U04. DESARROLLO DE APLICACIONES WEB UTILIZANDO CÓDIGO EMBEBIDO

1.- Autenticación de usuario y control de acceso.

- Activación de https en apache (*sudo a2enmod ssl && sudo a2ensite default-ssl)*
- Identificación de usuarios por grupos, según el grupo al que pertenece el usuario se le permite el acceso a unos recursos o a otros.

1.1.- Mecanismos de autenticación

En Apache 2:
Añadir el usuario dwes al fichero /etc/apache2/users:
sudo htpasswd -c /etc/apache2/users dwes (-c crea el fichero sólo se debe utilizar la 1ª vez)

Para indicar a Apache qué recursos tienen acceso restgringido, podemos crear un fichero ***.htaccess*** en el directorio en que se encuentren, con las siguientes directivas:

AuthName "Contenido Restringido"
AuthType Basic
AuthUserFile /etc/apache2/users
require valid-user

AuthName	Nombre de dominio que se usará en la autentificación. Si el cliente se autentifica correctamente, esa misma información de autentificación se utilizará automáticamente en el resto de las páginas del mismo dominio.
AuthType	Método de autentificación que se usará. Además del método Basic, Apache también permite utilizar el método Digest.
AuthUserFile	Ruta al archivo de credenciales que has creado con htpasswd.
Require	Permite indicar que sólo puedan acceder algunos usuarios o grupos de usuarios concretos. Si indicamos "valid-user", podrán acceder todos los usuarios que se autentifiquen correctamente.

Para que funcione lo anterior dentro del grupo de directivas ***<Directory /var/www/>...</Directory>***, hay que activar ***AllowOverride All***

Desde PHP se puede acceder a la información introducida por el usuario mediante el array superglobal $_SERVER

$_SERVER['PHP_AUTH_USER']	Nombre de usuario que se ha introducido.
$_SERVER['PHP_AUTH_PW']	Contraseña introducida.
$_SERVER['AUTH_TYPE']	Método HTTP usado para autentificar. Puede ser Basic o Digest.

Podemos conseguir el mismo efecto sin usar el fichero *.htaccess*, utilizando el siguiente código:

```php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic Realm="Contenido restringido"');
    header('HTTP/1.1 401 Unauthorized');
    echo "Usuario no reconocido!";
    exit;
}
```

La funcion ***header*** hace que el servidor envíe un error de *Acceso no autorizado* (código 401), de esta forma conseguimos el mismo efecto que con el fichero *.htaccess*.

header (string $string [, bool $replace = true [, int $http_response_code]]) : void
header() es usado para enviar encabezados HTTP sin formato. Ver la especificación » HTTP/1.1 specification para más información sobre encabezados HTTP.
Recuerde que **header()** debe ser llamado antes de mostrar nada por pantalla, etiquetas HTML, líneas en blanco desde un fichero o desde PHP. Es un error muy común leer código con funciones como include o require, u otro tipo de funciones de acceso de ficheros que incluyen espacios o líneas en blanco que se muestran antes de llamar a la función **header()**. Sucede el mismo problema cuando se utiliza un solo fichero PHP/HTML.

En el ejemplo de PHP anterior, se envía un error 401, lo que hace que el navegador pida nombre de usuario

1.2 Métodos de autenticación en una aplicación web

Si utilizamos el código del apartado anterior, usando la función ***header*** deberemos verificar nosotros que el nombre de usuario y contraseña son válidos. Esto lo haría el servidor si usásemos un fichero *.htaccess.*

Lo más lógico sería tener en una base de datos almacenados los nombres de usuarios y sus correspondientes *hashes* y verificar que el nombre de usuario y el *hash* de la contraseña coinciden con el que tenemos almacenados en dicha base de datos.

Como todavía no se ha visto el acceso a datos en el siguiente código se ve un ejemplo, en el cual ya se habría obtenido el *hash* almacenado en la base de datos:

```
function checkAuth() {
    $validAuth=[
        'user'=>'dwes',
'pass'=>'$6$vSeuwOWCW76fC421$ZT4EK2OtOcZY7E6ofRGzlimqxe72mJrIy74R.l3ZuRb8PHCNnUtj9xqK4tTdjs0qRVy5ZUWS3sbB25DQCBwX31'
    ];
    return isset($_SERVER['PHP_AUTH_USER'])
        && isset($_SERVER['PHP_AUTH_PW'])
        && $_SERVER['PHP_AUTH_USER']==$validAuth['user']
        && password_verify($_SERVER['PHP_AUTH_PW'], $validAuth['pass']);
}
if (!checkAuth()) {
    header('WWW-Authenticate: Basic Realm="Contenido restringido"');
    header('HTTP/1.1 401 Unauthorized');
    mensajeError("Usuario no reconocido!");
    exit;
}
```

Ejercicio 1. Realiza un formulario de alta de usuarios, en el que se pida nombre, apellido1, apellido2, usuario, contraseña y email. Una vez comprobado que el usuario no ha sido previamente dado de alta, guardaremos sus datos en formato JSON en un fichero de texto. Crear otra página en la que para entrar a su contenido (un mensaje: "Bienvenido nombre de usuario"), se verifique el nombre de usuario y contraseña válidos, accediendo al fichero JSON. No se deberá guardar la contraseña del usuario, sino un ***hash*** de la misma.

2.- Cookies

Las cookies son información en formato texto que el servidor almacena en el ordenador del cliente a través del navegador con el que se visita el sitio.

Podemos almacenar ***cookies*** en el navegador del cliente con la función de PHP ***setcookie***:

setcookie (string $name [, string $value = "" [, int $expires = 0 [, string $path = "" [, string $domain = "" [, bool $secure = **FALSE** [, bool $httponly = **FALSE**]]]]]]) : bool
setcookie (string $name [, string $value = "" [, array $options = []]]) : bool
setcookie() define una cookie para ser enviada junto con el resto de cabeceras HTTP. Como otros encabezados, cookies deben ser enviadas *antes* de cualquier salida en el script (este es un protocolo de restricción). Esto requiere que hagas llamadas a esta función antes de cualquier salida, incluyendo etiquetas *<html>* y *<head>* así como cualquier espacio en blanco. Una vez que las cookies se han establecido, se puede acceder a ellas en la siguiente página de carga con el array $_COOKIE. valores Cookie también pueden existir en $_REQUEST.

Ejemplo de uso:

```
   setcookie("timeStampUltimaVisita",getdate()[0],time()+365*24*60*60); // guardamos el timestamp de la visita por un año
   setcookie("numVisita",$contador,time()+365*24*60*60); //ídem con el número de visita
```

Como puede verse las cookies almacenan pares (*Nombre, Valor*) por el tiempo que expresemos en el tercer parámetro, en nuestro ejemplo:

Nombre	Valor	Tiempo Expiración
timeStampUltimaVisita	getdate()[0]	un año
numVisita	$contador	un año

Si se omite el tercer parámetro la cookie se eliminará al cerrarse el navegador. Se puede limitar el uso de la cookie para una determinada ruta dentro del servidor (parámetro ***path***).

Las cookies se transmiten del servidor al navegador a través de las cabeceras ***HTTP***, lo cual quiere decir que deben ir antes que cualquier sentencia ***echo*** o ***print***.

Cuando un cliente visita nuestro servidor, el navegador envía automáticamente todas las cookies que almacena de nuestro servidor. En *PHP* quedan almacenadas en la variable superglobal ***$_COOKIE***. Siguiendo con el ejemplo anterior, cuando el usuario vuelva a visitarnos dispondríamos de las variables ***$_COOKIE['timeStampUltimaVisita']*** y ***$_COOKIE['numVisita']***.

Se ha de tener en cuenta que el cliente podría tener deshabilitadas las *cookies*, o que en un momento dado puede borrarlas desde el navegador. Se puede eliminar una *cookie* usando la misma función ***setcookie*** y poniéndole una fecha de caducidad anterior a la fecha actual.

Ejercicio 2. Utilizando las cookies del ejemplo anterior, mostrar al usuario la fecha de su anterior visita y cuántos visitantes ha habido entre la última visita y la actual.

3. Sesiones

Las cookies permiten guardar información en el cliente pero necesitamos un mecanismo para guardar información sobre el cliente, que no queremos compartir con él, a lo largo de una sesión. Por ejemplo cuando un usuario se ha identificado, podemos guardar éste hecho en una variable de sesión (variable superglobal ***$_SESSION***).

Para cada cliente (navegador) que esté visitando nuestro sitio habrá una sesión distinta. Las sesiones se distinguen unas de otras por su identicador de sesión ***SID***. No necesitamos conocer el *SID* ya que el array ***$_SESSION*** se crea de forma independiente para cada cliente.

Existen dos maneras de mantener el *SID* entre las páginas web que visita un usuario, utilizando cookies o propagándolo en un parámetro de GET (en la URL). Por ejemplo;
http://10.10.100.11/PHP/DWES/Arrays/globals.php?PHPSESSID=1t02hu321hmbjout9pedheh06f

Es mejor utilizar cookies, ya que de esta forma no estará visible el *SID*. De todas formas aunque usemos cookies, si no usamos *https* un atacante podría capturar los paquetes de la conexión y obtener el identificador de la sesión e inyectar la cookie correspondiente en su navegador.

De todo esto no tenemos que preocuparnos ya que PHP lo hace de forma automática.

A la información que se almacena en el servidor sobre la sesión de un usuario se la conoce como *server side cookies*. Esta información no viaja entre el cliente y el servidor, salvo el *SID*, por lo que, como ya hemos dicho, debería usarse *https.*

Configuración

PHP incorpora soporte de sesiones por defecto. Podemos ajustar la configuración a través del fichero *php.ini* en las siguientes directivas:

session.use_cookies	Si está a **1** se usarán cookies. Si está a **0** propagación por URL
session.save_handler	Indica como se almcenan las sesiones, ***files*** (valor por defecto) en ficheros, ***mm*** en memoria, ***sqlite*** en una base de datos, ***user*** utilizando funciones que debe definir el programador.
session.name	***PHPSESSID*** por defecto
session.auto_start	Su valor por defecto es **0** que indica que hay que iniciar la sesión (***session_start***) para gestionar la sesión. Si fuese **1** se iniciarían las sesiones de forma automática
session.cookie_lifetime	Si se utiliza propagación del *SID* por URL se perderá al cerrar el navegador. Sin embargo si se utilizan cookies el *SID* no se perderá mientras exista la cookie. El valor por defecto es **0** y las cookies se destruyen cuando se cierra el navegador. Podríamos hacer que la sesión se prolongase incluso si se cierra el navegador, expresando un valor en segundos.
session.gc_maxlifetime	Tiempo en segundos que se matiene la sesión, aunque no haya ninguna actividad del usuario. Por defecto está asignado a **1440**, 24 minutos, si transcurre ese tiempo sin actividad por parte del usuario, se cerrará la sesión automáticamente.

Podemos ver el valor de todas las directivas de configuración del fichero php.ini con la función ***phpinfo***.

session_start - session_destroy

En el caso de que no esté activda la directiva **session.auto_start** (su valor por defecto es cero) deberemos iniciar la sesión llamando a la función ***session_start()***, si es la primera vez que la llamamos se inicia sesión y si ya la hubiésemos llamado se reanuda la sesión. Está función (a partir de PHP 5.3) devuelve ***false*** en caso de no poder iniciar o restaurar la sesión o ***true*** en caso contrario.

Si usamos cookies para la sesión, éstas se transmiten en las cabeceras de ***http*** con lo cual la llamada a ***session_start*** deberá estar al principio, antes de haber generado ***HTML***. Deberemos tener

en cuenta que todas las páginas que necesiten acceder a información guardada en la sesión (***$_SESSION)***, deberán iniciarse con la llamada a dicha función.

Si queremos guardar información de sesión bastará con crear una nueva clave en el array asociativo superglobal ***$_SESSION***, por ejemplo si queremos guardar el nombre de usuario con el que ha iniciado sesión en usuario: $_SESSION['user']=$_POST['usuario']. Mientras dure la sesión aunque el usuario visite otras páginas dentro de nuestro sitio, se mantendrá ese valor.

La sesión del usuario termina si éste cierra el navegador (siempre y cuando **session.cookie_lifetime** esté a **0** en el fichero *php.ini*). También podemos cerrarla de forma manual, por ejemplo cuando el usuario pulsa ***cerrar sesión*** en nuestra página, para ello podemos llamar a la función ***session_destroy()*** que elimina completamente la información de la sesión. Existe otra función ***session_unset***, que elimina las variables almacenadas en la sesión actual, pero que no elimina la información de la sesión, sería como un reseteo de la sesión.

Ejercicio 3. Modifica el ejercicio 1 para que ahora la autenticación se haga con un formulario en donde se pide nombre de usuario y contraseña, una vez verificado el usuario se iniciará una sesión en la que se guardará el nombre de usuario, lo cual indicará que el usuario ya se ha identificado en el sistema. Una vez identificado, muéstrale una página con 2 enlaces uno que le permita consultar la información de su perfil (nombre, apellidos, usuario, email) o modificarlo, haciendo uso del formulario que ya teníamos hecho. Da de alta un usuario admin, al que cuando acceda se le muestre además un enlace de dar de alta usuarios. Deberás asegurarte de que sólo el usuario admin puede hacer esto, incluso aunque un usuario acceda directamente al URL del formulario de altas.

Ejercicio 4. Crea una tienda on-line sencilla con un catálogo de productos y un carrito de la compra. Un catálogo de cuatro o cinco productos será suficiente. De cada producto se debe conocer al menos la descripción y el precio. Todos los productos deben tener una imagen que los identifique. Al lado de cada producto del catálogo deberá aparecer un botón Comprar que permita añadirlo al carrito. Si el usuario hace clic en el botón Comprar de un producto que ya estaba en el carrito, se deberá incrementar el número de unidades de dicho producto. Para cada producto que aparece en el carrito, habrá un botón Eliminar por si el usuario se arrepiente y quiere quitar un producto concreto del carrito de la compra. A continuación se muestra una captura de pantalla de una posible solución.

Ejercicio 5. Amplía el programa anterior de tal forma que se pueda ver el detalle de un producto. Para ello, cada uno de los productos del catálogo deberá tener un botón Detalle que, al ser accionado, debe llevar al usuario a la vista de detalle que contendrá una descripción exhaustiva del producto en cuestión. Se podrán añadir productos al carrito tanto desde la vista de listado como desde la vista de detalle.