AUTENTICACIÓN, COOKIES Y SESIONES CON PHP

Antonio Boronat Pérez
IES Joan Coromines (Benicarló)

Introducción

El protocolo usado en la Web, *http*, no conserva estados, es decir que dentro de un sitio Web un usuario puede visitar diferentes páginas y cada acceso será independiente de los otros. Pero en muchos casos es necesario disponer de información sobre la actividad de los usuarios mientras visitan un sitio Web, por ejemplo carros de la compra, respuestas a cuestionarios, selección de características de productos a buscar, etc. Para solucionar este problema está el concepto de *sesión* de forma que, independientemente de si el usuario debe o no autenticarse, nuestra aplicación será capaz de conservar información sobre la actividad de cada usuario durante el tiempo que permanece en el sitio Web, así aunque vaya cambiando de página nuestra aplicación podrá usar la información que se vaya generando.

Autenticación de Usuarios

Algunos sitios Web requieren que los usuarios estén registrados y para acceder deben presentar unas credenciales, normalmente usuario y contraseña introducidas mediante un formulario, aunque también se puede realizar mediante el uso de funciones de las cabeceras http. A continuación, se validan estas credenciales de forma que si son correctas dan acceso al sitio y si no se mostrará un error de validación.

Autenticación HTTP + PHP

En primer lugar veremos cómo realizar la validación mediante las cabeceras HTTP y PHP :
La base de este mecanismo de autenticación es la fusión de PHP y el protocolo HTTP. Para ello es necesario utilizar el método header de PHP,que nos permite enviar cabeceras HTTP. Al enviar dichas cabeceras debemos tener en cuenta que, tal y como establece el protocolo HTTP, las cabeceras deben enviarse al principio de la comunicación por lo que no podremos enviar al cliente ni un sólo carácter antes de ellas. Teniendo este hecho en cuenta, utilizaremos un script similar al siguiente para realizar la validación:

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
  header("WWW-Authenticate: Basic realm=\"WEB prueba\"");
  header("HTTP/1.0 401 Unauthorized");
  echo "Texto que se envía si se pulsa el botón Cancel\n";
  exit;
}
else {
```

```
//Ahora comprobamos si las credenciales son correctas, en este caso consultamos en una BD.
  include "funcion_conexion_bd.php";
  DEFINE ("SERVIDOR", "localhost");
  DEFINE ("USER", "root");
  DEFINE ("PASSWD", "");
  DEFINE ("BASE_DATOS", "valida_users");
  $con_bd = conexion_bd(SERVIDOR, USER, PASSWD,  BASE_DATOS);
  if($con_bd){
   $sql = "SELECT * FROM credenciales WHERE usuario = '" . $_SERVER['PHP_AUTH_USER'] ."'";
   if($res = mysqli_query($con_bd, $sql)) {
    if(mysqli_num_rows($res) >= 1){ // Ha encontrado al usuario
     $res_array = mysqli_fetch_all($res, MYSQLI_ASSOC);
     // Comprueba las contraseñas
     if($res_array[0]["password"] == $_SERVER['PHP_AUTH_PW']){
       header("Location: bienvenida.php");
     }
     else{
       unset($_SERVER['PHP_AUTH_USER']);
       unset($_SERVER['PHP_AUTH_PW']);
       header("WWW-Authenticate: Basic realm=\"My Realm\"");
       header("HTTP/1.0 401 Unauthorized");
       echo "Texto que se envía si se pulsa el botón Cancel\n";
       exit;
     }
    }
    else{
      unset($_SERVER['PHP_AUTH_USER']);
      unset($_SERVER['PHP_AUTH_PW']);
      header("WWW-Authenticate: Basic realm=\"My Realm\"");
      header("HTTP/1.0 401 Unauthorized");
      echo "Texto que se envía si se pulsa el botón Cancel\n";
      exit;
    }
   }
   else{
    echo "Error en la consulta: ". mysqli_error($con_bd) . "<br>";
   }
   $cierre_bd = @mysqli_close($con_bd);
 }
}
?>
```

El código anterior envía dos cabeceras HTTP que sirven para solicitar los datos de *login* al cliente mediante un formulario de sistema. En caso de cancelar la operación, el script seguiría su curso pero si se introdujera los datos se volvería a acceder a la misma página rellenando los valores $_SERVER[‘PHP_AUTH_USER’] y $_SERVER[‘PHP_AUTH_PW’] con los valores aportados por el cliente.
Es especialmente importante ser cuidadoso con el texto de las cabeceras ya que el protocolo HTTP y los servidores web pueden ser muy escrupulosos en cuanto a la sintaxis del mismo.

Validación Formulario + PHP

A continuación, veremos la toma de credenciales mediante formulario, una vez se han leído las credenciales se procederá a validarlas, para este proceso hay diferentes mecanismos, pero los más habituales se basan en el uso de valores almacenados en una base de datos o bien en acceder a sistemas más complejos como LDAP.
A continuación, veremos cómo aplicar estas dos soluciones a nuestras aplicaciones.

Validación de credenciales almacenadas en BD

```
<!DOCTYPE html>
<!--
Ejemplo para validar usuarios tomados de un formulario vi validados en una BD
-->
<html>
<body>
<form name="autenticar" method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>" >
       <label>Usuario:  </label> <input type="text" name="usuario" value="Usuario"><br>
       <label>Contraseña: </label> <input type="password" name="passwd"><br>
       <input type="submit" name="validar" value="Login">
       <input type="reset" name="cancelar" value="Cancelar">
</form>

<?php
// Formulario para validar usuarios contra una base de datos
  include "funcion_conexion_bd.php";

  DEFINE ("SERVIDOR", "localhost");
  DEFINE ("USER", "root");
  DEFINE ("PASSWD", "");
  DEFINE ("BASE_DATOS", "valida_users");
  $error_validacion= NULL;
  if(isset($_REQUEST["validar"])){
```

```
	if(isset($_REQUEST["usuario"])){
	$usuario= $_REQUEST["usuario"];
	}
	if(isset($_REQUEST["passwd"])){
	$passwd= $_REQUEST["passwd"];
	}
	// Conexión a la BD
	$con_bd = conexion_bd(SERVIDOR, USER, PASSWD,  BASE_DATOS);
	if($con_bd){
	$sql = "SELECT * FROM credenciales WHERE usuario = '" . $usuario ."'" ;
	if($res = mysqli_query($con_bd, $sql)) {
		if(mysqli_num_rows($res) >= 1){ // Ha encontrado al usario
		$res_array = mysqli_fetch_all($res, MYSQLI_ASSOC);
		// Comprueba las contraseñas
		if($res_array[0]["password"] == $passwd){
			header("Location: bienvenida.php");
		}
		else{
			$error_validacion= "Error en la validación del usuario<br>";
			echo $error_validacion;
		}
		}
	}
	else{
		echo "Error en la consulta: " . mysqli_error($con_bd) . "<br>";
	}
	$cierre_bd = @mysqli_close($con_bd);
	}
	else {
	echo "Error en la conexión a la BD: " . mysqli_error($con_bd) . "<br>";
	}
  }
?>
</body>
</html>
```

Para la validación de credenciales sobre un servidor de LDAP usaremos la utilidad *adLDAP* que nos proporciona una serie de funciones para la conexión y acceso a los datos almacenados en este servicio.

Validación de credenciales con PHP + LDAP

```
<!DOCTYPE html>
<!--Ejemplo para validar usuarios tomados de un formulario y validados en LDAP-->
```

```
<html>
<body>
<form name="autenticar" method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>" >
       <label>Usuario: </label> <input type="text" name="usuario" value="Usuario"><br>
       <label>Contraseña: </label> <input type="password" name="passwd"><br>
       <input type="submit" name="validar" value="Login">
       <input type="reset" name="cancelar" value="Cancelar">
</form>
<?php // Formulario para validar usuarios contra un servicio de LDAP con adLDAP
  include "adLDAP/src/adLDAP.php";
  $error_validacion= NULL;
  if(isset($_REQUEST["validar"])){
       if(isset($_REQUEST["usuario"])){ $usuario= $_REQUEST["usuario"];}
       if(isset($_REQUEST["passwd"])){ $passwd= $_REQUEST["passwd"];      }
       $adldap = new adLDAP(); // Conexión a LDAP
       if($adldap->authenticate($usuario, $passwd)){ //Valida user+password
         header("Location: bienvenida.php"); }
       else{
         $error_validacion= "Error en la validación del usuario<br>";
         echo $error_validacion; }
  }
?>
</body>
</html>
```

Para poder conectar con el servidor LDAP hemos de configurar sus parámetros en el fichero: *adLDAP/src/adLDAP.php* (Se deberá poner los valores que corresponda, los que se muestran son sólo un ejemplo):

`protected $accountSuffix = "DC=iesjc";`

`protected $baseDn = "DC=iesjc";`

`protected $domainControllers = array("10.2.200.189");` Pueden ser varios y el programa elige uno entre los indicados.

`protected $adminUsername = "CN=admin,";`

`protected $adminPassword = "Pues eso, lo que corresponda";`

`$this->ldapBind = @ldap_bind($this->ldapConnection, "UID=" . $username . ",OU=people," . $this->accountSuffix, $password);` Esta sentencia es la que se usará para consultar al servidor LDAP, está en el método *`authenticate`* que ejecutamos en nuestros programas. En ella debemos establecer la jerarquía de objetos LDAP para poder acceder a la información de nuestros usuarios.

Asegurar la Autenticación desde el Servidor Web

En los servidores Web podemos aplicar diferentes características que realicen, por un lado validación de usuarios para acceder a un sitio Web y por otro usar el protocolo HTTPS de forma que la información sensible, como usuario y contraseña se transmite cifrada para mejorar la seguridad del proceso.

Sitio Web con usuario y contraseña

El servidor Web Apache2 nos ofrece dos alternativas para restringir el acceso a un recurso mediante usuario y contraseña, uno básico en el que las credenciales están en texto claro y otra que usa cifrado y que se habilita con el módulo *mod_digest*. En lugar de enviar la contraseña en claro por la red, con Digest las contraseñas no viajan por la red. En lugar de eso, al pedir un recurso protegido, el servidor envía al cliente un número (noonce). Utilizando ese número "único", la URI del recurso solicitado y la contraseña, el navegador realiza un digest. Es decir, un cálculo basado en MD5 que le da un número difícil de obtener sin la contraseña. El servidor comprueba el digest utilizando para ello la información almacenada. Este mecanismo reduce el riesgo de captura de contraseñas.

Ejemplo de servidor virtual protegido con usuario y contraseña:

```
#Fichero de Configuración con Autenticación con Digest
#Cargamos el módulo necesario con:sudo a2enmod auth_digest
# reiniciar el servicio para que cargue el nuevo módulo:
# sudo systemctl restart apache2
<VirtualHost  profes.depinfo.iesjc>
ServerName  profes.depinfo.iesjc
ServerAdmin  administrador@depinfo.iesjc
DocumentRoot  /var/www/profes
ErrorLog  /tmp/profes_ERROR.log
TransferLog  /tmp/profes_ACCESS.log
<Directory  /var/www/profes>
      AuthType  Digest
      AuthName  profes
      # Indica el de dónde se toman las credenciales, por defecto
      # file, pero se puede indicar una BD, ldap, etc.
      AuthDigestProvider file
      #Fichero  con  los  Digest  de  los  usuarios  autorizados
      AuthtUserFile  /etc/apache2/passwd/digest_profes
      require  valid-user
</Directory>
</VirtualHost>
```

En el caso de crear un nuevo sitio Web debemos habilitarlo con:

```
sudo a2ensite fichero_conf_web.conf
sudo systemctl reload apache2
```

La gestión del fichero de digest en el servidor se realiza con la aplicación *htdigest*. Para añadir usuarios debemos ejecutar las siguientes órdenes:

```
# htdigest -c /etc/apache2/passwd/digest_profes profes usu1
# htdigest /etc/apache2/passwd/digest_profes profes usu2
```

La opción **c** en la primera orden creará el fichero de usuarios y contraseñas.
Fíjate que es necesario pasarle a la orden el nombre del entorno protegido (profes), este campo se denomina *realm* y su valor se corresponde al que hemos usado en *AuthName.*

Protocolo HTTPS

Las conexiones seguras se consiguen usando el protocolo https . Este protocolo es similar a http, pero utiliza algoritmos de cifrado SSL (en nuestro caso los proporcionados por el paquete OpenSSL).
Para conseguir que Apache nos envíe páginas cifradas se tiene que hacer una sencilla configuración, que a grandes rasgos consiste en generar el certificado que se presentará al cliente cuanto se conecte (en un servidor "profesional" este certificado se pediría a una agencia certificadora) y crear un servidor virtual que escuche en el puerto 443 (por defecto del protocolo https) y que presente el certificado.

Generación de certificados

En primer lugar se tiene que crear el directorio donde se guardarán los certificados. Este directorio se creará donde esté el resto de configuración de Apache, aunque también es habitual que se almacenen en los directorios que disponen las distribuciones de Linux (En Debian / Ubuntu están en /etc/ssl/certs/ para los certificados y /etc/ssl/private/ para las claves):

```
$sudo mkdir /etc/apache2/ssl
```

A continuación, se generará el certificado con la orden:

```
#make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /etc/apache2/ssl/apache.pem
```

el certificado estará en el fichero /etc/apache2/ssl/apache.pem . Este fichero contiene las claves necesarias para el intercambio de información cifrada, así que sólo falta la configuración del servidor virtual.

Servidor Virtual Seguro

Para el correcto funcionamiento sólo nos falta configurar un servidor virtual que escuche en el puerto 443 y presente el certificado que hemos generado para el cifrado de la conexión. Para eso seguiremos los pasos siguientes:

```
<VirtualHost *:443>
ServerName profes.depinfo.iesjc
ServerAdmin administrador@depinfo.iesjc
SSLEngine On
SSLCertificateFile /etc/apache2/ssl/apache.pem
DocumentRoot /var/www/profes
ErrorLog /var/log/apache2/error.log
LogLevel warn
CustomLog /var/log/apache2/access.log combined
</VirtualHost>
```

Para el uso del puerto 443, se activa el motor SSL y se indica donde está el fichero con el certificado y finalmente que al directorio se pide el uso de SSL.
En el fichero /etc/apache2/ports.conf se debe tener la directiva Listen 443 para que funcione correctamente, pero en circunstancias normales esta directiva ya estará puesta desde la instalación de Apache y activada en el momento que se cargó el módulo SSL.

Activando el Acceso Seguro

Ahora se ejecutará la orden para habilitar el módulo SSL:

```
$sudo a2enmod ssl
```

Y se reinicia el Apache para que cargue toda la configuración nueva:

```
$sudo systemctl restart apache2
```

Cookies

Según la Wikipedia las cookies son:
Una ***cookie***, **galleta** o **galleta informática** es una pequeña información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del usuario.
Sus principales funciones son:

- Llevar el control de usuarios: cuando un usuario introduce su nombre de usuario y contraseña, se almacena una galleta para que no tenga que estar introduciéndolas para cada página del servidor. Sin embargo, una *cookie* no identifica a una persona, sino a una combinación de computadora de la clase de computación-navegador-usuario.
- Conseguir información sobre los hábitos de navegación del usuario, e intentos de *spyware* (programas espía), por parte de agencias de publicidad y otros. Esto puede causar problemas de privacidad y es una de las razones por la que las *cookies* tienen sus detractores.

En PHP usamos la función *setcookie* para establecerlas en los navegadores cliente de nuestra aplicación. **setcookie()** define una cookie para ser enviada junto con el resto de las cabeceras de HTTP. Al igual que otras cabeceras, las cookies deben ser enviadas antes de que el script genere ninguna salida (es una restricción del protocolo). Ésto implica que las llamadas a esta función se coloquen antes de que se genere cualquier salida, incluyendo las etiquetas *<html>* y *<head>* al igual que cualquier espacio en blanco.
Sintaxis de *setcookie()*:

```
bool setcookie ( string $nombre [, string $valor [, int $expira = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false ]]]]]] );
```

Todos los argumentos exceptuando el argumento `$nombre` son opcionales. También puede reemplazar un argumento con un string vacío (*""*) para saltárselo. Ya que el argumento `$expira` es un entero, no puede pasarse por alto con un string vacío, en su lugar utilice un cero (*0*).
`$nombre` es el nombre de la cookie.
`$valor` corresponde al valor de la cookie y se almacena en el sistema cliente, a este valor podemos acceder con: `$_COOKIE[' nombre_cookie']`
`$expira` indica el final de la validez de la cookie, se expresa como un número de segundos a partir del momento actual. Se puede establecer como: *time()+Núm_segundos_validez*, por ejemplo para 24h: `time() + 60*60*24` Por defecto vale 0, con lo que la cookie se elimina al terminar la sesión, al salir del sitio Web.
`$path` establece el directorio a partir del cual está disponible esta cookie en nuestro servidor, por defecto es el directorio actual.
`$domain` establece el subdominio y subdominios subsecuentes para los que estará disponible la cookie, por ejemplo, si ponemos 'iesjc.org' estará disponible para este subdominio y dominios superiores como 'depinfo.iesjc.org' o 'www.depinfo.iesjc.org' .
`$secure` si vale `true`, sólo se enviará con https desde el cliente y la cookie sólo se creará si la conexión es de este tipo. Desde la aplicación en el servidor podemos decidir si envía este tipo de cookie usando $_SERVER["HTTPS"]. Por defecto `false`.
`$httponly` si vale `true` la cookie sólo será accesible por el protocolo web y no mediante algún tipo de lenguajes, como JavaScript. Por defecto, `false`.

La ejecución de *setcookie()* devuelve `true` o `false` en función si se completa correctamente o falla, pero no indica que se haya instalado en el navegador el cliente, por lo que puede devolver `true` y que el navegador rechace su instalación.
Una vez establecida la cookie se podrá acceder a ella las próxima vez que se ejecute la aplicación y usar su contenido mediante las variables array: *$_COOKIE* [...] o *$_REQUEST[...]* usando como índice el nombre se le dió.
Para borrar una cookie del navegador cliente se ejecuta *setcookie()* con el nombre y un tiempo negativo para que haya expirado, si se ha establecido path o dominio, se deben poner también.
Si lo que se desea es cambiar el valor de una cookie establecida se ejecuta *setcookie()* con el nuevo valor y este queda actualizado.
Cabe destacar que las cookies las gestionan los sitios Web, no sólo una aplicación, por tanto se pueden establecer y usar diferentes cookies establecidas por diferentes aplicaciones de un mismo sitio. Además, puede darse el caso que diferentes sitios den el mismo nombre a sus cookies con que pueden interferir en su funcionamiento, siendo recomendable establecer un sistema de nombres para evitar estas situaciones.
También se pueden crear arrays de cookies, de forma que el array toma el nombre de la cookie y se ejecutan tantos setcookie() como elementos tenga el array para establecerlas, en cambio para recuperarlas con *$_COOKIE* [...] o *$_REQUEST[...]* se reciben todas a la vez: (código del manual de PHP)

```
<?php
// crear las cookies
setcookie("cookie[tres]", "cookietres");
setcookie("cookie[dos]", "cookiedos");
setcookie("cookie[uno]", "cookieuno");

// imprimirlas luego que la página es recargada
if (isset($_COOKIE['cookie'])) {
    foreach ($_COOKIE['cookie'] as $name => $value) {
```

Ejemplo de uso de cookies con la aplicación de la calculadora, la cookie se usa para almacenar las operaciones que va realizando el usuario y se muestran en textarea, de forma que si el usuario quiere repetir alguna operación puede copiarla y pegarla en el input de la entrada de datos.

```
<?php
          //Mira si la Cookie ya existe y toma su valor
          if(isset($_COOKIE['previas'])){
               $previas = $_COOKIE['previas'];
          }
          else  {
               $previas = "";
          }
          if(isset($_REQUEST["num"])){
          //La var cadena se usa para mantener los datos introducidos
               $entrada = $_REQUEST["cadena"] . $_REQUEST["num"];
          }
          if(isset($_REQUEST["calc"])){
               // Convertimos la cadena en números y operaciones
          if(isset($_REQUEST["entrada"])){
               $cadena=$_REQUEST["entrada"]; //Si los val por teclado
```

```
        }
        else{
              $cadena= $_REQUEST["cadena"];
        }
        //preg_split extrae los operandos y los guarda en un array
              $operandos=preg_split("/[\+\-\*\/\(\)]/", $cadena);
              $pos = 0;
              foreach ($operandos as $oper){
                $pos+=strlen($oper);
                //Extrae el signo de la operación
                $operacion[] = substr($cadena,$pos ,1);
                $pos++; //Salta el signo que acaba de extraer
              }
              // Calcula las operaciones
              $aux = (float) $operandos[0];
              for($i=1; $i < count($operandos) ; $i++){
                switch ($operacion[$i-1]){
                    case '+': $aux = $aux + (float) $operandos[$i];
                    break;
                    case '-': $aux = $aux - (float) $operandos[$i];
                    break;
                    case '*': $aux = $aux * (float) $operandos[$i];
                    break;
                    case '/': $aux = $aux / (float) $operandos[$i];
                    break;
                }
              }
              // Pone el resultado
              $entrada = (string)$aux;
              //Guarda la cadena con la operación en un Cookie
              $previas = $previas . "\n" . $cadena;
              setcookie("previas", $previas, 0);
              // Con eval calcula y presenta el resultado
              //eval('$entrada='.$cadena. ';');
        }
        if(isset($_REQUEST["borrar"])){
              $entrada="";
        }
?>
```

Parte inicial del formulario de la calculadora donde se muestra el valor de la cookie:

```
<form name="calculadora" method="post" action="<?= $PHP_SELF ?>" >
  <table border="1">
  <tr>
    <td colspan="4"><textarea name="previas"><?=$previas?></textarea></td>
  </tr>
   <tr>
```

```
     <td colspan="4"> <input type="text" name="entrada" value="<?=
$entrada?>"> <input type="hidden" name="cadena" value="<?= $entrada
?>"> </td>
     <td> <input type="submit" name="borrar" value="cls"> </td>
   </tr>
```

Sesiones con PHP

Una *sesión* es un mecanismo de programación Web que nos permite conservar información sobre un usuario al pasar de una página a otra en nuestra aplicación. A diferencia de una *cookie*, los datos asociados a una sesión se almacenan en el servidor en lugar de en el cliente.
En muchos entornos de programación Web, las sesiones se implementan mediante una *cookie* que guarda un identificador del usuario en el servidor cada vez que pasa de una página Web a otra. El servidor Web almacena los datos de la sesión y se accede a ellos cada vez que se cambia de página usando el identificador de la cookie. Como el aceptar cookies depende del permiso del usuario, puede darse el caso que no podamos usar este método y por tanto la alternativa consiste en que el identificador se incorpore en la URL. En el caso de PHP, es el propio entorno quien elige el método más apropiado para que nuestra aplicación pueda gestionar las sesiones. En definitiva, es PHP quien se encarga de la gestión interna de las sesiones y nosotros sólo debemos encargarnos de crearlas, indicar la información que necesitamos que se conserve en la sesión y de su finalización.
A continuación mostramos las funciones que permiten la gestión de sesiones junto con el array *$_SESSION* que nos facilita el acceso a los datos que se comparten durante la sesión:
Iniciar una sesión:
$res = session_start([array de opciones]);
Esta función comprueba si el usuario ya tiene un sesión asociada, de forma que en caso afirmativo se establecerán las variables con los valores de la sesión en curso, o bien, si no la tiene creará una nueva asignándole un identificador. El valor devuelto será TRUE si se ha podido crear la sesión o FALSE si no se ha podido crear la sesión.
El `array de opciones`, es opcional y permite modificar los valores de configuración de las sesiones.
Como `session_start` intentará crear una cookie con el identificador de sesión, es importante que se ejecute antes de enviar el inicio de página, sinó se puede producir un error en que se indica que no se pueden iniciar la sesión porque la cabecera ya ha sido enviada.

```
<?php // Iniciar la sesión antes que se transmita algo al navegador
  // Crear o activar la sesión.
  session_start();
  // Guardar algún dato en las variables de sesión.
  $_SESSION['nombre'] = 'Homer S.';
?>
<!DOCTYPE html>
```

Consultar o modificar valores asociados a la sesión:

$valor_id = session_id([nuevo valor]);
Devuelve un string con el ID de la sesión, si se le pasa un valor, entonces cambia el ID por este.
$valor_nombre_variable_sesion = session_name([nuevo valor]);
Devuelve el nombre de la variable que almacena el ID de sesión, por defecto *PHPSESSID,* si se le pasa un parámetro cambia el nombre de esta variable.
Para el uso de los datos almacenados en la sesión, accedemos a la variable: *$_SESSION* todas las variables contenidas en este array se guardarán automáticamente durante la sesión, por ejemplo `$_SESSION["nombre"] = "Homer S.";` así tendremos una variable llamada nombre a lo largo de la sesión y podremos usarla en las diferentes páginas que formen nuestra aplicación.
La variable $_SESSION se puede usar en cualquier punto de la aplicación y también podemos eliminar elementos usando la función unset: `$_SESSION:unset($_SESSION["nombre"]);`
Con *session_abort();* se anulan los cambios que se haya realizado sobre las variables de sesión desde la llamada a `session_start()` en esta página hasta que se sale de ella. Los cambios no se aplican inmediatamente, pero se ve que los datos no han cambiado al pasar a otra página Web cuando al ejecutar `session_start()` para reiniciar la sesión podemos comprobar que las modificaciones de realizadas en la página donde estaba el `session_abort()` no están reflejados.
Con *session_reset();* se vuelve inmediatamente a los valores iniciales de las variables de sesión.
Para eliminar una sesión disponemos de *session_destroy();* al ejecutarla la sesión se elimina de forma que ya no estará disponible, pero los datos aún estarań disponibles en la página actual. Si ahora ejecutamos `session_id()` devolverá la cadena vacía, pero para que que los datos de la sesión sean borrados inmediatamente, se deberá borrar: `$_SESSION=array();`.
Posiblemente, si necesitamos eliminar la cookie de la sesión deberíamos usar `setcookie()`, por ejemplo: `(setcookie(session_name(),'',time()-1,'/');`, sino puede ocurrir que al crear una nueva sesión y no haber borrado la cookie se use el mismo ID de sesión en la nueva. Esta función devuelve TRUE si se ejecuta con éxito o FALSE en caso de error.
El estado de la sesión se puede comprobar con *session_status()* que devuelve un valor entero asociado a una de las constantes siguientes:

- `PHP_SESSION_DISABLED` indica que las sesiones están desactivadas
- `PHP_SESSION_NONE` las sesiones están habilitadas pero ninguna activa
- `PHP_SESSION_ACTIVE` las sesiones están habilitadas y hay una activa

Transmisión del Identificador de Sesión

Como se ha comentado anteriormente, el paso de la ID de una sesión entre una página y la siguiente en nuestra aplicación se puede realizar de dos formas, una usando una cookie y otra la información se envía en la URL, el mecanismo que elige uno u otro método forma parte del entorno PHP y es transparente al usuario. Ahora bien, hay tres variables de configuración que definen el uso de estos dos mecanismos:

- `session.use_cookies` Ésta si está activada (1, por defecto) intenta usar cookies para transferir el ID de la sesión.
- `session.use_trans_sid` Si está activada, puede usarse la URL para transferir la ID, por defecto está desactivada (0), ya que este método se considera inseguro y es preferible

el uso de cookies para esta tarea. Pero para que este método se use en el caso que no se permitan las cookies deberá estar activado (1).

- `session.use_only_cookies` Si está activada (1 por defecto) sólo se usarán las cookies para el paso de ID de sesión.

Esta configuración por defecto, implica que si el navegador cliente no acepta cookies no se podrán gestionar las sesiones. Esto nos llevará a decidir si, nuestras aplicaciones deben funcionar sólo con cookies o bien soportar los dos métodos, asumiendo los riesgos que puede implicar la transmisión de la ID de sesión en la URL. Estas variables se establecen en fichero *php.ini*. Si no deseamos modificar la configuración de PHP, pero en nuestra aplicación queremos que se usen sesiones independientemente de la configuración, podemos añadir nosotros la ID a la URL usando la constante *SID* que establece automáticamente PHP. Las construcciones de las URL en nuestra aplicación sería alguna de las siguientes:

- `<a href="destino.php?<?= SID ?>">Destino</a>`
- `<form action="destino.php?<?= SID ?>" method="post">`
- `header('Location: destino.php?'.SID);`

Ejemplo de creación de una sesión con la aplicación de usuarios con formulario + LDAP, crea una sesión y si el usuario se valida satisfactoriamente, se guarda en una variable y se puede usar su valor en otras páginas de la misma aplicación:

Página *index.php* de validación de usuarios con LDAP:

```
<?php
  // Crea una sesión
  session_start();
  // Si se valida correctamente, contendrá el valor introducido en el form.
  $_SESSION["usuario"] = "";
?>
<!DOCTYPE html>
<!--
Ejemplo para validar usurios tomados de un formulario y validados en LDAP
-->
<html>
       <head>
       <meta charset="UTF-8">
       <title></title>
       </head>
<body>
<form name="autenticar" method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>" >
       <label>Usuario:     </label>        <input        type="text"        name="usuario"
value="Usuario"><br>
       <label>Contraseña: </label> <input type="password" name="passwd"><br>
       <input type="submit" name="validar" value="Login">
       <input type="reset" name="cancelar" value="Cancelar">
</form>

<?php
// Formulario para validar usuarios contra una base de datos
   include "adLDAP/src/adLDAP.php";

  $error_validacion= NULL;
  if(isset($_REQUEST["validar"])){
       if(isset($_REQUEST["usuario"])){
         $usuario= $_REQUEST["usuario"];
       }
```

```
      if(isset($_REQUEST["passwd"])){
        $passwd= $_REQUEST["passwd"];
      }
      // Conexión a LDAP
      $adldap = new adLDAP();
      if($adldap->authenticate($usuario, $passwd)){
        $_SESSION["usuario"] = $usuario;
        header("Location: bienvenida.php");
      }
      else{
        $error_validacion= "Error en la validación del usuario<br>";
        echo $error_validacion;
      }
  }
?>
</body>
</html>
```

Página *bienvenida.php*:

```
<?php
  // Reactiva la sesión iniciada en la página index.php
  session_start();

  echo "Bienvenido " . $_SESSION["usuario"] . " a nuestro sitio Web";
?>
```