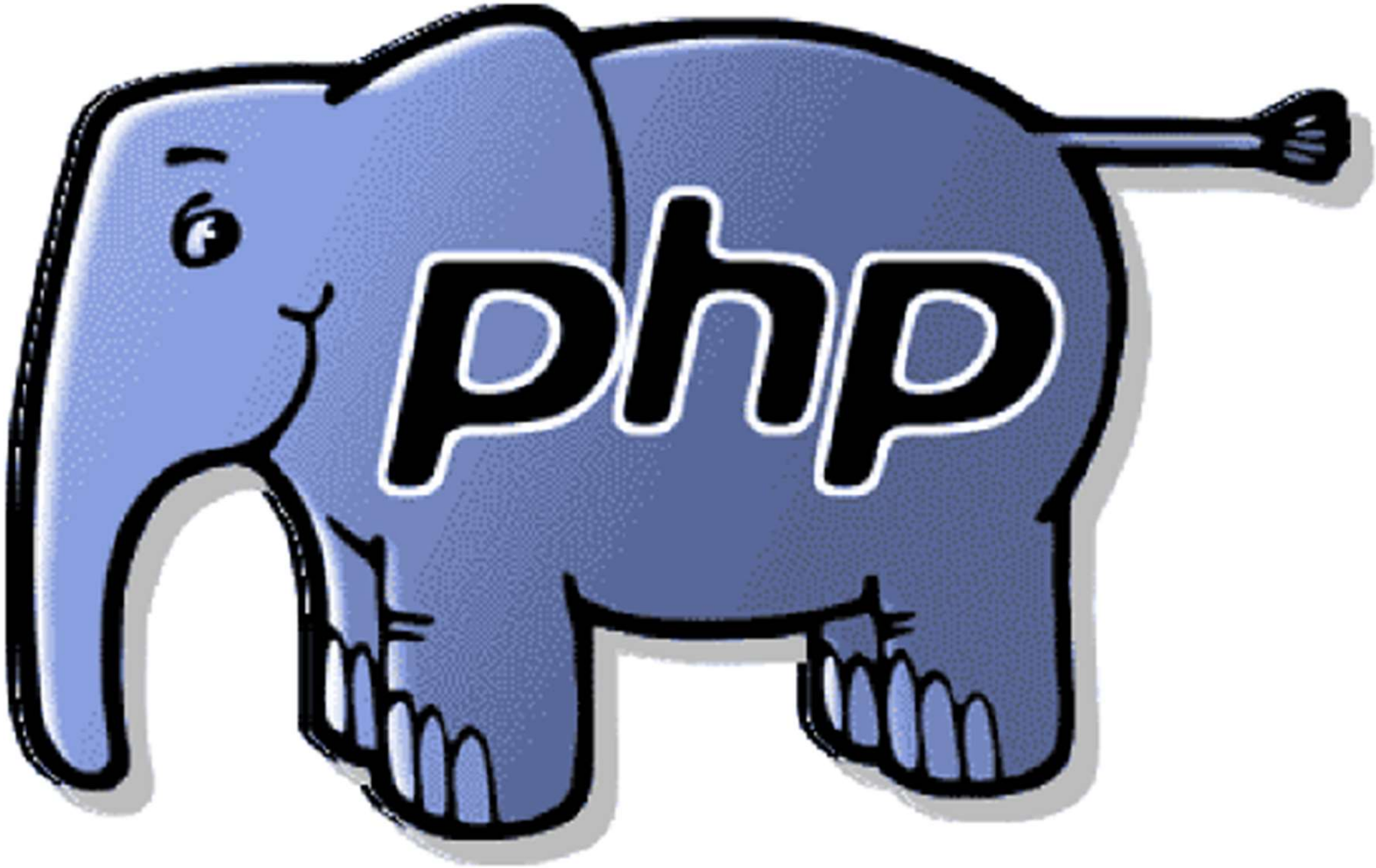
Desarrollo de aplicaciones web utilizando código embebido



Desarrollo Web en Entorno Servidor

Contenido

1. Autentificación de usuarios y control de acceso
2. Cookies
3. Manejo de sesiones
4. Ejemplo con sesiones

1. Autentificación de usuarios y control de acceso

A veces es importante **verificar la identidad de los extremos de una comunicación**. En el caso de una comunicación web, se debe identificar tanto al servidor en el que se aloja la 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. Para ello se utiliza un certificado válido, firmado por una autoridad confiable, y verificado por el navegador cuando se accede al sitio web; y métodos de cifrado para crear un canal seguro entre el navegador y el servidor.

Para **identificar a los usuarios** que visitan un sitio web, se pueden utilizar distintos métodos: DNI digital, certificados digitales de usuario.., pero el más utilizado es solicitar al usuario una información que solo él conoce: un nombre de usuario y una contraseña.

En las **aplicaciones web que acceden a bases de datos** es importante implantar 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.

Distinguir **la autentificación de los usuarios y el control de acceso,** de **la utilización de mecanismos para asegurar las comunicaciones** entre el usuario y el servidor web.

Aquí, la información de autentificación (nombre y contraseña) se enviará en texto plano desde el navegador hasta el servidor web. Esto es **inseguro, debe usarse un protocolo HTTPS** que permita cifrar las comunicaciones, pero ello excede este módulo.

1. Autentificación de usuarios y control de acceso

- **Mecanismos de autentificación**

El protocolo HTTP ofrece un método sencillo para autentificar a los usuarios:

- El servidor web debe proveer algún método para **definir los usuarios** que se utilizarán y cómo se pueden autentificar. Además, se tendrán que definir los recursos a los que se restringe el acceso y la lista de control de acceso aplicada. (*ACL - indica qué usuarios pueden utilizar cada objeto (fichero, directorio, etc.) y las acciones que pueden realizar con el mismo (lectura, escritura, borrado, etc.)*).
- Cuando **un usuario no autentificado 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 autentifique mediante su **nombre y contraseña**.
- La información de autentificación 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 Apache existe la utilidad **htpasswd** (fichero con usuarios y contraseñas). Para indicar a Apache qué recursos tienen acceso restringido se utiliza el fichero **.htaccess** (para aplicar la configuración de este fichero se debe configurar la directiva **AllowOverride**).

1. Autentificación de usuarios y control de acceso

En PHP se puede acceder a la información de autentificación HTTP que ha introducido el usuario utilizando el array superglobal **$_SERVER**.

Valor	Contenido
$_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.

Se puede usar la función **header** para forzar a que el servidor envíe un error de "Acceso no autorizado" (código 401). Así no es necesario utilizar ficheros .htaccess de Apache.

La función **header** envía encabezados HTTP (*bloque de datos que forma parte del protocolo HTTP y se envía antes de la información que se transmite. Permite especificar códigos de estado, acciones requeridas al servidor, tipo de información a transmitir*). Debe utilizarse antes de que se muestre nada por pantalla. Si no, se obtendrá un error.

(Ver ejemplo siguiente)

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 (habría que añadir código). Si se pulsa "Cancelar", se muestra un mensaje de error.

1. Autentificación de usuarios y control de acceso

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

(Ver: ej-aut.php)

- **Incorporación de métodos de autentificación a una aplicación web**

En el código anterior se solicitan credenciales HTTP, pero el servidor no verifica la información. Habrá que proveer un método para comprobar que las credenciales son correctas. El **método más simple** es comparar en el código PHP los datos introducidos con los datos requeridos:

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

1. Autentificación de usuarios y control de acceso

La solución anterior es poco adecuada. Una solución mejor es utilizar un **almacenamiento externo** para los nombres de usuario y sus contraseñas, una **base de datos**. La información de autentificación podrá estar aislada en su propia base de datos, o compartir espacio de almacenamiento con los datos que utilice la aplicación web.

(Utilizar "usuario.sql" para añadir la tabla usuarios a la base de datos "dwes")

NOTA: Aunque se podrían almacenar las contraseñas en texto plano, es mejor hacerlo en formato encriptado. En el script anterior, para el usuario "dwes" se almacena el hash MD5 correspondiente a la contraseña "abc123.".

En PHP se usa la función **md5** para calcular el hash MD5 de una cadena de texto.

Ejercicio: Utilizar PDO para modificar el ejemplo 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, volver a pedir las credenciales al usuario.

Se debe usar la función md5 para comprobar la contraseña. Si se introduce un usuario o contraseña incorrectos, el comportamiento depende del navegador utilizado; algunos pedirán las credenciales de forma indefinida, y otros un número limitado de veces.

(Ver: autentificación.php)

2. Cookies

- El protocolo HTTP es un protocolo sin estado. Cada vez que solicitamos una página a un servidor representa una conexión distinta.
- En una aplicación web casi siempre es necesario mantener el estado de la sesión, es decir, mantener algo que nos permita vincular una petición con otra.
- Las **cookies** son simples ficheros de texto mediante los cuales un sitio web almacena información en el ordenador del usuario (p.ej.: preferencias del usuario, como el idioma).
- Solamente el sitio que ha creado la cookie es capaz de volver a leerla.
- Si se envía solamente el nombre, la cookie será eliminada en el cliente.
- Su disponibilidad está controlada por el cliente. El **usuario** puede **deshabilitar** las cookies en el navegador.
- El manejo de las cookies en PHP es muy sencillo:
  - En el 1er paso se **envía** la cookie:

    llamar a la función **setcookie()** para crear la cookie en el cliente .
  - En las posteriores peticiones que recibamos de ese cliente vendrá incrustada la cookie => se **consulta** la información almacenada en ella.

    Se accede mediante la variable superglobal **$_COOKIE**.

2. Cookies

- Una cookie es un bloque de texto constituido por varios campos.
- Todos son opcionales excepto el nombre de la cookie.
  - **Nombre:** Nombre de la cookie.
  - **Valor:** Valor asociado a la cookie (se envía empleando la codificación URL).
  - **Fecha expiración:** Fecha de expiración de la cookie.
  - **Path:** Subconjunto de URLs para los que la cookie es válida.
  - **Dominio:** Rango de dominios para los que la cookie es válida en el servidor.
  - **Segura:** Indica si la cookie se debe transmitir exclusivamente sobre https.
- Tamaño máximo: 4Kb.

- **Crear cookie**

  setcookie($nombre,$valor,$fecha_expiración,$path_valido,$dominio,$segura)

Ejemplos:

$fecha_expiracion = time()+60*60*24*365; → dentro de 1 año

$fecha_expiracion = mktime(0,0,01,1,2015); → 1 Enero 2015

$path_valido = $_SERVER['REQUEST_URI']; → Sólo el path actual

$path_valido = "/"; → todo el sitio

$nombre = preferencias[idioma] -> se reciben los datos en un array

2. Cookies

P.ej.: almacenar en una cookie el nombre de usuario que se transmitió en las credenciales HTTP (no es aconsejable almacenar esta información en las cookies):

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

- **Borrar cookie**
  - Llamar a la función setcookie con una fecha anterior a la actual:

    setcookie("mail","",time()-1000,"/");
  - Enviar una cookie sólo con nombre:

    setcookie("mail");

- **Cookie de sesión**
  - Sólo existe mientras no cerremos la ventana del navegador.
  - Poner como **fecha de expiración de la cookie,** el valor **cero:**

    setcookie(,,0);

Ejemplo: almacenar en una cookie el último instante en que el usuario visitó la página. En la primera visita mostrar un mensaje de bienvenida. En caso contrario, mostrar la fecha y hora de la visita anterior. Utilizar la función setcookie para guardar el instante de la anterior visita y mostrar su contenido utilizando el array $_COOKIE. (Ver ejemplo: cookies.php)

3. Manejo de sesiones

Una forma de guardar información de cada usuario es utilizar **cookies** (la información se almacena en el cliente). Pero existen diversos problemas asociados a las cookies: el número que admite el navegador, su tamaño máximo, o que estén desactivadas.

Para salvar estos inconvenientes, existen las **sesiones**. La sesión hace referencia al conjunto de información relativa a un usuario concreto. Esta información puede ser: el nombre del usuario, preferencias del usuario, o los artículos de una cesta de compra online. En las sesiones la información se almacena en el servidor mediante las **variables de sesión**.

Cada usuario de una web tiene su propia información de sesión. Para distinguir una sesión de otra se usan los **identificadores de sesión** (**SID**). El SID se asigna a cada visitante de un sitio web y lo relaciona con toda la información que posee sobre él. Esa información se almacena en el servidor web, generalmente en ficheros.

Pero ¿dónde se almacena ese SID?. Existen dos maneras de mantener el SID:

- Utilizando cookies.
- Propagando el SID a través de la URL. Se añade como un parámetro:

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

  El SID es el valor del parámetro **PHPSESSID**.

3. Manejo de sesiones

Ninguna de las dos maneras es perfecta. Pese a ello, la utilización de cookies es el mejor método y el más utilizado.

En PHP el proceso de manejo de sesiones está automatizado. Cuando un usuario visita un sitio web, no es necesario verificar el SID. Tampoco hay que preocuparse de almacenar los SID en cookies, o de ir pasando el SID entre las páginas web del sitio. 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). 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 problema de seguridad. El SID puede ser conseguido por otra persona, y obtenida 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.

- **Configuración**

Por defecto, PHP incluye soporte de sesiones incorporado. Sin embargo, se deben conocer las siguientes directivas para la configuración del fichero **php.ini** (la función phpinfo() ofrece información sobre la configuración de las directivas de sesión):

3. Manejo de sesiones

Directiva	Significado
session.use_cookies	Indica si se deben usar cookies (1) o propagación en la URL (0) para almacenar el SID.
session.use_only_cookies	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).
session.save_handler	Se utiliza para indicar a PHP cómo debe almacenar los datos de la sesión del usuario. Existen cuatro opciones: en ficheros (`files`), en memoria (`mm`), en una base de datos SQLite (`sqlite`) o utilizando para ello funciones que debe definir el programador (`user`). El valor por defecto (`files`) funcionará sin problemas en la mayoría de los casos.
session.name	Determina el nombre de la cookie que se utilizará para guardar el SID. Su valor por defecto es `PHPSESSID`.
session.auto_start	Su valor por defecto es 0, y en este caso deberás usar la función `session_start` 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.
session.cookie_lifetime	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.
session.gc_maxlifetime	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.

3. Manejo de sesiones

- **Inicio y fin de una sesión**

El inicio de una sesión puede tener lugar de dos formas.

- Si está activada la directiva **session.auto_start** en la configuración de PHP, la sesión comenzará automáticamente cuando un usuario se conecte al sitio web. En caso de que ese usuario 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 (si se usa propagación del SID, no se podrán restaurar sesiones anteriores; el SID figurará en la URL y se pierde al cerrar el navegador).
- Si no se utiliza el inicio automático de sesiones, habrá que ejecutar la función **session_start** para indicar a PHP que inicie una nueva sesión o reanude la anterior. Esta función devuelve **false** en caso de no poder iniciar o restaurar la sesión, y **true** en caso contrario.

Como el inicio de sesión requiere utilizar cookies, y éstas se transmiten en los encabezados HTTP, para poder iniciar una sesión utilizando **session_start**, hay que llamar 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**.

3. Manejo de sesiones

Mientras la sesión permanece abierta, se puede utilizar la variable superglobal **$_SESSION** para añadir información a la sesión del usuario, o para acceder a la información almacenada en la sesión.

En PHP existen 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.

Ejemplo: Crear una página similar a la anterior, almacenando en la sesión de usuario los instantes de todas sus últimas visitas. Si es su primera visita, se muestra un mensaje de bienvenida. En caso contrario, se muestra la fecha y hora de todas sus visitas anteriores. Añadir un botón a la página que permita borrar el registro de visitas. Utilizar una variable de sesión para comprobar si el usuario se ha autentificado correctamente. De esta forma no hará falta comprobar las credenciales con la base de datos constantemente. Hacer la llamada a la función session_start() antes de utilizar la variable superglobal $_SESSION para acceder a la información de la sesión.

(Ver: sesiones.php)

3. Manejo de sesiones

Funciones de PHP para el manejo de sesiones - Resumen:

- **session_start ()**: Inicializa una sesión y le asigna un identificador de sesión único. Si no existe la crea y si ya existe la recupera (carga todas las variables de sesión).
- **session_destroy ()**: Cierra una sesión. No destruye ninguna de las variables globales asociadas con la sesión, ni destruye la cookie de sesión. Para destruir la sesión completamente, como desconectar al usuario, el id de sesión también debe ser destruido, es decir, se debe borrar la cookie de sesión.
- **session_name():** Devuelve el nombre de la sesión.
- **session_id()**: Devuelve y/o establece el identificador de la sesión.
- Registra o modifica una variable de sesión:

  $_SESSION['nombre'] = valor;
- Elimina una variable de sesión:

  unset ($_SESSION['nombre']);
- Comprueba si una variable está registrada:

  if (isset($_SESSION['nombre']))

4. Ejemplo con sesiones

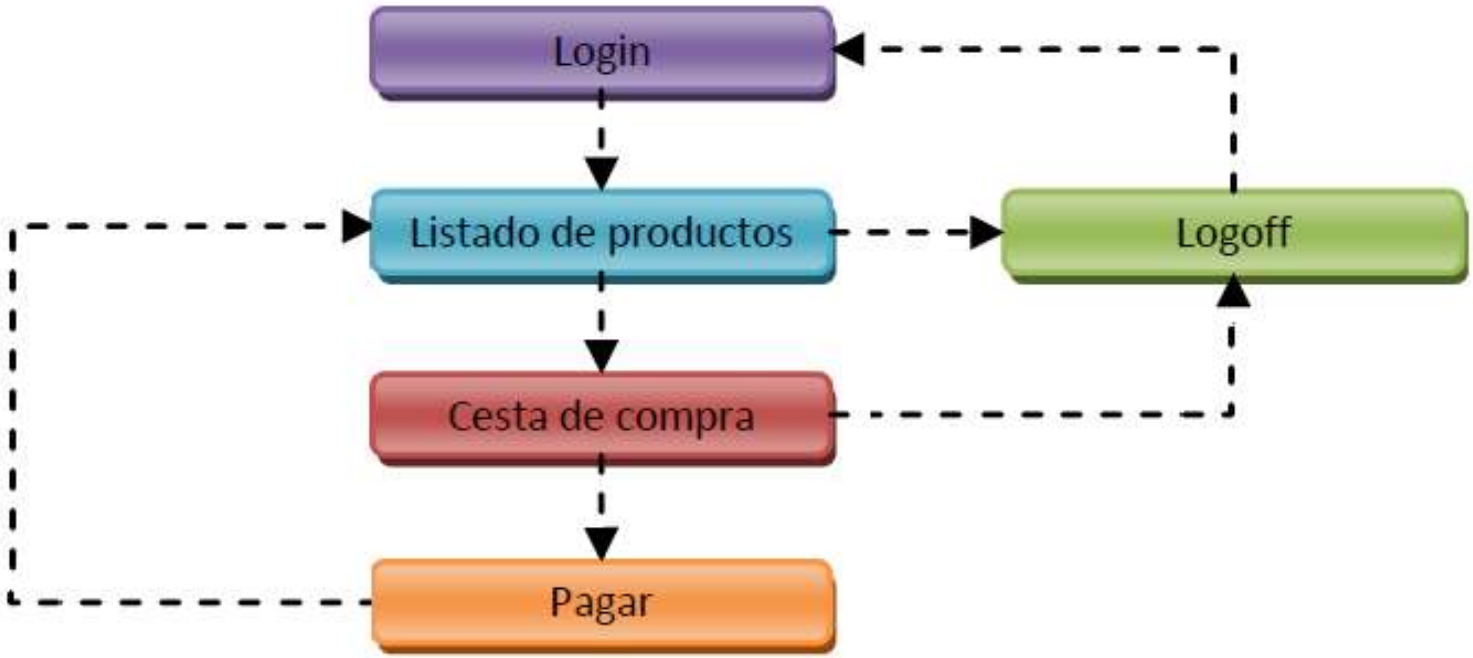
En este punto se verá paso a paso un ejemplo de utilización de sesiones para almacenar la información del usuario. Utilizará la base de datos "dwes", para crear un prototipo de una tienda web dedicada a la venta de productos de informática.

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

- **Login (login.php)**. Autentifica al usuario de la aplicación web. Los usuarios de la aplicación deberán autentificarse en 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, esta página debería ser la que permitiera al usuario escoger el método de pago y la forma de envío. En este ejemplo no se va a implementar como tal. Se mostrará un mensaje "Gracias por su compra" y se ofrecerá un enlace para comenzar de nuevo.
- **Logoff (logoff.php)**. Desconecta al usuario de la aplicación y redirige de forma automática a la pantalla de autentificación. No muestra ninguna información en pantalla.

4. Ejemplo con sesiones

El flujo del programa será el siguiente:



Además, se utilizará:

- una hoja de estilos, **tienda.css**, común a todas las páginas
- y una imagen, **cesta.png**,

para ofrecer una interface más amigable.

4. Ejemplo con sesiones

Conviene tener en cuenta que la aplicación que se propone **no es completamente funcional**. Habrá opciones que no se implementarán para simplificar el código.

- ✓ No se desarrollará la página con la información de pago.
- ✓ No se tendrá 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 el número máximo de productos por cada página.

- ➢ Ver ejemplo completo en: **cesta compra**

4. Ejemplo con sesiones

- **login.php**

Utilizará el manejo de sesiones para almacenar la identificación de los usuarios.

Se tomará la información de la base de datos "dwes", accediendo con PDO.

En la página se crea un formulario con dos campos, uno de tipo text (usuario), y otro de tipo password (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.

El formulario se creará mediante código HTML que irá dentro del cuerpo de la página (entre las etiquetas **<body>**). En el, existe un espacio para poner los mensajes de error que se produzcan, como la falta de algún dato necesario, o un mensaje de credenciales erróneas.

El código PHP debe figurar al comienzo de esta página, y se encargará de:

- Comprobar que se han introducido tanto el nombre de usuario como la contraseña.
- Conectarse a la base de datos.
- Comprobar las credenciales.
- Iniciar la sesión y almacenar en la variable de sesión **$_SESSION['usuario']** el nombre de usuario.
- Redirigir a la página del listado de productos.

4. Ejemplo con sesiones

- **productos.php**

La página se divide en varias zonas, definidas por etiquetas **< div>** en HTML:

- **encabezado**. Contiene únicamente el título de la página.
- **productos**. Contiene el listado de todos los productos de 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 **$_SESSION['cesta']** con los datos del nuevo producto. **$_SESSION['cesta']** es la variable de sesión donde se guardan los datos de los productos que va a comprar el usuario. Los datos del nuevo producto se incluyen a su vez como un array con dos elementos, el nombre y el precio.
- **cesta**. Muestra el código de los productos que se van añadiendo a la cesta. 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**.
- **pie**. Contiene un botón para desconectar al usuario. Va a la página **logoff.php**, que borra la sesión actual.

Tanto en esta página como en las demás, es necesario comprobar la variable de sesión **$_SESSION['usuario']** para verificar que el usuario se ha autentificado correctamente.

4. Ejemplo con sesiones

- **cesta.php**

Muestra un resumen de los productos que se han seleccionado junto al importe total de los mismos.

En la página figuran dos formularios que redirigen a otras páginas. El que contiene el botón "Pagar", redirige a la página **pagar.php**, y el que contiene el botón de desconexión, similar al que figuraba en **productos.php**, dirige a la página **logoff.php**.

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.

Al principio de esta página también será necesario verificar la variable de sesión $_SESSION['usuario'] para comprobar que el usuario se ha autentificado correctamente.

- **pagar.php**

Simula el pago. En este caso, lo único que hace es eliminar la variable de sesión **$_SESSION['cesta']** para permitir iniciar una nueva compra al usuario.

- **logoff.php**

Tanto desde **cesta.php** como desde **productos.php**, se 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, solo cierra la sesión del usuario, y redirige a la página de autentificación donde se puede iniciar una nueva sesión con el mismo o con otro usuario distinto.