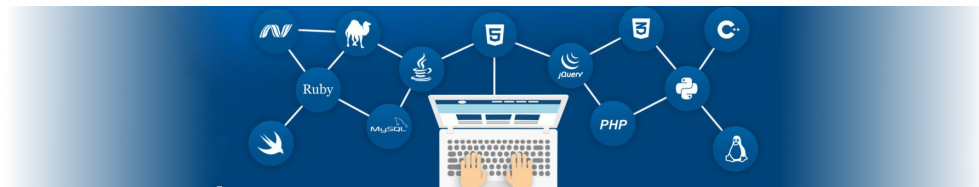


unidad didáctica 04

# Programación Web



## Índice

1. **duración y criterios de evaluación**
2. **variables de servidor**
3. **formularios**
  3. 1. validación
  3. 2. parámetros multivalor
  3. 3. volviendo a rellenar un formulario
  3. 4. subiendo archivos
    3. 4. 1. filtrado con php de tipos de ficheros subidos con html
    3. 4. 2. escritura de imágenes en carpeta del servidor
    3. 4. 3. seguridad de escritura de imagenes en carpeta del servidor
    3. 4. 4. extra 1: mostrar imagenes subidas con html
    3. 4. 5. extra 2: descargar ficheros subidos con html
4. **cabeceras de respuesta**
5. **gestión de estado**
  5. 1. cookies
  5. 2. sesión
6. **autenticación de usuarios**
7. **referencias**

# 1. duración y criterios de evaluación

---

**Duración estimada:** 12 sesiones

---

## **Resultado de aprendizaje y criterios de evaluación:**

4. Desarrolla aplicaciones Web embebidas en lenguajes de marcas analizando e incorporando funcionalidades según especificaciones.
  - a) *Se han identificado los mecanismos disponibles para el mantenimiento de la información que concierne a un cliente web concreto y se han señalado sus ventajas.*
  - b) *Se han utilizado sesiones para mantener el estado de las aplicaciones Web.*
  - c) *Se han utilizado cookies para almacenar información en el cliente Web y para recuperar su contenido.*
  - d) *Se han identificado y caracterizado los mecanismos disponibles para la autenticación de usuarios.*
  - e) *Se han escrito aplicaciones que integren mecanismos de autenticación de usuarios.*
  - f) *Se han realizado adaptaciones a aplicaciones Web existentes como gestores de contenidos u otras.*
  - g) *Se han utilizado herramientas y entornos para facilitar la programación, prueba y depuración del código.*

## 2. variables de servidor

PHP almacena la información del servidor y de las peticiones HTTP en seis arrays globales:

- `$_ENV`: información sobre las variables de entorno.
- `$_GET`: parámetros enviados en la petición GET.
- `$_POST`: parámetros enviados en el envío POST.
- `$_COOKIE`: contiene las cookies de la petición, las claves del array son los nombres de las cookies.
- `$_SERVER`: información sobre el servidor.
- `$_FILES`: información sobre los ficheros cargados via upload.

Si nos centramos en el array `$_SERVER` podemos consultar las siguientes propiedades:

- `PHP_SELF`: nombre del script ejecutado, relativo al document root (p.e.: `/tienda/carrito.php`).
- `SERVER_SOFTWARE`: (p.e.: `Apache`).
- `SERVER_NAME`: dominio, alias DNS (p.e.: `www.elche.es`).
- `REQUEST_METHOD`: GET.
- `REQUEST_URI`: URI, sin el dominio.
- `QUERY_STRING`: todo lo que va después de `?` en la URL (p.e.: `heroe=Batman&nombre=Bruce`).

Más información en <https://www.php.net/manual/es/reserved.variables.server.php>.

```

1 <?php
2     echo $_SERVER["PHP_SELF"]."<br>"; // /u4/401server.php
3     echo $_SERVER["SERVER_SOFTWARE"]."<br>"; // Apache/2.4.46 (Win64)
   OpenSSL/1.1.1g PHP/7.4.9
4     echo $_SERVER["SERVER_NAME"]."<br>"; // localhost
5
6     echo $_SERVER["REQUEST_METHOD"]."<br>"; // GET
7     echo $_SERVER["REQUEST_URI"]."<br>"; // /u4/401server.php?heroe=Batman
8     echo $_SERVER["QUERY_STRING"]."<br>"; // heroe=Batman

```

Otras propiedades relacionadas:

- `PATH_INFO`: ruta extra tras la petición (p.e.: si la URL es `http://www.php.com/php/pathInfo.php/algo/cosa?foo=bar`, entonces `$_SERVER['PATH_INFO']` será `/algo/cosa`).
- `REMOTE_HOST`: hostname que hizo la petición.
- `REMOTE_ADDR`: IP del cliente.
- `AUTH_TYPE`: tipo de autenticación (p.ej: `Basic`).
- `REMOTE_USER`: nombre del usuario autenticado.

Apache crea una clave para cada cabecera HTTP, en mayúsculas y sustituyendo los guiones por subrayados:

- `HTTP_USER_AGENT`: agente (navegador).
- `HTTP_REFERER`: página desde la que se hizo la petición.

```
1 <?php
2     echo $_SERVER["HTTP_USER_AGENT"]."<br>"; // Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88
Safari/537.36
```

## 3. formularios

A la hora de enviar un formulario, debemos tener claro cuando usar GET o POST:

- GET: los parámetros se pasan en la URL.
  - <2048 caracteres, sólo ASCII.
  - Permite almacenar la dirección completa (marcador / historial).
  - Idempotente: dos llamadas con los mismos datos siempre debe dar el mismo resultado.
  - El navegador puede cachear las llamadas.
- POST: parámetros ocultos (no encriptados).
  - Sin límite de datos, permite datos binarios.
  - No se pueden cachear.
  - No idempotente → actualizar la BBDD.

Así pues, para recoger los datos accederemos al array dependiendo del método del formulario que nos ha invocado:

```
1 <?php
2     $par = $_GET["parametro"]
3     $par = $_POST["parametro"]
```

Para los siguientes apartados nos vamos a basar en el siguiente ejemplo:

```
1 <form action="formulario.php" method="GET">
2     <p>
3         <label for="nombre">Nombre del alumno:</label>
4         <input type="text" name="nombre" id="nombre" value="" />
5     </p>
6     <p>
7         <input type="checkbox" name="modulos[]" id="modulosDWES" value="DWES" />
8         <label for="modulosDWES">Desarrollo web en entorno servidor</label>
9     </p>
10    <p>
11        <input type="checkbox" name="modulos[]" id="modulosDWEC" value="DWEC" />
12        <label for="modulosDWEC">Desarrollo web en entorno cliente</label>
13    </p>
14    <input type="submit" value="Enviar" name="enviar" />
15 </form>
```

### 3.1. validación

Respecto a la validación, es conveniente siempre hacer **validación doble**:

- En el cliente mediante JS.
- En servidor, antes de llamar a negocio, es conveniente volver a validar los datos.

```

1 <?php
2     if (isset($_GET["parametro"])) {
3         $par = $_GET["parametro"];
4         // comprobar si $par tiene el formato adecuado, su valor, etc...
5     }

```

### librerías de validación

Existen diversas librerías que facilitan la validación de los formularios, como son [respect/validation](#) o [particle/validator](#). Cuando estudiemos Laravel profundizaremos en la validación de forma declarativa.

## 3.2. parámetros multivalor

Existen elementos HTML que envían varios valores:

- `select multiple`
- `checkbox`

Para recoger los datos, el nombre del elemento debe ser un array.

```

1 <select name="lenguajes[]" multiple="true">
2     <option value="c">C</option>
3     <option value="java">Java</option>
4     <option value="php">PHP</option>
5     <option value="python">Python</option>
6 </select>
7
8 <input type="checkbox" name="lenguajes[]" value="c" />C<br />
9 <input type="checkbox" name="lenguajes[]" value="java" />Java<br />
10 <input type="checkbox" name="lenguajes[]" value="php" />Php<br />
11 <input type="checkbox" name="lenguajes[]" value="python" />Python<br />

```

De manera que luego al recoger los datos:

```

1 <?php
2     $lenguajes = $_GET["lenguajes"];
3
4     foreach ($lenguajes as $lenguaje) {
5         echo "$lenguaje <br />";
6     }

```

## 3.3. volviendo a rellenar un formulario

Un *sticky form* es un formulario que recuerda sus valores. Para ello, hemos de rellenar los atributos `value` de los elementos HTML con la información que contenían:

```

1 <?php
2     if (!empty($_POST['modulos']) && !empty($_POST['nombre'])) {
3         // Aquí se incluye el código a ejecutar cuando los datos son correctos
4     } else {
5         // Generamos el formulario
6         $nombre = $_POST['nombre'] ?? "";
7         $modulos = $_POST['modulos'] ?? [];

```

```

8  ?>
9
10 <form action="<?php echo $_SERVER['PHP_SELF'];?>" method="POST">
11     <p>
12         <label for="nombre">Nombre del alumno:</label>
13         <input type="text" name="nombre" id="nombre" value="<?= $nombre ?>" />
14     </p>
15     <p>
16         <input type="checkbox" name="modulos[]" id="modulosDWES" value="DWES"
17         <?php if(in_array("DWES",$modulos)) echo 'checked="checked"'; ?> />
18         <label for="modulosDWES">Desarrollo web en entorno servidor</label>
19     </p>
20     <p>
21         <input type="checkbox" name="modulos[]" id="modulosDWEC" value="DWEC"
22         <?php if(in_array("DWEC",$modulos)) echo 'checked="checked"'; ?> />
23         <label for="modulosDWEC">Desarrollo web en entorno cliente</label>
24     </p>
25     <input type="submit" value="Enviar" name="enviar"/>
26 </form>
27
28 <?php } ?>

```

### 3.4. subiendo archivos

Se almacenan en el servidor en el array `$_FILES` con el nombre del campo del tipo `file` del formulario.

```

1  <form enctype="multipart/form-data" action="<?php echo $_SERVER['PHP_SELF']; ?
  >" method="POST">
2      Archivo: <input name="archivoEnviado" type="file" />
3      <br />
4      <input type="submit" name="btnSubir" value="Subir" />
5  </form>

```

Configuración en `php.ini`:

- `file_uploads`: on / off.
- `upload_max_filesize`: 2M.
- `upload_tmp_dir`: directorio temporal. No es necesario configurarlo, cogerá el predeterminado del sistema.
- `post_max_size`: tamaño máximo de los datos POST. Debe ser mayor a `upload_max_filesize`.
- `max_file_uploads`: número máximo de archivos que se pueden cargar a la vez.
- `max_input_time`: tiempo máximo empleado en la carga (GET/POST y upload → normalmente se configura en 60).
- `memory_limit`: 128M.
- `max_execution_time`: tiempo de ejecución de un script (no tiene en cuenta el upload).



Veamos de qué información disponemos en el array `$_FILES` para una imagen llamada 'saludo.jpg' subida mediante nuestro formulario:

```
1 <?php
2     echo $_FILES['archivoEnviado']['name'];
3     echo $_FILES['archivoEnviado']['tmp_name'];
4     echo $_FILES['archivoEnviado']['type'];
5     echo $_FILES['archivoEnviado']['size'];
6 ?>
```

Mediante estos *echo's* se ha accedido a toda la información disponible del fichero en PHP. La información impresa sería la siguiente:

```
1 saludo
2 213mnuashduahs0923
3 image/jpg
4 120304
5 0
```

Para cargar los archivos, accedemos al array `$_FILES`:

```
1 <?php
2     if (isset($_POST['btnSubir']) && $_POST['btnSubir'] == 'Subir') {
3         if (is_uploaded_file($_FILES['archivoEnviado']['tmp_name'])) {
4             // subido con éxito
5             $nombre = $_FILES['archivoEnviado']['name'];
6             move_uploaded_file($_FILES['archivoEnviado']['tmp_name'],
7                 "./uploads/{$nombre}");
8             echo "<p>Archivo $nombre subido con éxito</p>";
9         }
10    }
```

Cada archivo cargado en `$_FILES` tiene:

- `name`: nombre.
- `tmp_name`: nombre temporal asignado al fichero por el servidor. Este nombre es único y permite identificarlo dentro de la carpeta de temporales.
- `size`: tamaño en bytes.
- `type`: tipo MIME.
- `error`: código de error de la subida, en nuestro caso 0 o `UPLOAD_ERR_OK` que indica que no se ha producido error alguno. [Códigos de error subida de fichero](#).

### 3.4.1. filtrado con php de tipos de ficheros subidos con html

Una vez sabemos cómo acceder a la información de los ficheros subidos, vamos a centrarnos en el filtrado de los tipos de ficheros aceptados. Limitar el tipo de fichero subido es altamente recomendable para evitar posibles problemas de seguridad.

*Para este ejemplo voy comprobar que la imagen subida sea en efecto una imagen con una de las extensiones más comunes y que su tamaño sea menor a 1 MB:*

```

1  <?php
2      $extensiones = ['image/jpg', 'image/jpeg', 'image/png'];
3      $max_tamanyo = 1024 * 1024 * 8;
4      if ( in_array($_FILES['archivoEnviado']['type'], $extensiones) ) {
5          echo 'Es una imagen';
6          if ( $_FILES['archivoEnviado']['size'] < $max_tamanyo ) {
7              echo 'Pesa menos de 1 MB';
8          }
9      }
10  ?>

```

### 3.4.2. escritura de imágenes en carpeta del servidor

Una vez tengamos nuestros ficheros filtrados vamos a proceder a guardarlos de forma permanente en una carpeta de nuestro servidor.

Las imagenes o ficheros subidos mediante los formularios HTML son almacenados siempre en una carpeta temporal del sistema, por lo tanto deberemos moverlos para poder guardarlos permanentemente.

Para trasladar los ficheros de la carpeta temporal directamente a nuestra carpeta elegida usaremos la función [move\\_uploaded\\_file\( origen, destino \)](#).

El siguiente ejemplo sería un script alojado en la carpeta raíz de nuestra web, p.e. index.php:

```

1  <?php
2      $ruta_indexphp = dirname(realpath(__FILE__));
3      $ruta_fichero_origen = $_FILES['archivoEnviado']['tmp_name'];
4      $ruta_nuevo_destino = $ruta_indexphp . '/uploads/' . $_FILES['imagen1']
5      ['name'];
6
7      if ( in_array($_FILES['archivoEnviado']['type'], $extensiones) ) {
8          echo 'Es una imagen';
9          if ( $_FILES['archivoEnviado']['size'] < $max_tamanyo ) {
10             echo 'Pesa menos de 1 MB';
11             if( move_uploaded_file ( $ruta_fichero_origen, $ruta_nuevo_destino )
12             ) {
13                 echo 'Fichero guardado con éxito';
14             }
15         }
16     }
17 }
18 ?>

```

- El nombre temporal del fichero subido, que se encuentra en la carpeta de temporales, en `$ruta_fichero_origen`.
- La ruta completa de destino del fichero, que se compone por una parte de la ruta raíz del script donde estamos trabajando (estamos programando en `index.php`) más el nombre de la carpeta que se ha creado para guardar las imagenes (`/uploads`) y por último el nombre definitivo que tendrá el fichero (*el nombre original del fichero*).

### 3.4.3. seguridad de escritura de imagenes en carpeta del servidor

Al guardar los archivos subidos por los usuarios en nuestro servidor, puede ocurrir que no filtremos los ficheros introducidos, o guardemos ficheros susceptibles de provocar problemas de seguridad. Para evitar problemas de este tipo lo mejor será incluir en la carpeta donde los almacenamos un pequeño *script htaccess* que evite la ejecución de código:

```

1 RemoveHandler .phtml .php3 .php .pl .py .jsp .asp .htm .shtml .sh .cgi .dat
2 RemoveType .phtml .php3 .php .pl .py .jsp .asp .htm .shtml .sh .cgi .dat

```

Con estas dos líneas en un fichero con extensión *.htaccess* evitaremos una posible ejecución de código por parte de usuarios malintencionados. Recuerda que debes incluir este fichero en la misma carpeta donde almacenas los ficheros.

Y ya está, con esto tendríamos terminado un **formulario para subir imagenes con php** totalmente funcional, con comprobaciones de seguridad para evitar subidas de ficheros inesperadas que puedan provocar problemas o hackeos inesperados.

### 3.4.4. extra 1: mostrar imagenes subidas con html

Mostrar las imagenes guardadas en nuestra carpeta de almacenamiento es sencillo, tan solo deberemos incluir la ruta hasta el fichero en una etiqueta IMG html:

```

1 

```

### 3.4.5. extra 2: descargar ficheros subidos con html

Si queremos incluir un enlace de descarga para el fichero almacenado, en vez de utilizar una etiqueta *IMG* usaremos una etiqueta para enlaces con el atributo *HREF* la ruta al fichero:

```

1 <a href="uploads/nombreImagen.jpg"> Descarga de la imagen </a>

```

Este enlace producirá que el usuario descargue el fichero en cuestión, no obstante si el archivo es por ejemplo una imagen o pdf, el usuario en vez de lograr una descarga directa visualizará el contenido, teniendo que descargarlo haciendo uso de la opción descargar del menú desplegable con clic derecho.

Evitar la visualización de ficheros es posible gracias a HTML5 y los navegadores más modernos: Chrome, Firefox, Opera. Deberemos incluir el atributo `download` en la etiqueta de enlace anterior (A), así, el enlace final para una descarga forzada quedaría así:

```

1 <a href="uploads/nombreImagen.jpg" download="nombreImagen">Descarga la
  imagen</a>

```

## 4. cabeceras de respuesta

Se devuelven mediante la función `header(cadena)`. Mediante las cabeceras podemos configurar el tipo de contenido, tiempo de expiración, redireccionar el navegador, especificar errores HTTP, etc.

Debe ser **lo primero a devolver**

El motivo es que en cuando un programa genera contenido HTML, el servidor genera automáticamente la información de estado y los campos de cabecera y a continuación envía el contenido generado. Si después el programa contiene una instrucción **header()**, se produce un **error** porque las cabeceras ya se han enviado (ni siquiera deja una linea en blanco).

```
1 <?php header("Content-Type: text/plain"); ?>
2 <?php header("Location: http://www.ejemplo.com/inicio.html");
3 exit();
```

**inspeccionando las cabeceras**

Se puede comprobar en las herramientas del desarrollador de los navegadores web mediante *Developer Tools* → *Network* → *Headers*.

Es muy común configurar las cabeceras para evitar consultas a la caché o provocar su renovación:

```
1 <?php
2   header("Expires: Sun, 31 Jan 2021 23:59:59 GMT");
3
4   // tres horas
5   $now = time();
6   $horas3 = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60 * 60 * 3);
7   header("Expires: {$horas3}");
8
9   // un año
10  $now = time();
11  $anyo1 = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365 * 86400);
12  header("Expires: {$anyo1}");
13
14  // se marca como expirado (fecha en el pasado)
15  $pasado = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");
16  header("Expires: {$pasado}");
17
18  // evitamos cache de navegador y/o proxy
19  header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
20  header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
21  header("Cache-Control: no-store, no-cache, must-revalidate");
22  header("Cache-Control: post-check=0, pre-check=0", false);
23  header("Pragma: no-cache");
```

## 5. gestión de estado

HTTP es un protocolo stateless, sin estado. Por ello, se simula el estado mediante el uso de *cookies*, *tokens* o la *sesión*. El estado es necesario para procesos tales como el carrito de la compra, operaciones asociadas a un usuario, etc... El mecanismo de PHP para gestionar la sesión emplea *cookies* de forma interna. Las *cookies* se almacenan en el navegador, y la sesión en el servidor web.

### 5.1. cookies

Las *cookies* se almacenan en el array global `$_COOKIE`. Lo que coloquemos dentro del array, se guardará en el cliente. Hay que tener presente que el cliente puede no querer almacenarlas.

Existe una limitación de 20 *cookies* por dominio y 300 en total en el navegador.

En PHP, para crear una cookie se utiliza la función `setcookie`:

```
1 <?php
2     setcookie(nombre [, valor [, expira [, ruta [, dominio [, seguro [, httponly
   ]]]]]]);
3     setcookie(nombre [, valor = "" [, opciones = [] ]] )
4 ?>
```

Destacar que el nombre no puede contener espacios ni el carácter `;`. Respecto al contenido de la cookie, no puede superar los 4 KB.

Por ejemplo, mediante *cookies* podemos comprobar la cantidad de visitas diferentes que realiza un usuario:

```
1 <?php
2     $accesosPagina = 0;
3     if (isset($_COOKIE['accesos'])) {
4         $accesosPagina = $_COOKIE['accesos']; // recuperamos una cookie
5         setcookie('accesos', ++$accesosPagina); // le asignamos un valor
6     }
7 ?>
```

#### inspeccionando las cookies

Si queremos ver que contienen las *cookies* que tenemos almacenadas en el navegador, se puede comprobar su valor en **Dev Tools** → **Application** → **Storage**

Ejemplo de cookies:

1. vamos a realizar un fichero html (`cookiesEjemplo1.html`) en el que:

```

1  <center>
2      <h2>elige idioma:</h2>
3      <a href="cookiesEjemplo1.php?idioma=va">
4          
5      </a>
6      <a href="cookiesEjemplo1.php?idioma=es">
7          
8      </a>
9      <a href="cookiesEjemplo1.php?idioma=en">
10         
11     </a>
12 </center>

```



2. Creamos fichero `cookiesEjemplo1.php` en el que comprueba si se ha enviado el parámetro `idioma`, si es así, crea una *cookie* de nombre `idioma`, valor `$_GET['idioma']`, tiempo 1 hora y que trabaja desde la raíz `\`. Además, se ha añadido una cabecera que redireccionará a la página `cookiesEjemplo1_b.php`:

```

1  <?php
2      if (isset($_GET['idioma'])){
3          setcookie('idioma',$_GET['idioma'], time()+3600, "/");
4          header("Location:cookiesEjemplo1_b.php");
5      }
6  ?>

```

3. En el fichero `cookiesEjemplo1_b.php` se comprueba el valor de la *cookie* `idioma`:

```

1  <?php
2      if (!$_COOKIE['idioma']){
3          header('Location:cookiesEjemplo1.html');
4      } else if ($_COOKIE['idioma']=="va"){
5          header('Location:cookiesEjemplo1Va.php');
6      } else if ($_COOKIE['idioma']=="es"){
7          header('Location:cookiesEjemplo1Es.php');
8      } else if ($_COOKIE['idioma']=="en"){
9          header('Location:cookiesEjemplo1En.php');
10     }
11 ?>

```

4. El fichero, por ejemplo, `cookiesEjemplo1Va.php` quedaría:

```

1 <h2> idioma en valencià </h2>
2 <br /> <br />
3 <a href="cookiesEjemplo2BorrarCookies.php">borrar cookies</a>

```

El tiempo de vida de las cookies puede ser tan largo como el sitio web en el que residen. Ellas seguirán ahí, incluso si el navegador está cerrado o abierto.

Para borrar una cookie se puede poner que expiren en el pasado:

```

1 <?php
2     setcookie(nombre, "", 1) // pasado

```

O que caduquen dentro de un periodo de tiempo determinado:

```

1 <?php
2     setcookie(nombre, valor, time() + 3600) // Caducan dentro de una hora

```

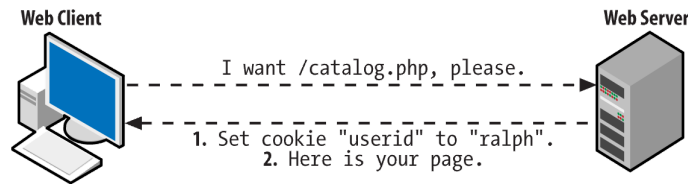
Ejemplo 2: Siguiendo con el ejemplo anterior de idiomas, podríamos crear el fichero `cookiesEjemplo2BorrarCookies.php` y, desde ahí, eliminar la *cookie* idioma:

```

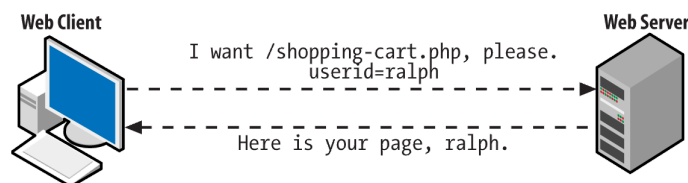
1 <?php
2     setcookie('idioma', '', time()-1, '/');
3     header('Location:cookiesEjemplo1.html');
4 ?>

```

#### First Request



#### Second Request



Se utilizan para:

- Recordar los inicios de sesión.
- Almacenar valores temporales de usuario.
- Si un usuario está navegando por una lista paginada de artículos, ordenados de cierta manera, podemos almacenar el ajuste de la clasificación.

La alternativa en el cliente para almacenar información en el navegador es el objeto [LocalStorage](#).

## 5.2. sesión

La *sesión* añade la gestión del estado a HTTP, almacenando en este caso la información en el servidor. Cada visitante tiene un ID de sesión único, el cual por defecto se almacena en una cookie denominada `PHPSESSID`. Si el cliente no tiene las *cookies* activas, el ID se propaga en cada URL dentro del mismo dominio. Cada sesión tiene asociado un almacén de datos mediante el array global `$_SESSION` en el cual podemos almacenar y recuperar información.

La sesión comienza al ejecutar un script PHP. Se genera un nuevo ID y se cargan los datos del almacén:



El trabajo con sesiones tiene tres partes:

- Creación o apertura de la sesión:

Cuando alguna página crea una sesión utilizando la cabecera `session_start()`; (recuerda que las cabeceras deben ir justo después de `<?php`), el servidor asocia al navegador del usuario un identificador de usuario único. El identificador se guarda en el usuario en forma de cookie o, si el navegador del usuario no permite la creación de cookies, añadiendo el identificador en la dirección de la página.

- Utilización de la sesión:

Si ya se ha creado la sesión, las páginas solicitadas por el mismo navegador pueden guardar y recuperar información en el servidor, información que se asocia al identificador de usuario, por lo que no es accesible a otros usuarios. La información se conserva hasta que el usuario o el servidor destruyan la sesión.

- Destrucción o cierre de la sesión:

Tanto el usuario como el servidor pueden cerrar la sesión. El usuario puede destruir la sesión cerrando el navegador. El servidor puede destruir la sesión cuando alguna página utilice la función `session_destroy()`; o al cabo de un tiempo determinado (definido mediante la función `session_set_cookie_params()`).

Las operaciones que podemos realizar con la sesión son:

```
1 <?php
2     session_start(); // carga la sesión
3     session_id() // devuelve el id
4     $_SESSION[clave] = valor; // inserción
5     session_destroy(); // destruye la sesión
6     unset($_SESSION[clave]); // borrado
```



Vamos a ver mediante un ejemplo cómo podemos insertar en una página datos en la *sesión* para posteriormente en otra página acceder a esos datos. Por ejemplo, en `sesion1.php` tendríamos:

```
1 <?php
2     session_start(); // inicializamos
3     $_SESSION["ies"] = "IES Mestre Ramon Esteve"; // asignación
4     $instituto = $_SESSION["ies"]; // recuperación
5     echo "Estamos en el $instituto ";
6     ?>
7     <br />
8     <a href="sesion2.php">Y luego</a>
```

Y posteriormente podemos acceder a la sesión en `sesion2.php`:

```
1 <?php
2     session_start();
3     $instituto = $_SESSION["ies"]; // recuperación
4     echo "Otra vez, en el $instituto ";
5     ?>
```

### configurando la sesión en php.ini

Las siguientes propiedades de `php.ini` permiten configurar algunos aspectos de la sesión:

- `session.save_handler`: controlador que gestiona cómo se almacena (valor `files`).
- `session.save_path`: ruta donde se almacenan los archivos con los datos (si tenemos un cluster, podríamos usar `/mnt/sessions` en todos los servidores de manera que apunten a una carpeta compartida).
- `session.name`: nombre de la sesión (PHSESSID).
- `session.auto_start`: Se puede hacer que se autocargue con cada script. Por defecto está deshabilitado.
- `session.cookie_lifetime`: tiempo de vida por defecto.

Más información en la [documentación oficial](#).

### cookie VS session

Las *sesiones* no deben confundirse con las *cookies*. Las *cookies* es un método que permite guardar información en el ordenador del cliente para recuperarla en el futuro; mientras que en las *sesiones* la información se mantiene en el servidor hasta que se cierra la sesión (por intervención del usuario o por tiempo). En el manual de PHP se ofrece un [capítulo dedicado a las sesiones](#).

Ejemplo: he aquí un ejemplo gráfico de utilización de sesiones. Programa de dos páginas que muestra gráficamente los votos recogidos por dos opciones.

- La primera página contiene un formulario con tres botones de tipo `submit` con el mismo atributo `name`.
  - Dos botones permiten votar a una u otra opción.
  - El tercer botón pone a cero los contadores de votos.

- La segunda página recibe el dato, modifica la variable de sesión que contiene el número de votos de la opción elegida (o ambas) y redirige a la primera página.
- Los dos números se guardan en dos variables de sesión. Si las variables de sesión no están definidas, se les dará el valor 0.
- Las franjas correspondientes a los votos se alargan de 10px en 10px y no tienen límite de tamaño.

```

1  <?php
2  // Accedemos a la sesión
3  session_name("sesion1");
4  session_start();
5
6  // Si algún contador no está guardado en la sesión, ponemos ambos a
   cero
7  if (!isset($_SESSION["a"]) || !isset($_SESSION["b"])) {
8      $_SESSION["a"] = $_SESSION["b"] = 0;
9  }
10 ?>
11
12 <!-- ... -->
13 <body>
14     <h2>Votar una opción</h2>
15     <form action="ps2.php" method="get">
16         <table>
17             <tr>
18                 <td style="vertical-align: top;">
19                     <button type="submit" name="accion" value="a"
20                         style="font-size: 60px; line-height: 50px; color:
   hwb(200 0% 0%);"> &#x2714; </button>
21                 </td>
22                 <td>
23 <?php
24 // Dibujamos la primera barra
25 print " <svg width=\"$_SESSION[a]\" height=\"50\">\n";
26 print "     <line x1=\"0\" y1=\"25\" x2=\"$_SESSION[a]\" y2=\"25\"
   stroke=\"hwb(200 0% 0%)\" stroke-width=\"50\" />\n";
27 print " </svg>\n";
28 ?>
29                 </td>
30             </tr>
31             <tr>
32                 <td>
33                     <button type="submit" name="accion" value="b"
34                         style="font-size: 60px; line-height: 50px; color:
   hwb(35 0% 0%)>&#x2714;
35                     </button>
36                 </td>
37             </tr>
38 // Dibujamos la segunda barra
39 print " <td>\n";
40 print " <svg width=\"$_SESSION[b]\" height=\"50\">\n";
41 print "     <line x1=\"0\" y1=\"25\" x2=\"$_SESSION[b]\" y2=\"25\"
   stroke=\"hwb(35 0% 0%)\" stroke-width=\"50\" />\n";
42 print " </svg>\n";

```

```

43 print " </td>\n";
44 ?>
45     </tr>
46 </table>
47 <p>
48     <button type="submit" name="accion" value="cero">Poner a
cero</button>
49 </p>
50 </form>
51 </body>
52 </html>

```

```

1 <?php
2 session_name("sesion1");
3 session_start();
4
5 // Si alguno de los números de votos no está guardado en la sesión,
redirigimos a la primera página
6 if (!isset($_SESSION["a"]) || !isset($_SESSION["b"])) {
7     header("Location:ps1.php");
8     exit;
9 }
10
11 $accion = $_GET['accion'];
12 // Dependiendo de la acción recibida, modificamos el número
correspondiente
13 if ($accion == "a") {
14     $_SESSION["a"] += 10;
15 } elseif ($accion == "b") {
16     $_SESSION["b"] += 10;
17 } elseif ($accion == "cero") {
18     $_SESSION["a"] = $_SESSION["b"] = 0;
19 }
20
21 // Volvemos al formulario
22 header("Location:ps1.php");

```

## Votar una opción

Haz clic en los botones para votar por una opción:



The image shows a voting interface. It features two rows of buttons. Each row contains a button with a large blue checkmark and a colored square. The top row has a blue square, and the bottom row has an orange square. Below these buttons is a button labeled 'Poner a cero'.

## 6. autenticación de usuarios

Una sesión establece una relación anónima con un usuario particular, de manera que podemos saber si es el mismo usuario entre dos peticiones distintas. Si preparamos un sistema de login, podremos saber quién utiliza nuestra aplicación.

Para ello, preparemos un sencillo sistema de autenticación:

- Mostrar el formulario login/password.
- Comprobar los datos enviados.
- Añadir el login a la sesión.
- Comprobar el login en la sesión para realizar tareas específicas del usuario.
- Eliminar el login de la sesión cuando el usuario la cierra.

Vamos a ver en código cada paso del proceso. Comenzamos con el archivo `index.php`:

```

1  <form action='login.php' method='post'>
2    <fieldset>
3      <legend>Login</legend>
4      <div><span class='error'><?php echo $error; ?></span></div>
5      <div class='fila'>
6        <label for='usuario'>Usuario:</label><br />
7        <input type='text' name='inputUsuario' id='usuario' maxlength="50" />
8      </div>
9      <div class='fila'>
10       <label for='password'>Contraseña:</label><br />
11       <input type='password' name='inputPassword' id='password'
12       maxlength="50" /><br />
13     </div>
14     <div class='fila'>
15       <input type='submit' name='enviar' value='Enviar' />
16     </div>
17   </fieldset>
18 </form>

```

Al hacer submit nos lleva a `login.php`, el cual hace de controlador:

```

1  <?php
2  // Comprobamos si ya se ha enviado el formulario
3  if (isset($_POST['enviar'])) {
4    $usuario = $_POST['inputUsuario'];
5    $password = $_POST['inputPassword'];
6
7    // validamos que recibimos ambos parámetros
8    if (empty($usuario) || empty($password)) {
9      $error = "Debes introducir un usuario y contraseña";
10     include "index.php";
11   } else {
12     if ($usuario == "admin" && $password == "admin") {
13       // almacenamos el usuario en la sesión

```

```

14         session_start();
15         $_SESSION['usuario'] = $usuario;
16         // cargamos la página principal
17         include "main.php";
18     } else {
19         // Si las credenciales no son válidas, se vuelven a pedir
20         $error = "Usuario o contraseña no válidos!";
21         include "index.php";
22     }
23 }
24 }

```

Dependiendo del usuario que se haya logueado, vamos a ir a una vista o a otra. Por ejemplo, en `main.php` tendríamos:

```

1  <?php
2      // Recuperamos la información de la sesión
3      if(!isset($_SESSION)) {
4          session_start();
5      }
6
7      // Y comprobamos que el usuario se haya autenticado
8      if (!isset($_SESSION['usuario'])) {
9          die("Error - debe <a href='index.php'>identificarse</a>.<br />");
10     }
11  ?>
12  <!DOCTYPE html>
13  <html lang="es">
14  <head>
15      <meta charset="UTF-8">
16      <meta name="viewport" content="width=device-width, initial-scale=1.0">
17      <title>Listado de productos</title>
18  </head>
19  <body>
20      <h1>Bienvenido <?= $_SESSION['usuario'] ?></h1>
21      <p>Pulse <a href="logout.php">aquí</a> para salir</p>
22      <p>Volver al <a href="main.php">inicio</a></p>
23      <h2>Listado de productos</h2>
24      <ul>
25          <li>Producto 1</li>
26          <li>Producto 2</li>
27          <li>Producto 3</li>
28      </ul>
29  </body>
30  </html>

```

Finalmente, necesitamos la opción de cerrar la sesión que colocamos en `logout.php`:

```
1 <?php
2     // Recuperamos la información de la sesión
3     session_start();
4
5     // Y la destruimos
6     session_destroy();
7     header("Location: index.php");
8 ?>
```

### autenticación en producción

En la actualidad la autenticación de usuario no se realiza gestionando la sesión directamente, sino que se realiza mediante algún framework que abstraer todo el proceso o la integración de mecanismos de autenticación tipo OAuth, como estudiaremos en la última unidad mediante *Laravel*.

## 7. referencias

---

- [Cookies en PHP](#)
- [Manejo de sesiones en PHP](#)