TEMA 3: DESARROLLO DE APLICACIONES WEB Y TECNICAS DE ACCESO A DATOS

Contenido

1.- CONTROL DE SESIONES

El protocolo que se utiliza para navegar por la web es http. Se dice que http es un protocolo sin estado (o sin memoria). La razón es que no almacena información alguna sobre la petición o la respuesta, lo que provoca que cada petición sea independiente de la anterior, no se tiene en cuenta lo realizado en la petición anterior.

Al principio, los desarrollos web no estaban preocupados por esta circunstancia ya que, esencialmente, servían páginas estáticas que contenía información simple. Sin embargo en la actualidad necesitamos otorgar memoria a nuestros desarrollos para adaptar nuestras páginas web al usuario concreto o a las acciones que este realiza. Eso permite una experiencia más rica al usuario o usuaria.

Una **sesión** es la navegación que hace un determinado usuario o usuario por un sitio web. La sesión se refiere a todas las peticiones que realiza el usuario referidas al mismo sitio web. Una sesión se inicia cuando el usuario acude a una página del sitio web y finaliza cuando el usuario abandona el sitio.

Durante toda la sesión el usuario puede haber establecido preferencias y elegido opciones que nos interese recordar. Eso es lo que se denomina el ***estado de una sesión***. Deberemos conseguir que esa información no se pierda para utilizarla tanto en beneficio del usuario como en el nuestro propio como propietarios del sitio.

1.1.- ALMACENAMIENTO DEL ESTADO DE UNA SESION.

Existen varias técnicas para almacenar el estado:

- **Uso de la dirección IP**. El array **$_SERVER** proporciona este dato del usuario entre otros (se puede usar ***$_SERVER["REMOTE_HOST"]*** o ***$_SERVER["REMOTE_ADDR"]***. Eso en principio permite identificarle; pero no es un método seguro ya que el usuario/a puede acceder desde servidores proxy o utilizando otras tecnologías que no nos permitan identificar con seguridad su dirección IP.
- **Uso de variables ocultas de formulario**. Cada vez que el usuario realice acciones que supongan datos a almacenar (los productos que va comprando, el usuario y contraseña con el que se identifica,…) los podemos pasar mediante campos de formulario oculto o bien añadiéndoles a la cadena URL de cada enlace por el sitio. Es posible usarlo siempre que tengamos la capacidad de identificar de forma única la sesión y de ser capaces de pasar los datos de enlace en enlace. El problema es que el paso por GET (el más cómodo de mantener mediante esta técnica) muestra los parámetros y eso es tremendamente inseguro.

  Además, por mucha pericia que tengamos con esta técnica, es muy pesada de utilizar y lastra el desarrollo del conjunto del sitio web.
- **Almacenar datos de sesión en bases de datos o ficheros en el servidor**. Permite incluso que el usuario abandone el sitio web y puede, sin embargo, mantener la sesión que abandonó. Lo malo es que si tenemos una enorme cantidad de usuarios en nuestro sitio web,

se convierte en un imposible mantener el sitio con esta técnica.

- **Uso de *cookies***. La idea es la misma que en la anterior, sólo que la información no se guarda en el servidor sino en un archivo en el cliente. Ese archivo es lo que se conoce como una cookie. El usuario no será ni siquiera consciente el almacenamiento de la cookie, pero en ella almacenaremos los datos de la sesión y podremos mantener esa información sobre el usuario.

  Lo malo es que la usuario o el usuario pueden bloquear las cookies o borrarlas y eso está fuera de nuestro control por lo que es una técnica que tiene también sus riesgos.

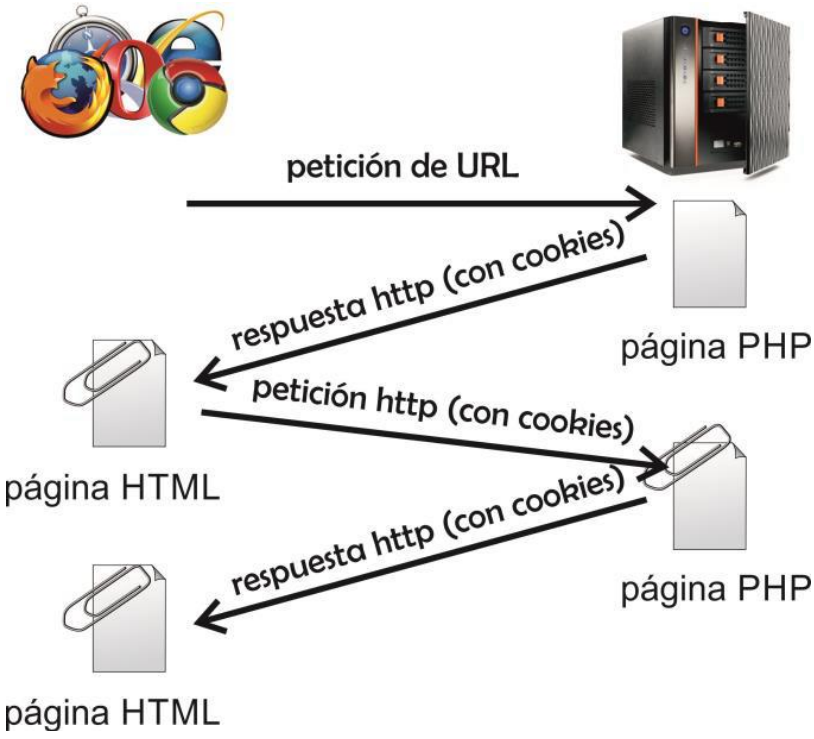
1.2.- COOKIES

Las cookies son indudablemente la opción más habitual para almacenar la información del usuario. Lo malo es lo ya comentado, el cliente de un sitio web puede prohibir o borrar las cookies almacenadas.

Las cookies son archivos de texto que almacenan información sobre el usuario y que se guardan en el propio ordenador del usuario. Contienen esencialmente nombres de variables y valores (de tipo string) de las mismas. Esas variables son las que se usan para guardar la información sobre el usuario.

Las cookies se pasan en la cabecera de la petición http por lo que su información viaja en el mismo paquete. En el momento que el servidor decide grabar los datos de la cookie, ésta viaja con el paquete en todas las peticiones y respuestas de esa sesión.

Los datos de la cookie viajan en la cabecera, por lo que el servidor debe almacenar dichos datos antes de hacer uso de cualquier instrucción en pantalla.



La función **setcookie** es la encargada de añadir (o borrar) una nueva cookie. Sintaxis:

```
setcookie(nombre [,valor [,expiración [,ruta [,dominio [, segura [sóloHttp ]]]]] )
```

Los parámetros son:

- ***nombre***. Es el nombre que le damos a la cookie. Después podremos consultarla para poder saber su valor.
- ***valor***. Valor que le damos a la cookie. Si no indicamos valor alguno, entonces estaremos borrando la cookie.
- ***expiración***. Por defecto toma el valor cero, que significa que la cookie caduca en cuanto se cierre el navegador con el que se creó. Otra posibilidad es indicar una fecha en formato Unix. Por ejemplo ***time()+600*** indicaría que la cookie expira dentro de 10 minutos
- ***dominio***. Si no se indica nada toma como dominio el dominio actual, pero se puede indicar otro.
- ***segura***. Por defecto vale **false**, si indicamos true; entonces indica que la cookie sólo se almacena si estamos comunicándonos mediante un protocolo seguro.
- ***soloHttp***. Disponible desde la versión 5.2 de PHP. Indica que la cookie sólo está disponible mediante el protocolo **http** y no, por ejemplo, desde **JavaScript**. Se supone que evita inseguridades al utilizar cookies; pero es un tanto discutible.

```
setcookie("visitas",1);
//graba la cookie visitas con valor 1 y que caduca con esta sesión
setcookie("usuario","andrei",time()+60*60*24)
//graba la cookie con nombre usuario que caducará al día siguiente
```

Para acceder a las cookies se usa el array superglobal **$_COOKIE.** Contiene las cookies almacenadas en el equipo del cliente, de modo que para acceder al valor de una cookie se indicará el nombre de la cookie como índice de este array. Por ejemplo ***$_COOKIE["visitas"]*** devolvería el valor de la cookie con nombre ***visitas*** (suponiendo que esté definida dicha cookie). Ejemplo de lectura de una cookie:

```
if(isset($_COOKIE["visitas"])){
      echo "esta es tu visita número".$_COOKIE["visitas"];
}
```

Cuando se vuelve a utilizar la función setcookie indicando el nombre de una cookie ya existente y un nuevo valor, el nuevo valor sobrescribe el anterior valor que tuviera dicha cookie.

Bajo la misma idea si usamos setcookie indicando un nombre de cookie existente y el valor cero o **false**, entonces se eliminará la cookie con ese nombre. Ejemplo:

```
setcookie("visitas",false); //elimina la cookie de nombre visitas
```

Podemos también eliminar cookies más antiguas indicando una fecha de caducidad anterior a la actual, por ejemplo:

```
setcookie("visitas",false,time()-60*60*24);
//elimina la cookie de nombre visitas
```

En la cookies no se pueden almacenar datos binarios, sólo strings. Es decir no es válido el código:

```
setcookie("notas",array(9,7,6,5)); //inválido: datos binarios
```

Por ello todos los programadores para almacenar datos binarios (como los arrays), crean mecanismos para convertir lo binario en texto. En general al proceso de convertir los datos en un formato almacenable se le conoce como serializar (traducción excesivamente literal del inglés serialize).

De hecho PHP incorpora una función que realiza automáticamente este proceso, que se llama precisamente serialize. A esta función se le pasa un dato binario y lo convierte en texto. De ese modo el array puede ser almacenado mediante la instrucción:

```
setcookie("notas",serialize(array(9,7,6,5))); //válido: datos binarios
```

El problema ahora es que cuando lo queramos leer, tenemos el problema contrario, convertir el texto en el binario correspondiente. Ese proceso es incluso más difícil; pero PHP lo facilita mediante la función contraria: unserialize. Para extraer el array del ejemplo usaríamos:

```
$variableArray=unserialize($_COOKIE["notas"]))
```

1.3.- SESIONES

PHP aporta un mecanismo de gestión de sesiones propio basado en el almacenamiento de información en el lado del servidor.

Comparadas con las cookies ofrecen estas ventajas

- Pueden funcionar aunque el usuario/a desactive la posibilidad de cookies (podrían pasar el identificador de sesión mediante GET).
- Almacenan más información que una cookie y además son capaces de almacenar datos binarios.
- Son más seguras puesto que los datos que se almacenan de la sesión lo hacen en el servidor y no en el cliente. En principio la seguridad de los servidores es mayor que la de los ordenadores de los usuarios

Y las desventajas:

- Son susceptibles al ataque conocido como **secuestro de sesión** (***session hijacking***), mediante el cual un usuario se puede hacer con el identificador de sesión de otro usuario y hacerse pasar por él.
- El identificador de sesión se almacena mediante cookies, por lo que si no nos damos cuenta podríamos tener la misma dependencia con la configuración que tenga el usuario en el navegador.

El manejo de sesiones parte de asignar un identificador a cada sesión. PHP dispone de la posibilidad de calcular dicho número de sesión para que sea único en cada sesión de usuario. Tras lo cual podemos identificar de forma unívoca a cada sesión. Ese identificador es el que se relacionará con los datos del usuario.

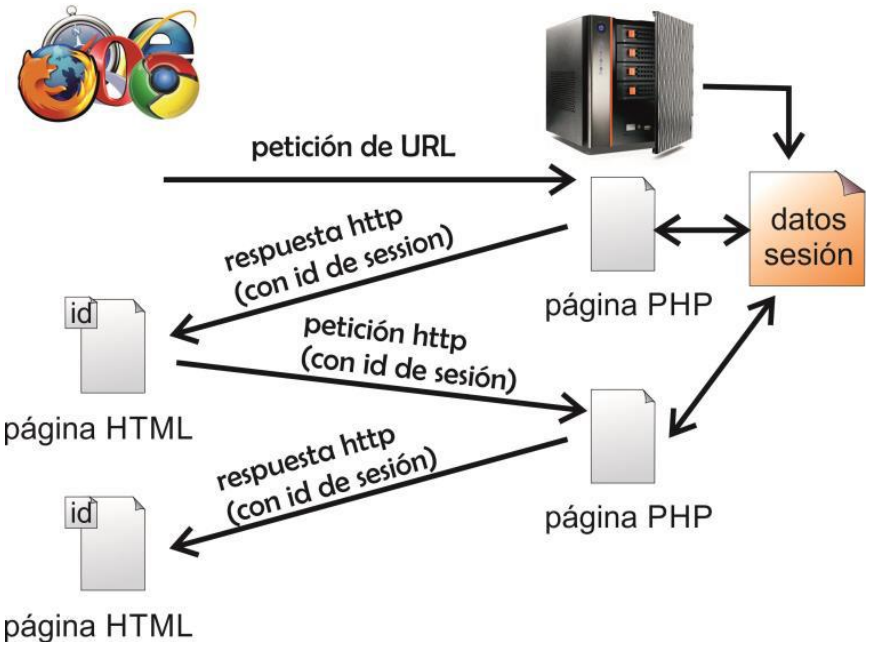
Los pasos son los siguientes:

(1) Se genera un id aleatorio. Para ello se invoca a la función **session_start()**.

(2) Se almacena cada dato de la sesión. Se asigna un nombre y un valor a cada dato de la sesión que deseemos almacenar. Para ello disponemos del array **$_SESSION**, cuyos índices son los nombres de las variables de sesión a las cuales asignamos valores. Los índices son strings, pero (a diferencia de lo que ocurre con las cookies), los valores pueden ser binarios (incluso podemos almacenar arrays directamente)

(3) En una página en la que deseemos recoger los datos almacenados de las sesiones, bastará con llamar a **session_start()** para poder recoger los valores de la sesión.

(4) Mediante el array $_SESSION podremos leer o escribir nuevos datos en la sesión



En realidad las sesiones son más sencillas de manejar que las cookies. El punto fundamental es el mecanismo para "pasar" el identificador de sesión. Disponemos de dos posibilidades:

- **Mediante una cookie**. Se crea una cookie cuyo nombre es el nombre de la sesión y cuyo valor es el identificador de la sesión. En PHP es la opción por defecto. Lo malo es que el usuario puede prohibir utilizar cookies; lo bueno es que es más difícil obtener el identificador de sesión.
  Para que esta opción funcione hay que modificar el archivo de configuración php.ini y colocar estos valores:

```
session.use_cookies 1
session.use_trans_id 0
```

- **Mediate GET**. En este caso se pasa el identificador de la sesión acompañando a cada petición mediante el uso de una cadena de consulta GET. Es decir que si nuestra sesión tiene el nombre de ***PHPID*** y el identificador fuera el ***8FR45237A89*** y estamos con la sesión ya iniciada y en la URL ***http://prueba.com/prog1.php*** la URL que tendremos será:

```
http://prueba.com/prog1.php?PHPID=8FR45237A89
```

  Para que esta opción funcione, deberemos modificar el archivo php.ini y hacer estos cambios:

```
session.use_cookies 0
session.use_trans_id 1
```

Normalmente la sesión caduca en cuanto el usuario abandona el navegador. Se puede hacer (sobre todo si el identificador es una cookie) que dure más, pero no es lo recomendable. Las sesiones deben caducar cuando el usuario cierra la sesión, de otro modo hay más posibilidad de inseguridad. Si necesitamos esa perdurabilidad de la sesión, lo lógico es usar cookies o bases de datos.

Una sesión comienza en la primera página PHP que invoque a la función **session_start**. Esta función no tiene argumentos, simplemente inicia la sesión (si no ha sido iniciada ya) lo cual implica:

(1) Crear un identificador aleatorio de sesión
(2) Crear el archivo en el que se almacenarán los datos de la sesión en el servidor
(3) Crear la variable global **$_SESSION**, array en el que se almacenarán los datos de la sesión.

Si la sesión ya había sido iniciada cuando se invocó a **sesión_start**, entonces no se crea una nueva sesión sino que:

(1) Se sigue utilizando la sesión anterior
(2) Permite el uso del array $_SESSION que contendrá los datos de la sesión

Con lo cual todas las páginas que quieran utilizar la sesión iniciada tienen que invocar a

session_start.

Para obtener, y cambiar si se quiere, los datos de la sesión se usa:

- **session_id**. Si la invocamos sin parámetros, devuelve el identificador de sesión actual. Admite que le pasemos un identificador de sesión con lo que ese se convierte en el identificador de la sesión actual. Poner nosotros números de sesión propios es peligroso en cuanto a que si no tenemos un buen algoritmo de cálculo del identificador podríamos tener dos sesiones
- **session_name**. Si la invocamos sin parámetros, devuelve el nombre de sesión actual (por defecto PHPID). Admite que le pasemos un nombre de sesión. Esto es útil para que no sea tan claro que estamos usando sesiones PHP ya que el nombre PHPID es conocido por todos los programadores PHP.

La directiva **session.name** en el archivo de configuración **php.ini** permite asignar otro nombre por defecto para las sesiones.

usar variables de sesión

```php
//para cambiar el nombre de la sesión
session_name("sesion");

//iniciar/cargar la sesión
session_start();

//recoger el nombre de la sesión y el id de la sesión.
echo session_name()." = ".session_id();
    //creo una variable de sesión llamada validado
$_SESSION["validado"]=true;

//comprobar si existe una variable de sesión
if (isset($_SESSION["validado"]))
    echo "el valor de la variable de sesion ".$_SESSION["validado"];

//borrar la variable de sessión
unset($_SESSION["validado"]);

//destruir la sesión.
//se eliminan todas las variables de sesión y se borra el id de sesión
session_destroy();
```

2.- SEGURIDAD Y AUTENTICACION.

2.1.- USUARIOS, PERFILES Y ROLES

Un aspecto complementario al manejo del estado en páginas web es el control de usuarios. Por lo general las aplicaciones web están diseñadas para que usuarios con distintos niveles de acceso puedan realizar diversas operaciones. Para ello es necesario orgarnizar dichos niveles de acceso.

Uno de los mecanismos mas usuales es a través de perfiles o roles. Así, un ***usuario*** cualquiera que entre y utilice nuestra aplicación tendrá asignado un ***role*** que indica cuales son las ***tareas o procesos*** que puede ejecutar.

Una vez visto el uso de sesiones, podemos pensar que una de las formas de controlar el acceso de los usuarios es a través de una variable de sesión que nos muestre el rol del usuario en curso. Este sería un método manual el cual sería fácilmente controlable si hablamos de una aplicación pequeña. Para aplicaciones mayores existen las ***listas de control de acceso*** (ACL).

Una lista de control de acceso es un concepto de seguridad informática usado para fomentar la separación de privilegios. Es una forma de determinar por medio de grupos y usuarios los permisos de acceso. Los permisos pueden determinar quien tiene privilegios de administrador, para leer un documento, escribir en una base de datos, imprimir, etc.

Normalmente las ACL se crean a través de unos recursos persistentes, bien almacenados en ficheros o en base de datos, con una estructura similar a:

- Usuarios: Tendrá la información de usuarios. Suele incluir datos de identificación (nick, contraseña) y datos descriptivos (nombre, dirección, nif, …)
- Roles: Identifica los diferentes roles disponibles en el sistema. Por ejemplo, administradores, operarios, usuarios, administrativos, …
- Permisos: Acciones que se quieren controlar como “puede administrar”, “puede crear usuarios”, “puede consultar facturas”, ….

Los elementos anteriores son los básicos. Según como se relacionen tendremos que usar nuevos elementos según las consideraciones:

- Si un usuario puede tener un solo role o varios posibles roles. Si tiene un solo role basta con indicar en el usuario el role asignado. Si un usuario puede tener varios roles debe usarse un elemento intermedio USUARIO_ROLE.
- Si los permisos no cambian, se puede optar por incluir los permisos en los roles. Si los permisos pueden variar y aumentar según necesidades, tendremos un elemento intermedio ROLE_PERMISOS.

Una vez creado las ACL, se tendrán que consultar dentro de cada acción y comprobar si se permite o no el acceso a la misma.

2.2.- AUTENTICACIÓN DE USUARIOS.

Cuando un usuario accede a una aplicación web, por lo general, lo primero es autenticarse, es decir, validarse para saber si puede o no acceder, que permisos tiene, que puede hacer…

La autenticación suele hacerse mediante la introducción de un nick y una contraseña y la verificación de estos en un servicio de autenticación:

- Servicio propio con nuestro ACL implementado en la aplicación.
- Servicio externo como facebook, twitter, gmail, … En este caso se usan protocolos estándares como OpenId o OAuth.

Si usamos nuestro servicio propio lo primero es recoger los datos de identificación. Podemos insertar un formulario para la recogida del usuario y contraseña y, a través del paso de parámetros realizar la autenticación.

HTTP aporta un mecanismo para la solicitud de las credenciales para autenticación. En este caso se usa header() para enviar un mensaje Authentication Required al navegador del cliente causando que se abra una ventana para ingresar usuario y password. Una vez se ha llenado el usuario y password, la URL contenida dentro del script PHP será llamada nuevamente con las variables predefinidas PHP_AUTH_USER, PHP_AUTH_PW, y AUTH_TYPE del array $_SERVER puestas con el nombre del usuario, password y el tipo de autenticación respectivamente.

Por ejemplo.

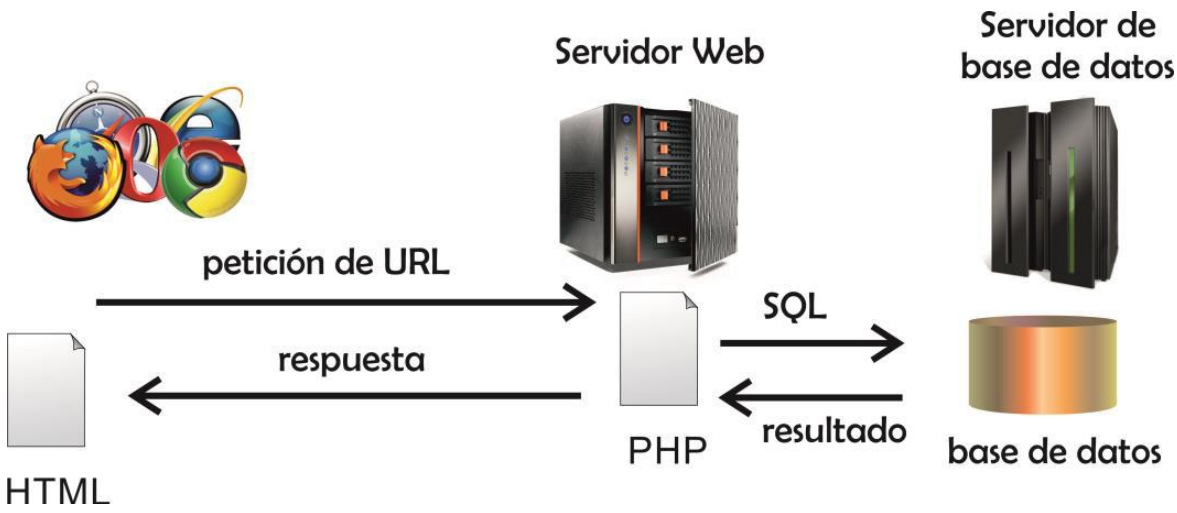
```php
if (!isset($_SERVER['PHP_AUTH_USER']))
  {
    header('WWW-Authenticate: Basic realm="Mensaje acceso"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Usuario incorrecto no puede acceder al sitio';
    exit;
  }
 else
  {
    echo "<p>Usuario {$_SERVER['PHP_AUTH_USER']}.</p>";
    echo "<p>Contrase&ntilde;a {$_SERVER['PHP_AUTH_PW']} </p>";
  }
```

Si no nos interesa esta forma se puede definir nuestro propio formulario en el que recogeremos el nick y la contraseña.

3.- TECNICAS DE ACCESO A DATOS

El uso de bases de datos es imprescindible en cualquier aplicación web de hoy en día. Algunas razones para utilizarlas son:

- **Proporcionan almacenamiento permanente**. Es decir, los datos que se almacenan resisten entre sesiones con lo cual son un método ideal cuando la información no se desea eliminar en ningún momento.
- **Añaden una capa más de seguridad**. Los datos están fuera del alcance directo del usuario ya que no se almacenan en las peticiones http. Los datos se encuentran en un servidor de base de datos con el que se comunica el servidor web. En esta comunicación no interviene el usuario. Aunque hay técnicas para intentar acceder a la base de datos usando trucos, lo cierto es que las bases de datos son un software más preparado para la seguridad que si nosotros gestionáramos directamente la información
- **Proporcionan herramientas avanzadas de gestión de datos y usuarios**. Los Sistemas Gestores de Bases de Datos son un software especializado en la gestión de datos, permite hacer consultas avanzadas, crear diferentes vistas y permitir su uso a ciertos usuarios, copias de seguridad, transacciones,…
- **Uso del lenguaje SQL**. El lenguaje más conocido para el manejo de datos en bases de datos relacionales, está disponible y esto significa que podremos usar toda su potencia en nuestras páginas web.



3.1.- MYSQL

PHP tiene un conjunto de librerías muy completo para trabajo con Bases de Datos lo que nos permite usar casi cualquier SGBD en nuestra aplicación. Sin embargo, de entre todas, destaca el SGBD MySQL, el cual se ha desarrollado casi a la par que PHP.

Para MySQL existen varias librerías:

- **PDO**. Es una capa de abstracción para acceso a bases de datos. Define una serie de objetos con una interfaz común. Según el SGBD se implementa una versión propia de cada objeto manteniendo su comportamiento general e implementando la funcionalidad propia.
- **Mysql**. API original para comunicación con MYSQL. Está formada por un conjunto de funciones. Está versión está declarada como obsoleta desde PHP 5.5 y se ha eliminado en la versión 7.0.
- **Mysqli**. Sustituye a la extensión mysql introduciendo una interfaz dual: programación procedimental y programación orientada a objetos.

Actualmente se aconseja usar PDO o mysqli para acceder a bases de datos MYSQL. Usaremos mysqli.

Como se ha indicado, mysqli aporta una interfaz dual: basada en procedimientos (al estilo de mysql) u orientada a objetos. El funcionamiento es semejante y se aconseja no mezclar ambas sintaxis. Un ejemplo podría ser

```
//Sintaxis procedimental
//Establece una conexión a un servidor MySQL
$mysqli    =    mysqli_connect("ejemplo.com",    "usuario",    "contraseña",
"basedatos");

//Compruebo si se ha establecido o no la conexión.
if (mysqli_connect_errno($mysqli))
    {
        echo "Fallo al conectar a MySQL: " . mysqli_connect_error();
        exit;
    }

//establece la página de códigos del cliente
$mysqli->set_charset("utf8");

//o mediante la ejecución de una sentencia
$mysqli->query("SET NAMES 'utf8'");


//Ejecuto una sentencia SQL
$consulta  =  mysqli_query($mysqli,  "SELECT  'Un  mundo  lleno  de  '  A  AS
mensaje FROM DUAL");

//obtengo una fila del conjunto resultado
$fila = mysqli_fetch_assoc($consulta);
echo $fila['mensaje'];

//cierro la conexión a la base de datos
mysqli_close($mysqli);

//Sintaxis Orientada a Objetos
//Establece una conexión a un servidor MySQL
$mysqli    =    new    mysqli("ejemplo.com",    "usuario",    "contraseña",
"basedatos");

//Compruebo si se ha establecido o no la conexión.
if ($mysqli->connect_errno)
    {
```

```
        echo "Fallo al contenctar a MySQL: " . $mysqli->connect_error;
        exit;
    }

//Ejecuto una sentencia SQL
$consulta =  $mysqli->query("SELECT 'elecciones para complacer a todos.'
AS mensaje FROM DUAL");

//obtengo una fila del conjunto resultado
$fila = $consulta->fetch_assoc();
echo $fila['mensaje'];

//cierro la conexión a la base de datos
$mysqli->close();
```

Para trabajar con MySQLi usaremos la forma orientada a objetos. En esta librería destacan fundamentalmente dos clases:

- **Mysqli**: Representa la conexión a una base de datos. Aporta métodos para establecer la conexión, ejecutar consultas/sentencias, iniciar/confirmar/deshacer transacción.
- **Mysqli_result**: Representa el conjunto de resultados a partir de una consulta a la base de datos.

Para establecer una conexión a la base de datos se usa la clase mysqli. Lo primero será crear una instancia para lo que se tiene el constructor:

```
__construct ([ string $host = ini_get("mysqli.default_host") [, string $username =
ini_get("mysqli.default_user") [, string $passwd = ini_get("mysqli.default_pw") [,
string $dbname = "" [, int $port = ini_get("mysqli.default_port") [, string $socket
= ini_get("mysqli.default_socket") ]]]]] )
```

Por ejemplo para conectarnos al servidor local (localhost) a la base de datos ‘prueba’ con el usuario ‘root’ y contraseña ‘contra’ se haría

```
$mysqli    =    mysqli_connect("ejemplo.com",    "usuario",    "contraseña",
"basedatos");
```

Para ***cerrar la conexión*** se usa el método close()

```
mysqli_close($mysqli);
```

Hay una configuración de PHP que hace que se lancen Excepciones/Errores en vez de warnings y que el código indicado para control de errores no funcione. Se usa para una depuración más profunda del código y no se aconseja en desarrollo. Se indica mediante la función **mysqli_report**. En nuestro caso, para que el código de control de errores funcione usaremos la opción **MYSQLI_REPORT_OFF**. Llamaremos por tanto a esta función para deshabilitar este funcionamiento

```
mysqli_report(MYSQLI_REPORT_OFF);
```

Otro aspecto a tener en cuenta es la codificación de los caracteres. En HTML definimos como código de página UTF-8 (nos permite generar contenido en todos los idiomas del mundo). La base de datos la definiremos en una configuración utf8 (utf8mb3_spanish2_ci o utf8mb4_spanish2_ci). Solo nos queda indicarle al cliente mysqli que la conexión sea en utf8. Para ello podemos usar el método set_charset de mysqli o ejecutar la sentencia "set names 'UTF-8' ".

```
//establece la página de códigos del cliente
$mysqli->set_charset("utf8");

//o mediante la ejecución de una sentencia
$mysqli->query("SET NAMES 'utf8'");
```

Al utilizar bases de datos hay muchos posibles errores que pueden ocurrir. De hecho si no funciona la conexión, todo lo demás que nuestra aplicación quisiera realizar, no sería posible. Podemos capturar el error mediante propiedades:

- *int $connect_errno*: Código de error de la última conexión.
- *String $connect_error*: Mensaje de error de la última conexión.

```
if ($mysqli->connect_errno)
    {
        echo "error al conectar nº {$mysqli->connect_errno}: ".
                "{$mysqli->connect_error}";
        exit;
    }
```

El siguiente paso es **ejecutar sentencias** SQL. Podemos ejecutar dos tipos de sentencias: aquellas que devuelven un conjunto resultado (sentencia select) y aquellas que no devuelven un conjunto resultado (insert, update, delete, create, alter, drop).

En ambos caso se usa el método query.

mixed ***mysqli::query*** *(string $query [, int $resultmode = MYSQLI_STORE_RESULT])*

Esta función devolverá false si se ha producido un error, un objeto de tipo mysqli_result para sentencias de tipo select y true para el resto de sentencias.

Si se tiene un **error** se puede consultar en las propiedades $errno (número de error) y $error (mensaje de error).

```
$sentencia='select nombre,dni from usuarios';

$consulta=$mysqli->query($sentencia);

if ($mysqli->errno!=0)
 {
    echo "error  nº {$mysqli->errno}: {$mysqli->error}";
```

```
    exit;

 }

$sentencia='update usuarios set edad=edad+1';
if (!$mysqli->query($sentencia))
 {
    echo "error  nº {$mysqli->errno}: {$mysqli->error}";
    exit;

 }
```

Se puede saber el número de **filas afectadas** por la ejecución de la última sentencia mediante la propiedad $affected_rows. En el caso de sentencias insert/delete/update representa cuantas filas se han insertado/borrado/actualizado. En el caso de sentencias select, es el número de **filas devueltas** (equivalente a la propiedad $num_rows).

Si se ha ejecutado una **sentencia de inserción** se puede saber el código de la clave principal generada de forma automática mediante la propiedad $insert_id.

```
if ($mysqli->affected_rows==0)
    {
        echo "no se ha actualizado ninguna fila";
    }
```

El siguiente paso en una sentencia select es recorrer el conjunto resultado a través del objeto **mysqli_result** devuelto en el método query.

Básicamente se realiza obteniendo todas las filas (método fetch_all) o devolviendo fila a fila (fetch_array, fetch_assoc, fetch_object, etc).

Si se obtienen todas las fila se usa el método fetch_all.

mixed ***mysqli_result::fetch_all*** ([*int* `$resulttype` = *MYSQLI_NUM*])

Devolviendo las filas como un array indexado (*MYSQLI_NUM)*, un array asociativo (*MYSQLI_ASSOC*) o ambos (*MYSQLI_BOTH*)

```
$filas= $consulta->fetch_all(MYSQLI_ASSOC);

foreach($filas as $fila)
{
    print_r($fila);
}
```

Si se desea obtener fila a fila se puede usar cualquiera de los métodos fetch_xxx, obteniendo el resultado como array indexados, array asociativo, objeto, etc.

```
$sentencia='select nombre,dni from usuarios';

$consulta=$mysqli->query($sentencia);
```

```php
if ($mysqli->errno!=0)
    {
        echo "error  nº {$mysqli->errno}: {$mysqli->error}";
        exit;
    }

//se recorre fila a fila, devolviéndola como array asociativo
while ($fila=$consulta->fetch_assoc())
{
    echo "El nombre {$fila["nombre"]} con dni {$fila["dni"]}<br>\n";
}
```

Las funciones serían:

- mixed **mysqli_result::fetch_array** ([int `$resulttype` = MYSQLI_BOTH]). Devuelve un array asociativo (parámetro a MYSQLI_ASSOC), un array indexado (MYSQLI_NUM) o un array con ambas formas (MYSQLI_BOTH).
- array **mysqli_result::fetch_asso**c (void). Devuelve un array asociativo.
- object **mysqli_result::fetch_object** ([string $class_name [, array $params]]). Devuelve una clase.
- mixed **mysqli_result::fetch_row** (void). Devuelve un array indexado.

3.2.- SEGURIDAD EN BASE DE DATOS.

En la actualidad la mayoría de las aplicaciones desarrolladas usan de algún modo Bases de Datos. Los hackers usan diversas vulnerabilidades para alcanzar su objetivo de tomar el control u obtener información de ellas. Es por ello, que se deben tomar medidas para limitar el impacto de un ataque en nuestras aplicaciones.

A nivel de Bases de Datos se puede trabajar en cuatro aspectos: diseño de la base de datos, conexión a una base de datos, almacenamiento cifrado e inyección de SQL.

Diseño de la Base de Datos:

Lo primero que se hace es crear la Base de Datos. Cuando se crea, ésta es asignada a un propietario (el que ejecutó la sentencia de creación). Normalmente, sólo el propietario (o un superusuario) puede hacer cualquier cosa con los objetos de esa base de datos.

Las aplicaciones **nunca** deben **conectarse** a la base de datos **como** con el **propietario** o el **superusuario**, ya que estos usuarios pueden ejecutar cualquier sentencia.

Por tanto, se deben crear distintos usuarios con permisos muy limitados a los objetos de dicha base de datos (sólo aquellos necesarios para poder obtener la funcionalidad deseada) de forma que si un atacante obtiene las credenciales del usuario se minimice lo que puede hacer.

Conexión a una Base de Datos

Establecer la conexión sobre SSL para cifrar la comunicación entre el servidor/cliente.

Almacenamiento cifrado

SSL protege los datos que se envían entre el servidor/cliente. Cuando un atacante obtiene acceso a una base de datos, SSL no te protege. El atacante puede ver todos los datos incluso aquellos más sensibles.

En este caso, la forma de mitigar esta amenaza es cifrar la base de datos, algo que no todos los SGBD lo permiten.

La forma más sencilla para evitar este problema es crear tu propia API para que cifre/descifre la información cuando se almacene/recupere la misma de la Base de Datos.

Si no usamos lo anterior, en caso de datos que deban estar realmente ocultos (no se muestra su representación real) como contraseñas de usuarios, se usarán representaciones seguras de la misma. En PHP podemos usar funciones como password

Inyección de código

Las consultas SQL pueden ser manipuladas y convertirse así en métodos por los que un atacante puede tomar control de nuestra Base de Datos.

Normalmente, usamos formularios en los que se solicita información a los usuarios. Si el programador da por buenos los datos enviados, un atacante podría modificar los datos indicados en el formulario de forma que se generase una sentencia con la que obtuviera información o incluso tomara el control, es lo que se conoce como inyección de código.

Por ejemplo, tenemos un formulario en el que se solicita un email con el siguiente código al recibir el formulario:

```
$conex = new mysqli($servidor, $usuario, $contrasenia, $bd);

$email = $_POST["email"];

//sentencia que obtiene todos los datos del usuario
//con el email indicado
$sentencia = "select * from usuarios where email='$email'";

//ejecuto la sentencia
$consulta= $conex->query($sentencia);

//recorro el usuario con ese email
while ($fila=$consulta->fetch_assoc())
{

}
```

Un usuario normal introduciría una dirección de correo, por ejemplo, profesor@iespedroespinosa.es. La sentencia que se ejecutaría sería

```
“select datos from tabla where email = ‘profesor@iespedroespinosa.es’ “
```

Un atacante podría introducir en el formulario el siguiente texto “*’ or 1=1 or “*”. En este

caso obtendría los datos de todos los usuarios. La sentencia que generaríamos sería

```
“select datos from tabla where email = ‘’ or 1=1 or ‘’ ”,
```

Para prevenir el ataque de inyección de código se aconseja:

- ***usar sentencias preparadas y parametrizadas***. De esta forma, al indicar los parámetros se indican el tipo de los mismos y se convierten de forma automática al ejecutar la sentencia.

```php
$conex = new mysqli($servidor, $usuario, $contrasenia, $bd);

$email = $_POST["email"];

//sentencia que obtiene todos los datos del usuario
//con el email indicado
//preparo la sentencia
$sentencia = $conex->prepare("select * from usuarios where email=?");

//asigno el parámetro
//en la cadena se indica el tipo de cada parámetro
//s cadena, i entero, d float, b blob
$sentencia->bind_param("s",$email);

//ejecuto la sentencia
$sentencia->execute();

//obtengo el resultado
$consulta=$sentencia->get_result();

//recorro el usuario con ese email
while ($fila=$consulta->fetch_assoc())
{

}
```

- si no se usan sentencias con parámetros

  a) ***convertir todo dato*** al tipo apropiado usando funciones de php (intval, floatval, r…)

  b) ***escapar las cadenas***, en mysqli con **escape_string**. Con esta función se escapan varios caracteres como ‘ y “ para que no tengan un significado especial (por ejemplo ‘ es fin de cadena en mysql)

```php
$conex = new mysqli($servidor, $usuario, $contrasenia, $bd);

$email = $_POST["email"];

//email es una cadena, escapo la cadena
$email= $conex->escape_string($email);

//si tuviera entero usaría
```

```php
//$num=intval($num);

//sentencia que obtiene todos los datos del usuario
//con el email indicado
$sentencia = "select * from usuarios where email='$email'";

//ejecuto la sentencia
$consulta= $conex->query($sentencia);

//recorro el usuario con ese email
while ($fila=$consulta->fetch_assoc())
{

}
```

4.- OPERACIONES BÁSICAS EN UNA APLICACIÓN

Cuando se desarrolla una aplicación hay una serie de acciones a desarrollar. Estas operaciones incluyen:

- Diseño de la estructura e interfaz de la aplicación.
- Soporte de autenticación de usuario. En este apartado se incluyen tres elementos:
    a. Gestión de usuarios. Realizado mediante una ACL. Aquí se incluyen las operaciones para gestión de usuarios, roles y permisos.
    b. Solicitud de credenciales. Solicitud de usuario/contraseña cuando se accede a la aplicación.
    c. Validación y control de usuario en la aplicación. Una vez indicadas las credenciales, la comprobación de las mismas y su uso a lo largo de toda la ejecución de la aplicación.
- Mantenimiento de elementos (CRUD). Operaciones de creación, modificación, consulta y borrado sobre elementos en la aplicación.

4.1.- DISEÑO DE LA ESTRUCTURA E INTERFAZ DE LA APLICACIÓN

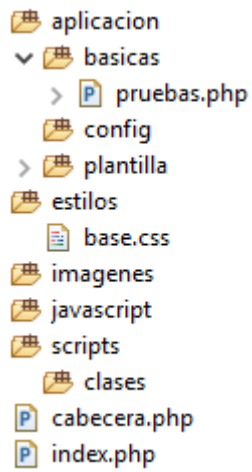
4.1.A.- DISEÑO DE LA ESTRUCTURA DE LA APLICACIÓN

El desarrollo de una aplicación implica unos costes que se deben minimizar para obtener el máximo rendimiento. Esto implica que operaciones como la creación de nuevas ventanas o la modificación de la interfaz (si fuera necesario) no supongan un coste elevado. Todo esto se puede conseguir mediante un diseño correcto de nuestra aplicación.

Si usamos un framework para desarrollar nuestra aplicación, el propio framework aporta los mecanismos para estructurar correctamente la aplicación.

Si no usamos un framework, lo mejor es preparar nuestra propia estructura que nos facilite la creación y mantenimiento de la aplicación.

En primer lugar creamos una estructura de carpetas y archivos. Como ejemplo podríamos tener una estructura semejante a:



Carpeta aplicación: En ella colocaremos los distintos ficheros de nuestra aplicación. Lo mejor es crear una carpeta en la que colocar todos los ficheros relacionados. Por ejemplo, para la gestión de articulos podemos crear la carpeta /aplicacion/articulos. Dentro de ella colocaremos los ficheros listarArticulos.php, nuevoArticulo.php, verArticulo.php, etc.

En aplicación tambien tenemos dos carpetas: config y plantilla. En config situamos archivos de configuración. Por ejemplo, un archivo con los datos de acceso a la base de datos. En plantilla podemos situar el/los ficheros necesarios para definir la plantilla del sitio web. Si tenemos una plantilla con la interfaz de la aplicación, será fácil modificarla en el futuro.

En la carpeta estilos colocaríamos los distintos ficheros de estilos a

usar.

En la carpeta imágenes colocaríamos las imágenes que usáramos.

En la carpeta javascript podríamos colocar todos los ficheros de javascript que necesitaramos.

En la carpeta scripts colocaríamos nuestos ficheros con aquella funcionalidad general. Por ejemplo, nuestra clases básicas necesarias para que funcione la aplicación pero que no implementa la lógica del programa (ACL, ControlAcceso, BD, Command, Sesiones, etc). Aquí podemos también colocar librerías externas que necesitemos como componentes para PDF, para convertir a EXCEL, ACCESS, etc.

Una vez creada la estructura de carpetas, el siguiente paso es crear la estructura de las páginas. Se debe intentar separar lo más posible la lógica de la aplicación de la interfaz. Para ello hay numerosas librerías que nos lo permiten hacer. Si no queremos usar ninguna api externa deberemos generar nuestra propia API para definir la intefaz.

Lo mejor es estudiar el diseño de página que deseamos y crear funciones (o una clase con métodos estáticos) que se encargen del diseño de la página. Como ejemplo podríamos tener el fichero /aplicación/plantilla/plantilla.php con la siguiente estructura.

```
<?php

function paginaError($mensaje)
{
  header("HTTP/1.0 404 $mensaje");
  inicioCabecera("PRACTICA");
  finCabecera();
  inicioCuerpo("ERROR");
  echo "<br />\n";
  echo $mensaje;
  echo "<br />\n";
  echo "<br />\n";  echo "<br />\n";
  echo "<a href='/index.php'>Ir a la pagina principal</a>\n";

  finCuerpo();
}

function inicioCabecera($titulo)
{
?>
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="utf-8">

        <!-- Always force latest IE rendering engine (even in intranet) &
Chrome Frame
        Remove this if you use the .htaccess -->
                                    <meta     http-equiv="X-UA-Compatible"
 content="IE=edge,chrome=1">

        <title><?php echo $titulo ?></title>
```

```
        <meta name="description" content="">
        <meta name="author" content="Administrador">

            <meta  name="viewport"  content="width=device-width;  initial-
scale=1.0">

         <!-- Replace favicon.ico & apple-touch-icon.png in the root of
your domain and delete these references -->
        <link rel="shortcut icon" href="/favicon.ico">
        <link rel="apple-touch-icon" href="/apple-touch-icon.png">

        <link rel="stylesheet" type="text/css" href="/estilos/base.css">
<?php
}

function finCabecera()
{
?>
    </head>
<?php
}

function inicioCuerpo($cabecera)
{
    global $acceso;

?>
    <body>
        <div id="documento">

            <header>
                <h1 id="titulo"><?php echo $cabecera;?></h1>
            </header>

            <div id="barraLogin">

            </div>
            <div id="barraMenu">
                <ul>
                    <li><a href="/index.php">Inicio</a></li>
                 </ul>

            </div>

            <div>
<?php
}

function finCuerpo()
{
?>
                <br />
                <br />
            </div>
            <footer>
                <hr width="90%"  />
                <div>
```

```
                    &copy; Copyright  by Profesor
                </div>
            </footer>
        </div>
    </body>
</html>
<?php
}
```

Las funciones definidas son:

- inicioCabecera($titulo): Escribe las estiquetas correspondientes a <html> y <head>. Aquí se añaden enlaces a javascript y a hojas de estilo, es decir, todos aquellos elementos que queremos que se cargen en todas las páginas. Además se crea una etiqueta title con el $titulo indicado
- finCabecera(). Cierra la etiqueta head
- inicioCuerpo($cabecera). Inicia el body e introduce una línea de cabecera ($cabecera).
- finCuerpo(). Define un pie para las páginas y cierra las etiquetas body y html

Para completar la plantilla se tendrá el archivo con los estilos de nuestra página (/estilos/base.css)

```
@charset "UTF8";

body {
    background-color: cyan;
    font-size: 16px;
}

#documento{
    margin: 0 auto;
    background-color: #8080ff;
    width: 700px;
}
#cabecera{
    background-color: #14e737;
    padding: 5px;
}
#cabecera #logo{
    width: 75px;
    height: 50px;
}

#cabecera #titulo{
    font-size: 1.3em;
    font-style: italic;
}

.boton{
    background-color: blue;
    color:white;
    border:thin black solid;
}
```

```
.boton a{
    text-decoration: none;
    colo:white;
}

.boton a:visited{
    text-decoration: none;
    color:white;
}

.boton a:hover{
    background-color:cyan;
    color: black;
}

.boton:hover{
    background-color:cyan;
    color: black;
}

.tabla{
    width: 100%;
    border-collapse: collapse;
    border: none;
    background-color: white;
}

.tabla th{
    background-color: teal;
    color: blue;
}
.tabla tr.par{
    background-color: black;
    color: white;

}

.tabla tr.impar{

}

#pie{
    background-color: #81e714;
    color: blue;
    font-size: 1.5 em;
    padding: 5px;
}

.for_texto{
    border:blue thin solid;
    background-color: white;
    padding: 2px;
    margin:2px;
    border-radius: 4px;
}
.error{
    background-color: white;
```

```
    color:red;
    border:thin solid red;
    padding:5px;
}
```

Se crean dos archivos más.

- ***Cabecera.php:*** este fichero se encarga de definir funciones básicas como la autocarga de clase o el modo de trabajo. Además se encarga de crear e inicializar todas las variables y clases necesarias para que trabaje mi aplicación.

```
<?php
define("RUTABASE", dirname(__FILE__));
//define("MODO_TRABAJO","produccion"); //en "produccion o en desarrollo
define("MODO_TRABAJO","desarrollo"); //en "produccion o en desarrollo

if (MODO_TRABAJO=="produccion")
    error_reporting(0);
    else
        error_reporting(E_ALL);

spl_autoload_register(function ($clase){
    $ruta=RUTABASE."/scripts/clases/";
    $fichero=$ruta."$clase.php";

    if (file_exists($fichero))
        {
            require_once($fichero);
        }
      else
        {
            throw new Exception("La clase $clase no se ha encontrado.");
        }
});

include(RUTABASE."/aplicacion/plantilla/plantilla.php");
include(RUTABASE."/aplicacion/config/acceso_bd.php");

//creo todos los objetos que necesita mi aplicación
```

- ***Index.php***: Punto de acceso inicial a mi aplicación.

```
<?php
include_once(dirname(__FILE__)."/cabecera.php");


inicioCabecera("APLICACION PRUEBA");
cabecera();
finCabecera();

inicioCuerpo("APLICACION PRUEBA");
cuerpo();
```

```
finCuerpo();


// *********************************************************

function cabecera()
{
}

function cuerpo()
{
?>

            <a     href="/aplicacion/articulos/verArticulos.php">     Ver
articulos</a><br>


<?php

}
```

Con esto se pretende aplicar el paradigma **Modelo-Vista-Controlador**. En nuestro caso, se separa la parte correspondiente a la vista (funciones de diseño de la página) del controlador (parte inicial del fichero). En la vista se indica todo lo que se va a mostrar al usuario y en el controlador se hacen todas las operaciones y obtienen los datos a mostrar en la vista.

4.1.B.- ESTRUCTURA BÁSICA DE UNA PÁGINA

A partir del fichero anterior, una página cualquiera tendría la forma:

```
<?php
include_once(dirname(__FILE__)."/cabecera.php");


inicioCabecera("APLICACION PRUEBA");
cabecera();
finCabecera();

inicioCuerpo("APLICACION PRUEBA");
cuerpo();
finCuerpo();


// *********************************************************

function cabecera()
{

}

function cuerpo()
{
```

```
}
```

Para cada página particular se llaman a las funciones de la plantilla además de definir dos propias:

- cabecera(). Nos permite definir código html que se incluirá en la cabecera (head)
- cuerpo(). Se usa para definir el contenido propio de la página.

4.1.C.- PAGINA CON FORMULARIO

Otro punto a tener en cuenta es como actuar cuando se tiene un formulario. Se puede optar por:

- Un fichero para el formulario y otro para la acción del formulario
- Un fichero único con el formulario y la acción del mismo.

Todo parámetro que se reciba debe validarse y/o convertirse al tipo apropiado. Con ello evitamos posibles problemas tanto al almacenar en BD como con ataques por injeccion de código.

A continuación muestro una página en la que se pide los datos de un nuevo artículo y se valida.

```
<?php
include_once(dirname(__FILE__)."/../../cabecera.php");

//inicializaciones
$datos=["nombre"=>"",
        "precio"=>0
        ];
$errores=[];



//comprobar si se ha dado a insertar
if ($_POST)
    {
        $nombre="";
        if (isset($_POST["nombre"]))
            {
                $nombre=$_POST["nombre"];
                $nombre=strtolower(trim($nombre));
            }
        if ($nombre=="")
            $errores["nombre"][]="Debe indicarse un nombre";

        if (strlen($nombre)>20)
           $errores["nombre"][]="El  nombre  no  puede  tener  mas  de  30
caracteres";
```

```
        $datos["nombre"]=$nombre;

        $precio=0;
        if (isset($_POST["precio"]))
            {
                $precio=floatval($_POST["precio"]);
            }

        if ($precio==0)
            $errores["precio"][]="Debe indicarse un precio";

        $datos["precio"]=$precio;


        if (!$errores)  //no hay errores hago la insercion
            {
                //codigo para el nuevo articulo
                $codigo=1; //codigo

                //se guarda el articulo

                //ir a ver el articulo insertado
                header("location: verArticulo.php?id=$codigo");
                exit;
            }


    }


inicioCabecera("PRUEBA: articulos");
cabecera();
finCabecera();

inicioCuerpo("NUEVO ARTICULO");
cuerpo($datos,$errores);
finCuerpo();


// ******************************************************

function cabecera()
{


}


function cuerpo($datos,$errores)
{


    ?>
    <br>
    <br>
    <br>
```

```
    <?php
    formulario($datos,$errores);
}

function formulario($datos,$errores)
{
    if ($errores)
    {   //mostrar los errores
        echo "<div class='error'>";
        foreach($errores as $clave=>$valor)
        {
            foreach($valor as $error)
                echo "$clave => $error<br>".PHP_EOL;
        }
        echo "</div>";

    }

    ?>
    <form action="" method="post">
        <label for="nombre">Nombre: </label>
        <input type="text" name="nombre" id="nombre"
            value="<?php echo $datos["nombre"];?>" size=21
        maxlength="20"><br>
        <label for="precio">Precio: </label>
        <input type="text" name="precio" id="precio"
                value="<?php echo $datos["precio"];?>" size=4
            maxlength="3"><br>

        <input type="submit" class="boton" value="Crear">
    </form>
    <?php
}
```

Para hacer el código más claro puede crearse una función que se encarga de validar el envío del formulario.

Deben usarse al mínimo las variables globales y dejarlas únicamente para variables que se pueden considerar casi “constantes” y que no se modifican.

4.2.- AUTENTICACIÓN DE USUARIOS.

Normalmente en toda aplicación se realizan operaciones que no todo usuario puede realizar. Por ejemplo en una tienda virtual, el administrador o un trabajador de la empresa podría cambiarle el nombre a un artículo. Sin embargo, un comprador nunca debe poder cambiarlo.

Para realizar la autenticación se necesita:

a. Gestión de usuarios. Realizado mediante una ACL. Aquí se incluyen las operaciones para gestión de usuarios, roles y permisos. Esto ya se ha visto anteriormente en el

apartado 2.1

b. Solicitud de credenciales. Solicitud de usuario/contraseña cuando se accede a la aplicación. Esto se ha visto anteriormente en el apartado 2.2.
c. Validación y control de usuario en la aplicación. Una vez indicadas las credenciales, la comprobación de las mismas y su uso a lo largo de toda la ejecución de la aplicación.

El apartado c) Validación y control de usuario se suele realizar mediante una clase **Acceso** que guarde los datos del usuario actualmente registrado y los permisos que tiene. Además esta clase aporta métodos tanto para asignar como para consultar toda esa información.

Por ejemplo, se podría tener la clase Acceso con la siguiente definición:

- $Validado: Hay actualmente un usuario validado.
- $Nick: Nick del usuario.
- $Nombre: nombre del usuario.
- $PuedeAcceder: Tiene permiso de acceso.
- $PuedeConfigurar: Tiene permiso de configuración.
- registrarUsuario($nick, $nombre, $puedeAcceder, $puedeConfigurar). Sirve para guardar en la sesión la información del usuario validado.
- quitarRegistroUsuario(). Hace que no haya ningún usuario registrado en el sistema
- hayUsuario(). Devuelve true si hay un usuario validado y false en caso contrario.
- puedeAcceder(). Devuelve true si puede acceder el usuario validado y false en caso contrario.
- puedeConfigurar() Devuelve true si puede configurar el usuario validado y false en caso contrario.
- getNick(), getNombre()

Una vez que se tiene esta clase en las página se llama a los métodos apropiados para permitir o no mostrar la página

```
//validar el acceso a la pagina
//si no hay usuario, tiene que validarse
if (!$acceso->hayUsuario())
    {
        pedirLogin();
        exit;
    }

if (!$acceso->puedePermiso(2))
    {
        paginaError("Acceso no permitido");
        exit;
    }
```

Esta clase también puede usarse para crear enlaces o mostrar partes concretas.

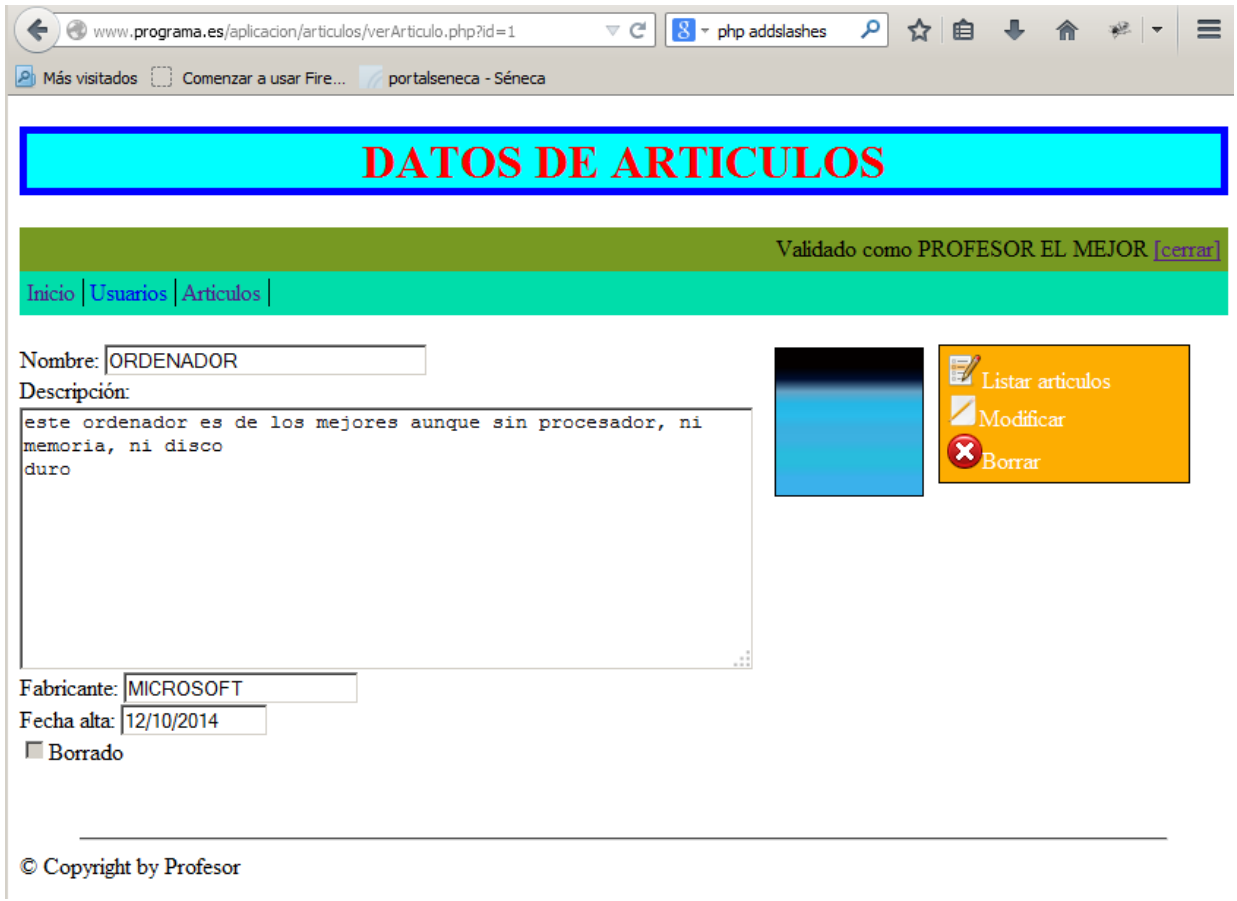
```
if ($acceso->hayUsuario() &&
    $acceso->puedePermiso(2))
    {
    ?>
        <br />
            <a  href='/aplicacion/articulos/modificarArticulo.php?id=<?php
echo $fila["cod_articulo"];?>'>
                <img src='/imagenes/24x24/modificar.png'></a>
    <?php
    }
```

4.3.- CRUD

En cualquier aplicación en la que se tenga Base de Datos, se hace necesario la gestión de los diferentes elementos con unas operaciones básicas: alta, consulta, modificación y borrado o en inglés create, read, update and remove (CRUD).

Normalmente defino una pantalla que nos sirve de consulta en bloques. En ella se puede consultar todos los elementos existentes. Como ejemplo voy a tomar una pantalla de articulos. Para cada elemento se tienen las operaciones de consulta/borrado/modificación que no es más que enlaces a las páginas apropiadas. En esta pantalla se suelen mostrar opciones de filtrado y de paginación.

La página de consulta muestra los datos de un elemento determinado. Se incluyen las operaciones a realizar sobre el elemento.

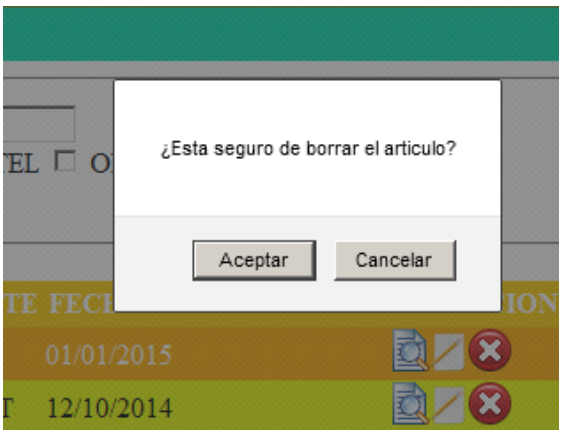


La página de modificación nos permite cambiar las propiedades del elemento. Para facilitar el trabajo se suelen dar una serie de facilidades para el usuario:

- Se cargan inicialmente los datos del elemento.
- Al guardar los cambios, si hay algún error en los datos introducidos se vuelve a mostrar el formulario con mensajes de error apropiados y mostrando de nuevo los datos introducidos.
- Se valida todo lo introducido y se muestran errores apropiados.
- Si todo es correcto se almacena y seabre la ventana de consulta del elemento.

La página de creación es esencialmente igual a la de modificación salvo que no se cargan los valores iniciales.

El borrado se puede hacer con una página separada de confirmación del elemento o simplemente con una confirmación javascript, eso si, el borrado se realiza en otro fichero al hacerse en el servidor.



En relación con el borrado, se tiende a hacer un borrado lógico (marca borrado S/N) en vez de físico de los elementos. Así se puede tener un historial sobre los elementos borrados que de otra forma desaparecería.

Se puede observar ese borrado lógico en las pantalla anteriores con la propiedad Borrado de articulo.