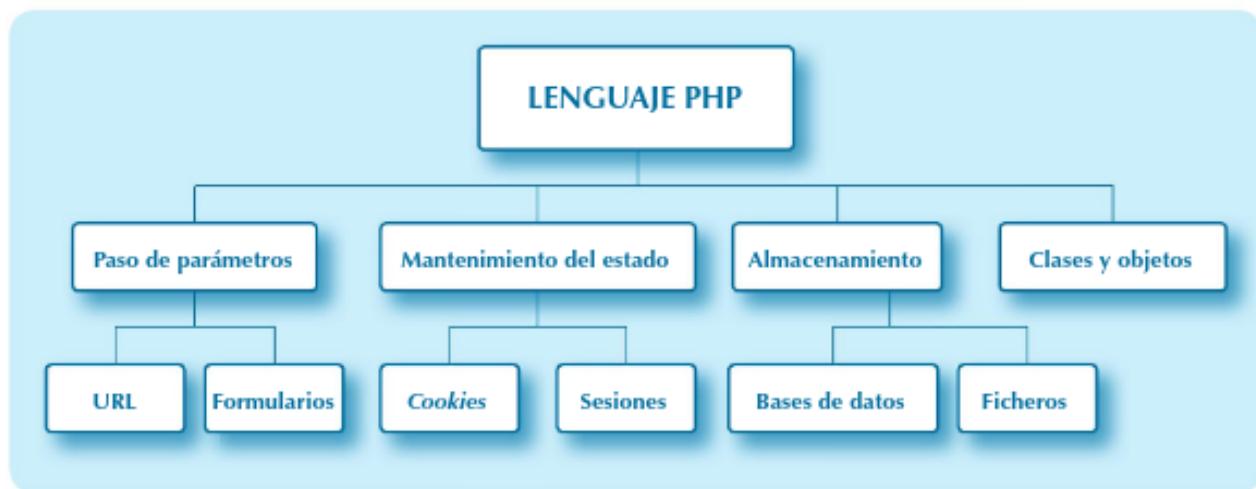


# Desarrollo de aplicaciones web con PHP

## Objetivos

- ✓ Conocer los mecanismos de paso de parámetros a un *script*.
- ✓ Procesar y validar formularios.
- ✓ Utilizar *cookies*.
- ✓ Aprender a emplear las sesiones y las variables de sesión.
- ✓ Manejar bases de datos.
- ✓ Enviar correos desde PHP.
- ✓ Dominar las herramientas de pruebas y depuración.

## Mapa conceptual



## Glosario

**Base de datos no relacional.** Sistema gestor de bases de datos que no sigue el modelo relacional. Por ejemplo, bases de datos XML.

**Cookie.** Fichero que almacenan los servidores en los clientes. Puede almacenar información sobre la última visita o las preferencias del usuario, entre otras cosas.

**Formulario.** Elemento que permite al usuario introducir datos que se envían servidor.

**Sesión.** Las sesiones permiten que las páginas de una misma aplicación comparten información.

**XML.** eXtensible Markup Language, “lenguaje de marcas extensibles”. Es un estándar del W3C.

**XPath.** El lenguaje XPath se utiliza para buscar información en ficheros XML. Es un estándar del W3C.

**XSD.** Lenguaje basado en XML para validar documentos XML. Validar consiste en verificar que un fichero tiene una estructura determinada. Es un estándar del W3C.

**XSL/XSLT.** Lenguaje basado en XML para realizar transformaciones sobre documentos XML. El resultado puede ser XML, HTML... Es un estándar del W3C.

### 3.1. Paso de parámetros

El protocolo HTTP define ocho métodos o verbos para establecer una comunicación entre cliente y servidor. Los más habituales son GET (obtener) y POST (enviar). El método GET es el que se utiliza habitualmente para solicitar páginas web a un servidor, por ejemplo, al seguir

un vínculo o introducir una dirección a mano en la barra del navegador. El método POST se utiliza sobre todo para enviar formularios al servidor.

Cuando se usa el método GET se pueden pasar parámetros al servidor en la URL. A la ruta normal para acceder a una página se le añade el carácter "?" como indicador de que empieza la lista de parámetros. Cada parámetro tiene un nombre, a la izquierda del igual, y un valor, a la derecha. Los argumentos están separados entre sí por el carácter *ampersand*. Por ejemplo, si se accede con el navegador a:

`http://localhost/cap3/hola_nombre.php?nombre=Ana`

Se solicita al servidor el fichero **/cap3/hola\_nombre.php** del servidor *localhost* y se le pasa un parámetro llamado "nombre" con valor "Ana".

Con la siguiente URL se añadiría otro parámetro para el apellido:

`http://localhost/cap3/hola_nombre.php?nombre=Ana&apellido=Luna`

Se puede acceder a los argumentos de la URL desde un bloque PHP usando el *array* superglobal `$_GET`, que tiene un elemento por cada argumento presente en la URL. El nombre del argumento será la clave del elemento del *array*.

El fichero **hola\_nombre.php** muestra un mensaje personalizado usando el valor del parámetro nombre.

```
<?php  
echo "Hola ". $_GET["nombre"];
```

Si se accede con la ruta

`http://localhost/cap3/hola_nombre.php?nombre=Ana,`

se obtendrá el mensaje "Hola Ana".

Si se accede con

`http://localhost/cap3/hola_nombre.php,`

se obtendrá:

`Notice: Undefined index: nombre in C:\xampp\htdocs\cap3\ej3_1.php on line 2 Hola`

Para controlar si los parámetros se han pasado correctamente se pueden utilizar las funciones `empty()` o `is_null()`. Las dos devuelven TRUE cuando el parámetro no está presente en la URL. La diferencia está en los parámetros presentes, pero sin valor, por ejemplo:

`http://localhost/cap3/hola_nombre.php?nombre`

En este caso, `empty($_GET["nombre"])` devuelve TRUE, pero `is_null($_GET["nombre"])` devuelve FALSE.

El ejemplo **hola\_comprobacion.php** mejora el anterior para mostrar un mensaje de error si no se pasa el parámetro nombre.

```
<?php
    if (empty($_GET["nombre"])) {
        echo "Error, falta el parámetro nombre";
    }else {
        echo "Hola ". $_GET["nombre"];
    }
}
```

### Actividad propuesta 3.1



Escribe un fichero que reciba dos parámetros, num1 y num2, y muestre su suma. Hay que comprobar que los dos argumentos existan y sean números.

## 3.2. Formularios

Los formularios HTML son la forma más habitual de enviar datos a un servidor. Permiten que el usuario rellene varios campos mediante diferentes tipos de controles (campos de texto, botones de radio...) y lo envíe al servidor al pulsar un botón. El servidor procesa los datos del formulario y genera la respuesta.

Un formulario sencillo de *login* en HTML se escribiría así:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Formulario de login</title>
        <meta charset = "UTF-8">
    </head>
    <body>
        <form action = "login_basico.php" method = "POST">
            <input name = "usuario" type = "text">
            <input name = "clave" type = "password">
            <input type = "submit">
        </form>
    </body>
</html>
```

El atributo **action** del formulario especifica la ruta del *script* al que se enviará el formulario para que lo procese. Si se usa una ruta relativa, se toma como referencia la localización en el servidor del fichero que contenga el formulario.

El atributo **method** especifica el método HTPP, que se usará para la petición. En los formularios lo habitual es utilizar POST, pero también es posible utilizar GET. Si se utiliza POST, los parámetros no aparecen en la URL.

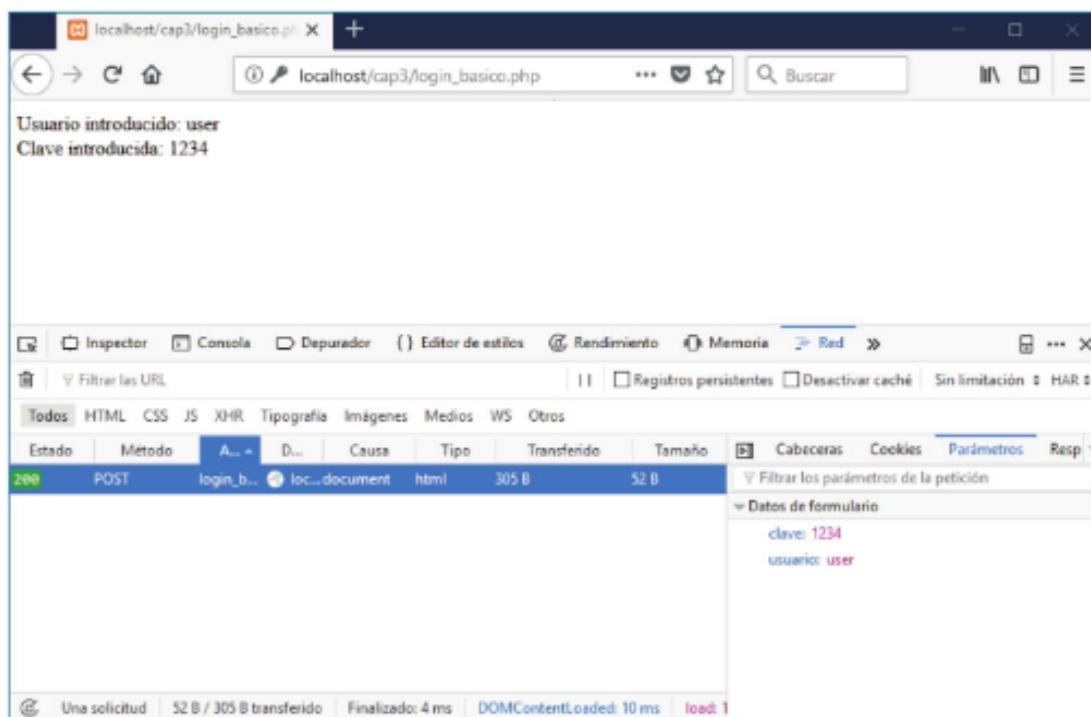
Dentro del elemento **form** se introducen los campos, que tendrá que llenar el usuario, y los botones de envío y para limpiar los campos. Para los campos que se envían hay que usar el atributo **name**, que sirve para identificarlo dentro del *script*. El envío se produce al pulsar el botón.

En el *script* al que se envía el formulario los parámetros están disponibles en el *array* superglobal `$_POST`. La clave de cada argumento dentro del *array* es el atributo `name` del elemento correspondiente en el formulario.

El fichero **login\_basico.php**, al que se envía el formulario anterior, muestra los valores introducidos en él:

```
<?php
echo "Usuario introducido: ". $_POST['usuario']. "<br>";
echo "Clave introducida: ". $_POST['clave'];
```

Al usar el método POST los parámetros no se muestran en la URL, pero se pueden consultar desde la consola del navegador. En Firefox está disponible en Menú > Desarrollador Web > Red. Una vez abierta, muestra las peticiones que se realizan desde el navegador. Si se selecciona la correspondiente al envío del formulario, se pueden consultar los parámetros en la parte derecha.



**Figura 3.1**  
Herramientas de desarrollador en Firefox.

### 3.2.1. Formulario de *login*

En general un formulario de *login* se encarga de comprobar los datos introducidos y, según sean correctos o no, da acceso al sistema al usuario o muestra un mensaje de error. El fichero **login\_falso.php** se encarga de procesar un formulario de *login*. Para simular un formulario de *login*, hay que comprobar que el usuario y la contraseña sean correctos. Si el usuario es “usuario” y la clave es “1234”, se redirige a la página de bienvenida. En caso contrario, lo hace a una página de error. Para la redirección se usa la función `header`, que sirve para escribir en la cabecera de la respuesta HTTP.

```
<?php
/*si va bien redirige a bienvenido.html
si va mal, mensaje de error */
if ($_POST['usuario']=="usuario" and $_POST["clave"]=="1234"){
    header("Location:bienvenido.html");
} else {
    header("Location:error.html");
}
```

**RECUERDA**

- ✓ Hay que enviar las cabeceras antes de empezar con el cuerpo de la respuesta. Esto implica que hay que utilizar la función `header()` antes de que se empiece a escribir la salida. Si se intenta llamar a `header()` después de haber realizado un `echo`, se producirá un error.

### 3.2.2. Formulario y procesamiento en un solo fichero

En ocasiones el formulario HTML y el bloque PHP que lo procesa se integran en un solo fichero, en el que hay que distinguir entre dos casos. Cuando se accede al formulario para rellenarlo y cuando se envía para procesarlo.

Cuando se accede a la página usando el método GET, es decir, introduciendo la dirección en el navegador, al seguir un vínculo o como resultado de una redirección con `header(Location:)`, se muestra el formulario. En cambio, si se accede mediante POST quiere decir que el cliente está enviado el formulario. Se puede diferenciar entre los dos métodos de acceso consultando `$_SERVER["REQUEST_METHOD"]`.

El ejemplo **form\_en\_uno.php** une los dos ficheros del anterior en uno solo y, en caso de error:

- Muestra un mensaje apropiado encima del formulario.
- Mantiene el valor introducido en el campo usuario.

```
<?php
/* si va bien redirige a principal.php si va mal, mensaje de error */
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if($_POST['usuario']=="usuario" and $_POST["clave"]=="1234"){
        header("Location: principal.php");
    }else{
        $err = true;
    }
}
?>
<!DOCTYPE html>
<html>
```

```

<head>
    <title>Formulario de login</title>
    <meta charset = "UTF-8">
</head>
<body>
    <?php if(isset($err)) {
        echo "<p> Revise usuario y contraseña</p>";
    }?>
    <form method = "POST"
        action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
        <label for = "usuario">Usuario</label>
        <input value = "<?php if(isset($usuario))echo $usuario;?>">
        id = "usuario" name = "usuario" type = "text">
        <label for = "clave">Clave</label>
        <input id = "clave" name = "clave" type = "password">
        <input type = "submit">
    </form>
</body>
</html>

```

**TOMA NOTA**

Cuando el formulario llama al mismo fichero se recomienda usar:

`action = "<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"`

en lugar del nombre del fichero como aparece en el ejemplo. La variable *superglobal* `$_SERVER["PHP_SELF"]` contiene el nombre del fichero y la función `htmlspecialchars()` sirve para filtrar los caracteres por seguridad.

El bloque inicial de PHP con la comprobación de usuario y contraseña se ejecuta solo cuando se accede por el método POST, es decir, al enviar el formulario.

En ese caso, se comprueban los datos y, si son correctos, se reenvía a la página de bienvenida. Con el reenvío termina la ejecución de **form\_en\_uno.php**. Si no son correctos, se crea la variable `$err` con valor TRUE y el script continúa. El bloque de HTML que contiene el formulario se mostrará tanto si el método es GET como si es POST y la comprobación de datos falló.

Hay un segundo bloque de PHP antes del formulario para imprimir un mensaje de error si la variable `$err` existe, es decir, si se ha enviado el formulario con datos incorrectos. Si se accede por GET el bloque se ejecuta, pero la condición no se cumple y no se muestra el mensaje.

### 3.2.3. Subida de ficheros

Un caso especial en los formularios es la subida de ficheros al servidor. En el formulario hay que usar el atributo `enctype="multipart/form-data"` y el método POST. Para el fichero se usa una etiqueta `<input type = "file">`. Con este control se abre una ventana para que el usuario pueda escoger un fichero en su equipo.

```
<form action="procesar_subida.php" method="post"
      enctype = "multipart/form-data">
    Escoja un fichero
    <input type="file" name="fichero">
    <input type="submit" value="Subir fichero">
</form>
```

En el *script* que recibe el formulario, la variable global `$_FILES` contiene información sobre el fichero que se está subiendo. Se trata de un *array* bidimensional. La primera dimensión identifica el fichero según el atributo `name` en el formulario, en la segunda las claves son:

- `name`: el nombre del fichero en el cliente.
- `size`: el tamaño del fichero en *bytes*.
- `type`: el tipo MIME del fichero.
- `tmp_name`: el nombre temporal con el que se ha subido al servidor.
- `error`: código de error asociado a la subida.

El fichero se almacena en principio en el directorio temporal del servidor y se puede mover al directorio que se desee con la función:

```
bool move_uploaded_file ( $fichero, $destino )
```

Si el fichero no se mueve del directorio temporal, el servidor se encargará de eliminarlo.

El ejemplo **procesar\_subida.php** se encarga de comprobar que el fichero no pase cierto límite de tamaño. Si se cumple la condición, se mueve al directorio “subidos” (hay que crear el directorio para que el ejercicio funcione).

```
<?php
$tam = $_FILES["fichero"]["size"];
if($tam > 256 *1024){
    echo "<br>Demasiado grande";
    return;
}
echo "Nombre del fichero: ". $_FILES["fichero"]["name"];
echo "<br>Nombre temporal del fichero en el servidor: ".
    $_FILES["fichero"]["tmp_name"];
$res = move_uploaded_file($_FILES["fichero"]["tmp_name"],
    "subidos/".$_FILES["fichero"]["name"]);
if($res){
    echo "<br>Fichero guardado";
} else {
    echo "<br>Error";
}
```

El formulario de envío sería el siguiente:

```
<!DOCTYPE html>
<html>
    <body>
        <form action="procesar_subida.php" method="post"
        enctype="multipart/form-data">
            Escoja un fichero
            <input type="file" name="fichero">
            <input type="submit" value="Subir fichero">
        </form>
    </body>
</html>
```

### 3.3. Cookies

Las *cookies* son pequeños ficheros que dejan los servidores web en los ordenadores de los clientes. Pueden almacenar información sobre la fecha de la última visita o preferencias de idioma, por ejemplo. Cuando un cliente realiza una petición web, envía al servidor las *cookies* que pudiera tener de este.

Para manejar las *cookies* se usa la función `setcookie()`, que tiene la siguiente cabecera:

```
bool setcookie (string $name [, string $value = "" [, int $expire = 0 [, string $path = "" [, string
    $domain = "" [, bool $secure = FALSE [, bool $httponly = FALSE ]]]]]])
```

Los tres primeros argumentos son los más importantes:

1. El primer argumento es el nombre de la *cookie*.
2. El segundo, el valor que se le quiere dar.
3. El tercero es la fecha en la que expira la *cookie*. Se especifica como una fecha Unix, es decir, el número de segundos pasados desde el comienzo de 1970. Normalmente se utiliza la función `time()`, que devuelve la fecha actual y se le suma un periodo de tiempo expresado en segundos.

Por ejemplo, para crear la *cookie* “visitas” con valor “1” y duración de un día se utiliza:

```
setcookie('visitas', '1', time() + 3600 * 24);
```

Para la fecha de caducidad se toma la actual, que es el valor que devuelve `time()` y se le suma `3.600 * 24`, el número de segundo que hay en 24 horas.

Para destruir una *cookie* se usa la función `setcookie()` con una fecha límite anterior a la actual:

```
setcookie('visitas', '1', time() - 3600 * 24);
```

#### RECUERDA

- ✓ Las *cookies* se envían como cabeceras de las peticiones HTTP. Hay que enviar las cabeceras antes de empezar con el cuerpo de la respuesta. Esto implica que hay que utilizar la función `setcookie()` antes de que se empiece a escribir la salida. Si se intenta llamar a `setcookie()` después de haber realizado un `echo`, se producirá un error.

Las *cookies* que envía el cliente están disponibles en `$_COOKIES`, tomando como clave el nombre que se les dio con `setcookie()`. Es importante señalar que, después de crear una *cookie*, esta no está disponible en `$_COOKIES` hasta la siguiente petición del cliente. El siguiente ejemplo daría error:

```
<?php
setcookie('nueva', "valor", time() + 3600 * 24);
echo $_COOKIES["nueva"];
```

El ejemplo **contador\_visitas.php** utiliza una *cookie* para almacenar el número de veces que un usuario ha visitado la página. Si la *cookie* no existe, la crea con valor “1”. Si ya existe, lee su valor para mostrar el mensaje y reescribe la *cookie* sumando 1 al valor. Como el valor es una cadena, hay que transformarlo a entero.

```
<?php
if (!isset($_COOKIE['visitas'])) { // si no existe
    setcookie('visitas', '1', time() + 3600 * 24);
    echo "Bienvenido por primera vez";
} else { // si existe
    $visitas = (int) $_COOKIE['visitas'];
    $visitas++; // se reescribe incrementada
    setcookie('visitas', $visitas, time() + 3600 * 24);
    echo "Bienvenido por $visitas vez";
}
```

La primera vez que se acceda al *script* se obtendrá la salida “Bienvenido por primera vez”. La siguiente, “Bienvenido por 2 vez”, y así sucesivamente.

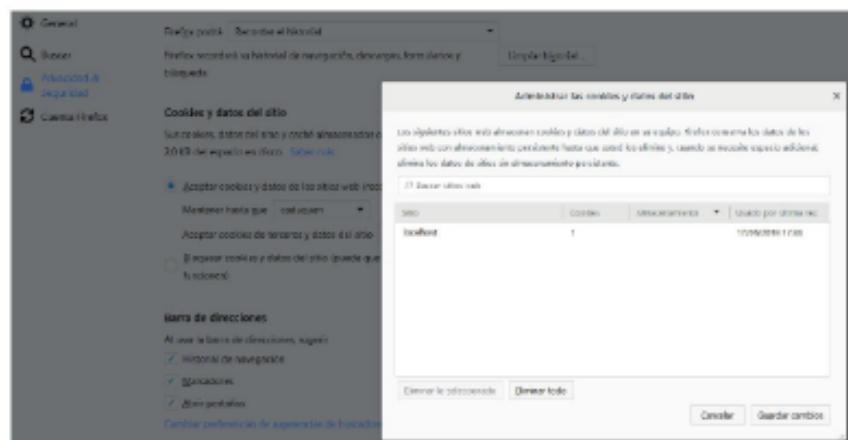
### Actividad propuesta 3.2



Añade un vínculo para borrar la *cookie* al ejemplo anterior.

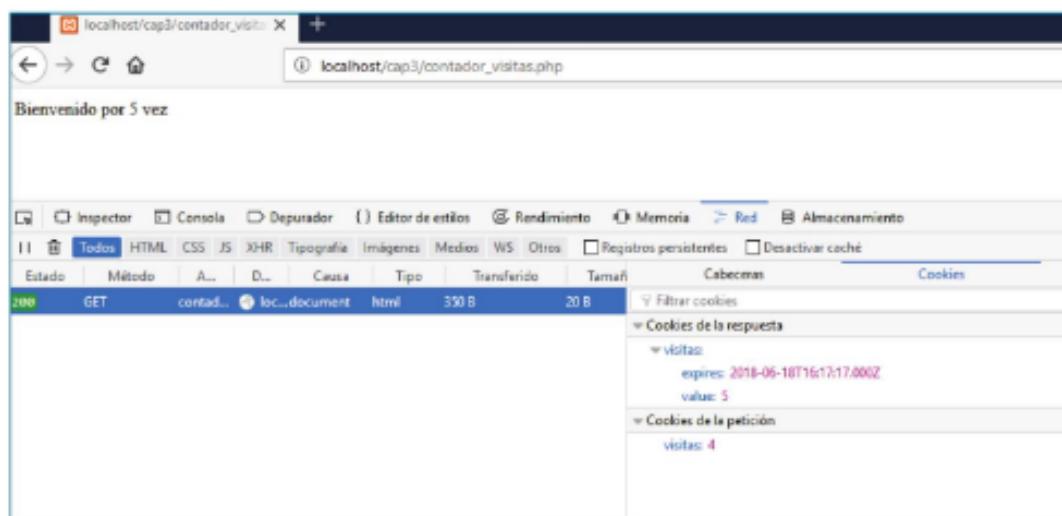
Los navegadores ofrecen la posibilidad de ver las *cookies* almacenadas en el ordenador. En Firefox se accede a ellas mediante el menú Opciones, en la sección Privacidad y Seguridad. Hay un apartado

**Figura 3.2**  
Administrar *cookies* en Firefox.



llamado “Cookies y datos del sitio” con un botón “Administrar datos”. Al pulsarlo se muestra la lista de *cookies* agrupadas por el servidor de origen. Después de acceder al ejemplo **contador\_visitas.php**, habrá una *cookie* del servidor *localhost*. También es posible eliminar las *cookies*, o borrar el historial de navegación.

Desde las herramientas de desarrollador se pueden ver las *cookies* que envía el cliente y la que crea el servidor con el valor incrementado en uno:



**Figura 3.3**  
Cookies desde las herramientas de desarrollador.



### Actividad propuesta 3.3

Crea una página web con un formulario para elegir el idioma en el que se muestra, inglés o español. Almacena la elección del usuario con una *cookie* para que la siguiente vez que el usuario se conecte la página aparezca directamente en su idioma. Si la *cookie* no existe, la página se mostrará en español.

## 3.4. Sesiones. Seguridad: usuarios y roles

Como HTTP es un protocolo sin estado, las diferentes peticiones de un cliente al servidor son independientes, no están relacionadas entre sí. Para asociarlas, se utilizan las sesiones.

Al iniciar una sesión el servidor asigna y envía al usuario un identificador de sesión. En las siguientes peticiones el usuario envía al servidor ese identificador, de manera que el servidor sabe que se trata del mismo usuario.

Para controlar la sesión el servidor deja una *cookie* con el id de sesión en el cliente, que se elimina al cerrarla. Si se borran manualmente las *cookies* del navegador, se cierran las sesiones abiertas. Se puede configurar el servidor para controlar las sesiones sin *cookies*, pero no es lo habitual.

Para crear una sesión se utiliza la función `start_session()`. Si no hay una sesión activa la crea, si la hay, el script que llama a la función se une a ella. Una vez creada la sesión es posible

utilizar la variable superglobal `$_SESSION` para compartir información entre los *scripts* que comparten sesión. Es un *array* que almacena las variables de sesión definidas por el usuario, basta con añadir elementos de la manera usual.

```
$_SESSION["nombre"] = valor;
```

El ejemplo **sesiones\_uso\_basico.php** es muy sencillo.

```
<?php
session_start();
if (!isset($_SESSION['count'])) {
    $_SESSION['count'] = 0;
} else {
    $_SESSION['count']++;
}
echo "hola ".$_SESSION['count'];
echo "<br><a href='sesiones_uso_basico2.php'>Siguiente</a>";
```

Crea la sesión y, si no existe ya, una variable de sesión `$_SESSION["count"]` con valor 0. Si existe, le suma 1. Muestra un vínculo a **sesiones\_uso\_basico2.php**. Este segundo fichero se une a la sesión y muestra el valor de la variable. Accediendo a ambos se puede comprobar que realmente se trata de la misma variable.

```
<?php
session_start();
echo "La variable count vale: ".$_SESSION['count'];
```

En las aplicaciones web es habitual que se cree una sesión al hacer *login*. El ejemplo **sesiones1\_login.php** amplía el del apartado 3.2.1. Si el *login* es correcto, se crea una sesión, una variable de sesión para el nombre de usuario y redirige **sesiones1\_principal.php**.

Como en el ejemplo anterior, se usa una función para simular la base de datos. Admite dos usuarios, “usuario” y “admin”, ambos con clave “1234”. Si los datos son correctos, devuelve un *array* con el nombre y el rol del usuario, si no devuelve FALSE.

```
<?php
/*formulario de login
habitual
si va bien abre sesión, guarda el nombre de usuario y redirige a principal.
php
si va mal, mensaje de error */
function comprobar_usuario($nombre, $clave){
if($nombre === "usuario" and $clave === "1234"){


```



**Figura 3.4**  
Diagrama para el ejemplo de sesiones.

```

$usu['nombre'] = "usuario";
$usu['rol'] = 0;
return $usu;
}elseif($nombre === "admin" and $clave === "1234"){
    $usu['nombre'] = "admin";
    $usu['rol'] = 1;
    return $usu;
}else return FALSE;
}
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $usu = comprobar_usuario($_POST['usuario'], $_POST['clave']);
    if($usu==FALSE){
        $err = TRUE;
        $usuario = $_POST['usuario'];
    }else{
        session_start();
        $_SESSION['usuario'] = $_POST['usuario'];
        header("Location: sesiones1_principal.php");
    }
}
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Formulario de login</title>
        <meta charset = "UTF-8">
    </head>
    <body>
        <?php if(isset($_GET["redirigido"])){echo "<p>Haga login para continuar</p>";}
        ?>
        <?php if(isset($err) and $err == true){echo "<p> revise usuario y contraseña</p>";}
        ?>
        <form method = "POST" action = "<?php echo htmlspecialchars($_
        SERVER["PHP_SELF"]);?>" >
            Usuario
            <input value = "<?php if(isset($usuario))echo $usuario;?>" id = "usuario" name = "usuario" type = "text">
            Clave
            <input id="clave" name = "clave" type = "password">
            <input type = "submit">
        </form>
    </body>
</html>

```

Lo primero que hace **sesiones1\_principal.php** es unirse a la sesión. Todos los ficheros que quieran acceder a las variables de sesión tienen que llamar a `session_start()`. Después,

comprueba que realmente se haya hecho *login* viendo si la variable `$_SESSION['usuario']` está definida. Si no lo está redirige al formulario de *login*. De esta manera se bloquea el acceso a **sesiones1\_principal.php** si no se ha hecho *login* antes. Si la variable está definida, muestra un mensaje personalizado.

```
<?php
    session_start();
    if(!isset($_SESSION['usuario'])){
        header("Location:sesiones1_login.php?redirigido=true");
    }
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Página principal</title>
        <meta charset = "UTF-8">
    </head>
    <body>
        <?php echo "Bienvenido ".$_SESSION['usuario'];?>
        <br><a href = "sesiones1_logout.php"> Salir <a>
    </body>
</html>
```

Para cerrar la sesión se utiliza la función `session_destroy()`. El vínculo “Salir” lleva a **sesiones1\_logout.php**, que cierra la sesión y redirige a la página de *login*.

```
<?php
    session_start(); // unirse a la sesión
    $_SESSION = array();

    session_destroy(); // eliminar la sesión
    // eliminar la cookie

    setcookie(session_name(), 123, time() - 1000);
    header("Location: sesiones1_login.php");
```

Para cerrar la sesión no basta con llamar a `session_destroy()`. También hay que borrar las variables de sesión y la *cookie*.

### 3.5. Envío de correo electrónico

Aunque la función `mail()` permite enviar correos directamente, es habitual usar alguna librería que se ocupe de los detalles del formato. Una de las más habituales es `phpMailer`. Se puede instalar utilizando `composer`:

```
composer require phpmailer/phpmailer
```

En principio es posible enviar un correo utilizando la configuración de `sendmail` (en Linux) o un servidor SMTP local, pero en la práctica los filtros *antispam* hacen que no lleguen los correos enviados desde servidores que no estén registrados correctamente.

Si no se dispone de un servidor de correo en Internet, la opción más cómoda para enviar un correo electrónico es utilizar una cuenta de Gmail o similar. En Gmail hay que activar la opción “Permitir aplicaciones menos seguras” en la sección de ajustes de la cuenta.

## Recurso web

www

Para permitir las aplicaciones menos seguras, consulta:

<https://support.google.com/accounts/answer/6010255?hl=es>

En **ejemplo\_correo.php** se muestra cómo enviar un correo a través del servidor de Google.

- Hay que introducir usuario y clave de Google (líneas 13 y 18).
- Los destinatarios se añaden con `AddAddress()` (líneas 24-25)

```
1 <?php
2 use PHPMailer\PHPMailer\PHPMailer;
3 require "vendor/autoload.php";
4 $mail = new PHPMailer();
5 $mail->IsSMTP();
6 // cambiar a 0 para no ver mensajes de error
7 $mail->SMTPDebug = 2;
8 $mail->SMTPAuth = true;
9 $mail->SMTPSecure = "tls";
10 $mail->Host = "smtp.gmail.com";
11 $mail->Port = 587;
12 // introducir usuario de google
13 $mail->Username = "";
14 // introducir clave
15 $mail->Password = "";
16 $mail->SetFrom('user@gmail.com', 'Test');
17 // asunto
18 $mail->Subject = "Correo de prueba";
19 // cuerpo
20 $mail->MsgHTML('Prueba');
21 // adjuntos
22 $mail->addAttachment("empleado.xls");
23 // destinatario
24 $address = "destino@servidor.com";
25 $mail->AddAddress($address, "Test");
26 // enviar
27 $resul = $mail->Send();
28 if(!$resul) {
29     echo "Error". $mail->ErrorInfo;
```

```

30 } else {
31     echo "Enviado";
32 }

```

## 3.6. Bases de datos relacionales

En PHP hay *drivers* para manejar los sistemas gestores de bases de datos más extendidos. También hay varias extensiones que proveen una capa de abstracción sobre la base de datos, entre las que destaca PHP Data Objects (PDO). Permite manejar diferentes bases de datos con una interfaz común. Tiene la ventaja de que si se cambia de base de datos no hay que modificar el código.

### 3.6.1. Conexión a la base de datos

El primer paso para trabajar con una base datos es obtener una conexión a la misma. Para representar la conexión se usa un objeto de clase PDO. El constructor es:

```
public PDO::__construct ( string $dsn [, string $username [, string $passwd [, array $options ]]] )
```

El primer parámetro es una cadena que especifica qué *driver* hay que usar, la localización y el nombre de la base de datos. Para una base de datos MySQL se usaría:

```
mysql:dbname=<base de datos>;host=<ip o nombre>
```

Los siguientes parámetros son el nombre de usuario y clave para acceder a la base de datos. El último parámetro, opcional, es un *array* de opciones.

Si se puede establecer la conexión, se usará el nuevo objeto PDO para manejar la base de datos. Si no se puede conectar con la base de datos, el constructor lanza una excepción PDOException.

```

<?php
$cadenaConexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    $bd = new PDO($cadenaConexion, $usuario, $clave);
    $bd->close();
} catch (PDOException $e) {
    echo 'Error con la base de datos: ' . $e->getMessage();
}

```

En principio, al acabar el *script* se cierra la conexión a la base de datos, pero también es posible usar conexiones persistentes, que no se cierran automáticamente al terminar el *script*. Quedan abiertas y si se vuelven a utilizar no es necesario reestablecer la conexión, por lo que pueden ser más eficientes en determinadas ocasiones. Para crear una conexión persistente se utiliza la opción PDO::ATTR\_PERSISTENT en el constructor:

```
$bd = new PDO($cadena_conexion, $usuario, $clave, array(PDO::ATTR_PERSISTENT => true));
```

## TOMA NOTA



Los ejemplos de esta sección utilizan la base datos empresa, que contiene la tabla: Usuarios(Código, Nombre, Clave, Rol)

- Código es la clave primaria, un campo autonumérico.
- Nombre es el nombre de usuario y está marcado como *unique*.
- Clave es una cadena con la clave de acceso al sistema.
- Rol es un entero que representa el rol del usuario dentro del sistema.

### 3.6.2. Recuperación y presentación de datos

El método `query($cad)` de la clase PDO ejecuta la cadena que recibe como argumento en la base de datos, como se haría desde la línea de comando SQL. El argumento `$cad` tiene que ser una instrucción SQL válida. Devuelve FALSE si hubo algún error o un objeto PDOStatement si la cadena se ejecutó con éxito.

Si se trata de una consulta, es posible recorrer las filas devueltas con un `foreach`. En cada iteración del bucle se tendrá una fila, representada como un *array* en que las claves son los nombres que aparecen en la cláusula `select`.

```
<?php
$cadenaConexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    $bd = new PDO($cadenaConexion, $usuario, $clave);
    echo "Conexión realizada con éxito";
    $sql = 'SELECT nombre, clave, rol FROM usuarios';
    $usuarios = $bd->query($sql);
    echo $usuarios->rowCount(). "<br>";
    foreach ($usuarios as $row) {
        print $row['nombre']. "\t";
        print $row['clave']. "\t";
    }
}
```



## Actividad propuesta 3.4

Escribe un fichero que reciba el código de un usuario y muestre por pantalla todos sus datos.

También es posible obtener instrucciones preparadas que permiten utilizar parámetros. Se inicializan una sola vez con el método `prepare()` y luego se ejecutan las veces que sea necesario con `execute()`, con diferentes valores para los parámetros. Las instrucciones preparadas permiten reutilizar las consultas, previenen la inyección de código y mejoran el rendimiento.

Hay dos opciones para indicar los parámetros de la consulta, por posición y por nombre. En el primer caso se utiliza el símbolo de interrogación para indicar un parámetro. Al ejecutarla, se asocian por orden los símbolos de interrogación con los valores del `array` que se pasa como argumento a `execute()`.

```
$preparada = $bd->prepare("select nombre from usuarios where
    rol = ?");
$preparada->execute( array(0));
echo "Usuarios con rol 0: ". $preparada->rowCount(). "<br>";
foreach ($preparada as $usu) {
    print "Nombre: ". $usu['nombre']. "<br>";
}
```

Si se utilizan nombres en la instrucción preparada, utilizando: `nombre`, se deberán usar esos nombres como claves del `array` de `execute()`.

```
$preparada_nombre=$bd->prepare("select nombre from usuarios where rol
    =:rol");
$preparada_nombre->execute(array(':rol' => 0));
echo "Usuarios con rol 0: ". $preparada->rowCount(). "<br>";
foreach ($preparada_nombre as $usu) {
    print "Nombre: ". $usu['nombre']. "<br>";
}
```

### Actividad propuesta 3.5



Modifica el formulario de `login` para que compruebe usuario y contraseña usando la tabla `usuarios` de la base de datos `empresa`.

### 3.6.3. Inserción, borrado y actualización

Para insertar, borrar o actualizar simplemente hay que ejecutar la sentencia SQL correspondiente. Puede ser una sentencia preparada o no.

```
<?php
// datos conexión
$cadenaConexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    // conectar
    $bd = new PDO($cadenaConexion, $usuario, $clave);
```

```

echo "Conexión realizada con éxito<br>";
// insertar nuevo usuario
$ins = "insert into usuarios(nombre, clave, rol)
        values('Alberto', '33333', '1');";
$resul = $bd->query($ins);
//comprobar errores
if($resul) {
    echo "insert correcto <br>";
    echo "Filas insertadas: ". $resul->rowCount(). "<br>";
} else print_r( $bd -> errorinfo());
// para los autoincrementos
echo "Código de la fila insertada". $bd->lastInsertId()."<br>";
// actualizar
$upd = "update usuarios set rol = 0 where rol = 1";
$resul = $bd->query($upd);
//comprobar errores
if($resul){
    echo "update correcto <br>";
    echo "Filas actualizadas: ". $resul->rowCount(). "<br>";
} else print_r($bd -> errorinfo());
// borrar
$del = "delete from usuarios where nombre = 'Luisa'";
$resul = $bd->query($del);
//comprobar errores
if($resul){
    echo "delete correcto <br>";
    echo "Filas borradas: ". $resul->rowCount(). "<br>";
} else print_r($bd -> errorinfo());
} catch (PDOException $e) {
    echo 'Error con la base de datos: '. $e->getMessage();
}

```

Para que el ejemplo funcione, tiene que estar activada la opción de *autocommit* en la base de datos.

### 3.6.4. Transacciones

Una transacción consiste en un conjunto de operaciones que deben realizarse de forma atómica. Es decir, o se realizan todas o no se realiza ninguna. Por ejemplo, una transferencia de saldo entre dos clientes implica dos operaciones:

- Quitar saldo al cliente que envía la transferencia.
- Aumentar el saldo del cliente que recibe la transferencia.

Si por cualquier motivo la segunda operación falla, hay que deshacer la primera operación para que el sistema quede en un estado consistente. Las dos operaciones forman una única transacción.

Para indicar el comienzo de una transacción se utiliza el método `beginTransaction()`. La transacción se salva usando el método `commit()`. Con el método `rollBack()` se deshacen las operaciones realizadas desde que se inició la transacción.

En el ejemplo **transaccion.php** se realizan dos inserciones como una transacción. Como la segunda falla (tiene un valor repetido para un campo *unique*), la primera se deshace.

```

<?php
$cadenaConexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    $bd = new PDO($cadenaConexion, $usuario, $clave);
    echo "Conexión realizada con éxito<br>";
    // comenzar la transacción
    $bd->beginTransaction();
    $ins = "insert into usuarios (nombre, clave, rol)
            values('Fernando', '33333', '1')";
    $resul = $bd->query($ins);
    // se repite la consulta
    // falla porque el nombre es unique
    $resul = $bd->query($ins);
    if(!$resul){
        echo "Error: ". print_r($bd->errorinfo());
        // deshace el primer cambio
        $bd->rollback();
        echo "<br>Transacción anulada<br>";
    }else{
        // si hubiera ido bien
        $bd->commit();
    }
} catch (PDOException $e) {
    echo 'Error al conectar: '. $e->getMessage();
}

```

### 3.7. Bases de datos no relacionales

Las bases de datos no relacionales o no-SQL están cada vez más extendidas. Como su propio nombre indica, no siguen el modelo relacional. En las bases de datos no relacionales los modelos de datos son, en general, más flexibles y es más fácil realizar cambios en el esquema a lo largo del proceso de desarrollo.

En el caso de MongoDB, el elemento básico de almacenamiento es el documento. Los documentos se agrupan dentro de colecciones. Una base de datos cuenta con una o más colecciones.

Los documentos en MongoDB siguen un formato similar a JSON (incluye algunos tipos de datos adicionales) llamado BSON, Binary JSON. Por ejemplo,

```
{ nombre: "Pedro", edad: 22 }
```

Buscando una analogía con el modelo relacional, los documentos serían filas y las colecciones, tablas. Pero, al contrario que en el modelo relacional, no todos los documentos dentro de una colección tienen que tener la misma estructura. Dentro de una tabla, todas las filas tienen que tener las mismas columnas, aunque algunos valores puedan ser nulos. Las colecciones de MongoDB no imponen restricciones de ese tipo.

#### 3.7.1. Instalación y puesta marcha de MongoDB

Hay que instalar:

- MongoDB Community Server, la versión libre de MongoDB.

- MongoDB Compass, una aplicación para manejar la base de datos con interfaz gráfica.
- El *driver* de PHP para MongoDB.

**Recursos web**

Para descargar MongoDB Community Server:  
<https://docs.mongodb.com/manual/administration/install-community/>

Para MongoDB Compass:  
<https://www.mongodb.com/products/compass>

Para empezar, hay que instalar el servidor MongoDB. En la configuración de la instalación, por defecto:

- El servidor escucha en el puerto 27017.
- La ruta de instalación en Windows es C:\Program Files\MongoDB.
- No es necesario utilizar usuario y contraseña para conectarse.

Una vez instalado, el servidor se pone en marcha ejecutando mongod.exe, que se encuentra en el subdirectorio bin dentro de la ruta de instalación del programa (en Windows, la ruta por defecto es C:\Program Files\MongoDB\Server\3.6\bin).

A continuación, se instala MongoDB Compass, que ofrece una interfaz gráfica para manejar el servidor. Al arrancar la aplicación, lo primero que hay que hacer es conectarse al servidor. Si se ha seguido la configuración por defecto, no hace falta cambiar nada, solo pulsar el botón Connect, en la esquina inferior derecha.

Al conectar, la aplicación muestra las bases de datos del servidor (llamado MyCluster). Hay tres bases de datos ya creadas: admin, config y local.

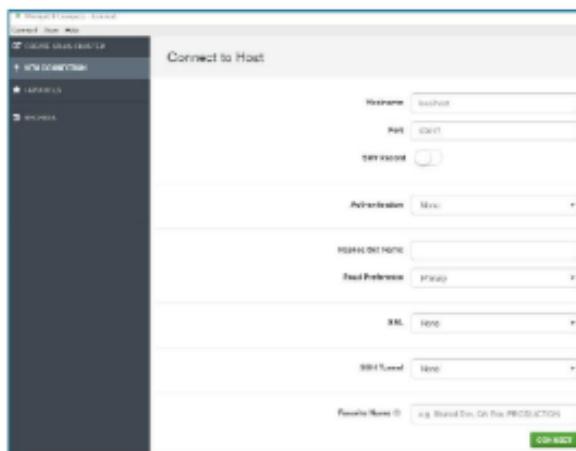


Figura 3.5  
Conexión a MongoDB.

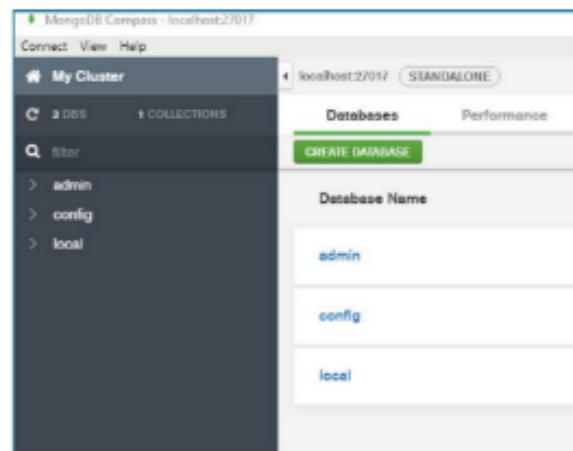
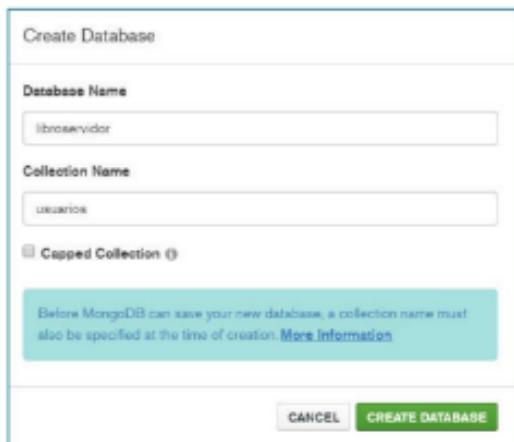


Figura 3.6  
Bases de datos predefinidas.

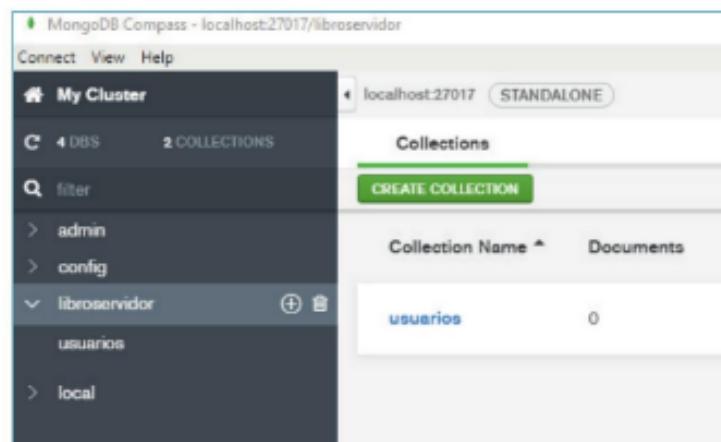
Para crear una nueva se usa el botón “CREATE DATABASE”. Al pulsarlo, se abre una ventana que pide el nombre de la nueva base de datos y también el de una colección. Es obligatorio

introducir ambos campos. Para poder utilizar los ejemplos posteriores hay que crear una base de datos llamada *libroservidor* y una colección llamada *usuarios*.

Al volver a la pantalla principal aparecerá la nueva base de datos, y si se pulsa sobre su nombre se verá también la colección.



**Figura 3.7**  
Creación de una nueva base de datos.



**Figura 3.8**  
Colección creada.

Finalmente, para poder conectarse a un servidor MongoDB desde PHP hay que instalar el *driver* correspondiente, lo que se hace utilizando **composer**. Se debe ejecutar el comando:

```
composer require mongodb/mongodb
```

### 3.7.2. Conexión desde PHP

Con la librería de MongoDB para PHP es muy fácil realizar las operaciones habituales. El siguiente ejemplo inserta documentos en la colección creada en el apartado anterior.

Primero se conecta al servidor usando la clase `MongoDB\Client()` y se selecciona la base de datos `libroservidor`. Luego se insertan documentos en `usuarios` utilizando los métodos `insertOne()` e `insertMany()`.

```
<?php
require 'vendor/autoload.php';
$cliente = new MongoDB\Client("mongodb://localhost:27017");
$bd = $cliente->libroservidor;
try {
    $res = $bd->usuarios->insertOne( [ 'nombre' => 'Ana', 'clave' =>
        '1234', 'saldo' => 1000 ] );
    echo "Id del último registro: ". $res->getInsertedId(). "<br>";
    $res = $bd->usuarios->insertMany( [
        [ 'nombre' => 'Paco', 'clave' => '1234', 'saldo' => 100],
        [ 'nombre' => 'Nuria', 'clave' => '1234', 'saldo' => 30],
    ] );
    echo "Documentos insertados: ". $res->getInsertedCount(). "<br>";
    print_r($res->getInsertedIds());
}
```

```

} catch (Exception $e) {
    print ($e);
}

```

Para consultar se usan los métodos `find()` y `findOne()`. El segundo devuelve solo un resultado, se suele utilizar cuando se sabe que solo habrá uno, como al buscar por clave primaria. Admiten criterios de búsqueda utilizando un *array*, como se puede ver a continuación.

```

<?php
require 'vendor/autoload.php';
try {
    $cliente = new MongoDB\Client("mongodb://localhost:27017");
    $bd = $cliente->libroservidor;
    // devuelve todos los usuarios
    echo "Todos los usuarios". "<br>";
    $usuarios = $bd->usuarios->find();
    foreach($usuarios as $usuario){
        var_dump($usuario);
    }
    // usuarios con nombre Ana
    echo "Usuarias con nombre 'Ana'". "<br>";
    $usuarios = $bd->usuarios->find([ 'nombre' => 'Ana' ]);
    foreach($usuarios as $usuario){
        var_dump($usuario);
    }
    // solo devuelve el primero que encuentre
    echo "Usuaria con nombre 'Ana'". "<br>";
    $ana = $bd->usuarios->findOne([ 'nombre' => 'Ana' ]);
    var_dump($ana);
}catch (Exception $e) {
    print ($e);
}

```

Para actualizar se usan los métodos `update()` y `updateOne()`.

```

<?php
require 'vendor/autoload.php';
try {
    $cliente = new MongoDB\Client("mongodb://localhost:27017");
    $bd = $cliente->libroservidor;
    /* pone a 7000 el saldo del usuario con nombre 'Ana'*/
    $updateResult = $bd->usuarios->updateOne(
        [ 'nombre' => 'Ana' ],
        [ '$set' => [ 'saldo' => '7000' ] ]
    );
}catch (Exception $e) {
    print ($e);
}

```

Para borrar se usan los métodos `delete()` y `deleteOne()`.

```
<?php
require 'vendor/autoload.php';
try {
    $cliente = new MongoDB\Client("mongodb://localhost:27017");
    $bd = $cliente->libroservidor;
    /* pone a 7000 el saldo del usuario con nombre 'Ana'*/
    $updateResult = $bd->usuarios->deleteOne(
        [ 'nombre' => 'Paco' ]);
    echo "Documentos restantes después de borrar: ".$bd->
    usuarios->count();
} catch (Exception $e) {
    print ($e);
}
```

### 3.8. Ficheros

En PHP hay varias librerías para manejo de ficheros. La más importante es `Filesystem`, que incluye funciones para leer y escribir ficheros y para manejar el sistema de archivos. Son funciones muy similares a las de manejo de ficheros en C.

Para abrir un fichero se usa la función `fopen()`, indicando la ruta del fichero y el modo en el que se abre. Si hay algún error, la función devuelve FALSE; en otro caso devuelve un puntero para manejar el fichero. En el siguiente ejemplo se muestran ambos casos.

```
<?php
$fich = fopen("fichero_ejemplo.txt", "r");
if ($fich === FALSE){
    echo "No se encuentra fichero_ejemplo.txt<br>";
} else{
    echo "fichero_ejemplo.txt se abrió con éxito<br>";
}
$fich = fopen("fichero_no_existe.txt", "r");
if ($fich === FALSE){
    echo "No se encuentra fichero_no_existe.txt<br>";
} else{
    echo " fichero_no_existe.txt se abrió con éxito<br>";
}
```

El primer fichero sí existe, pero el segundo no. Al ejecutarlo la salida será:

```
fichero_ejemplo.txt se abrió con éxito Warning: fopen(fichero_no_existe.txt): failed to open stream: No such file or directory in C:\xampp\htdocs\cap3\ficheros_fopen.php on line No se encuentra fichero_no_existe.txt
```

El mensaje de Warning lo pone el propio servidor y, en principio, no se mostraría en producción. Los modos de apertura más importantes se recogen en el cuadro 3.1.

**CUADRO 3.1**  
**Modos de apertura de ficheros**

Modo	Descripción
r	Solo lectura. Si el fichero no existe, devuelve FALSE
r+	Lectura y escritura. Si el fichero no existe, devuelve FALSE
w	Solo escritura. Si el fichero no existe, se crea; si existe, se trunca (es decir, se borra el contenido anterior). Si no puede crearlo, devuelve FALSE
w+	Lectura y escritura. Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE
a	Solo escritura. Las escrituras se realizan siempre al final de fichero (append). Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE
a+	Lectura y escritura. Las escrituras se realizan siempre al final de fichero. Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE

Lectura y escritura se realizan a partir del indicador de posición del fichero. Cuando se abre un fichero, el indicador se sitúa al inicio de este, salvo si se ha abierto en modo “a” o “a+”, que se sitúa al final. Cuando se lee o escribe, el puntero avanza para situarse al final del último carácter o *byte* leído o escrito.

En el siguiente ejemplo, se abre un fichero en modo lectura y se lee carácter a carácter usando la función `fgetc()`, que devuelve el siguiente carácter a partir del indicador de posición.

```
<?php
$fich = fopen("fichero_ejemplo.txt", "r");
if ($fich === FALSE){
    echo "No se encuentra el fichero o no se pudo leer<br>";
} else{
    while( !feof($fich) ){
        $car = fgetc($fich);
        echo $car;
    }
}
fclose($fich);
```

Para detectar el final del fichero se usa la función `feof()`, que devuelve TRUE cuando ya se ha llegado al final. Cuando se termina de trabajar con el fichero, se cierra con `fclose()`.

Para leer un fichero que sigue un formato determinado se puede usar la función `fscanf()`, que lee una línea del fichero y le aplica un formato. Hay dos maneras de usarla. En la primera, se le pasan dos parámetros y devuelve un *array* con los valores leídos.

```
$valores = fscanf($fichero, $formato);
```

También se le pueden pasar variables adicionales para que almacene en ellas los valores leídos en lugar de devolverlos. En este caso devuelve el número de valores leídos correctamente.

```
$valores = fscanf($fichero, $formato, $var1, ...);
```

### Ejemplo de formato

Supongamos que tenemos un fichero de texto que representa una matriz de 4 filas y 4 columnas.

```
23 234 611 5
1233 565 123 5
123 54 757 12
77 88 9 99
```

Para indicar que cada línea está formada por 4 números separados por espacios se usa como formato:

```
"%d %d %d %d"
```

Cada %d representa un número en formato decimal, y se deja un espacio entre ellas.

El siguiente ejemplo muestra cómo recorrer un fichero con el formato anterior utilizando fscanf() de las dos maneras posibles. Antes de empezar con la segunda vuelta, sitúa el indicador de posición de nuevo al principio del fichero usando rewind().

```
<?php
$fich = fopen("matriz.txt", "r");
if ($fich === False){
    echo "No se encuentra el fichero o no se pudo leer<br>";
} else{
    while( !feof($fich) ){
        $num = fscanf($fich, "%d %d %d %d");
        echo "$num[0] $num[1] $num[2] $num[3] <br>";
    }
}
rewind($fich);
while( !feof($fich) ){
    $num = fscanf($fich, "%d %d %d %d", $num1,$num2,$num3,$num4);
    echo "$num1 $num2 $num3 $num4 <br>";
}
fclose($fich);
```

También son útiles las funciones file\_get\_contents() y file\_put\_contents(). La primera devuelve una cadena con el contenido de un fichero. La segunda hace lo contrario, escribe datos en un fichero. La ventaja que tienen es que funcionan directamente a partir de la ruta del fichero, así que no hace falta llamar a fopen() y fclose().

```
<?php
$contenido = file_get_contents("fichero_ejemplo.txt");
echo "Contenido del fichero: $contenido<br>";
$res = file_put_contents("fichero_salida.txt",
    "Fichero creado con file_put_contents");
if($res){
    echo "Fichero creado con éxito";
} else{
    echo "Error al crear el fichero";
}
```

En el cuadro 3.2 se resumen las funciones más útiles para trabajar con ficheros.

**CUADRO 3.2**  
**Funciones de ficheros**

Función	Descripción
fgets(\$fich)	Devuelve una cadena con los caracteres desde el indicador de posición
fputs(\$fich, \$cad)	Escribe una cadena en el fichero
fseek(\$fich, \$pos)	Sitúa el cursor del fichero en la posición indicada
ftell(\$fich)	Devuelve el indicador de posición del fichero
rewind(\$fich)	Sitúa el indicador de posición al principio del fichero
fopen(\$ruta, modo)	Abre un fichero
fclose(\$fich)	Cierra un fichero
fread(\$fich, \$longitud)	Lee \$longitud bytes
fwrite(\$fich, \$cadena)	Escritura una cadena en un fichero
copy(\$origen, \$destino)	Copia un fichero
unlink(\$borra)	Borra un fichero
move(\$actual, \$nuevo)	Mueve un fichero
filesize(\$ruta)	Devuelve el tamaño del fichero en bytes
Filetype(\$ruta)	Devuelve el tipo de fichero
is_file(\$ruta)	Para comprobar si la ruta corresponde a un fichero
is_dir(\$ruta)	Para comprobar si la ruta corresponde a un directorio
rename(\$actual,\$nuevo)	Cambia el nombre a un fichero

### 3.8.1. Ficheros XML

También hay librerías específicas para facilitar el trabajo con ficheros XML. La librería SimpleXML viene activada por defecto y tiene funciones para filtrar y recorrer ficheros XML.

La función `simplexml_load_file()` lee un fichero XML y devuelve un objeto de clase `SimpleXMLElement`. El fichero se manipula a través de este objeto. Por ejemplo, si se recorre como un *array*, se obtienen los hijos del elemento raíz.

```
<?php
$datos = simplexml_load_file("empleados.xml");
echo "<br>";
if($datos==false){
    echo "Error al leer el fichero";
}
foreach($datos as $valor){
    print_r($valor);
    echo "<br>";
}
```

El método `Xpath()` permite seleccionar ciertos elementos usando una expresión XPath. En el siguiente ejemplo se selecciona solo los elementos `<edad>`.

```
<?php
$datos = simplexml_load_file("empleados.xml");
$edades = $datos->xpath("//edad");
foreach($edades as $valor){
    print_r($valor);
    echo "<br>";
}
```

Validar con esquemas XSD es muy sencillo con la clase `DOMDocument`. El siguiente ejemplo valida `empleados.xml` usando el esquema `departamento.xsd`.

```
<?php
$dept = new DOMDocument();
$dept->load('empleados.xml');
$res = $dept->schemaValidate('departamento.xsd');
if ($res){
    echo "El fichero es válido";
}
else {
    echo "Fichero no válido";
}
```

También es posible utilizar transformaciones XSLT con la librería del mismo nombre. El siguiente ejemplo utiliza una transformación para mostrar un fichero XML como una tabla HTML. Tanto la transformación como el fichero XML se cargan utilizando la clase `DOMDocument`. La transformación se realiza con un objeto `XSLTProcessor`, que se asocia con la transformación usando el método `importStylesheet()`.

```
<?php
// cargar el fichero a transformar
$dept = new DOMDocument();
$dept->load('empleados.xml' );
// cargar la transformacion
$transformacion = new DOMDocument();
```

```

$transformacion->load( 'departamento.xslt' );
// crear el procesador
$procesador = new XSLTProcessor();
// asociar el procesador y la transformacion
$procesador-> importStylesheet($transformacion);
// transformar
$transformada = $procesador->transformToXml($dept);
echo $transformada;

```

Los ejemplos de este apartado usan el fichero **empleados.xml**. Representa la información de un departamento y sus empleados.

```

<?xml version="1.0" encoding="UTF-8"?>
<departamento nombre="Marketing" jefeDep="e101">
    <empleado codEmple="e101">
        <nombre>Ana</nombre>
        <apellidos>Frutos Jarama</apellidos>
        <edad>28</edad>
        <carnetConducir/>
    </empleado>
    <empleado codEmple="e102">
        <nombre>Antonio</nombre>
        <apellidos>Miguel Bustos</apellidos>
        <edad>23</edad>
    </empleado>
</departamento>

```

El esquema para validarla es este:

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="departamento">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="empleado" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="jefeDep" type="xs:string" use="required"/>
            <xs:attribute name="nombre" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="empleado">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="nombre" type="xs:string"/>
                <xs:element name="apellidos" type="xs:string"/>
                <xs:element name="edad" type="xs:string"/>
                <xs:element name="carnetConducir" minOccurs = "0"
                           type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>

```

```

<xs:attribute name="codEmple" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:schema>

```

La transformación es la siguiente:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="/">
        <html>
            <head>
                <title>Empleados del departamento</title>
            </head>
            <body>
                <table>
                    <tr>
                        <th>Nombre</th>
                        <th>Apellidos</th>
                        <th>Edad</th>
                    </tr>
                    <xsl:for-each select="departamento/empleado">
                        <tr>
                            <td>
                                <xsl:value-of select="nombre"/>
                            </td>
                            <td>
                                <xsl:value-of select="apellidos"/>
                            </td>
                            <td>
                                <xsl:value-of select="edad"/>
                            </td>
                        </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>

```

### 3.9. Pruebas

Para automatizar las pruebas de código, la herramienta más utilizada en PHP es PHPUnit, que sigue una línea similar a JUnit. Se puede instalar con *composer*:

```
composer require --dev phpunit/phpunit
```

El elemento base de PHPUnit es el caso de prueba, una clase que hereda de la clase `TestCase` de la librería. Dentro de un caso de prueba hay varias pruebas.

Para probar una clase de nombre `Clase` se creará una clase llamada `ClaseTest`. Esta clase contendrá una serie de métodos, que contendrán las pruebas que se quieren realizar. Estos métodos tienen que ser públicos. La clase también puede tener otros métodos auxiliares que no sean pruebas.

Para indicar que un método es una prueba hay dos opciones:

- Que el nombre empiece por “test”.

```
public testExcepcion(){ ... }
```

- Utilizar un bloque de comentarios específico con la anotación `@test`. Son bloques de comentario que empiezan por “`/**`”. Estos bloques se sitúan justo encima del método al que se refieren.

```
/**
 * @test
 */
public static function excepcion();
```

Dentro de estos métodos, la mayoría de las comprobaciones se realizan utilizando aserciones, métodos definidos en la clase `TestCase`, que comprueban si se cumplen o no ciertas condiciones. Si alguna de las aserciones del método no se cumple, se considera que la prueba ha fallado.

Por ejemplo, la clase `MatematicasTest` sirve para probar la clase `Matematicas`, en concreto el método factorial. El primer método, `testCero()`, se encarga de verificar si la función calcula correctamente el factorial de 0. Si la función devuelve 1, la aserción se cumple y se considerará que la prueba ha ido bien. En caso contrario, la prueba falla.

El segundo método se encarga de comprobar que, si se le pasa un argumento negativo, la función genera una excepción. Si la función no genera la excepción, la prueba falla. Se puede hacer con el método `expectException()` o con la anotación `@expectedException`.

```
<?php
require "vendor/autoload.php";
require "Matematicas.php";
use PHPUnit\Framework\TestCase;
class MatematicasTest extends TestCase{
    public function testCero(){
        $this->assertEquals(1,Matematicas::factorialEx(0));
    }
    /**
     * @test
     * @expectedException InvalidArgumentException
     */
    public static function Excepcion(){
        return Matematicas::factorialEx(-1);
    }
}
```

Para ejecutar un caso de prueba, se ejecuta PHPUnit desde la línea de comandos.

```
C:\xampp\htdocs\cap3\> \vendor\bin\phpunit MatematicasTest.php
```

La salida en este caso es:

```
2/2 100%
2 test / 2 assertions
```

Es decir, se han realizado dos pruebas (que contienen dos aserciones) y las dos han sido correctas.

La clase `TestCase` incluye muchos más métodos para hacer comprobaciones. Algunos de los más útiles están recogidos en el cuadro 3.3.

### CUADRO 3.3 Algunos métodos básicos de PHPUnit

Aserción / método	Descripción
<code>assertLessThan(\$limite, \$valor)</code>	Comprueba que \$valor sea menor que \$limite
<code>assertGreaterThanOrEqual(\$limite, \$valor)</code>	Comprueba que \$valor sea mayor que \$limite
<code>assertStringStartsWith(\$prefijo, \$cadena)</code>	Comprueba que \$cadena empiece por \$prefijo
<code>assertArrayHasKey(\$clave, \$array)</code>	Comprueba que \$array tenga un elemento con clave \$clave
<code>assertInstanceOf(Clase, \$objeto)</code>	Comprueba que \$objeto sea una instancia de Clase
<code>assertFileExists(\$ruta)</code>	Comprueba que la ruta exista
<code>assertNull(\$var)</code>	Comprueba que \$var sea <i>null</i>
<code>assertCount(\$num, \$conjunto)</code>	Comprueba que \$conjunto tenga \$num elementos
<code>expectException(Clase)</code>	Comprueba que el método genere una excepción de clase Clase

## 3.10. Depuración de errores

Aunque PHP incluye un depurador a partir de la versión 5.6, lo habitual es usar un depurador externo integrado en un IDE. Los depuradores más extendidos son ZendDebugger y Xdebug. Ambos están disponibles sin instalaciones adicionales en Eclipse PDT (ver capítulo 1).

El proceso de depuración es similar al de otros lenguajes:

- Para depurar un *script*, hay que establecer al menos un punto de ruptura (*breakpoint*).
- Al comenzar la depuración, el *script* se ejecutará hasta el primero. A partir de ahí, es posible ejecutar la siguiente línea de código pulsando F6.
- Si se trata de una función, pulsando F5 se entra en el código de esta.

- A la derecha se muestran las variables del script; sus valores se van actualizando al ejecutar más instrucciones.

**Figura 3.9**  
Depurador de Eclipse PDT.

Name	Value
<class>	Matematicas
\$num	0
\$_COOKIE	Array [0]
\$_ENV	Array [0]
\$_FILES	Array [0]
\$_GET	Array [0]
\$_POST	Array [0]
\$_REQUEST	Array [0]
\$_SERVER	Array [51]
\$_GLOBALS	Array [12]

## Resumen

- Los métodos HTTP más habituales son GET y POST. Ambos permiten enviar parámetros.
- GET es el método que se usa para solicitar una página al servidor. Los parámetros enviados por GET se añaden a la URL y están disponibles en `$_GET`.
- En los formularios es habitual usar POST para enviar los datos. El script que reciba el formulario podrá acceder a ellos en el array `$_POST`.
- Las claves de `$_POST` coinciden con los atributos *name* de los controles del formulario.
- Hay que tener cuidado de no ejecutar funciones que afecten a la cabecera de la respuesta, como `setcookie()`, después de haber comenzado a generar la salida del script.
- Las cookies permiten al servidor almacenar información en cliente. Este puede borrarlas cuando quiera, ya que son parte del historial de navegación.
- Las páginas de una misma aplicación pueden compartir información mediante las sesiones. Las variables de sesión se almacenan en el array `$_SESSION`.
- PHP cuenta con drivers para las bases de datos más extendidas. La librería PDO se encarga de los detalles específicos de la base de datos.
- Hay varias librerías para trabajar con ficheros XML. Desde PHP es posible utilizar esquemas XSD, transformaciones XSL/XSLT y búsquedas con XPath.
- Para automatizar las pruebas la librería más extendida es PHPUnit.

## Ejercicios propuestos



1. Crea una página web con un formulario para elegir el color de fondo. Almacena la elección del usuario con una cookie para que la siguiente vez que el usuario se conecte la página aparezca directamente con ese color.

2. Amplia el ejemplo de la sección 3.4 para que la página principal incluya un nuevo vínculo, “Zona Admin”, solo visible para los usuarios con rol 1.
3. Escribe un programa que muestre una tabla con los datos de la tabla Usuarios.
4. Confecciona un formulario para insertar usuarios en la tabla Usuarios.
5. A partir del ejemplo del apartado 3.5, crea una función para enviar correos.
6. Realiza un formulario para enviar correos que utilice esa función.

## ACTIVIDADES DE AUTOEVALUACIÓN

1. Los parámetros que se pasan por la URL están disponibles en:  
 a) \$\_GET.  
 b) \$\_POST.  
 c) Ambos.
2. En \$\_FILES['fichero']['name'] se almacena:  
 a) El nombre en el cliente del archivo que se está subiendo.  
 b) El nombre temporal en el servidor del archivo que se está subiendo.  
 c) El nombre en el que se almacenará el fichero en el servidor.
3. En Mongo DB los documentos se almacenan como:  
 a) BSON.  
 b) XML.  
 c) Se convierten a una fila de la tabla correspondiente.
4. Para borrar una cookie se utiliza la función:  
 a) deletecookie().  
 b) setcookie().  
 c) erasecookie().
5. ¿Cuál de estos elementos no se envía en las cabeceras?  
 a) Redirecciones.  
 b) Datos de formulario.  
 c) Cookies.
6. El cliente envía al servidor:  
 a) Todas las cookies que tiene.  
 b) Todas las cookies que tiene de ese servidor.  
 c) Las cookies que el servidor le solicita.
7. Para cerrar la sesión hay que:  
 a) Llamar a sesion\_destroy().  
 b) Llamar a sesion\_destroy(), borrar las variables de sesión y la cookie de sesión.  
 c) Llamar a sesion\_destroy() o borrar las variables de sesión.

8. Para automatizar las pruebas se utiliza la librería:

- a) PHPUnit.
- b) PHPMailer.
- c) PDO.

9. Al abrir un fichero con modo de apertura "r+":

- a) Si el fichero no existe, lo crea.
- b) Si el fichero no existe, devuelve error.
- c) Si el fichero existe, devuelve error.

10. Al abrir un fichero con modo de apertura "w":

- a) Si el fichero no existe, lo crea.
- b) Si el fichero no existe, devuelve error.
- c) Si el fichero existe, devuelve error.

#### SOLUCIONES:

1. **a** **b** **c**

2. **a** **b** **c**

3. **a** **b** **c**

4. **a** **b** **c**

5. **a** **b** **c**

6. **a** **b** **c**

7. **a** **b** **c**

8. **a** **b** **c**

9. **a** **b** **c**

10. **a** **b** **c**