Desarrollo Web en Entorno Servidor

Programación en PHP

José A. Lara

Sesiones, seguridad y autenticación

- En esta unidad vamos a tratar los conceptos necesarios para poder manejar sesiones y cookies, así como las diferencias entre ambos conceptos. Podremos aplicarlos en la seguridad y control de acceso básico para nuestras aplicaciones.
- Cualquier aplicación moderna tiene una zona de acceso restringido así como un registro e incluso una diferenciación de uso por roles de usuario. No es nada complejo implementarlo y con PHP es muy sencillo con la combinación y uso de sesiones y cookies.

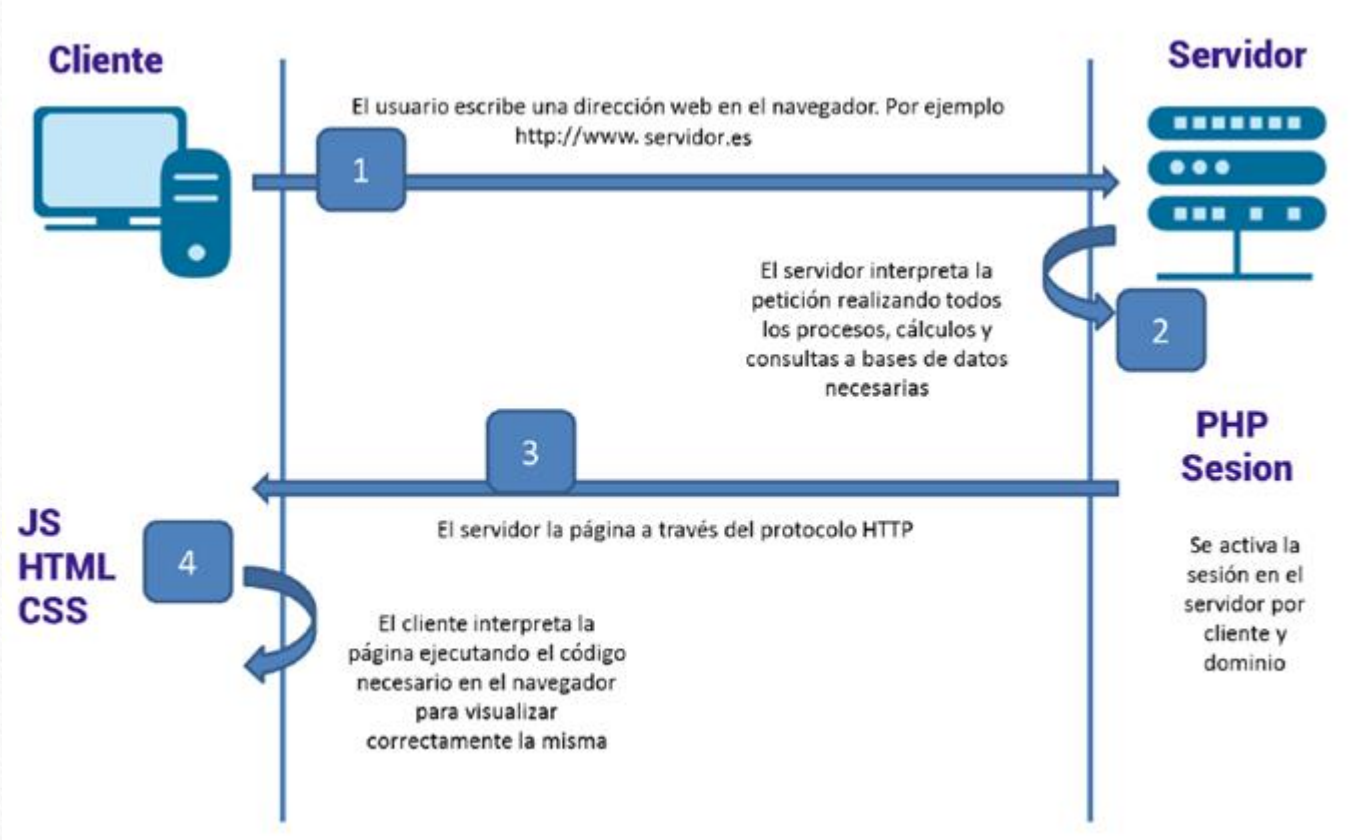
Sesiones, seguridad y autenticación

- Las sesiones nos permiten controlar desde la parte de servidor información referente para una petición de cliente.
- Las cookies nos permiten controlar información desde la parte de navegador o de cliente.
- La combinación de ambas propiedades nos permite realizar un control de acceso y seguridad completo para una determinada aplicación.
- El uso de sesiones y cookies no será complejo ya que pertenecen al entorno de superglobals, es decir variables que no hace falta crear ni referenciar y que son arrays asociativos que nos permiten almacenar y gestionar información, tal y como hacíamos con GET o POST.

Sesiones, seguridad y autenticación

Sesiones vs. Cookies

- Las sesiones, al igual que las cookies que veremos en el siguiente apartado, se encuentran dentro de lo que PHP denomina superglobals.



Sesiones, seguridad y autenticación

Sesiones vs. Cookies

- Como vemos en el anterior ejemplo, una vez que el cliente realiza la petición y esta alcanza el servidor, se activa **una nueva sesión** EN EL SERVIDOR, **por cliente y por dominio**. Es importante, y por eso lo remarcamos, que **la sesión se controla** por y en el SERVIDOR.

Sesiones, seguridad y autenticación

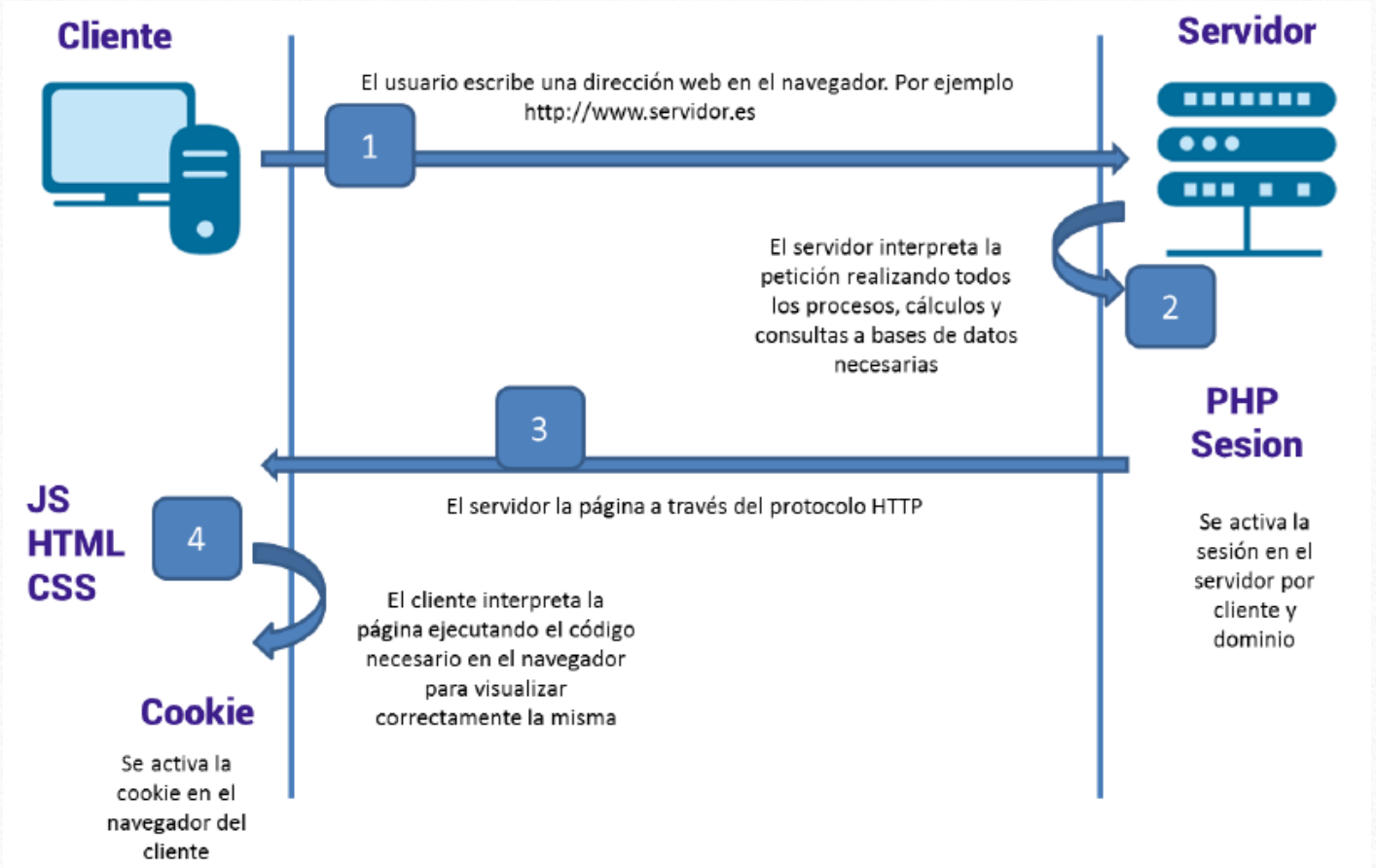
Sesiones vs. Cookies

- Esta **sesión** estará por lo tanto **activa** por dos motivos:
  - Siempre que el cliente mantenga el **navegador abierto**, independientemente de que todas las pestañas o conexiones estén cerradas contra ese servidor. Cuando el navegador se cierre se cerrará la sesión.
  - Hasta que el **servidor cierre la sesión**, bien porque haya pasado un **tiempo programado**, bien porque el **cliente envíe esa acción** (el típico **logout**), bien porque el **servidor decida** por funcionamiento de la aplicación cerrar la sesión.

Sesiones, seguridad y autenticación

Sesiones vs. Cookies

- El caso de las cookies es diferente:



Sesiones, seguridad y autenticación

Sesiones vs. Cookies

- Las cookies se **crean** y se **mantienen en el cliente**, es más, la cookie se crea en un **navegador** en concreto, esto es, si nosotros tenemos abiertos dos navegadores diferentes, Chrome y Firefox, por ejemplo, se pueden perfectamente tener dos cookies diferentes.

Sesiones, seguridad y autenticación

Sesiones vs. Cookies

- Así pues:
  - Las **sesiones** están relacionadas con el **servidor**, las **cookies** con el **cliente**.
  - Las sesiones se **cierran** al cerrar el navegador, las cookies **no se destruyen** al cerrar el navegador.
  - Las **sesiones** son **temporales** (mientras dura la sesión o bien el navegador abierto), las **cookies** son **permanentes** hasta que son destruidas.

Sesiones, seguridad y autenticación

Superglobal

- Algunas variables predefinidas en PHP son "superglobales", lo que significa que están disponibles en todos los ámbitos a lo largo del script. No es necesario emplear global $variable; para acceder a ellas dentro de las funciones o métodos.

Sesiones, seguridad y autenticación

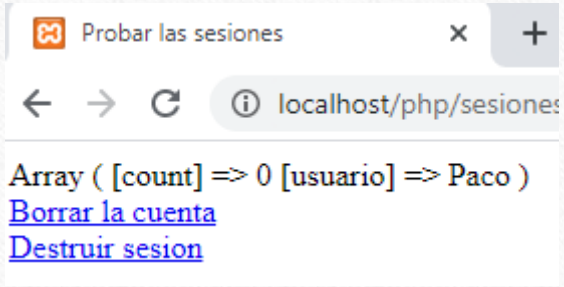
Sesiones

- Las sesiones son una forma sencilla de almacenar datos para usuarios de manera individual usando un **ID de sesión único**. Esto se puede usar para hacer **persistente** la **información de estado** entre peticiones de páginas.
- Los ID de sesiones normalmente son enviados al navegador mediante **cookies de sesión**, y el ID se usa para recuperar los datos de sesión existente.
- La **ausencia** de un ID o una cookie de sesión permite a PHP poder crear una **nueva sesión** y generar un nuevo ID de sesión.

Sesiones, seguridad y autenticación

Sesiones

- Veamos cómo podemos usar la superglobal $_SESSION para el almacenamiento de información.




```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Probar las sesiones</title>
  </head>
  <body>
    <?php
      session_start();
      if (!isset($_SESSION['count'])) {
        $_SESSION['count']=0;
        $_SESSION['usuario']="Paco";
        print_r($_SESSION);
      } else {
        $_SESSION['count']++;
        print_r($_SESSION);
      }
     ?>
     <br>
     <a href="borrarCuenta.php">Borrar la cuenta</a>
     <br>
     <a href="destruirSesion.php">Destruir sesion</a>
  </body>
</html>
```

Sesiones, seguridad y autenticación

Sesiones

1. Para **activar la sesión** desde PHP y por lo tanto unir la sesión del usuario con la sesión almacenada en el servidor debemos utilizar la función **session_start();** Si no utilizamos esta función, la sesión no se relacionará con el cliente.
2. Utilizando el array **$_SESSION**, podremos añadir, modificar y/o eliminar datos a dicha sesión.

Sesiones, seguridad y autenticación

Ejercicio

- Una vez que hemos visto el inicio de creación de las sesiones:
  1. Crearemos dentro de una estructura de directorios dentro de nuestro servidor un nuevo fichero denominado numVisitas.php
  2. Iniciamos la sesión con session_start
  3. Añadimos una variable numVisitas dentro de la superglobal $_SESSION
  4. Añadimos 1 siempre y cuando la sesión esté iniciada.

Sesiones, seguridad y autenticación

Destrucción de sesiones

- Una de las tres formas que teníamos para destruir una sesión era desde el servidor, bien porque el usuario nos indique que quiere eliminar la sesión y toda la información relacionada, o bien porque la aplicación así lo requiera.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Destruir sesion</title>
  </head>
  <body>
    <?php
      session_start();
      $_SESSION=[];
      print_r($_SESSION);
      session_destroy();
    ?>
  </body>
</html>
```

Sesiones, seguridad y autenticación

Destrucción de sesiones

- El primer y gran uso que le damos al uso de las sesiones es **integrarlo con el control de acceso** a las aplicaciones, los **roles** y la **seguridad**.
- Vamos a dar las pautas más importantes para la incorporación de las sesiones al sistema de seguridad.

Sesiones, seguridad y autenticación

Paso 1: Objeto Seguridad

- El primer paso será seguir el **modelo orientado a objetos** que estamos utilizando a lo largo de todo el contenido y generar una **nueva clase** que controle y gestione la parte de **sesión**. Esta clase se encontraría dentro de lo que se denomina **Controlador** en el Patrón MVC.
- Sin entrar a describir por completo la clase, las partes importantes que tendría serían:

```php
function __construct() {
    //Arrancamos la sesion
    session_start();
    if(isset($_SESSION["usuario"]))
        $this->usuario=$_SESSION["usuario"];
}
```

Sesiones, seguridad y autenticación

Paso 1: Objeto Seguridad

- Como vemos, al concentrar en **una sola clase** el manejo de las sesiones, colocaremos en el constructor de esta clase la activación de la clase, de esta forma se realizará siempre y sólo se realizará una vez.

```
$seguridad = new Seguridad();
```

- Al tener la clase definida, la utilización será muy sencilla, **creando un nuevo objeto** allí donde se vaya a necesitar.

Sesiones, seguridad y autenticación

PASO 2: El formulario de registro

- El formulario de registro puede ser tan sencillo como el que se muestra en la imagen o tan complejo como la aplicación lo requiera:

Sesiones, seguridad y autenticación

PASO 2: El formulario de registro

- Una de las particularidades que vemos en el anterior formulario y que sucede en un 100% de formularios de registro es la aparición de un **doble campo de comprobación de la contraseña** de registro del usuario.

```
<label for="pass0">Contraseña</label>
<input type="password" id="pass0" name="pass0" placeholder="Contraseña..">
<label for="pass1">Repite Contraseña</label>
<input type="password" id="pass1" name="pass1" placeholder="Contraseña..">
```

- Como observamos, los campos son de tipo password, pero lo importante en este caso es que el **name** de ambos campos **sea diferente** para poder recogerlos mediante $_POST y poderlos comparar.

Sesiones, seguridad y autenticación

PASO 3: La codificación de la contraseña

- Como observamos en el anterior ejemplo la contraseña es **texto plano** que recibimos a través de los campos pass0 y pass1. ¿Cómo se codifican dichas contraseñas en nuestra base de datos para que sea seguro?
- Utilizaremos funciones ya programadas en php tipo hash como es **sha** o **md5**.

https://stackoverflow.com/questions/16713810/how-secure-is-md5-and-sha1

Sesiones, seguridad y autenticación

PASO 3: La codificación de la contraseña

- Ejemplo sha

```php
$str = 'apple';
if (sha1($str) === 'd0be2dc421be4fcd0172e5afceea3970e2f3d940') {
    echo "Quiere una manzana roja o verde?";
}
```

- Ejemplo md5

```php
$str = 'apple';
if (md5($str) === '1f3870be274f6c49b3e31a0c6728957f') {
    echo "Quiere una manzana roja o verde?";
}
```

Sesiones, seguridad y autenticación

PASO 3: La codificación de la contraseña

- Como vemos, el resultado es muy parecido, produciendo en ambos casos hash que ahora sí almacenaremos en la base de datos. A su vez en la base de datos tendremos preparado un campo para almacenar el password que será de tipo **VARCHAR** si estamos trabajando con MySQL.

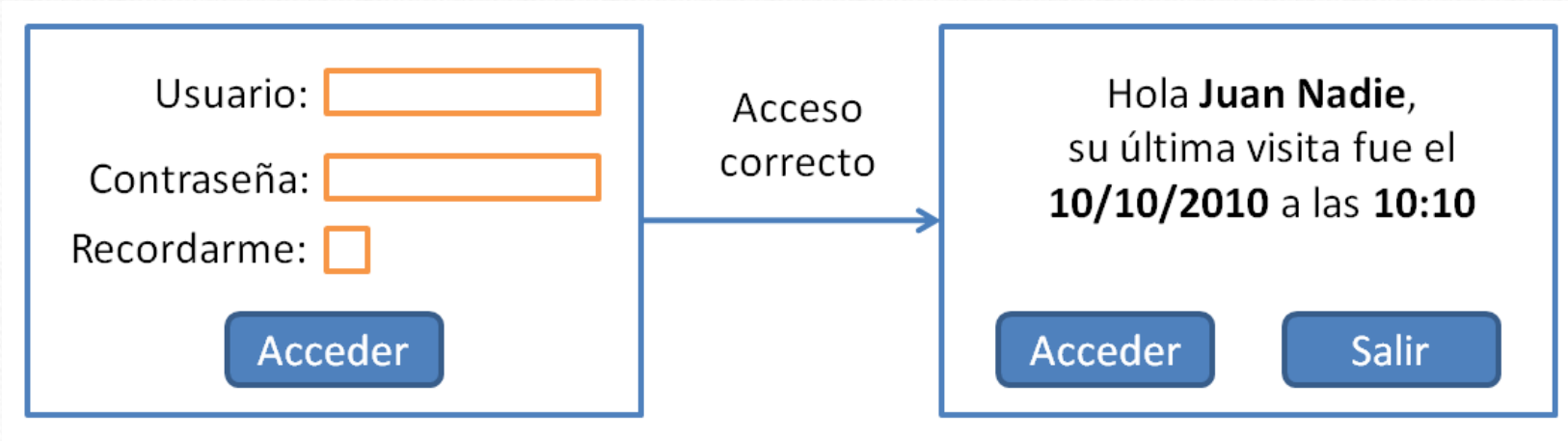
Sesiones, seguridad y autenticación

PASO 4: El formulario de acceso

- El formulario de acceso nos permite **recoger la información de un usuario ya registrado** en la base de datos y comprobar que existe en la misma con dicho usuario y dicha password.
- El único proceso a tener en cuenta en este paso es que en el momento de **comprobar la contraseña**, la misma se encuentra codificada en la base de datos, normalmente con un sistema de encriptación tipo MD5 o SHA, tal y como hemos visto en el PASO3.

Sesiones, seguridad y autenticación

PASO 4: El formulario de acceso



Sesiones, seguridad y autenticación

PASO 4: El formulario de acceso

- Así pues, en este punto:
  1. Recogeremos los datos del formulario.
  2. Recogeremos los datos del usuario en la BBDD.
  3. Comprobaremos la contraseña almacenada con la contraseña que hemos recogido del formulario (recordemos que esa contraseña está como texto plano y por lo tanto deberemos aplicarle la misma función de codificación que en el caso de registro, sha1 o md5).

Sesiones, seguridad y autenticación

PASO 5: Activación de la sesión

- Una vez que el usuario se encuentra logado ya que lo hemos comprobado en el paso anterior, deberemos **activar la sesión** y **añadir el usuario** a dicha sesión:

```php
function __construct() {
//Arrancamos la sesion
    session_start();
    if(isset($_SESSION["usuario"]))
        $this->usuario=$_SESSION["usuario"];
}
public function getUsuario(){ return $this->usuario; }
public function addUsuario($usuario,$pass,$remember=false){
//Generando la variable de sesion
    $_SESSION["usuario"]=$usuario;
    $this->usuario=$usuario;
    //Almacenaremos el user/pass cookies
    if($remember) {
        setcookie("usuario",$usuario,time()+(60*60));
        setcookie("pass",$pass,time()+(60*60));
    }
}
```

Sesiones, seguridad y autenticación

PASO 5: Activación de la sesión

- Vemos en el anterior código varias funciones que intervienen en este paso:
  - La primera ya la habíamos visto anteriormente ya que el **constructor** activa la sesión y comprueba si el $_SESSION[“usuario”] se encuentra activo.
  - **addUsuario** permite añadir el usuario a la sesión una vez que se ha logado.
  - **getUsuario** permite recoger el usuario.

Sesiones, seguridad y autenticación

PASO 6: Proteger una página

- Una vez que tenemos implementado todo el mecanismo anterior es muy sencillo proteger cualquier página con el siguiente código:

```php
include "./lib/Seguridad.php";
$seguridad=new Seguridad();
if($seguridad->getUsuario()==null){
    header('Location: login.php');
    exit;
}
```

Sesiones, seguridad y autenticación

PASO 6: Proteger una página

- Como vemos, antes de cargar absolutamente nada de la página a proteger, antes de la etiqueta <HTML>, introducimos el anterior código y en el que:
  - Primero incluimos la clase **Seguridad** y **creamos el objeto** (por lo que se activa la sesión).
  - **Comprobamos** que exista el usuario.
  - **Si no existe redirigimos** a una zona de la aplicación no protegida.

Sesiones, seguridad y autenticación

PASO 7: LogOut

- Realizar el logout es muy sencillo ya que únicamente debemos crear una **función** dentro de la clase seguridad que destruya la información dentro de la sesión:

```php
public function logOut(){
    $_SESSION=[];
    session_destroy();
}
```

Sesiones, seguridad y autenticación

Cookies

- Mediante las sesiones, podemos generar información particular/individual de un usuario que mantenemos **lo que dure la sesión abierta**. Es por lo tanto una **información temporal** que perderemos o bien cuando el usuario cierre la sesión, o bien cuando queramos borrar dicha sesión borrando todos los datos dentro de la sesión generados.
- Tenemos un segundo mecanismo de interacción con el navegador, utilizar las **cookies**.

Sesiones, seguridad y autenticación

Cookies

- Según la definición de php.net, las **cookies** son un mecanismo por el que se **almacenan datos en el navegador remoto** para monitorizar o identificar a los usuarios que vuelvan al sitio web.
- En este caso, **SÍ almacenamos información en el equipo del usuario**, y por lo tanto aunque el usuario cierre el navegador dicha información permanecerá durante cierto tiempo.

Sesiones, seguridad y autenticación

Cookies

- Esto tiene sus partes **positivas** y sus partes **negativas**:
  - Tenemos un **mecanismo sencillo** para poder almacenar información y además que sea dependiente del usuario que realiza la petición.
  - La información **podrá ser alcanzada por cualquier otro software**, y por lo tanto NO debemos almacenar información personal, sensible y que pueda ser utilizada posteriormente para el uso malicioso contra el usuario.

Sesiones, seguridad y autenticación

Cookies: Creación y recuperación

- **Crear una nueva cookie y recuperarla** va a ser una tarea tremendamente sencilla. Para ello seguiremos las indicaciones y el ejemplo que la misma documentación oficial de php.net nos ofrece:

```
Setting new cookie
==============================
<?php
    setcookie("name","value",time()+$int);
    /*name is your cookie's name value is cookie's value
    $int is time of cookie expires*/
?>
Getting Cookie
==============================
<?php
    echo $_COOKIE["your cookie name"];
?>
```

Sesiones, seguridad y autenticación

Cookies

```
Updating Cookie
==============================
<?php
    setcookie("color","red");
    echo $_COOKIE["color"];
    /*color is red*/ /* your codes and functions*/
    setcookie("color","blue");
    echo $_COOKIE["color"];
    /*new color is blue*/
?>
Deleting Cookie
==============================
<?php
    unset($_COOKIE["yourcookie"]); /*Or*/
    setcookie("yourcookie","yourvalue",time()-1);
    /*it expired so it's deleted*/
?>
```

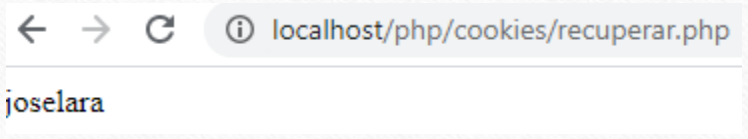
Sesiones, seguridad y autenticación

Cookies

- Como podemos observar en los ejemplos y en la documentación, una cookie permanecerá activa en nuestro navegador el **tiempo** que hayamos programado.

```php
<?php
    setcookie("user","joselara",time()+3600);
    /*name is your cookie's name
    value is cookie's value
    $int is time of cookie expires*/
?>
```

```php
<?php
    echo $_COOKIE["user"];
?>
```



Sesiones, seguridad y autenticación

Ejercicio

Una vez que hemos vista el inicio de creación de las cookies:

1. Crearemos dentro de una estructura de directorios dentro de nuestro servidor un nuevo fichero denominado numVisitas.php
2. Iniciamos la cookie numVisitas con el valor 0.
3. Añadimos 1 siempre y cuando la cookie exista.