Desarrollo Web en entorno servidor

Unidad 3:
Aplicaciones Web en PHP
Cookies y sesiones

Tabla de contenido

1. Cookies

Una cookie es un fichero de texto que un sitio web guarda en el entorno del usuario del navegador. Su uso más típico es el almacenamiento de las preferencias del usuario para que no tenga que volver a indicarlas la próxima vez que visite el sitio. Ejemplos de datos que se guardan son el idioma en que se deben mostrar las páginas o la fecha de la última visita.

Cuando un cliente realiza una petición web, envía al servidor las cookies que pudiera tener de este.

Las cookies se transmiten entre el navegador y el servidor web utilizando los encabezados del protocolo HTTP. Por ello, las sentencias que manejas las cookies deben enviarse antes de que el navegador muestre información alguna en pantalla.

Para manejar las cookies se usa la función setcookie(), que tiene los siguientes parámetros en su cabecera:

```
Bool: setcookie(string $name
[, string $value = ""
[, int $expires = 0
[,  string $path = ""
[,  string $domain = ""
[,  bool $secure = false
[,  bool $httponly = false]]]]]])
```

Los tres primeros parámetros de la función son los más importantes, y contendrán la siguiente información:

- Name: El nombre de la cookie.
- Value: El valor de la cookie.
- Expires: La fecha en que la cookie expira. Se expresa como una fecha UNIX, es decir, el número de segundos pasados desde el comienzo de 1970. Normalmente se utiliza la función `time()`, que devuelve la fecha actual y se le suma un periodo de tiempo expresado en segundos. Si queremos que una cookie tenga una duración de un día, entonces enviaremos como argumento `time()+3600*24`

El resto de los argumentos de la función setcookie() hacen referencia a la siguiente información:

- Path: La ruta dentro del servidor en la que la cookie estará disponible. El valor por defecto es el directorio actual en donde se está configurando la cookie.
- Domain: El dominio o subdominio en el que la cookie está disponible.
- Secure: Indica que la cookie sólo debiera transmitirse por una conexión segura HTTPS desde el cliente. Cuando se configura como true, la cookie sólo se creará si es que existe una conexión segura.
- Httponly: Cuando es true la cookie será accesible sólo a través del protocolo HTTP. Esto significa que la cookie no será accesible por lenguajes de scripting, como JavaScript.

La función `setcookie()` devolverá true si se ha ejecutado correctamente, independientemente de si el usuario ha aceptado o no dicha cookie. Devolverá false en caso de que haya habido algún tipo de problema en la creación de la cookie, como por ejemplo alguna salida anterior a la función `setcookie()`.

Como decíamos anteriormente, cuando accedemos a un sitio web, el navegador envía de forma automática al servidor todo el contenido de las cookies que almacena relativas a ese sitio en concreto. Desde PHP podemos acceder a esta información por medio del array `$_COOKIE`.

Es importante recalcar que la primera vez que se hace una petición por parte del cliente, la información de las cookies que creamos en esta primera petición no está disponible en `$_COOKIE`.

Podemos ver un ejemplo en `cookie_mal.php`.

```
1:  <?php
2:  setcookie('nueva', "valor", time() + 3600 * 24);
3:  echo $_COOKIE['nueva'];
```

La primera vez que accedemos nos da un error en la línea 3, las siguientes veces que accedamos, no.

Veamos el ejemplo, `contador_visitas.php`.

```
 1:  <?php
 2:      if(!isset($_COOKIE['visitas'])){
 3:          setcookie('visitas', '1', time() + 3600 * 24);
 4:          echo "Bienvenido por primera vez";
 5:      }else{
 6:          $visitas = (int) $_COOKIE['visitas'];
 7:          $visitas++;
 8:          setcookie('visitas', $visitas, time() + 3600 * 24);
 9:          echo "Bienvenido por $visitas vez";
10:      }
```

En este ejemplo buscamos la cookie visitas, línea 2. Como la primera vez que hemos accedido no existe, la creamos y mostramos un mensaje de bienvenida, líneas 3 y 4. Si volvemos a acceder entonces pasamos al bloque `else`. En la línea 6 vemos una conversión de tipos forzada, ya que el valor que guarda la cookie es un string, pero en nuestro caso queremos manipularla como un `integer`. Incrementamos, en la línea 7, el número de visitas y creamos en la línea 8 de nuevo la cookie visitas con el nuevo valor. Por último, mostramos el mensaje de bienvenida con el número de veces que hemos accedido.

Otras conversiones forzadas de tipo, según la variable que queramos manipular, son:

- (int), (integer) - forzado a integer
- (bool), (boolean) - forzado a boolean
- (float), (double), (real) - forzado a float
- (string) - forzado a string
- (array) - forzado a array
- (object) - forzado a object
- (unset) - forzado a NULL

Para destruir una cookie se usa la función `setcookie()` con una fecha límite anterior a la actual.

Ejercicio 1

Modificar el ejemplo `contador_visitas.php` para que la cookie se destruya tras acceder 3 veces.

Los navegadores ofrecen la posibilidad de ver las cookies que tenemos almacenadas. Según el servidor, se accede de diferente manera.

En Chrome las podemos ver en la siguiente ruta: chrome://settings/cookies/detail?site=localhost o a través de “Configuración / Privacidad y seguridad / Cookies y otros datos de sitios / Ver todas las cookies y datos del sitio” y buscando localhost

En Edge las podemos ver en la siguiente ruta: edge://settings/cookies/detail?site=localhost o a través de “Configuración / Cookies y permisos del sitio / Administra y elimina cookies y datos del sitio / ver todas las cookies y datos del sitio” y buscando localhost.

Ejercicio 2

Crea una página web con un formulario para elegir el idioma en el que se muestra, inglés o español. Almacena la elección del usuario con una cookie para que la siguiente vez que el usuario se conecte la página aparezca directamente en su idioma. Si la cookie no existe, la página se mostrará en español.[1]

2. Sesiones. Seguridad: usuarios y roles.

Como acabamos de ver, una forma para guardar información particular de cada usuario es utilizar cookies. Sin embargo, existen diversos problemas asociados a las cookies, como el número de ellas que admite el navegador, su tamaño máximo o si el usuario las acepta o no.

Por otra parte, como HTTP es un protocolo sin estado, las diferentes peticiones de un cliente al servidor son independientes, no están relacionadas entre sí.

Para solventar estos inconvenientes, existen las sesiones. El término sesión hace referencia al conjunto de información relativa a un usuario concreto. Esta información puede ser tan simple como el nombre del propio usuario, o más compleja, como los artículos que ha depositado en la cesta de compra de una tienda online.

Al iniciar una sesión el servidor asigna y envía al usuario un identificador de sesión (SID). En las siguientes peticiones el usuario envía al servidor ese identificador, de manera que el servidor sabe que se trata del mismo usuario. De esta forma, el servidor web puede relacionarlo con toda la información que posee sobre él.

Para mantener el SID entre las páginas de un sitio web que visita el usuario existen dos procedimientos:

A) Utilizando cookies.
B) Propagando el SID en un parámetro de la URL.

El mejor método y el más utilizado es la utilización de cookies. Propagar el SID como parte de la URL conlleva mayores desventajas, como la imposibilidad de mantener el SID entre distintas sesiones, o el hecho de que compartir la URL con otra persona implica compartir también el identificador de sesión.

Por tanto, para controlar la sesión de un usuario, el servidor deja una cookie con el id de sesión en el cliente, que se elimina al cerrarla. Si se borran manualmente las cookies del navegador, se cierran

[1] Mas información sobre las páginas web multi-idioma en Páginas multi-idioma con PHP (desarrolloweb.com)

las sesiones abiertas. Se puede configurar el servidor para controlar las sesiones sin cookies, pero no es lo habitual.

Para crear una sesión se utiliza la función `session_start()`. Si no hay una sesión activa la crea, es decir, crea una cookie con el SID. Si ya hay una sesión activa, es decir, existe la cookie con el SID, el script que llama a la función se une a la sesión.

Una vez creada la sesión es posible utilizar la variable superglobal `$_SESSION` para compartir información entre los scripts que compartan sesión. `$_SESSION` es un array que almacena las variables de sesión definidas por el usuario, basta con añadir elementos de la manera usual.

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

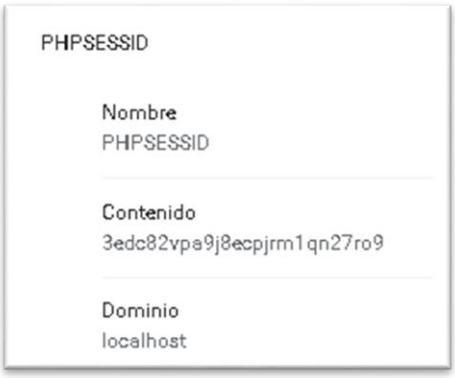
Veamos el siguiente ejemplo, `sesiones_uso_basico.php`.

```
1:  <?php
2:      session_start();
3:      if (!isset($_SESSION['count'])) {
4:        $_SESSION['count'] = 0;
5:      } else {
6:        $_SESSION['count']++;
7:      }
8:      echo "hola " .$_SESSION['count'];
9:      echo "<br><a href='sesiones_uso_basico2.php'>Siguiente</a>";
```

En la línea 2 creamos la sesión y, si no existe ya, una variable de sesión `$_SESSION['count']` con valor 0. En caso de que ya exista, le suma 1. En la línea 9 mostramos un vínculo a `sesiones_uso_basico2.php`. Este segundo fichero se une a la sesión y muestra el valor de la variable. Accediendo a ambos se puede comprobar que realmente se trata de la misma variable.

```
1:  <?php
2:      session_start();
3:      echo "La variable count vale: " . $_SESSION['count'];
```

Podemos encontrar en el navegador la cookie creada con la primera llamada a session_start(). En la siguiente figura mostramos un ejemplo de dicha cookie en Google Chrome.



Vemos el nombre de la cookie creada, PHPSESSID; el contenido de la cookie, es decir, el valor del SID; y el resto de los datos correspondientes a las cookies.

Volviendo al caso de los formularios de login de las aplicaciones web, es habitual que tras validar los datos de acceso se cree una sesión, además, esta sesión está ligada a un cierto tipo de perfil: administrador, usuario, gestor, etc.

Los ejemplos que vienen a continuación muestran un ciclo completo de *Login -> Principal -> Logout* redireccionando tanto Principal como *Logout* a *Login* en caso de querer acceder directamente a ellos.

Comenzamos viendo el formulario de Login, `sesiones1_login.php`

```
 1: <?php
 2: function comprobar_usuario($nombre, $clave){
 3:   if($nombre === "usuario" and $clave === "1234"){
 4:      $usu['nombre'] = "usuario";
 5:      $usu['rol'] = 0;
 6:      return $usu;
 7:   }elseif($nombre === "admin" and $clave === "1234"){
 8:           $usu['nombre'] = "admin";
 9:           $usu['rol'] = 1;
10:           return $usu;
11:   }else return false;
12:  }
13:  if ($_SERVER["REQUEST_METHOD"] == "POST") {
14:     $usu = comprobar_usuario($_POST['usuario'], $_POST['clave']);
15:     if($usu==false){
16:       $err = true;
17:          $usuario = $_POST['usuario'];
18:     }else{
19:       session_start();
20:          $_SESSION['usuario'] = $_POST['usuario'];
21:       header("Location: sesiones1_principal.php");
22:     }
23:  }
24:  ?>
25:  <!DOCTYPE html>
26:  <html>
27:   <head>
28:    <title>Formulario de login</title>
29:    <meta charset = "UTF-8">
30:   </head>
31:    <body>
32:  <?php if(isset($_GET["redirigido"])){
33:    echo "<p>Haga login para continuar</p>";
34:    }?>
35:  <?php if(isset($err) and $err == true){
36:           echo "<p> revise usuario y contraseña</p>";
37:   }?>
38:         <form action = "<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"
method = "POST">
39:          Usuario
40:          <input value = "<?php if(isset($usuario))echo $usuario;?>"
41:          id = "usuario" name = "usuario" type = "text">
```

```
42:    Clave
43:    <input id = "clave" name = "clave" type = "password">
44:    <input type = "submit">
45:  </form>
46:    </body>
47: </html>
```

Este ejemplo se basa en `validacion_redireccionado.php` en el que hemos introducido diversos cambios:

1.- Hemos añadido entre las líneas 1 y 11 una función (`comprobar_usuario()`) para comprobar los datos de acceso del usuario. Si son ciertos nos devolverá un array con el tipo de usuario y su rol, sin son falsos entonces devolverá Falso.

2.- Hemos modificado la parte if original de `validacion_redireccionado.php`, líneas 12 a la 23, para hacer uso de la función `comprobar_usuario()`. Si los datos introducidos son correctos, entonces iniciamos sesión y somo redirigidos a `sesiones1_principal.php`.

3.- Hemos modificado el HTML para incorporar una opción que nos reenvíe de nuevo al formulario de login en el caso de que hayamos accedido al `sesiones1_principal.php` sin haber introducido los datos de acceso previamente.

El ejemplo `sesiones1_principal.php` está basado en `redireccionado.php`.

```
1:  <?php
 2:     session_start();
 3:     if(!isset($_SESSION['usuario'])){
 4:         header("Location: sesiones1_login.php?redirigido=true");
 5:     }
 6:  ?>
 7:  <!DOCTYPE html>
 8:  <html>
 9:     <head>
10:         <title>Página principal</title>
11:         <meta charset = "UTF-8">
12:     </head>
13:     <body>
14:         <?php echo "Bienvenido ".$_SESSION['usuario'];?>
15:         <br><a href = "sesiones1_logout.php"> Salir <a>
16:     </body>
17:  </html>
```

Hemos realizado los siguientes cambios:

1.- Hemos añadido la función `session_start()` para unirnos a la sesión anterior, y, en caso de no haber hecho login, es decir, que la variable `$_session` no contenga el dato del usuario, entonces, somos redirigidos al formulario de login.

2.- Hemos añadido el vínculo “Salir”, que nos lleva a `sesiones1_logout.php`, el cual cierra la sesión y redirige a la página de login.

```
1:  <?php
2:    session_start();
3:    if(!isset($_SESSION['usuario'])){
```

```
4:          header("Location: sesiones1_login.php?redirigido=true");
5:      }else{
6:          session_destroy();
7:          setcookie(session_name(), 123, time() - 1000); // eliminar la cookie
8:          header("Location: sesiones1_login.php");
9:      }
```

Para cerrar sesión llamamos a la función `session_destroy()`, la cual borra cualquier dato en `$_SESSION`, y borramos la cookie de sesión correspondiente con la sentencia de la línea 7.

Ejercicio 3

Modificar sesiones1_principal.php para que se muestre el mensaje “Zona de administrador” solo visible para los usuarios con rol 1.