entra en un sitio abre una sesión, que es independiente de la sesión de otros usuarios. En la sesión de un usuario podemos almacenar todo tipo de datos, como su nombre, productos de un hipotético carrito de la compra, preferencias de visualización o trabajo, páginas por las que ha pasado, etc. Todas estas informaciones se guardan en lo que denominamos variables de sesión. Estas **variables de sesión**, al contrario que las cookies, **se almacenan en el servidor.**

Dicho todo lo anterior, cabe preguntarse por qué son necesarias las sesiones. La respuesta la tenemos en el funcionamiento del protocolo HTTP. Este protocolo permite la comunicación entre una aplicación cliente (navegador) y un servidor web a través de solicitudes a las que el servidor responde de manera independiente, es decir, sin posibilidad de establecer ningún tipo de vínculo con peticiones anteriores realizadas desde el mismo lugar y por el mismo usuario. Se dice que el protocolo HTTP es un protocolo sin estado.

Visto que las comunicaciones que se establecen a través del protocolo HTTP son independientes las unas de las otras, lo cual cierra la posibilidad de establecer cualquier tipo de comunicación entre una serie de páginas, las sesiones vienen a cubrir este tipo de requerimiento, ofreciendo un mecanismo para el mantenimiento ubicuo de esa información compartida, al margen del funcionamiento de este tipo de protocolo.

En esencia, una sesión parece servir para lo mismo que una cookie, por tanto, debemos preguntarnos ¿Cuando debemos usar cookies en una aplicación?

- Si el único propósito de un sitio web es, simplemente, ofrecer páginas de información a usuarios distintos, sin importar su historial de navegación, ni sus preferencias, no tiene sentido aplicar el concepto de sesión que estamos analizando. Entonces usar cookies. Ejemplos: Búsquedas, opciones de configuración parciales, incluso carrito de la compra que no quiera mantener en distintos navegadores… Recuerda las cookies se almacenan el navegador
- Si por el contrario, nos interesan la aplicaciones web en las que, dadas sus características, se hace necesario disponer de la posibilidad de mantener cierta información a lo largo del proceso de navegación entre las páginas que la integran, se hace imprescindible el uso de sesiones para el desarrollo de las mismas. Ejemplos de esto podrían ser tanto la banca on-line -en la que cada usuario dispone de un área de navegación personalizada-. Carrito de la compra anteriormente haya realizado un login. Recuerda: Las sesiones son una zona compartida de almacenamiento en el servidor para dicha conexión.

3.1. Sesiones en PHP

Para empezar, un buen soporte de sesiones debería encargarse, al menos, de los siguientes aspectos:

- Seguimiento de sesiones: detección de cuando dos peticiones independientes forman parte de la misma sesión.
- Almacenamiento de información: Posibilidad de guardar información (aparte del identificador) relacionada con las sesiones.

La gestión de sesiones en PHP funciona combinando los mecanismos basados en variables ocultas y cookies descritos en el apartado anterior.

Cuando un usuario accede por primera vez en una página PHP preparada para crear una sesión, a este se le asigna un identificador único que será utilizado por PHP como clave para recuperar la matriz $_SESSION salvado específicamente para el usuario particular. El ID es un código único que, en el momento de la creación de la sesión se guarda automáticamente por PHP en una cookie o si las cookies están inhabilitadas, anexa a las URL generadas por el script PHP.

De esta manera, el programador puede ocuparse sólo de inicializar la sesión y guardar los datos. Obviamente, este comportamiento puede ser controlado a través de la configuración del archivo php.ini; de esta forma, podría suceder que el ID de sesión automático a través de URLs relativas no esté habilitado. En este caso PHP viene a generar una constante llamada SID que contiene la clave de sesión precedida del nombre. Basta con anexar esta constante a nuestra URL para obtener un sistema totalmente funcional.
La inicialización de una sesión, como ya se mencionó, se hace automáticamente por
PHP. Escribiremos:

```
$_SESSION['nombre'] = $valor;
```

para asegurarnos de que la variable "nombre" sesión tiene valor asignado, y que este valor es específico para el usuario que acaba

Por fortuna para nosotros, las funciones de gestión de sesiones que proporciona PHP actúan a un nivel de abstracción mayor y se encargan, por sí solas, de comprobar el soporte de cookies del navegador y de seleccionar que mecanismo de gestión de sesiones deben emplear. Nosotros sólo debemos encargarnos de manejarlas

Ejemplo

```
-------------------------- pagina1.php ----------------------------
<?php
session_start();

$_SESSION['svar']='Hola a todos';

echo 'El contenido de $_SESSION[\'svar\'] es ' .
$_SESSION['svar'] . '<br>';
?>
<a href=”pagina2.php”>Siguiente</a>

-------------------------- pagina2.php ----------------------------
<?php
session_start();

echo 'El contenido de $_SESSION[\'svar\'] es ' .

$_SESSION['svar'] . '<br>';
unset ($_SESSION['svar']);
?>
```

```
<a href=”pagina3.php”>Siguiente</a>

-------------------------- pagina3.php ------------------------------
<?php
session_start();

echo 'El contenido de $_SESSION[\'svar\'] es ' .
$_SESSION['svar'] . '<br>';

session_destroy();
?>
<a href=”pagina1.php”>Inicio</a>
```

Los pasos básicos del uso de sesiones son los siguientes:

1. Iniciar una sesión
2. Registrar variables de sesión
3. Utilizar variables de sesión
4. Anular las variables registradas y eliminar la sesión

Estos pasos no siempre tienen lugar al mismo tiempo y en la misma secuencia, ya que alguno puede producirse más de una vez. El código anterior es un sencillo ejemplo de uso de sesiones siguiendo los pasos básicos que acabamos de enumerar.

3.2 Iniciar una sesión

Antes de poder utilizar las funcionalidades de sesiones, es necesario iniciar una sesión. La forma más rápida y simple de hacerlo es mediante una llamada a la función **session_start()**.

Esta función comprueba si existe un identificador de sesión (recibido por cookies o mediante GET/POST). Si todavía no existe, simplemente crea uno. En caso de que ya exista, se encarga de recuperar las variables de sesión registradas previamente para que podamos utilizarlas.

Es aconsejable invocar **session_start()** al principio de todos los scripts que utilicen sesiones.

Podemos obviar la llamada a **session_start()** configurando PHP para que lo haga automáticamente cuando alguien visite nuestro sitio web.
La directiva de configuración de php.ini que controla este comportamiento es
session.auto_start

Pero esto es una mala praxis.

3.3 Registrar variables de sesión

PHP pone a nuestra disposición la variable $_SESSION, una matriz asociativa que servirá de contenedora para las variables de sesión.

Esta es una variable 'superglobal', o global automática. Esto simplemente quiere decir que está disponible en todos los contextos a lo largo de un script. No necesitamos hacer global **$_SESSION**; para acceder a ella dentro de funciones o métodos

Por lo tanto, el registro de variables de sesión consiste simplemente en añadir elementos a la matriz **$_SESSION**

```
$_SESSION['svar']='Hola a todos';
```

¿Dónde se almacenan las variables de sesión?
PHP pasa el identificador de sesión entre diferentes página, pero ¿Dónde se almacenan el resto de datos asociados a la sesión (lo que hemos denominado variables de sesión)?
Por defecto, el contenido de dichas variables se almacena en ficheros especiales en el servidor, usando un fichero por sesión.
El fichero tiene por nombre el identificador de la sesión, y almacena las variables en la forma nombre=valor.
La configuración de cómo y dónde se almacenan las variables de sesión se configura en el archivo php.ini, con las directivas:

```
session.save_handler = files
session.save_path = '/xampp/sessions'
```

3.4 Utilizar variables de sesión

Igual de sencillo que resulta añadir variables de sesión es utilizar dichas variables para acceder a su contenido y/o modificarlo.

Previo a cualquier acceso a las variables de sesión, deberemos iniciar la sesión mediante una llamada a session_start(), la cual realizará la carga de todas las variables previamente registradas en la matriz $_SESSION.

Para acceder a las variables de sesión tan sólo tenemos que acceder a los elementos de la matriz $_SESSION,

```
$session_var = $_SESSION['svar']
```

Para comprobar si se han inicializado las variables de sesión, debemos utilizar los métodos habituales (isset, empty), con la matriz $_SESSION, si lo que queremos comprobar es la existencia o no de una determinada variable de sesión.

```
if (isset ($_SESSION['svar']) === true) {...}
```

3.5 Anular las variables registradas y eliminar la sesión

Cuando hayamos finalizado con una variable de sesión, es decir, ya no necesitemos pasarla más entre páginas ni sea necesario acceder a su valor, podemos anular su registro.

Para anular una variable de sesión es recomendable utilizar la función unset() sobre el elemento de la matriz $_SESSION

```
unset ($_SESSION['svar']);
```

```
Por contra, si lo que queremos es anular el registro de todas las variables,
en vez de utilizar unset sobre la matriz $_SESSION, usaremos
$_SESSION = array ();
```

Cuando hayamos finalizado con una sesión, primero deberíamos anular el registro de todas sus variables y posteriormente invocar la función session_destroy(). Esta función destruye todos los datos asociados con la sesión, pero no elimina las variables asociadas ni la cookie.

- Destruye todas las variables de la sesión $_SESSION = array ();
- Finalmente destruye la sesión: session_destroy();

3.6 Ejemplo básicos de sesiones

Contador de sesiones

```
<?php
session_start();
$_SESSION['nombre'] = 'Juan';
?>

<?php
session_start();
echo "El nombre del usuario es: ".$_SESSION['nombre'];
?>

<?php
session_start();
if(isset($_SESSION['contador'])){
      $_SESSION['contador']++;
} else {
      $_SESSION['contador'] = 1;
}
echo "Nos has visitado ".$_SESSION['contador']." veces.";
?>
```

login con sesiones

```
---- pagina sesion3a.php-------------------------------
<html>
      <head>
            <title>Ejercicio</title>
            <meta charset = "UTF-8" />
      </head>
      <body>

            <?php
            if ($_GET[error] == "si") {
                  echo "Tu usuario o/y tu contraseña no son correctos,
inténtalo de nuevo.<br/><br/>";
            }
            elseif ($_GET[error] == "fuera") {
                  echo "No puedes entrar directamente en esta página.
Introduce usuario y contraseña.<br/><br/>";
            }
            ?>
```

```
            <form action="sesion3b.php" method="post">
                  <label for "nombre">Nombre de Usuario</label>
                  <input type="text" name="nombre" placeholder="Tu Nombre!!" /
>

                  <label for "pass">Tu Contraseña</label>
                  <input type="password" name="pass" />
                  <br/>
                  <input type="submit" value="Entrar!" />
            </form>

      </body>
</html>
```

```
---- pagina sesion3b.php------------------------------
<?
$usuariook = 'pedro';
$passok = 'abcd';

if ($_POST['nombre'] == $usuariook && $_POST['pass'] == $passok) {
      session_start();
      $_SESSION["verificado"] = "si";
      echo "Tienes acceso a la página privada";
      echo "<a href='sesion3c.php'>Ve a ver el contenido privado!!</a>";
} else {
      header ("Location: sesion3a.php?error=si");
}

?>

---- pagina sesion3c.php------------------------------
<?
session_start();
if(isset($_SESSION['verificado'])){
      echo "Esta es tu página privada";
} else {
      header ("Location: sesion3a.php?error=fuera");
}
?>
```

3.7 Directivas básicas de configuración de sesiones en php.ini

El sistema de control de sesiones soporta varias opciones de configuración (algunas de las cuales ya hemos adelantado) que podemos modificar en nuestro archivo
php.ini. Vamos a ver el significado y utilidad de las más importantes.

session.save_handler: Define el nombre del controlador que se utilizará para almacenar y recuperar los datos asociados a la sesión. Su valor por defecto es files.

session.save_path: Define el argumento que se pasa al controlador de almacenamiento. Para el controlador files, esta es la ruta donde se crean los archivos.
Nosotros hemos utilizado /xampp/sessions

session.name: Especifica el nombre de la sesión, que además se utiliza como nombre de la cookie . Su valor por defecto es PHSESSID.

session.auto_start: Determina si la sesión se debe inicializar de forma automática en todos los scripts,

evitando de esta manera la necesidad de invocar a session_start(). Su valor por defecto es 0 (no iniciar automáticamente)

session.cookie_lifetime: Tiempo de vida de cookie de sesión. Por tanto tiempo de vida de la sesión activa.

3.8. Autenticación con sesiones.

Se conoce como autenticación, al proceso por el cual se determina si una persona o cosa es quien o lo que dice ser.
En ningún caso debe confundirse el término autenticación con el término autentificación, ya que aunque están relacionados, la autenticación es un proceso más completo y costoso.

La autenticación se suele dividir a su vez en dos procesos:

- Identificación, la fase de identificación es aquella en la que el usuario o cliente aporta al sistema sus datos identificativos, o lo que es lo mismo, la fase en la que el cliente dice quien es.
- Autentificación, el objetivo de esta segunda fase es verificar la identidad que aportó el cliente durante la fase de identificación. Para llevar a cabo la fase de autentificación se recurre a contrastar la identidad mediante datos que sólo pueda conocer el cliente (contraseñas), algo que posee el cliente (tarjetas de acceso) o características físicas únicas del cliente (sistemas biométricos).

A grandes rasgos, el proceso que comúnmente se sigue es el siguiente:

1. El servidor solicita la información de logado al cliente.
2. El cliente procede a rellenar la información, y ésta se envía al servidor mediante los mecanismos de comunicación establecidos.
3. El servidor recibe la información, y la contrasta contra algún tipo de origen de datos.
4. Llegados a este punto existen dos posibilidades:
   a) Si la información es válida, el servidor procede a marcar la sesión de algún modo indicando que se trata de un usuario válido y, si es oportuno, indicando los permisos y/o privilegios de los que goza.
   b) Si la información es inválida, el servidor notifica al cliente que los datos no son válidos, y comienza de nuevo el proceso de autenticación.

Todo esto para hacer lo que viene a ser un login de toda la vida.
La mejor forma de ver como sucede todo el proceso es mediante un ejemplo. En este ejemplo vamos a utilizar el control de sesiones en el que probablemente sea su ámbito más extendido de uso, el seguimiento de usuarios autenticados a través de un mecanismo de inicio de sesión.

Ejemplo login.php

Login.php

```
<?php
$usuario="";
$errores=[];
//RECIBIR DATOS
if(isset($_POST['usuario'])){
      //FUENTE DE DATOS
      $arrayUsuarios=[["Luis","123"],["Rosa","345"]];
      $usuario=$_POST['usuario'];
      //COMPROBAR ERRORES
      if(empty($_POST['password'])||empty($_POST['usuario']) ) {
            $errores[]="Debe rellenar los datos";
      }
      $encontrado=false;
      $i=0;
      while ($i<count($arrayUsuarios)&& !$encontrado){
            if ($arrayUsuarios[$i][0]==$_POST['usuario'] &&
                  $arrayUsuarios[$i][1]==$_POST['password']) $encontrado=true;
            $i++;
      }

      if($encontrado){// SI TODO BIEN
            //creo y almaceno sesion
                  session_start();
                  $_SESSION['usuario']=$usuario;
                   header ("location:inicio.php");
                  }
      else $errores[]="Usuario o contraseña no existen"; // SI NO LO ENCUENTO
}
?>

<!DOCTYPE html>
<html lang="es">
<head>
      <meta charset="UTF-8">
      <title>Registrar datos</title>
</head>
<body>
    <?php
      //MOSTRAR ERRORES
      for($i=0;$i<count($errores);$i++){
                  echo $errores[$i]."<br>";
      }
    ?>
      <form action="<?=$_SERVER['PHP_SELF']?>" method="POST">
            <label>Usuario </label><input type="usuario" name="usuario"
value="<?=$usuario?>"><br/><br/>
            <label>Password</label><input type="password" name="password"><br/
><br/>
            <input type="submit" name="Comprobar" value="Enviar">
      </form>
</body>
</html>

----------------------------Inicio.php--------------------------------
<?php
session_start();
if (!isset ($_SESSION['usuario'])){
            $_SESSION = array();
```

```
            session_destroy();
            header("Location:login.php");
}else{
            echo "Bienvenida usuario: " $_SESSION['usuario'];
}
```

3.9. Logout o eliminación de sesión.

Muchas veces es deseable un punto de menú en nuestra aplicación para salirnos de ella y por tanto eliminar los datos de sesión. Este es el script de finalización de la sesión. En este comprobamos si estamos en la sesión, para borrar todas las variables y eliminar la sesión.

En cualquier caso, siempre acabamos con una redirección a la página de login.

```
---------------------------- logout.php ----------------------------
<?php
session_start();

if (isset ($_SESSION['usuario']))
{
      $_SESSION = array();
      session_destroy();
}
header("Location:login.php");
?>
```

3.9. Envío de datos complejos entre páginas

Los ejemplos anteriores son básicos y los datos que enviamos también. Pero ¿habrá algún problema si los datos a enviar son complejos?
Prueba el siguiente código y comprueba que es lo que está ocurriendo

Código a probar

```
-------------- clases.php-----------------------------------
<?php
class Alumno
{
  private $_nombre;
  private $apellido;
  public function __construct($nombre,$apellido)
  {
      $this->nombre = $nombre;
      $this->apellido = $apellido;
  }

   public function __toString()
   {
      return $this->nombre . ' ' .$this->apellido;
   }
}

class Escuela
{
  private $nombre;
  private $alumnos = array();
```

```php
public function __construct($nombre)
  {
      $this->nombre = $nombre;
  }
  public function addAlumno($alumno)
  {
      $this->alumnos[] = $alumno;
  }
  public function __toString()
  {
      $retorno = $this->nombre.'\n' ;

      foreach ($this->alumnos as $alumno) {
          $retorno .= $alumno .' ';
          /* Es lo mismo que decir
           * $retorno .= $alumno->__toString() .' ';
           * solo que el objeto sabe cómo convertirse en String,
           * puesto que detecta cuando se hace una operación de suma de
cadenas
           * con el punto ".". operador concatenar vamos  */
      }
      return $retorno;
  }
}


----------------------------escuela.php-------------------------
<?php
session_start();
include "clases.php";
$alu1 = new Alumno('Alan','Moore');
$alu2 = new Alumno('Neil','Gaiman');
$alu3 = new Alumno('Terry','Pratchett');
$escuela = new Escuela('Los Padres Salesianos');
$escuela-> addAlumno($alu1);
$escuela-> addAlumno($alu2);
$escuela-> addAlumno($alu3);
echo $escuela ;
echo "PRIMERA ESCUELA <BR>";
echo $escuela ;

$_SESSION['escuela']=$escuela;
?>

  <form action="escuela2.php" method="POST">
      <input type="submit" name="Ingresar" value="Enviar">
  </form>

----------------------------escuela2.php-------------------------
<?php
session_start();
include "clases.php";

$escuela2=$_SESSION['escuela'];
echo "ESTA ES LA ESCUELA 2<BR>";
echo $escuela2;
?>
```

¿Se puede? ¿No se puede? Qué es lo que ocurre.