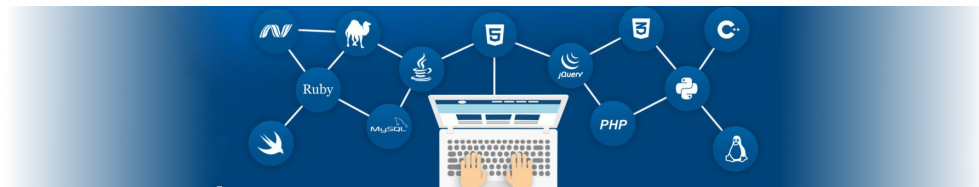


unidad didáctica 04

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**
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).

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`: ruta temporal.
- `size`: tamaño en bytes.
- `type`: tipo MIME.
- `error`: si hay error, contiene un mensaje. Si ok → 0.

4. cabeceras de respuesta

Debe ser lo primero a devolver. 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.

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

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     // tres horas
4     $now = time();
5     $horas3 = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60 * 60 * 3);
6     header("Expires: {$horas3}");
7     // un año
8     $now = time();
9     $anyo1 = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365 * 86440);
10    header("Expires: {$anyo1}");
11    // se marca como expirado (fecha en el pasado)
12    $pasado = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");
13    header("Expires: {$pasado}");
14    // evitamos cache de navegador y/o proxy
15    header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
16    header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
17    header("Cache-Control: no-store, no-cache, must-revalidate");
18    header("Cache-Control: post-check=0, pre-check=0", false);
19    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**

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.

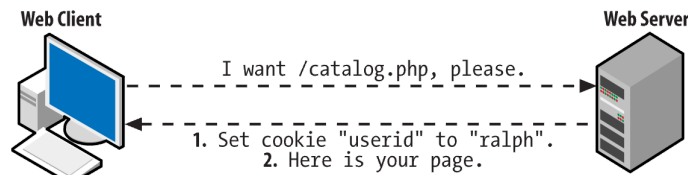
Para borrar una cookie se puede poner que expiren en el pasado:

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

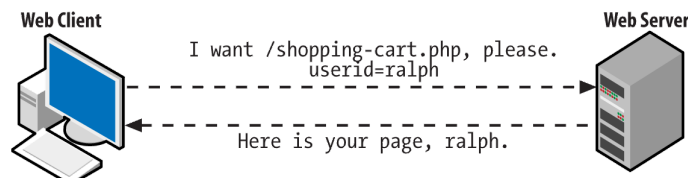
O que caduquen dentro de un periodo de tiempo determinado:

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

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.

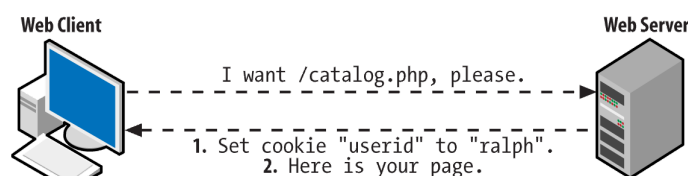
La alternativa en el cliente para almacenar información en el navegador es el objeto [LocalStorage](#).

5.2. sesión

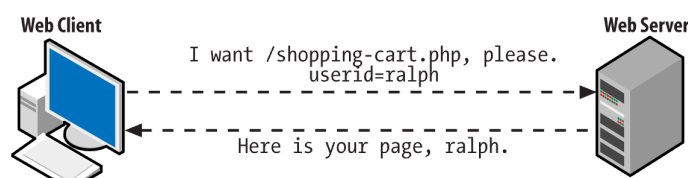
La sesión añade la gestión del estado a HTTP, almacenando en este caso la información en el servidor. Cada visitante tiene un ID de sesión único, el cual por defecto se almacena en una cookie denominada PHPSESSID. Si el cliente no tiene las cookies activas, el ID se propaga en cada URL dentro del mismo 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:

First Request



Second Request



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 como 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 Severo Ochoa"; // 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 (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 (PHPSESSID).
- `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](#).

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 quien 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](#)