

TEMA 4

Contenido

1.- Autenticación de usuarios y control de acceso.....	1
1.1.- Mecanismos de autenticación (I).....	2
1.1.1.- Mecanismos de autenticación (II).....	3
1.2.- Incorporación de métodos de autenticación a una aplicación web.....	5
2.- Cookies.....	7
3.- Manejo de sesiones.....	10
3.1.- Configuración.....	11
3.2.- Inicio y fin de una sesión.....	12
3.3.- Gestión de la información de la sesión (I).....	14
3.3.1.- Gestión de la información de la sesión (II).....	16
3.3.2.- Gestión de la información de la sesión (III).....	19
3.3.3.- Gestión de la información de la sesión (IV).....	21
4.- Herramientas para depuración de código.....	24
4.1.- Instalación de herramientas de depuración.....	25
4.2.- Depuración de código en PHP.....	26

Desarrollo de aplicaciones web con PHP.

Caso práctico

Va siendo hora de comenzar a dar pasos firmes en el desarrollo del nuevo proyecto, y en BK Programación ultimán los preparativos y se toman las últimas decisiones para establecer el método que seguirán en su desarrollo.

Mientras, **Carlos** revisa los conocimientos adquiridos y decide hacer una lista con todo lo aprendido hasta el momento y otra con todo lo que aún no sabe hacer. El saldo final es bastante positivo: ya sabe utilizar variables, hacer llamadas a funciones, estructurar el código, utilizar formularios web, trabajar con bases de datos... ¡Incluso ha diseñado una pequeña página para gestionar su colección de comics!

Entre lo que no sabe hacer todavía hay algo que le llama la atención: no sabe cómo mantener los datos entre las distintas páginas de una aplicación. Es decir, la única forma que conoce es utilizando un formulario web..., o bases de datos. Pero tiene que haber otras maneras más adecuadas. Algún modo sencillo de mantener información del usuario dentro de la aplicación web.

De nuevo tendrá que pedir consejo a **Juan**.

1.- Autenticación de usuarios y control de acceso.

Caso práctico

Tal y como **Esteban** les ha explicado, el objetivo principal del proyecto no es crear una página web pública, con información sobre la empresa. Necesitan una aplicación web con un objetivo más específico: permitir a clientes y a empleados conocer información sobre los productos de la empresa.

Juan sabe que con estas condiciones, uno de los puntos fundamentales con los que deberá tratar en el nuevo proyecto es el control de acceso a la aplicación web. Todos los usuarios que accedan habrán de identificarse para poder acceder a las páginas del sitio web. Además, en función de si el usuario es un cliente o un empleado, habrá que darle acceso a una o a otra información.

Juan nunca ha programado sitios web con autenticación de los usuarios. Además, se encuentra terminando otro proyecto, y no dispone de demasiado tiempo. Por este motivo, le pide a **Carlos** que se documente sobre el tema para poder decidir el camino a tomar.

Muchas veces es importante verificar la identidad de los dos extremos de una comunicación. En el caso de una comunicación web, existen métodos para identificar tanto al servidor en el que se aloja el sitio web, como al usuario del navegador que se encuentra en el otro extremo.

Los sitios web que necesitan emplear identificación del servidor, como las tiendas o los bancos, utilizan el protocolo HTTPS. Este protocolo requiere de un certificado válido, firmado por una autoridad confiable, que es verificado por el navegador cuando se accede al sitio web. Además, HTTPS utiliza métodos de cifrado para crear un canal seguro entre el navegador y el servidor, de tal forma que no se pueda interceptar la información que se transmite por el mismo.

Para identificar a los usuarios que visitan un sitio web, se pueden utilizar distintos métodos como el DNI digital o certificados digitales de usuario (*documento digital que contiene información acerca del usuario como el nombre o la dirección. Esa información está firmada por otra entidad, llamada entidad certificadora, que debe ser de confianza y garantiza que la información que contiene es cierta*), pero el más extendido es solicitar al usuario cierta información que solo él conoce: la combinación de un nombre de usuario y una contraseña.

En la unidad anterior aprendiste a utilizar aplicaciones web para gestionar información almacenada en bases de datos. En la mayoría de los casos es importante implantar en este tipo de aplicaciones web, las que acceden a bases de datos, algún mecanismo de control de acceso que

obligue al usuario a identificarse. Una vez identificado, se puede limitar el uso que puede hacer de la información.

Así, puede haber sitios web en los que los usuarios autenticados pueden utilizar sólo una parte de la información (como los bancos, que permiten a sus clientes acceder únicamente a la información relativa a sus cuentas). Otros sitios web necesitan separar en grupos a los usuarios autenticados, de tal forma que la información a la que accede un usuario depende del grupo en que éste se encuentre. Por ejemplo, una aplicación de gestión de una empresa puede tener un grupo de usuarios a los que permite visualizar la información, y otro grupo de usuarios que, además de visualizar la información, también la pueden modificar.

Debes distinguir la autenticación de los usuarios y el control de acceso, de la utilización de mecanismos para asegurar las comunicaciones entre el usuario del navegador y el servidor web. Aunque ambos aspectos suelen ir unidos, son independientes.

En los ejemplos de esta unidad, la información de autenticación (nombre y contraseña de los usuarios) se envía en texto plano desde el navegador hasta el servidor web. **Esta práctica es altamente insegura y nunca debe usarse sin un protocolo como HTTPS que permita cifrar las comunicaciones con el servidor web.** Sin embargo, la configuración de servidores web que permitan usar el protocolo HTTPS para cifrar la información que reciben y transmiten no forma parte de los contenidos de este módulo. Por este motivo, durante esta unidad utilizaremos únicamente el protocolo no seguro HTTP.

1.1.- Mecanismos de autenticación (I).

El protocolo HTTP ofrece un método sencillo para autenticar a los usuarios. El proceso es el siguiente:

- ✓ El servidor web debe proveer algún método para definir los usuarios que se utilizarán y cómo se pueden autenticar. Además, se tendrán que definir los recursos a los que se restringe el acceso y qué lista de control de acceso (ACL - lista de permisos sobre un objeto (fichero, directorio, etc.), que indica qué usuarios pueden utilizar el objeto y las acciones concretas que pueden realizar con el mismo (lectura, escritura, borrado, etc.)) se aplica a cada uno.
- ✓ Cuando un usuario no autenticado intenta acceder a un recurso restringido, el servidor web responde con un error de "Acceso no autorizado" (código 401).
- ✓ El navegador recibe el error y abre una ventana para solicitar al usuario que se autentique mediante su nombre y contraseña.
- ✓ La información de autenticación del usuario se envía al servidor, que la verifica y decide si permite o no el acceso al recurso solicitado. Esta información se mantiene en el navegador para utilizarse en posteriores peticiones a ese servidor.



En el servidor web Apache, el que has estado usando en anteriores unidades, existe una utilidad en línea de comandos, **htpasswd**, que permite almacenar en un fichero una lista de usuarios y sus respectivas contraseñas. La información relativa a las contraseñas se almacena cifrada; aun así, es conveniente crear este fichero en un lugar no accesible por los usuarios del servidor web.

<http://httpd.apache.org/docs/2.0/es/howto/auth.html>

Por ejemplo, para crear el fichero de usuario y añadirle el usuario "dwes", puedes hacer:

```
sudo htpasswd -c users dwes
```

e introducir la contraseña correspondiente a ese usuario.

La opción `-c` indica que se debe crear el fichero, por lo que solo deberás usarla cuando introduzcas el primer usuario y contraseña. Fíjate que en el ejemplo anterior, el fichero se crea en la ruta `/etc/apache2/users`, que en principio no es accesible vía web.

```
sam@ubuntu-profe: /etc/apache2
Archivo Editar Ver Terminal Ayuda
sam@ubuntu-profe:/etc/apache2$ sudo htpasswd -c users dwes
New password:
Re-type new password:
Adding password for user dwes
sam@ubuntu-profe:/etc/apache2$
```

Para indicarle al servidor Apache qué recursos tienen acceso restringido, una opción es 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
```

El significado de cada una de las directivas anteriores es el siguiente:

Directiva	Significado
AuthName	Nombre de dominio que se usará en la autenticación. Si el cliente se autentifica correctamente, esa misma información de autenticación se utilizará automáticamente en el resto de las páginas del mismo dominio.
AuthType	Método de autenticación 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.

Además tendrás que asegurarte de que en la configuración de Apache se utiliza la directiva `AllowOverride` para que se aplique correctamente la configuración que figura en los ficheros `.htaccess`.

<http://httpd.apache.org/docs/2.0/es/mod/core.html#allowoverride>

La sentencia `sudo htpasswd -c users admin` añade un nuevo usuario con nombre "admin" al fichero "users" que hemos creado anteriormente.

Verdadero Falso

Al incluir la opción `-c` lo que hacemos es crear un nuevo fichero, con lo cual eliminamos el contenido anterior del mismo.

1.1.1.- Mecanismos de autenticación (II).

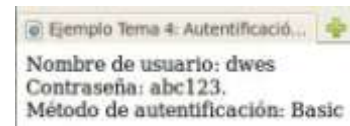
Desde PHP puedes acceder a la información de autenticación HTTP que ha introducido el usuario utilizando el array **superglobal** `$_SERVER`.

Valor	Contenido
<code>\$_SERVER['PHP_AUTH_USER']</code>	Nombre de usuario que se ha introducido.
<code>\$_SERVER['PHP_AUTH_PW']</code>	Contraseña introducida.
<code>\$_SERVER['AUTH_TYPE']</code>	Método HTTP usado para autenticar. Puede ser Basic o Digest.

Es decir, que si creas una página web que muestre los valores de estas variables, y preparas el servidor web para utilizar autenticación HTTP, cuando accedas a esa página con el usuario "dwes" obtendrás algo como lo siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 4 : Desarrollo de aplicaciones web con PHP -->
<!-- Ejemplo: Autenticación HTTP -->
<html>
```

```
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Ejemplo Tema 4: Autenticación HTTP</title>
<link href="dwes.css" rel="stylesheet" type="text/css">
</head>
<body>
<?php
echo "Nombre de usuario: ".$_SERVER['PHP_AUTH_USER']."<br />";
echo "Contraseña: ".$_SERVER['PHP_AUTH_PW']."<br />";
echo "Método de autenticación: ".$_SERVER['AUTH_TYPE']."<br />";
?>
</body>
</html>
```



Si no introduces un usuario/contraseña válidos, el navegador te mostrará el error 401.



Además, en PHP puedes usar la función `header` para forzar a que el servidor envíe un error de "Acceso no autorizado" (código 401). De esta forma no es necesario utilizar ficheros `.htaccess` para indicarle a Apache qué recursos están restringidos. En su lugar, puedes añadir las siguientes líneas en tus páginas PHP:

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic Realm="Contenido restringido"');
    header('HTTP/1.0 401 Unauthorized');
    echo "Usuario no reconocido!";
    exit;
}
?>
```

La función `header` envía encabezados HTTP (bloque de datos que forma parte del protocolo HTTP y se envía antes de la información propia que se transmite. Permite especificar códigos de estado, acciones requeridas al servidor, o el tipo de información que se transmite), pero debe utilizarse antes de que se muestre nada por pantalla. En caso contrario, obtendrás un error.

<http://es.php.net/manual/es/function.header.php>

Con el código anterior, la página envía un error 401, lo que fuerza al navegador a solicitar las credenciales de acceso (nombre de usuario y contraseña). Si se introducen, se ejecuta el resto de la página y se muestra su contenido. En este caso, **habría que añadir algún código para comprobar que el nombre de usuario y contraseña son válidos**, tal y como veremos a continuación. Si se pulsa el botón "Cancelar", se muestra el mensaje de error que se indica.

Modifica la página anterior utilizando la función header para que solicite las credenciales al usuario.

Tendrás que crear una página similar a la anterior, y añadir el código para forzar el error 401 antes de cualquier otro.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 4 : Desarrollo de aplicaciones web con PHP -->
<!-- Ejemplo: Función header para autenticación HTTP -->
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic Realm="Contenido restringido"');
    header('HTTP/1.0 401 Unauthorized');
    echo "Usuario no reconocido!";
    exit;
}
?>
<html>
<head>
```

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Ejercicio: Función header para autenticación HTTP</title>
<link href="dwes.css" rel="stylesheet" type="text/css">
</head>
<body>
<?php
echo "Nombre de usuario: ".$_SERVER['PHP_AUTH_USER']."<br />";
echo "Contraseña: ".$_SERVER['PHP_AUTH_PW']."<br />";
?>
</body>
</html>
```

1.2.- Incorporación de métodos de autenticación a una aplicación web.

Si utilizas la función `header` para forzar al navegador a solicitar credenciales HTTP, el usuario introducirá un nombre y una contraseña. Pero el servidor no verificará esta información; deberás ser tú quien provea un método para comprobar que las credenciales que ha introducido el usuario son correctas.

El método más simple es incluir en el código PHP de tu página las sentencias necesarias para comparar los datos introducidos con otros datos fijos. Por ejemplo, para permitir el acceso a un usuario "dwes" con contraseña "abc123.", puedes hacer:

```
<?php
if ($_SERVER['PHP_AUTH_USER'] != 'dwes' || $_SERVER['PHP_AUTH_PW'] != 'abc123.') {
    header('WWW-Authenticate: Basic Realm="Contenido restringido"');
    header('HTTP/1.0 401 Unauthorized');
    echo "Usuario no reconocido!";
    exit;
}
?>
```

Recuerda que el código PHP no se envía al navegador, por lo que la información de autenticación que introduzcas en el código no será visible por el usuario. Sin embargo, esto hará tu código menos portable. Si necesitas modificar el nombre de usuario o la contraseña, tendrás que hacer modificaciones en el código. Además, no podrás permitir al usuario introducir su propia contraseña. Una solución mejor es utilizar un almacenamiento externo para los nombres de usuario y sus contraseñas. Para esto podrías emplear un fichero de texto, o mejor aún, una base de datos. La información de autenticación podrá estar aislada en su propia base de datos, o compartir espacio de almacenamiento con los datos que utilice tu aplicación web.

Si quieres almacenar la información de los usuarios en la base de datos "dwes", tienes que crear una nueva tabla en su estructura. Para ello, revisa y ejecuta estas sentencias SQL.

```
-- Seleccionamos la base de datos
USE dwes;

-- Creamos la tabla
CREATE TABLE usuarios (
    usuario VARCHAR(20) NOT NULL PRIMARY KEY,
    contraseña VARCHAR(32) NOT NULL
) ENGINE = INNODB;

-- Creamos el usuario dwes
INSERT INTO usuarios (usuario, contraseña) VALUES
('dwes', 'e8dc8ccd5e5f9e3a54f07350ce8a2d3d');
```

Aunque se podrían almacenar las contraseñas en texto plano, es mejor hacerlo en formato encriptado. En el ejemplo anterior, para el usuario "dwes" se almacena el hash MD5 (*método para generar un resumen de un texto o un documento, de tal forma que a partir del resumen obtenido no es posible recuperar el texto original, ni hallar otro texto a partir del cual se obtenga el mismo resumen. Se llama hash al resumen obtenido al aplicar una función hash. Una de las funciones hash más extendidas es MD5, que genera 128 bits como resumen (normalmente se representa mediante una cadena de texto de 28 caracteres o mediante 32 dígitos hexadecimales)*) correspondiente a la contraseña "abc123.". En PHP puedes usar la función `md5` para calcular el hash MD5 de una cadena de texto.

<http://es.php.net/manual/es/function.md5.php>

Utiliza la extensión MySQLi para modificar el ejercicio anterior, de tal forma que las credenciales del usuario se comprueben con la información de la nueva tabla "usuarios" creada en la base de datos "dwes". Si no existe el usuario, o la contraseña es incorrecta, vuelve a pedir las credenciales al usuario.

Revisa la solución propuesta. Fíjate que se debe usar la función md5 para comprobar la contraseña. Si introduces un usuario o contraseña incorrectos, el comportamiento depende del navegador que utilices; algunos te pedirán las credenciales de forma indefinida, y otros un número limitado de veces.

```
<?php
// Si el usuario aún no se ha autenticado, pedimos las credenciales
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="Contenido restringido");
    header("HTTP/1.0 401 Unauthorized");
    exit;
}
// Si ya ha enviado las credenciales, las comprobamos con la base de datos
else {
    // Conectamos a la base de datos
    @ $dwes = new mysqli("localhost", "dwes", "abc123.", "dwes");
    $error = $dwes->connect_errno;
    // Si se estableció la conexión
    if ($error == null) {
        // Ejecutamos la consulta para comprobar si existe
        // esa combinación de usuario y contraseña
        $sql = "SELECT usuario FROM usuarios
                WHERE usuario='$ SERVER[PHP AUTH USER]' AND
                contrasena=md5('$ SERVER[PHP AUTH PW]')";
        $resultado = $dwes->query($sql);
        // Si no existe, se vuelven a pedir las credenciales
        if($resultado->num_rows == 0) {
            header('WWW-Authenticate: Basic realm="Contenido restringido");
            header("HTTP/1.0 401 Unauthorized");
        }
        exit;
    }
    $resultado->close();
    $dwes->close();
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 4 : Desarrollo de aplicaciones web con PHP -->
<!-- Ejemplo: Utilización de MySQL para autenticación HTTP -->
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Ejemplo Tema 4: Utilización de MySQL para autenticación HTTP</title>
<link href="dwes.css" rel="stylesheet" type="text/css">
</head>
<body>
<?php
echo "Nombre de usuario: " . $_SERVER['PHP_AUTH_USER'] . "<br />";
echo "Hash de la contraseña: " . md5($_SERVER['PHP_AUTH_PW']) . "<br />";
?>
</body>
</html>
```

Las dos posibilidades que hemos visto para solicitar al usuario que se autentique vía HTTP son la creación del fichero ".htaccess", y la utilización de la función header de PHP. ¿Cuál de esas dos formas será preferible si quieres que los privilegios de acceso a tu aplicación varíen en función del día de la semana (por ejemplo, unos usuarios que puedan acceder de lunes a viernes y otros distintos el fin de semana)?



El fichero .htaccess



La función header

Si utilizas la función header para enviar los encabezados HTTP, debes escribir tú el código que decide si el usuario se ha autenticado correctamente y se le permite acceder o no. Por tanto, puedes incluir las reglas adicionales que quieras, por ejemplo, darle o no acceso en función del día de la semana.

2.- Cookies.

Caso práctico

Una vez resuelto el tema de la autenticación, el siguiente objetivo de **Carlos** es aprender a gestionar las cookies. **Juan** le ha explicado qué son y cómo funcionan, y seguramente las tengan que utilizar en la aplicación que están desarrollando.

Sabe que muchos de los sitios web que visita las utilizan, porque ha probado a desactivarlas en su navegador, y le han empezado a aparecer mensajes pidiéndole que las vuelva a activar. Se ha propuesto averiguar para qué las utilizan, y cómo las podrán gestionar desde PHP.

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 (por ejemplo, el idioma en que se deben mostrar las páginas), para que no tenga que volver a indicárselas la próxima vez que visite el sitio.

Si utilizas Firefox como navegador, puedes instalar la extensión Web Developer para obtener información sobre las páginas web que visitas. Entre sus características te permite consultar y editar las cookies almacenadas en el mismo.

<https://addons.mozilla.org/es/firefox/addon/web-developer/>

En PHP, para almacenar una cookie en el navegador del usuario, puedes utilizar la función `setcookie`. El único parámetro obligatorio que tienes que usar es el nombre de la cookie, pero admite varios parámetros más opcionales.

<http://es.php.net/manual/es/function.setcookie.php>

Por ejemplo, si quieres almacenar en una cookie el nombre de usuario que se transmitió en las credenciales HTTP (es solo un ejemplo, no es en absoluto aconsejable almacenar información relativa a la seguridad en las cookies), puedes hacer:

```
setcookie("nombre_usuario", $_SERVER['PHP_AUTH_USER'], time()+3600);
```

Los dos primeros parámetros son el nombre de la cookie y su valor. El tercero es la fecha de caducidad de la misma (una hora desde el momento en que se ejecute). En caso de no figurar este parámetro, la cookie se eliminará cuando se cierre el navegador. Ten en cuenta que también se pueden aplicar restricciones a las páginas del sitio que pueden acceder a una cookie en función de la ruta.

Las cookies se transmiten entre el navegador y el servidor web de la misma forma que las credenciales que acabas de ver; utilizando los encabezados del protocolo HTTP. Por ello, las sentencias `setcookie` deben enviarse antes de que el navegador muestre información alguna en pantalla.

El proceso de recuperación de la información que almacena una cookie es muy simple. Cuando accedes a un sitio web, el navegador le envía de forma automática todo el contenido de las cookies que almacene relativas a ese sitio en concreto. Desde PHP puedes acceder a esta información por medio del array `$_COOKIE`.

Siempre que utilices cookies en una aplicación web, debes tener en cuenta que en última instancia su disponibilidad está controlada por el cliente. Por ejemplo, algunos usuarios deshabilitan las cookies en el navegador porque piensan que la información que almacenan puede suponer un potencial problema de seguridad. O la información que almacenan puede llegar a perderse porque el usuario decide formatear el equipo o simplemente eliminarlas de su sistema.

Si una vez almacenada una cookie en el navegador quieres eliminarla antes de que expire, puedes utilizar la misma función `setcookie` pero indicando una fecha de caducidad anterior a la actual.

Sobre el mismo ejercicio anterior, almacena en una cookie el último instante en que el usuario visitó la página. Si es su primera visita, muestra un mensaje de bienvenida. En caso contrario, muestra la fecha y hora de su anterior visita.

Revisa la solución propuesta. Deberás utilizar la función `setcookie` para guardar el instante de su anterior visita y mostrar su contenido utilizando el array `$_COOKIE`.

```
<?php
// Si el usuario aún no se ha autenticado, pedimos las credenciales
if( !isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="Contenido restringido");
    header("HTTP/1.0 401 Unauthorized");
    exit;
}
// Si ya ha enviado las credenciales, las comprobamos con la base de datos
else {
    // Conectamos a la base de datos
    @ $dwes = new mysqli("localhost", "dwes", "abc123.", "dwes");
    $error = $dwes->connect_errno;
    // Si se estableció la conexión
    if ($error == null) {
        date_default_timezone_set('Europe/Madrid');
        // Ejecutamos la consulta para comprobar si existe
        // esa combinación de usuario y contraseña
        $sql = "SELECT usuario FROM usuarios
            WHERE usuario='{$_SERVER['PHP_AUTH_USER']}' AND
            contraseña=md5('{$_SERVER['PHP_AUTH_PW']}')";
        $resultado = $dwes->query($sql);
        // Si no existe, se vuelven a pedir las credenciales
        if($resultado->num rows == 0) {
            header('WWW-Authenticate: Basic realm="Contenido restringido");
            header("HTTP/1.0 401 Unauthorized");
            exit;
        } else {
            if (isset($_COOKIE['ultimo login'])) {
                $ultimo_login = $_COOKIE['ultimo login'];
            }
            setcookie("ultimo_login", time(), time()+3600);
        }
        $resultado->close();
        $dwes->close();
    }
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 4 : Desarrollo de aplicaciones web con PHP -->
<!-- Ejemplo: Cookies en autenticación HTTP -->
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Ejemplo Tema 4: Cookies en autenticación HTTP</title>
<link href="dwes.css" rel="stylesheet" type="text/css">
</head>
<body>
<?php
    if ($error == null) {
        echo "Nombre de usuario: ".$SERVER['PHP_AUTH_USER']."<br />";
        echo "Hash de la contraseña: ".md5($SERVER['PHP_AUTH_PW'])."<br />";
        if (isset($ultimo_login))
            echo "Ultimo login: " . date("d/m/y \a \l\a\s H:i", $ultimo_login);
        else
            echo "Bienvenido. Esta es su primera visita.";
    }
    else
        echo "Se ha producido el error $error.<br />";
?>
</body>
</html>
```

¿Cuál es la duración por defecto de una cookie si no se indica la fecha de caducidad, como en la siguiente llamada a la función setcookie?

```
setcookie("idioma", "español");
```



Hasta que se cierre el navegador del usuario.



1 hora.

Cuando se acaba la sesión del usuario, al cerrar el navegador, se borra la cookie.

3.- Manejo de sesiones.

Caso práctico

Carlos ha aprendido a utilizar las cookies, y cuáles son sus limitaciones. Está claro que les serán de utilidad, pero también que no cubren las necesidades de una aplicación web. Al viajar en las cabeceras HTTP, la información que pueden mantener está muy limitada.

Aunque sabe que aún le queda mucho por aprender, Juan le ha comentado que con los conocimientos que acaba de adquirir ya tiene una buena base para dar los primeros pasos programando en PHP. Dice que sólo le falta aprender cómo funcionan las sesiones para empezar a desarrollar aplicaciones web completas por su cuenta.

Animado por estos comentarios, Carlos decide dar un paso más. Veamos cómo funcionan las sesiones en PHP.

Como acabas 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, o su tamaño máximo. 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.

Cada usuario distinto de un sitio web tiene su propia información de sesión. Para distinguir una sesión de otra se usan los identificadores de sesión (SID). Un SID es un atributo que se asigna a cada uno de los visitantes de un sitio web y lo identifica. De esta forma, si el servidor web conoce el SID de un usuario, puede relacionarlo con toda la información que posee sobre él, que se mantiene en la sesión del usuario. Esa información se almacena en el servidor web, generalmente en ficheros aunque también se pueden utilizar otros mecanismos de almacenamiento como bases de datos.

Como ya habrás supuesto, la cuestión ahora es: ¿y dónde se almacena ese SID, el identificador de la sesión, que es único para cada usuario? Pues existen dos maneras de mantener el SID entre las páginas de un sitio web que visita el usuario:

- ✓ Utilizando cookies, tal y como ya viste.
- ✓ Propagando el SID en un parámetro de la URL. El SID se añade como una parte más de la URL, de la forma:

<http://www.misitioweb.com/tienda/listado.php&PHPSESSID=34534fg4ffg34ty>

En el ejemplo anterior, el SID es el valor del parámetro `PHPSESSID`.

Ninguna de las dos maneras es perfecta. Ya sabes los problemas que tiene la utilización de cookies. Pese a ello, es el mejor método y el más utilizado. 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.

La buena noticia, es que el proceso de manejo de sesiones en PHP está automatizado en gran medida. Cuando un usuario visita un sitio web, no es necesario programar un procedimiento para ver si existe un SID previo y cargar los datos asociados con el mismo. Tampoco tienes que utilizar la función `setcookie` si quieres almacenar los SID en cookies, o ir pasando el SID entre las páginas web de tu sitio si te decides por propagarlo. Todo esto PHP lo hace automáticamente.

A la información que se almacena en la sesión de un usuario también se le conoce como cookies del lado del servidor (server side cookies). Debes tener en cuenta que aunque esta información no viaja entre el cliente y el servidor, sí lo hace el SID, bien como parte de la URL o en un encabezado HTTP si se guarda en una cookie. En ambos casos, esto plantea un posible problema de seguridad. El SID puede ser conseguido por otra persona, y a partir

del mismo obtener la información de la sesión del usuario. La manera más segura de utilizar sesiones es almacenando los SID en cookies y utilizar HTTPS para encriptar la información que se transmite entre el servidor web y el cliente.

3.1.- Configuración.

Por defecto, PHP incluye soporte de sesiones incorporado. Sin embargo, antes de utilizar sesiones en tu sitio web, debes configurar correctamente PHP utilizando las siguientes directivas en el fichero `php.ini` según corresponda:

Directiva	Significado
<code>session.use_cookies</code>	Indica si se deben usar cookies (1) o propagación en la URL (0) para almacenar el SID.
<code>session.use_only_cookies</code>	Se debe activar (1) cuando utilizas cookies para almacenar los SID, y además no quieres que se reconozcan los SID que se puedan pasar como parte de la URL (este método se puede usar para usurpar el identificador de otro usuario).
<code>session.save_handler</code>	Se utiliza para indicar a PHP cómo debe almacenar los datos de la sesión del usuario. Existen cuatro opciones: en ficheros (<code>files</code>), en memoria (<code>mm</code>), en una base de datos SQLite (<code>sqlite</code>) o utilizando para ello funciones que debe definir el programador (<code>user</code>). El valor por defecto (<code>files</code>) funcionará sin problemas en la mayoría de los casos.
<code>session.name</code>	Determina el nombre de la cookie que se utilizará para guardar el SID. Su valor por defecto es <code>PHPSESSID</code> .
<code>session.auto_start</code>	Su valor por defecto es 0, y en este caso deberás usar la función <code>session_start</code> para gestionar el inicio de las sesiones. Si usas sesiones en el sitio web, puede ser buena idea cambiar su valor a 1 para que PHP active de forma automática el manejo de sesiones.
<code>session.cookie_lifetime</code>	Si utilizas la URL para propagar el SID, éste se perderá cuando cierres tu navegador. Sin embargo, si utilizas cookies, el SID se mantendrá mientras no se destruya la cookie. En su valor por defecto (0), las cookies se destruyen cuando se cierra el navegador. Si quieres que se mantenga el SID durante más tiempo, debes indicar en esta directiva ese tiempo en segundos.
<code>session.gc_maxlifetime</code>	Indica el tiempo en segundos que se debe mantener activa la sesión, aunque no haya ninguna actividad por parte del usuario. Su valor por defecto es 1440. Es decir, pasados 24 minutos desde la última actividad por parte del usuario, se cierra su sesión automáticamente.

La función `phpinfo`, de la que ya hablamos con anterioridad, te ofrece información sobre la configuración actual de las directivas de sesión.

En la documentación de PHP tienes información sobre éstas y otras directivas que permiten configurar el manejo de sesiones.
<http://es.php.net/manual/es/session.configuration.php>

SESSION

Session Support	enabled
Registered name handlers:	file user redis
Registered serializer handlers	php php_serialize

Directive	Local Value	Master Value
<code>session.auto_start</code>	0	0
<code>session.bug_compat_42</code>	On	On
<code>session.bug_compat_warn</code>	On	On
<code>session.cache_expire</code>	180	180
<code>session.cache_limiter</code>	nocache	nocache
<code>session.cookie_domain</code>	no value	no value
<code>session.cookie_httponly</code>	Off	Off
<code>session.cookie_lifetime</code>	0	0
<code>session.cookie_path</code>	/	/
<code>session.cookie_secure</code>	Off	Off
<code>session.cookie_samesite</code>	no value	no value
<code>session.cookie_timeout</code>	0	0
<code>session.gc_divisor</code>	100	100
<code>session.gc_maxlifetime</code>	1440	1440
<code>session.gc_probability</code>	1	1
<code>session.hash_bits_per_character</code>	5	5
<code>session.hash_function</code>	0	0
<code>session.name</code>	PHPSESSID	PHPSESSID
<code>session.referer_check</code>	no value	no value
<code>session.save_handler</code>	file	file
<code>session.save_path</code>	/tmp/	/tmp/
<code>session.serialize_handler</code>	php	php
<code>session.use_cookies</code>	On	On
<code>session.use_only_cookies</code>	Off	Off
<code>session.use_strict_mode</code>	0	0

Si la información del usuario que quieres almacenar incluye contenido privado como una contraseña, ¿qué utilizarías, cookies o la sesión del usuario?



La sesión del usuario.



Cookies.

La sesión del usuario se almacena y se procesa en el servidor web, por lo que no es necesario transmitirla hasta el ordenador del usuario, lo que aumenta su seguridad.

3.2.- Inicio y fin de una sesión.

El inicio de una sesión puede tener lugar de dos formas. Si has activado la directiva `session.auto_start` en la configuración de PHP, la sesión comenzará automáticamente en cuanto un usuario se conecte a tu sitio web. En caso de que ese usuario ya haya abierto una sesión con anterioridad, y esta no se haya eliminado, en lugar de abrir una nueva sesión se reanudará la anterior. Para ello se utilizará el SID anterior, que estará almacenado en una cookie (recuerda que si usas propagación del SID, no podrás restaurar sesiones anteriores; el SID figura en la URL y se pierde cuando cierras el navegador).

Si por el contrario, decides no utilizar el inicio automático de sesiones, deberás ejecutar la función `session_start` para indicar a PHP que inicie una nueva sesión o reanude la anterior. Aunque anteriormente esta función devolvía siempre `true`, a partir de la versión 5.3.0 de PHP su comportamiento es más coherente: devuelve `false` en caso de no poder iniciar o restaurar la sesión.

Como el inicio de sesión requiere utilizar cookies, y éstas se transmiten en los encabezados HTTP, debes tener en cuenta que para poder iniciar una sesión utilizando `session_start`, tendrás que hacer las llamadas a esta función antes de que la página web muestre información en el navegador.

Además, todas las páginas que necesiten utilizar la información almacenada en la sesión, deberán ejecutar la función `session_start`.

Mientras la sesión permanece abierta, puedes utilizar la variable superglobal `$_SESSION` para añadir información a la sesión del usuario, o para acceder a la información almacenada en la sesión. Por ejemplo, para contar el número de veces que el usuario visita la página, puedes hacer:

```
<?php
// Iniciamos la sesión o recuperamos la anterior sesión existente
session start();
// Comprobamos si la variable ya existe
if (isset($_SESSION['visitas']))
    $_SESSION['visitas']++;
else
    $_SESSION['visitas'] = 0;
?>
```

Si en lugar del número de visitas, quisieras almacenar el instante en que se produce cada una, la variable de sesión "visitas" deberá ser un array y por tanto tendrás que cambiar el código anterior por:

```
<?php
// Iniciamos la sesión o recuperamos la anterior sesión existente
session start();
// En cada visita añadimos un valor al array "visitas"
$_SESSION['visitas'][] = mktime();
?>
```

Aunque como ya viste, puedes configurar PHP para que elimine de forma automática los datos de una sesión pasado cierto tiempo, en ocasiones puede ser necesario cerrarla de forma manual en un momento determinado. Por ejemplo, si utilizas sesiones para recordar la información de autenticación, deberás darle al usuario del sitio web la posibilidad de cerrar la sesión cuando lo crea conveniente.

En PHP tienes dos funciones para eliminar la información almacenada en la sesión:

- ✓ `session_unset()`. Elimina las variables almacenadas en la sesión actual, pero no elimina la información de la sesión del dispositivo de almacenamiento usado.
- ✓ `session_destroy()`. Elimina completamente la información de la sesión del dispositivo de almacenamiento.

Crea una página similar a la anterior, almacenando en la sesión de usuario los instantes de todas sus últimas visitas. Si es su primera visita, muestra un mensaje de bienvenida. En caso contrario, muestra la fecha y hora de todas sus visitas anteriores. Añade un botón a la página que permita borrar el registro de visitas.

Utiliza también una variable de sesión para comprobar si el usuario se ha autenticado correctamente. De esta forma no hará falta comprobar las credenciales con la base de datos constantemente.

Revisa la solución propuesta. Acuérdate de que debes hacer una llamada a la función `session_start()` antes de utilizar la variable superglobal `$_SESSION` para acceder a la información de la sesión.

```
<?php
// Si el usuario aún no se ha autenticado, pedimos las credenciales
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="Contenido restringido"');
    header("HTTP/1.0 401 Unauthorized");
    exit;
}
// Vamos a guardar el usuario en una variable de sesión
// si no existe, aún no se ha autenticado
session_start();
if (!isset($_SESSION['usuario'])) {
    // Conectamos a la base de datos
    @ $dwes = new mysqli("localhost", "dwes", "abc123.", "dwes");
    $error = $dwes->connect_errno;
    // Si se estableció la conexión
    if ($error == null) {
        // Ejecutamos la consulta para comprobar si existe
        // esa combinación de usuario y contraseña
        $sql = "SELECT usuario FROM usuarios
            WHERE usuario='{$_SERVER['PHP_AUTH_USER']}' AND
            contraseña=md5('{$_SERVER['PHP_AUTH_PW']}')";
        $resultado = $dwes->query($sql);
        // Si no existe, se vuelven a pedir las credenciales
        if($resultado->num_rows == 0) {
            header('WWW-Authenticate: Basic realm="Contenido restringido"');
            header("HTTP/1.0 401 Unauthorized");
            exit;
        }else
            $_SESSION['usuario'] = $_SERVER['PHP_AUTH_USER'];
        $resultado->close();
        $dwes->close();
    }
}
// Si ya está autenticado
else {
    // Comprobamos si se ha enviado el formulario de limpiar el registro
    if (isset($_POST['limpiar']))
        unset($_SESSION['visita']);
    else
        $_SESSION['visita'][] = time();
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 4 : Desarrollo de aplicaciones web con PHP -->
<!-- Ejemplo: Cookies en autenticación HTTP -->
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <title>Ejemplo Tema 4: Cookies en autenticación HTTP</title><link href="dwes.css"
rel="stylesheet" type="text/css">
```

```

</head>
<body>
  <?php
    if ($error == null) {
      echo "Nombre de usuario: ".$SERVER['PHP_AUTH_USER']."<br />";
      echo "Hash de la contraseña: ".md5($_SERVER['PHP_AUTH_PW'])."<br />";
      if (count($_SESSION['visita']) == 0)
        echo "Bienvenido. Esta es su primera visita.";
      else {
        date_default_timezone_set('Europe/Madrid');
        foreach($_SESSION['visita'] as $v)
          echo date("d/m/y \a \l\ \a\s H:i", $v) . "<br />";
      }
    }
    <form id='vaciar' action='<?php echo $_SERVER['PHP_SELF'];?>' method='post'>
      <input type='submit' name='limpiar' value='Limpiar registro' />
    </form>
    <?php
      }
    }else
      echo "Se ha producido el error $error.<br />";
    ?>
  </body>
</html>

```

Si usamos el inicio de sesión automático, la sesión de un usuario se inicia en cuanto se autentifica correctamente en el servidor web.



Verdadero



Falso

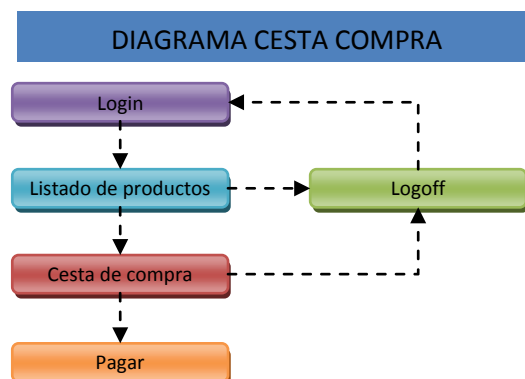
Con el inicio de sesión automático, la sesión del usuario se inicia en cuanto se conecta al sitio web, esté o no autenticado. No tiene nada que ver una cosa con otra. Un usuario no autenticado puede tener una sesión de la misma forma que otro que sí se haya autenticado correctamente

3.3.- Gestión de la información de la sesión (I).

En este punto vas a ver paso a paso un ejemplo de utilización de sesiones para almacenar la información del usuario. Utilizarás la base de datos "dwes", creada anteriormente, para crear un prototipo de una tienda web dedicada a la venta de productos de informática.

Las páginas de que constará tu tienda online son las siguientes:

- ✓ **Login** ([login.php](#)). Su función es autenticar al usuario de la aplicación web. Todos los usuarios de la aplicación deberán autenticarse utilizando esta página antes de poder acceder al resto de páginas.
- ✓ **Listado de productos** ([productos.php](#)). Presenta un listado de los productos de la tienda, y permite al usuario seleccionar aquellos que va a comprar.
- ✓ **Cesta de compra** ([cesta.php](#)). Muestra un resumen de los productos escogidos por el usuario para su compra y da acceso a la página de pago.
- ✓ **Pagar** ([pagar.php](#)). Una vez confirmada la compra, la última página debería ser la que permitiera al usuario escoger el método de pago y la forma de envío. En este ejemplo no la vas a implementar como tal. Simplemente mostrará un mensaje de tipo "Gracias por su compra" y ofrecerá un enlace para comenzar una nueva compra.
- ✓ **Logoff** ([logoff.php](#)). Esta página desconecta al usuario de la aplicación y redirige al usuario de forma automática a la pantalla de autenticación. No muestra ninguna información en pantalla, por lo que no es visible para el usuario.



Recuerda poner a las páginas los nombres que aquí figuran, almacenando todas en la misma carpeta. Si cambias algún nombre o mueves alguna página de lugar, los enlaces internos no funcionarán.

Aunque el aspecto de la aplicación no es importante para nuestro objetivo, utilizaremos una hoja de estilos, `tienda.css`, común a todas las páginas y una imagen, `cesta.png`, para ofrecer un interface más amigable.

```
.error {
  font-family: Verdana, Arial, sans-serif;
  font-size: 0.7em;
  color: #900;
  background-color : #ffff00;
}

.divisor {
  clear:both;
  height:0;
  font-size: 1px;
  line-height: 0px;
}

#login fieldset {
  position: absolute;
  left: 50%;
  top: 50%;
  width: 230px;
  margin-left: -115px;
  height: 160px;
  margin-top: -80px;
  padding:10px;
  border:1px solid #ccc;
  background-color: #eee;
}

#login legend {
  font-family : Arial, sans-serif;
  font-size: 1.3em;
  font-weight:bold;
  color:#333;
}

#login .campo {
  margin-top:8px;
  margin-bottom: 10px;
}

#login label {
  font-family : Arial, sans-serif;
  font-size:0.8em;
  font-weight: bold;
}

#login input[type="text"], #login
input[type="password"] {
  font-family : Arial, Verdana, sans-serif;
  font-size: 0.8em;
  line-height:140%;
  color : #000;
  padding : 3px;
  border : 1px solid #999;
  height:18px;
  width:220px;
}

#login input[type="submit"] {
  width:100px;
  height:30px;
  padding-left:0px;
}

.pagproductos body, .pagcesta body {
  font: 100% Verdana, Arial, Helvetica,
  sans-serif;
  background: #666;
```

```
margin: 0;
padding: 0;
text-align: center;
color: #000000;
}

#contenedor {
  width: 90%;
  background: #fff;
  margin: 0 auto;
  border: 1px solid #000;
  text-align: left;
}

#encabezado {
  padding: 0 10px;
  border-top-width: thin;
  border-right-width: thin;
  border-bottom-width: thin;
  border-left-width: thin;
  border-top-style: solid;
  border-right-style: solid;
  border-bottom-style: solid;
  border-left-style: solid;
  background-color: #9cf;
}

#encabezado h1 {
  margin: 0;
  padding: 10px 0;
}

.pagproductos #cesta {
  float: right;
  width: 12em;
  background-color: #588;
}

.pagproductos #cesta h3 {
  margin-left: 10px;
  margin-right: 10px;
}

.pagproductos #cesta p {
  margin-left: 10px;
  margin-right: 10px;
  font-size: 10px;
}

.pagproductos #cesta input[type="submit"] {
  margin-left: 10px;
  margin-right: 10px;
  margin-bottom: 3px;
  width:100px;
  height:30px;
  padding-left:0px;
}

.pagproductos #productos {
  margin: 0 10em 0 10px;
}

.pagcesta #productos {
  margin: 10px;
  font-size: 12px;
}

.pagproductos #productos input,
.pagproductos #productos p {
  font-size: 10px;
}
```

```
#productos .codigo {
  width: 20%;
  float: left;
}

#productos .nombre {
  width: 60%;
  float: left;
}

#productos .precio {
  width: 20%;
  text-align: right;
  font-weight: bold;
}
```

```
}

#pie {
  padding: 0 10px;
  background-color: #99ccff;
  border-top-width: thin;
  border-right-width: thin;
  border-bottom-width: thin;
  border-left-width: thin;
  border-top-style: solid;
  border-right-style: solid;
  border-bottom-style: solid;
  border-left-style: solid;
}
```



Antes de comenzar ten en cuenta que la aplicación que vas a desarrollar no es completamente funcional. Además de no desarrollar la página con la información de pago, habrá algunas opciones que no tendrás en cuenta para simplificar el código. Por ejemplo:

- ✓ No tendrás en cuenta la posibilidad de que el usuario compre varias unidades de un mismo producto.
- ✓ Una vez añadido un producto a la cesta de compra, no se podrá retirar de la misma. La única posibilidad será vaciar toda la cesta y comenzar de nuevo añadiendo productos.
- ✓ No se mostrarán imágenes de los productos, ni será posible ver el total de la compra hasta que ésta haya finalizado.
- ✓ Se muestran todos los productos en una única página. Sería preferible filtrarlos por familia y mostrarlos en varias páginas, limitando a 10 o 20 productos el número máximo de cada página.

Aunque reduzcamos en este ejemplo la funcionalidad de la tienda, te animamos a que una vez finalizado el mismo, añadas por tu cuenta todas aquellas opciones que quieras. Recuerda que la mejor forma de aprender programación es... ¡programando!

3.3.1.- Gestión de la información de la sesión (II).

La primera página que vas a programar es la de autenticación del usuario (`login.php`). Para variar, utilizarás las capacidades de manejo de sesiones de PHP para almacenar la identificación de los usuarios. Además, utilizaremos la información de la tabla "`usuarios`" en la base de datos "`dwes`", accediendo mediante PDO en lugar de MySQLi.

Vas a crear en la página un formulario con dos campos, uno de tipo text para el usuario, y otro de tipo password para la contraseña. Al pulsar el botón Enviar, el formulario se enviará a esta misma página, donde se compararán las credenciales proporcionadas por el usuario con las almacenadas en la base de datos. Si los datos son correctos, se iniciará una nueva sesión y se almacenará en ella el nombre del usuario que se acaba de conectar.

Vamos por pasos. El código HTML para crear el formulario, que irá dentro del cuerpo de la página (entre las etiquetas `<body>`) será el siguiente:

```
<div id='login'>
<form action='login.php' method='post'>
<fieldset >
  <legend>Login</legend>
```

```

<div><span class='error'><?php echo $error; ?></span></div>
<div class='campo'>
  <label for='usuario' >Usuario:</label><br/>
  <input type='text' name='usuario' id='usuario' maxlength="50" /><br/>
</div>
<div class='campo'>
  <label for='password' >Contraseña:</label><br/>
  <input type='password' name='password' id='password' maxlength="50" /><br/>
</div>
<div class='campo'>
  <input type='submit' name='enviar' value='Enviar' />
</div>
</fieldset>
</form>
</div>

```

Fíjate que existe un espacio para poner los posibles mensajes de error que se produzcan, como la falta de algún dato necesario, o un error de credenciales erróneas.

El código PHP que debe figurar al comienzo de esta misma página (antes de que se muestre cualquier texto), se encargará de:

Comprobar que se han introducido tanto el nombre de usuario como la contraseña.

```

if (isset($_POST['enviar'])) {
    $usuario = $_POST['usuario'];
    $password = $_POST['password'];

    if (empty($usuario) || empty($password))
        $error = "Debes introducir un nombre de usuario y una contraseña";
    else {

```

Conectarse a la base de datos.

```

    try {
        $opc = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
        $dsn = "mysql:host=localhost;dbname=dwes";
        $dwes = new PDO($dsn, "dwes", "abc123.", $opc);
    }
    catch (PDOException $e) {
        die("Error: " . $e->getMessage());
    }

```

Comprobar las credenciales.

```

$sql = "SELECT usuario FROM usuarios WHERE usuario='$usuario' " .
    "AND contrasena='" . md5($password) . "'";

if($resultado = $dwes->query($sql)) {
    $fila = $resultado->fetch();
    if ($fila != null) {

```

Iniciar la sesión y almacenar en la variable de sesión `$_SESSION['usuario']` el nombre de usuario.

```

session_start();
$_SESSION['usuario']=$usuario;

```

Redirigir a la página del listado de productos.

```

header("Location: productos.php");

```

Revisa el código completo de la página login.php y comprueba su funcionamiento antes de seguir con las demás.

```

<?php
// Comprobamos si ya se ha enviado el formulario
if (isset($_POST['enviar'])) {
    $usuario = $_POST['usuario'];
    $password = $_POST['password'];

    if (empty($usuario) || empty($password))
        $error = "Debes introducir un nombre de usuario y una contraseña";
    else {
        // Comprobamos las credenciales con la base de datos
        // Conectamos a la base de datos
        try {

```

```

        $opc = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
        $dsn = "mysql:host=localhost;dbname=dwes";
        $dwes = new PDO($dsn, "dwes", "abc123.", $opc);
    }
    catch (PDOException $e) {
        die("Error: " . $e->getMessage());
    }

    // Ejecutamos la consulta para comprobar las credenciales
    $sql = "SELECT usuario FROM usuarios " .
        "WHERE usuario='$usuario' " .
        "AND contraseña='" . md5($password) . "'";

    if($resultado = $dwes->query($sql)) {
        $fila = $resultado->fetch();
        if ($fila != null) {
            session start();
            $_SESSION['usuario']=$usuario;
            header("Location: productos.php");
        }
        else {
            // Si las credenciales no son válidas, se vuelven a pedir
            $error = "Usuario o contraseña no válidos!";
        }
        unset($resultado);
    }
    unset($dwes);
}
}
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 4 : Desarrollo de aplicaciones web con PHP -->
<!-- Ejemplo Tienda Web: login.php -->
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Ejemplo Tema 4: Login Tienda Web</title>
<link href="tienda.css" rel="stylesheet" type="text/css">
</head>
<body>
<div id='login'>
<form action='login.php' method='post'>
<fieldset >
<legend>Login</legend>
<div><span class='error'><?php echo $error; ?></span></div>
<div class='campo'>
<label for='usuario' >Usuario:</label><br/>
<input type='text' name='usuario' id='usuario' maxlength="50" /><br/>
</div>
<div class='campo'>
<label for='password' >Contraseña:</label><br/>
<input type='password' name='password' id='password' maxlength="50" /><br/>
</div>
<div class='campo'>
<input type='submit' name='enviar' value='Enviar' />
</div>
</fieldset>
</form>
</div>
</body>
</html>

```

En el código anterior, la sesión del usuario se inicia solo si proporciona un nombre de usuario y contraseña correctos. ¿Se podría haber iniciado la sesión al principio del código, aunque el usuario no proporcione credenciales?



Sí



No

Sí, se puede iniciar la sesión de igual forma, aunque si el usuario no se autentifica no tendríamos ninguna información que incluir dentro de la misma.

3.3.2.- Gestión de la información de la sesión (III).

Cuando un usuario proporciona unas credenciales de inicio de sesión correctas (recuerda que tú ya habías añadido el usuario "dwes" con contraseña "abc123."), se le redirige de forma automática a la página del listado de productos (`productos.php`) para que pueda empezar a hacer la compra. Esa es la página que vas a programar a continuación.



```
<?php
// Recuperamos la información de la sesión
session_start();

// Y comprobamos que el usuario se haya autenticado
if (!isset($_SESSION['usuario'])) {
    die("Error - debe <a href='login.php'>identificarse</a>.<br />");
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 4 : Desarrollo de aplicaciones web con PHP -->
<!-- Ejemplo Tienda Web: productos.php -->
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <title>Ejemplo Tema 4: Listado de Productos</title>
    <link href="tienda.css" rel="stylesheet" type="text/css">
</head>

<body class="pagproductos">

<div id="contenedor">
    <div id="encabezado">
        <h1>Listado de productos</h1>
    </div>
    <div id="cesta">
        <h3> Cesta</h3>
        <hr />
    </div>
<?php
// Comprobamos si se ha enviado el formulario de vaciar la cesta
if (isset($_POST['vaciar'])) {
    unset($_SESSION['cesta']);
}

// Comprobamos si se ha enviado el formulario de añadir
if (isset($_POST['enviar'])) {
    // Creamos un array con los datos del nuevo producto
    $producto['nombre'] = $_POST['nombre'];
    $producto['precio'] = $_POST['precio'];
    // y lo añadimos
    $_SESSION['cesta'][$_POST['producto']] = $producto;
}

// Si la cesta está vacía, mostramos un mensaje
$cesta_vacia = true;
if (count($_SESSION['cesta'])==0) {
    print "<p>Cesta vacía</p>";
}

// Si no está vacía, mostrar su contenido
else {
    foreach ($_SESSION['cesta'] as $codigo => $producto)
        print "<p>$codigo</p>";
    $cesta_vacia = false;
}
```

```

}
?>
<form id='vaciar' action='productos.php' method='post'>
  <input type='submit' name='vaciar' value='Vaciar Cesta'
    <?php if ($cesta vacia) print "disabled='true'"; ?>
  />
</form>
<form id='comprar' action='cesta.php' method='post'>
  <input type='submit' name='comprar' value='Comprar'
    <?php if ($cesta vacia) print "disabled='true'"; ?>
  />
</form>
</div>
<div id="productos">
<?php
  try {
    $opc = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
    $dsn = "mysql:host=localhost;dbname=dwes";
    $dwes = new PDO($dsn, "dwes", "abc123.", $opc);
  }
  catch (PDOException $e) {
    $error = $e->getCode();
    $mensaje = $e->getMessage();
  }

  if (!isset($error)) {
    $sql = <<<SQL
SELECT cod, nombre_corto, PVP
FROM producto
SQL;
    $resultado = $dwes->query($sql);
    if($resultado) {
      // Creamos un formulario por cada producto obtenido
      $row = $resultado->fetch();
      while ($row != null) {
        echo "<p><form id='{$row['cod']}' action='productos.php' method='post'>";
        // Metemos ocultos los datos de los productos
        echo "<input type='hidden' name='producto' value='".$row['cod']."' />";
        echo "<input type='hidden' name='nombre' value='".$row['nombre corto']."' />";
        echo "<input type='hidden' name='precio' value='".$row['PVP']."' />";
        echo "<input type='submit' name='enviar' value='Añadir' />";
        echo " {$row['nombre_corto']}: ";
        echo $row['PVP']." euros.";
        echo "</form>";
        echo "</p>";
        $row = $resultado->fetch();
      }
    }
  }
?>

</div>
<br class="divisor" />
<div id="pie">
  <form action='logout.php' method='post'>
    <input type='submit' name='desconectar' value='Desconectar usuario <?php echo
$_SESSION['usuario']; ?>' />
  </form>
<?php
  if (isset($error)) {
    print "<p class='error'>Error $error: $mensaje</p>";
  }
?>
</div>
</div>
</body>
</html>

```

La página se divide en varias zonas, cada una definida por una etiqueta **< div>** en el código HTML:

- ✓ **encabezado.** Contiene únicamente el título de la página.
- ✓ **productos.** Contiene el listado de todos los productos tal y como figuran en la base de datos. Cada producto figura en una línea (nombre y precio).
Se crea un formulario por cada producto, con un botón "Añadir" que envía a esta misma página los datos código, nombre y precio del producto. Cuando se abre la página, se comprueba si se ha

enviado este formulario, y si fuera así se añade un elemento al array asociativo `$_SESSION['cesta']` con los datos del nuevo producto.

```
// Comprobamos si se ha enviado el formulario de añadir
if (isset($_POST['enviar'])) {
    // Creamos un array con los datos del nuevo producto
    $producto['nombre'] = $_POST['nombre'];
    $producto['precio'] = $_POST['precio'];
    // y lo añadimos
    $_SESSION['cesta'][$_POST['producto']] = $producto;
}
```

El array `$_SESSION['cesta']` es la variable de sesión en la que guardaremos los datos de todos los productos que va a comprar el usuario. Fíjate que los datos del nuevo producto se incluyen a su vez como un array con dos elementos, el precio y el nombre.

- ✓ **cesta.** Muestra el código de los productos que se van añadiendo a la cesta.

```
// Si la cesta está vacía, mostramos un mensaje
$cesta_vacia = true;
if (count($_SESSION['cesta'])==0) {
    print "<p>Cesta vacía</p>";
}

// Si no está vacía, mostrar su contenido
else {
    foreach ($_SESSION['cesta'] as $codigo => $producto)
        print "<p>$codigo</p>";
    $cesta_vacia = false;
}
```

Contiene dos formularios. Uno para vaciar la cesta (botón "Vaciar Cesta"), dirigido a esta misma página, y otro para realizar la compra (botón "Comprar"), que dirige a la página `cesta.php`. Para vaciar la cesta, se incluye en la página el siguiente código:

```
// Comprobamos si se ha enviado el formulario de vaciar la cesta
if (isset($_POST['vaciar'])) {
    unset($_SESSION['cesta']);
}
```

- ✓ **pie.** Contiene un botón para desconectar al usuario actual. Llama a la página `logout.php`, que borra la sesión actual.

Además, tanto en esta página como en todas las demás, es necesario comprobar la variable de sesión `$_SESSION['usuario']` para verificar que el usuario se ha autenticado correctamente. Para ello, debes incluir el siguiente código al inicio de la página.

```
<?php
// Recuperamos la información de la sesión
session_start();

// Y comprobamos que el usuario se haya autenticado
if (!isset($_SESSION['usuario'])) {
    die("Error - debe <a href='login.php'>identificarse</a>.<br />");
}
?>
```

Si el usuario no se ha autenticado, se muestra un mensaje de error junto con un enlace a la página `login.php`.

3.3.3.- Gestión de la información de la sesión (IV).

Si desde la página del listado de productos, el usuario pulsa sobre el botón "Comprar", se le dirige a la página de la Cesta de la compra (`cesta.php`), en la que se le muestra un resumen de los productos que ha

Cesta de la compra		
ITEM1101111LD	Samsung 3110 16.1LED 2430 1GB 200GB BATA BT W7R	288.00
ITEM2222222OB	Kegona 3.6módulo 90B	18.20
ITEM3333333AZ	Cerveza Dos EEBB azul	194.70
Precio total: 481.90		
<input type="button" value="Comprar"/>		
<input type="button" value="Desconectar usuario actual"/>		

seleccionado junto al importe total de los mismos.

```
<?php
// Recuperamos la información de la sesión
session_start();

// Y comprobamos que el usuario se haya autenticado
if (!isset($_SESSION['usuario'])) {
    die("Error - debe <a href='login.php'>identificarse</a>.<br />");
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 4 : Desarrollo de aplicaciones web con PHP -->
<!-- Ejemplo Tienda Web: cesta.php -->
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>Ejemplo Tema 4: Cesta de la Compra</title>
<link href="tienda.css" rel="stylesheet" type="text/css">
</head>

<body class="pagcesta">

<div id="contenedor">
<div id="encabezado">
<h1>Cesta de la compra</h1>
</div>
<div id="productos">
<?php
$total = 0;
foreach($_SESSION['cesta'] as $codigo => $producto) {
    echo "<p><span class='codigo'>$codigo</span>";
    echo "<span class='nombre'>${producto['nombre']}</span>";
    echo "<span class='precio'>${producto['precio']}</span></p>";
    $total += $producto['precio'];
}
?>

<hr />
<p><span class='pagar'>Precio total: <?php print $total; ?> €</span></p>
<form action='pagar.php' method='post'>
<p>
<span class='pagar'>
<input type='submit' name='pagar' value='Pagar' />
</span>
</p>
</form>
</div>
<br class="divisor" />
<div id="pie">
<form action='logout.php' method='post'>
<input type='submit' name='desconectar' value='Desconectar usuario <?php echo
$_SESSION['usuario']; ?>' />
</form>
</div>
</div>
</body>
</html>
```

En la página figuran dos formularios que simplemente redirigen a otras páginas. El que contiene el botón "Pagar", que redirige a la página `pagar.php`, que en nuestro caso lo único que debe hacer es eliminar la sesión del usuario. Y el que contiene el botón de desconexión, que es similar al que figuraba en la página `productos.php`, y dirige a la página `logout.php`, que cierra la sesión del usuario.

```
<?php
// Recuperamos la información de la sesión
session_start();
unset($_SESSION['cesta']);

die("Gracias por su compra.<br />Quiere <a href='productos.php'>comenzar de nuevo</a>?");
?>
```


Los datos que figuran en la página se obtienen todos de la información almacenada en la sesión del usuario. No es necesario establecer conexiones con la base de datos. El código que sirve para mostrar el listado de los productos seleccionados es el siguiente:

```
<?php
$total = 0;
foreach($_SESSION['cesta'] as $codigo => $producto) {
    echo "<p><span class='codigo'>$codigo</span>";
    echo "<span class='nombre'>${producto['nombre']}</span>";
    echo "<span class='precio'>${producto['precio']}</span></p>";
    $total += $producto['precio'];
}
?>

<hr />
<p><span class='pagar'>Precio total: <?php print $total; ?> €</span></p>
```

Recuerda que al principio de esta página, también será necesario verificar la variable de sesión `$_SESSION['usuario']` para comprobar que el usuario se ha autenticado correctamente.

Tanto desde esta página como desde la página del listado de productos, se le ofrece al usuario la posibilidad de cerrar la sesión. Para ello se le dirige a la página `logoff.php`, que no muestra nada en pantalla. Su código es el siguiente:

```
<?php
// Recuperamos la información de la sesión
session_start();

// Y la eliminamos
session_unset();
header("Location: login.php");
?>
```

Tras eliminar la sesión, redirige al usuario a la página de autenticación donde puede iniciar una nueva sesión con el mismo o con otro usuario distinto.

En las páginas anteriores, `productos.php` y `cesta.php`, se ha incluido un botón para desconectar al usuario cerrando su sesión. ¿Se podría utilizar un enlace a la página `logoff.php` en lugar de crear un botón dentro de un formulario?



Sí



No

La única función del botón es abrir la página `logoff.php`. No necesitamos enviar nada a ésta página, por lo que se podría haber logrado lo mismo con un simple enlace.

4.- Herramientas para depuración de código.

Caso práctico

Con lo que ha aprendido, **Carlos** se ha aventurado por su cuenta en la programación de una aplicación web sencilla. Ahora ya sabe cómo y dónde almacenar la información que genera cada página, para poderla utilizar más adelante. Ha cogido la página que había creado para catalogar su colección de comics, y ha decidido convertirla en una aplicación de verdad. Empieza a estructurar el código que tenía y decide programar algunas páginas más y crear un menú para moverse entre ellas. Y entonces... comienzan a surgir los problemas: páginas que no muestran lo que se suponía que deberían mostrar, errores inexplicables cuando se intenta conectar a la base de datos, listados que algunas veces funcionan y otras no,...

Tras muchas horas intentando encontrar y solucionar los problemas que van apareciendo, busca en Internet y descubre que existen algunas utilidades que pueden ayudarle a arreglar los errores del código. No se trata de que no vaya a cometer fallos nunca más, sino de que cuando los cometa tenga una forma más cómoda y rápida de resolverlos.

Carlos está descubriendo por su cuenta cómo funciona la depuración de código en lenguaje PHP.

Seguramente en la aplicación de tienda online que acabas de programar te hayas encontrado con algunos problemas. Algún código que no funcionaba, problemas de conexión a la base de datos, o páginas que no mostraban lo que deberían.

Con lo que has visto hasta ahora ya puedes empezar a desarrollar algunas aplicaciones web sencillas en lenguaje PHP. Sin embargo, cuando empiezan a surgir problemas tienes que ingeniártelas para poder descubrir qué es lo que está fallando. Puedes poner trozos de código en las páginas que desarrollas para mostrar información interna de la aplicación, que muestren en pantalla la secuencia en que se ejecutan las líneas de código, o el contenido de las variables que usas. Para ello puedes utilizar `echo`, `print`, `print_r` o `var_dump`.

En la documentación de PHP puedes consultar información sobre la función `var_dump`.
<http://php.net/manual/es/function.var-dump.php>

Por ejemplo, si quieres ver el contenido de la variable superglobal `$_SERVER`, puedes introducir una línea como:

```
<?php print_r($_SERVER); ?>
```

Este método de depuración es muy tedioso y requiere hacer cambios constantes en el código de la aplicación. También existe la posibilidad de que una vez encontrado el fallo, te olvides de eliminar de tus páginas algunas de las líneas creadas a propósito para la depuración, con los consiguientes problemas.

Si quieres conocer más sobre los principios de la depuración, y la utilización de comandos como `echo` o `print_r` para depurar tu código, puedes consultar este artículo de php-hispano.
<http://www.php-hispano.net/articulos/debug-en-php.html>

Si ya has programado anteriormente en otros lenguajes, seguramente conoces algunas herramientas de depuración específicas que se integran con el entorno de desarrollo, permitiéndote por ejemplo detener la ejecución cuando se llega a cierta línea y ver el contenido de las variables en ese momento. Al usar estas herramientas minimizas o eliminas por completo la necesidad de introducir líneas específicas de depuración en tus programas y agilizas el procedimiento de búsqueda de errores.

De las herramientas disponibles que posibilitan la depuración en lenguaje PHP, en esta unidad aprenderás a configurar y utilizar la extensión Xdebug. Es una extensión de código libre, bajo licencia

propia (The Xdebug License, basada en la licencia de PHP) que se integra perfectamente con el IDE que estás usando, NetBeans.

Si la depuración está activada, Xdebug controla la ejecución de los guiones en PHP. Puede pausar, reanudar y detener los programas en cualquier momento. Cuando el programa está pausado, Xdebug puede obtener información del estado de ejecución, incluyendo el valor de las variables (puede incluso cambiar su valor).

Xdebug es un servidor que recibe instrucciones de un cliente, que en nuestro caso será NetBeans. De esta forma, no es necesario que el entorno de desarrollo esté en la misma máquina que el servidor PHP con Xdebug.

4.1.- Instalación de herramientas de depuración.

Concretamente, vas a aprender a preparar un sistema Ubuntu para depurar aplicaciones en lenguaje PHP utilizando Xdebug y el IDE Netbeans. Los pasos que verás a continuación los puedes seguir también utilizando el tutorial multimedia que figura al final de esta unidad.

Aunque existen diversas formas para realizar la instalación de Xdebug. Probablemente la más sencilla es a través del repositorio de extensiones PECL, del que ya hablamos en la unidad 2. Necesitas por tanto asegurarte de que PECL está disponible en tu instalación de PHP. Para instalarlo desde los repositorios de Ubuntu, ejecuta desde una consola:

```
sudo apt-get install php-pear
```

Es conveniente mantener actualizados los repositorios ejecutando `sudo apt-get update` antes de instalar paquetes.

Además, para poder compilar las extensiones que descargues con PECL, debes instalar el paquete con las herramientas de desarrollo `php5-dev`. Incluye dos programas necesarios para instalar Xdebug: `phpize` y `php-config`.

```
sudo apt-get install php5-dev
```

Si todo fue bien, ya tienes el sistema preparado para instalar Xdebug.

```
sudo pecl install xdebug
```

Al finalizar la instalación verás un mensaje como el siguiente:

```
Build process completed successfully
Installing '/usr/lib/php5/20090626-lfs/xdebug.so'
install ok: channel://pecl.php.net/xdebug-2.1.1
configuration option "php_ini" is not set to php.ini location
You should add "extension=xdebug.so" to php.ini
usr@ubuntu-profe:/bin$
```

En esa pantalla figura (tras "`Installing`") la ruta en que se ha instalado la extensión (archivo `xdebug.so`). Además te indica que debes modificar el archivo `php.ini` para activarla. Abre por tanto el archivo `php.ini` de tu instalación de PHP, y añade las siguientes líneas al final.

```
[xdebug]
zend_extension="/usr/lib/php5/20090626-lfs/xdebug.so"
; Opciones de configuración
xdebug.remote_enable=on
xdebug.remote_handler=dbgp
xdebug.remote_host=localhost
xdebug.remote_port=9000
```

Ajusta la ruta anterior al archivo `xdebug.so` para que sea la misma que has obtenido en tu sistema tras instalar la extensión.

Por último, recuerda reiniciar el servidor web siempre que modifiques el archivo `php.ini` para aplicar los cambios.

```
sudo /etc/init.d/apache2 restart
```

Si necesitas instalar Xdebug en sistemas Windows, puedes encontrar información al respecto en Internet. En la página web de la extensión Xdebug tienes información en lenguaje inglés.
<http://xdebug.org/docs/install>

4.2.- Depuración de código en PHP.

Entre las características de depuración que incorpora Xdebug destacan:

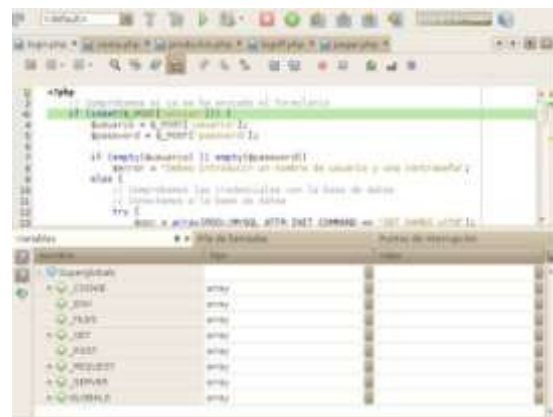
- ✓ Creación de registros con las llamadas a funciones que se produzcan, incluyendo parámetros y valores devueltos.
- ✓ Creación de registros de optimización, que permitan analizar el rendimiento de los programas.
- ✓ Depuración remota.

Vamos a centrarnos en la depuración remota utilizando NetBeans como cliente.

Utilizando los menús, botones, o la combinación de teclas adecuada, puedes iniciar la depuración del proyecto actual (Ctrl+F5, comenzando por el archivo definido en la configuración de ejecución del mismo) o directamente la del archivo que tengas abierto en el editor (Ctrl+May+F5).



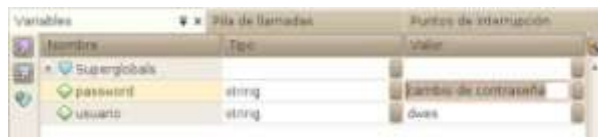
Una vez iniciada la depuración, se abre la aplicación en el navegador y se modifica la apariencia del editor. En las barras de herramientas y de menús, puedes controlar la ejecución con las opciones habituales (continuar, detener, paso a paso, etc.). En la parte inferior se muestran tres zonas nuevas con información sobre las variables que haya definidas en ese instante, la pila de llamadas y los puntos de interrupción.



En cualquier momento puedes añadir o eliminar un punto de interrupción en una línea de código simplemente pulsando con el ratón en la zona izquierda del editor, sobre los números de las líneas.



También puedes modificar el valor de las variables editando directamente sobre la zona en que se muestran.



Mientras estás depurando una aplicación, la ejecución se detendrá antes de ejecutar la primera línea de cada una de las páginas que se vayan abriendo. Éste es el comportamiento por defecto en NetBeans. En las opciones del entorno (Herramientas – Opciones – PHP) puedes modificarlo para que



se detenga solo en los puntos de ruptura que hayas indicado.

Al instalar la extensión Xdebug en PHP, se instala automáticamente el cliente para NetBeans, que permite gestionar la ejecución del código.



No



Sí

No es necesario instalar el cliente de NetBeans para Xdebug, ya viene incluido con la instalación por defecto de NetBeans