unidad didáctica 04

Programación Web

Índice

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 autentificación de usuarios.*

   e) *Se han escrito aplicaciones que integren mecanismos de autentificación 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.
- `$_POST`: parámetros enviados en el envio POST.
- `$_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.

```php
<?php
    echo $_SERVER["PHP_SELF"]."<br>"; // /u4/401server.php
    echo $_SERVER["SERVER_SOFTWARE"]."<br>"; // Apache/2.4.46 (Win64) OpenSSL/1.1.1g PHP/7.4.9
    echo $_SERVER["SERVER_NAME"]."<br>"; // localhost

    echo $_SERVER["REQUEST_METHOD"]."<br>"; // GET
    echo $_SERVER["REQUEST_URI"]."<br>"; // /u4/401server.php?hero=Batman
    echo $_SERVER["QUERY_STRING"]."<br>"; // hero=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.

```php
<?php
    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:

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

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

```html
<form action="formulario.php" method="GET">
  <p>
    <label for="nombre">Nombre del alumno:</label>
    <input type="text" name="nombre" id="nombre" value="" />
  </p>
  <p>
    <input type="checkbox" name="modulos[]" id="modulosDWES" value="DWES" />
    <label for="modulosDWES">Desarrollo web en entorno servidor</label>
  </p>
  <p>
    <input type="checkbox" name="modulos[]" id="modulosDWEC" value="DWEC" />
    <label for="modulosDWEC">Desarrollo web en entorno cliente</label>
  </p>
  <input type="submit" value="Enviar" name="enviar" />
</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.

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

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.

```php
<select name="lenguajes[]" multiple="true">
    <option value="c">C</option>
    <option value="java">Java</option>
    <option value="php">PHP</option>
    <option value="python">Python</option>
</select>

<input type="checkbox" name="lenguajes[]" value="c" />C<br />
<input type="checkbox" name="lenguajes[]" value="java" />Java<br />
<input type="checkbox" name="lenguajes[]" value="php" />Php<br />
<input type="checkbox" name="lenguajes[]" value="python" />Python<br />
```

De manera que luego al recoger los datos:

```php
<?php
    $lenguajes = $_GET["lenguajes"];

    foreach ($lenguajes as $lenguaje) {
        echo "$lenguaje <br />";
    }
```

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:

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

```php
?>

<form action="<?php echo $_SERVER['PHP_SELF'];?>" method="POST">
  <p>
   <label for="nombre">Nombre del alumno:</label>
   <input type="text" name="nombre" id="nombre" value="<?= $nombre ?>" />
  </p>
  <p>
   <input type="checkbox" name="modulos[]" id="modulosDWES" value="DWES"
   <?php if(in_array("DWES",$modulos)) echo 'checked="checked"'; ?> />
   <label for="modulosDWES">Desarrollo web en entorno servidor</label>
  </p>
  <p>
   <input type="checkbox" name="modulos[]" id="modulosDWEC" value="DWEC"
   <?php if(in_array("DWEC",$modulos)) echo 'checked="checked"'; ?> />
   <label for="modulosDWEC">Desarrollo web en entorno cliente</label>
  </p>
  <input type="submit" value="Enviar" name="enviar"/>
</form>

<?php } ?>
```

3.4. subiendo archivos

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

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

```php
<?php
    echo $_FILES['archivoEnviado']['name'];
    echo $_FILES['archivoEnviado']['tmp_name'];
    echo $_FILES['archivoEnviado']['type'];
    echo $_FILES['archivoEnviado']['size'];
?>
```

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

```
saludo
213mnuashduahs0923
image/jpg
120304
0
```

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

```php
<?php
  if (isset($_POST['btnSubir']) && $_POST['btnSubir'] == 'Subir') {
     if (is_uploaded_file($_FILES['archivoEnviado']['tmp_name'])) {
        // subido con éxito
        $nombre = $_FILES['archivoEnviado']['name'];
        move_uploaded_file($_FILES['archivoEnviado']['tmp_name'],
"./uploads/{$nombre}");

        echo "<p>Archivo $nombre subido con éxito</p>";
     }
  }
```

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 súbida sea en efecto una imagen con una de las extensiones más comunes y que su tamaño sea menor a 1 MB:

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

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 raiz de nuestra web, p.e. index.php:

```php
<?php
  $ruta_indexphp = dirname(realpath(__FILE__));
  $ruta_fichero_origen = $_FILES['archivoEnviado']['tmp_name'];
  $ruta_nuevo_destino = $ruta_indexphp . '/uploads/' . $_FILES['imagen1']['name'];

  if ( in_array($_FILES['archivoEnviado']['type'], $extensiones) ) {
     echo 'Es una imagen';
     if ( $_FILES['archivoEnviado']['size']< $max_tamanyo ) {
        echo 'Pesa menos de 1 MB';
        if( move_uploaded_file ( $ruta_fichero_origen, $ruta_nuevo_destino ) ) {
                        echo 'Fichero guardado con éxito';
        }
     }
  }
?>
```

- 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 raiz 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:

```
RemoveHandler .phtml .php3 .php .pl .py .jsp .asp .htm .shtml .sh .cgi .dat
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:

```
<img src="uploads/nombreImagen.jpg" />
```

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:

```
<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:

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

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.

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

```php
<?php
  header("Expires: Sun, 31 Jan 2021 23:59:59 GMT");
  // tres horas
  $now = time();
  $horas3 = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 60 * 60 * 3);
  header("Expires: {$horas3}");
  // un año
  $now = time();
  $anyo1 = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", $now + 365 * 86440);
  header("Expires: {$anyo1}");
  // se marca como expirado (fecha en el pasado)
  $pasado = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");
  header("Expires: {$pasado}");
  // evitamos cache de navegador y/o proxy
  header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
  header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
  header("Cache-Control: no-store, no-cache, must-revalidate");
  header("Cache-Control: post-check=0, pre-check=0", false);
  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 limitació de 20 cookies por dominio y 300 en total en el navegador.

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

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

Destacar que el nombre no puede contener espacios ni el caracter ;. 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:

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

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**

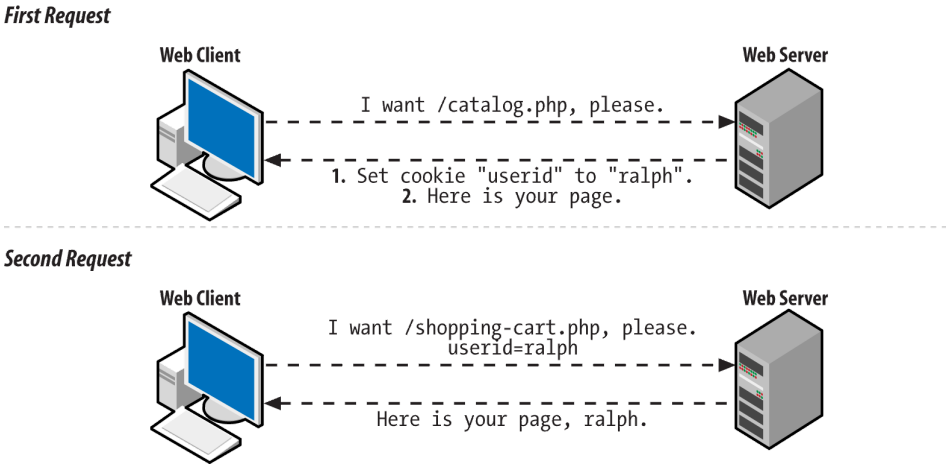
El tiempo de vida de las cookies puede ser tan largo como el sitio web en el que residen. Ellas seguirán ahí, incluso si el navegador está cerrado o abierto.

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

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

O que caduquen dentro de un periodo de tiempo deteminado:

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



Se utilizan para:

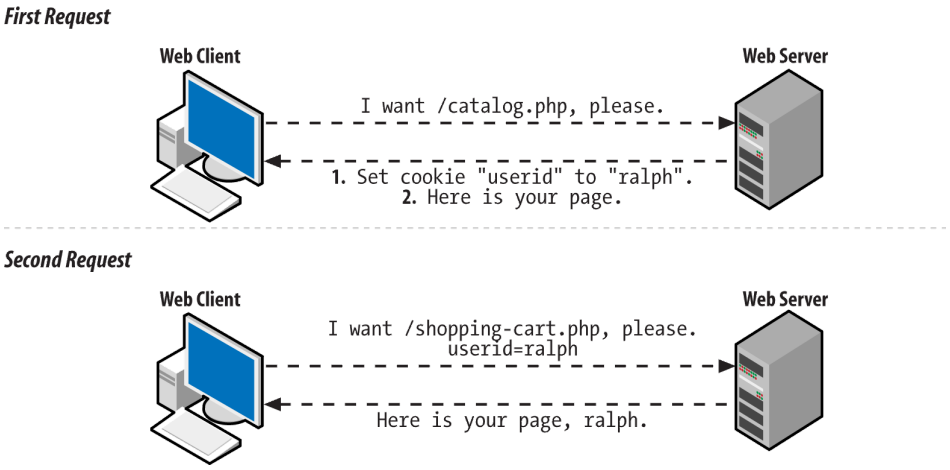
- Recordar los inicios de sesión.
- Almacenar valores temporales de usuario.
- Si un usuario está navegando por una lista paginada de artículos, ordenados de cierta manera, podemos almacenar el ajuste de la clasificación.

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:



Las operaciones que podemos realizar con la sesión son:

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

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

```php
<?php
    session_start(); // inicializamos
    $_SESSION["ies"] = "IES Severo Ochoa"; // asignación
    $instituto = $_SESSION["ies"]; // recuperación
    echo "Estamos en el $instituto ";
    ?>
    <br />
    <a href="sesion2.php">Y luego</a>
```

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

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

configurando la sesión en php.ini

Las siguiente 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 servidor de manera que apuntan 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.

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`:

```
<form action='login.php' method='post'>
  <fieldset>
    <legend>Login</legend>
    <div><span class='error'><?php echo $error; ?></span></div>
    <div class='fila'>
        <label for='usuario'>Usuario:</label><br />
        <input type='text' name='inputUsuario' id='usuario' maxlength="50" />
<br />
    </div>
    <div class='fila'>
        <label for='password'>Contraseña:</label><br />
        <input type='password' name='inputPassword' id='password'
maxlength="50" /><br />
    </div>
    <div class='fila'>
        <input type='submit' name='enviar' value='Enviar' />
    </div>
  </fieldset>
  </form>
```

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

```
<?php
// Comprobamos si ya se ha enviado el formulario
if (isset($_POST['enviar'])) {
    $usuario = $_POST['inputUsuario'];
    $password = $_POST['inputPassword'];

    // validamos que recibimos ambos parámetros
    if (empty($usuario) || empty($password)) {
        $error = "Debes introducir un usuario y contraseña";
        include "index.php";
    } else {
        if ($usuario == "admin" && $password == "admin") {
            // almacenamos el usuario en la sesión
```

```
            session_start();
            $_SESSION['usuario'] = $usuario;
            // cargamos la página principal
            include "main.php";
        } else {
            // Si las credenciales no son válidas, se vuelven a pedir
            $error = "Usuario o contraseña no válidos!";
            include "index.php";
        }
    }
}
```

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

```
<?php
    // Recuperamos la información de la sesión
    if(!isset($_SESSION)) {
        session_start();
    }

    // Y comprobamos que el usuario se haya autentificado
    if (!isset($_SESSION['usuario'])) {
       die("Error - debe <a href='index.php'>identificarse</a>.<br />");
    }
?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Listado de productos</title>
</head>
<body>
    <h1>Bienvenido <?= $_SESSION['usuario'] ?></h1>
    <p>Pulse <a href="logout.php">aquí</a> para salir</p>
    <p>Volver al <a href="main.php">inicio</a></p>
    <h2>Listado de productos</h2>
    <ul>
        <li>Producto 1</li>
        <li>Producto 2</li>
        <li>Producto 3</li>
    </ul>
</body>
</html>
```

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

```php
<?php
    // Recuperamos la información de la sesión
    session_start();

    // Y la destruimos
    session_destroy();
    header("Location: index.php");
?>
```

autenticación en producción

En la actualidad la autenticación de usuario no se realiza gestionando la sesión directamente, sino que se realiza mediante algún framekwork que abstrae 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