Unidad 5

Desarrollo de aplicaciones web
con PHP

1. Variables del servidor en PHP

Las **variables de servidor** nos permiten acceder al tema relacionados con el servidor como: el host del servidor, el encabezado, el software del servidor, etc. Conocer estas variables nos puede ser muy útil en el desarrollo de nuestras aplicaciones.

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

1. Variables del servidor en PHP

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

PHP_SELF: nombre del script ejecutado, relativo al document root

SERVER_SOFTWARE: (p.ej: Apache)

SERVER_NAME: dominio, alias DNS

REQUEST_METHOD: GET

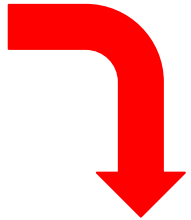
REQUEST_URI: URI, sin el dominio

QUERY_STRING: todo lo que va después de ? en la URL

1. Variables del servidor en PHP

```php
<?php
echo $_SERVER["PHP_SELF"]."<br>";
echo $_SERVER["SERVER_SOFTWARE"]."<br>";
echo $_SERVER["SERVER_NAME"]."<br>";

echo $_SERVER["REQUEST_METHOD"]."<br>";
echo $_SERVER["REQUEST_URI"]."<br>";
echo $_SERVER["QUERY_STRING"]."<br>";
?>
```



/DWEES/Unidad4/variableservidorejemplo.php
Apache/2.4.54 (Win64) OpenSSL/1.1.1p PHP/8.1.10
localhost
GET
/DWEES/Unidad4/variableservidorejemplo.php

2. Formularios

En la unidad 1 vimos una introducción a los formularios.
Vimos que la creación de formularios se divide en 2 pasos:

1. Generar un formulario con action = ‘archivo.php’ method=GET
2. En el archivo .php leer los datos con $_GET [‘nombreVar’]

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

- **GET**: los parámetros se pasan en la URL. Características:
    - <2048 caracteres, sólo ASCII
    - Permite almacenar la dirección completa (marcador / historial)
    - Idempotente: dos llamadas con los mismos datos siempre deben dar el mismo resultado
    - El navegador puede cachear las llamadas

2. Formularios

En la unidad 1 vimos una introducción a los formularios.
Vimos que la creación de formularios se divide en 2 pasos:

1. Generar un formulario con action = ‘archivo.php’ method=GET
2. En el archivo .php leer los datos con $_GET [‘nombreVar’]

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

- **POST:** parámetros ocultos (no encriptados). Características:
  - Sin límite de datos, permite datos binarios.
  - No se pueden cachear
  - No idempotente → actualizar la BBDD

2. Formularios

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"]
```

2. Formularios

Vamos a trabajar con el siguiente ejemplo de formulario:

```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>
```

2.1 Validación de formularios

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

- En el cliente mediante JS (JavaScript)
- 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...
}
```

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.

2.2 Parámetro multivalor

Existen elementos HTML que envían varios valores:

- Select multiple
- Checkbox

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

```
<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 />
```

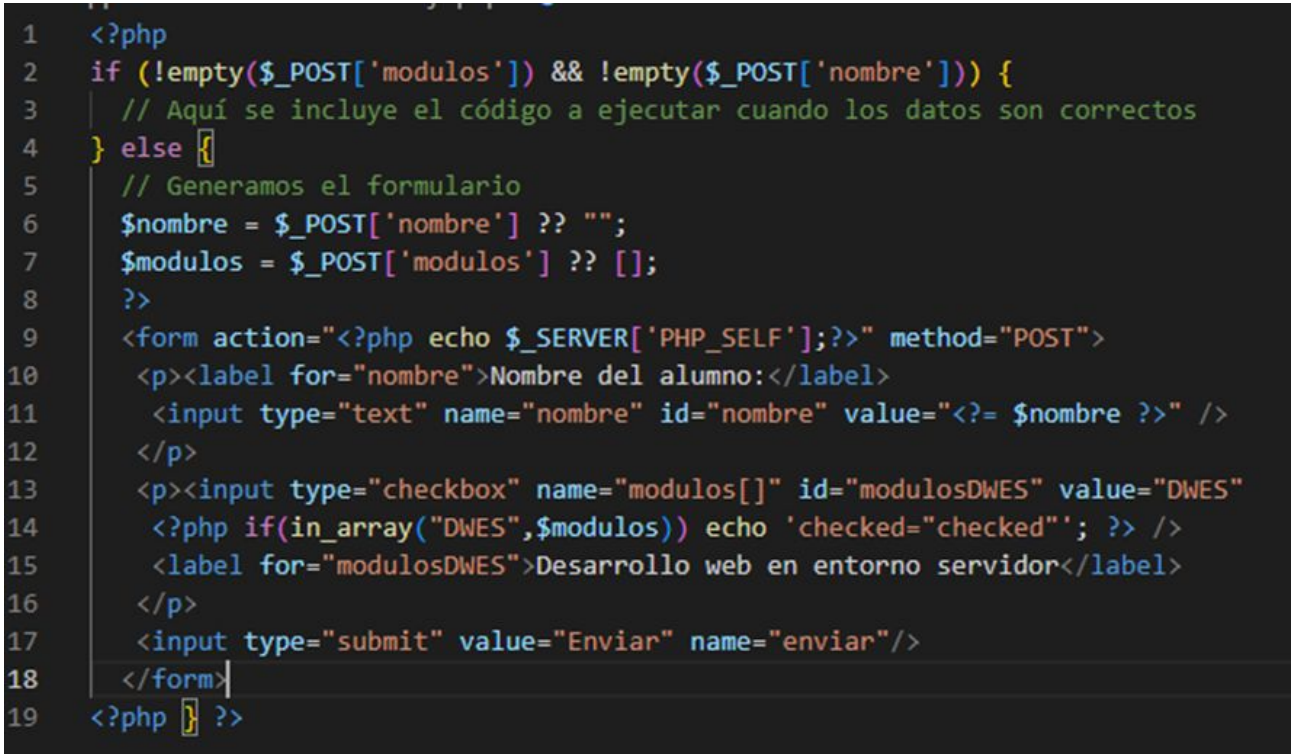
`<select>` representa un control que muestra un menú de opciones. Las opciones contenidas en el menú son representadas por elementos `<option>`.

`multiple` indica que se pueden seleccionar múltiples opciones de la lista.

`<input type=''checkbock”>` es un elemento de entrada que te permite insertar un vector o array de valores. El atributo `value` es usado para definir el valor enviado por el checkbox.

2.3 Rellenar formularios

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'] ?? [];
  ?>
  <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>
   <input type="submit" value="Enviar" name="enviar"/>
  </form>
<?php } ?>
```

2.4 Subiendo archivos

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

```html
<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>
```

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>";
    }
}
?>
```

2.4 Subiendo archivos

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.

Actividades propuestas

- **Classroom → Unidad 5: Tarea 1**
- **Importante: en la carpeta DWEES creada anteriormente, crea una carpeta que se llame U5Tarea1**

3. Cabeceras de respuesta

Debe ser lo primero a devolver. Se devuelve mediante la función `header`.

Mediante las cabeceras podemos configurar el tipo de contenido, tiempo de expiración, redireccionar el navegador, especificar errores HTTP, etc.

Por ejemplo:

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

Saldrá:

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

3. Cabeceras de respuesta

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");
```

4. Gestión del estado

HTTP es un protocolo stateless, sin estado. Es decir, no almacena información sobre operaciones anteriores ni se hace referencia a ellas. Cada operación se lleva a cabo desde cero, como si fuera la primera vez.

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.

4.1 Cookies

Una **cookie** es un fichero de texto que un sitio web guarda en el entorno del usuario del navegador. Su uso más típico es el almacenamiento de las preferencias del usuario (por ejemplo, el idioma en que se deben mostrar las páginas) para que no tenga que volver a indicarlas la próxima vez que visite el sitio.

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

4.1 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 almacenar una cookie en el navegador del usuario, puedes utilizar la función `setcookie`

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

El único **parámetro obligatorio** que tienes que usar es el **nombre de la cookie.**
Admite varios parámetros más opcionales: https://www.php.net/manual/es/function.setcookie.php

Destacar que el nombre no puede contener espacios ni el caracter **;**

Respecto al contenido de la cookie, no puede superar los 4 KB.

4.1 Cookies

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

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

4.1 Cookies

Tiempo de vida de una cookie

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

O que caduquen dentro de un periodo de tiempo determinado:

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

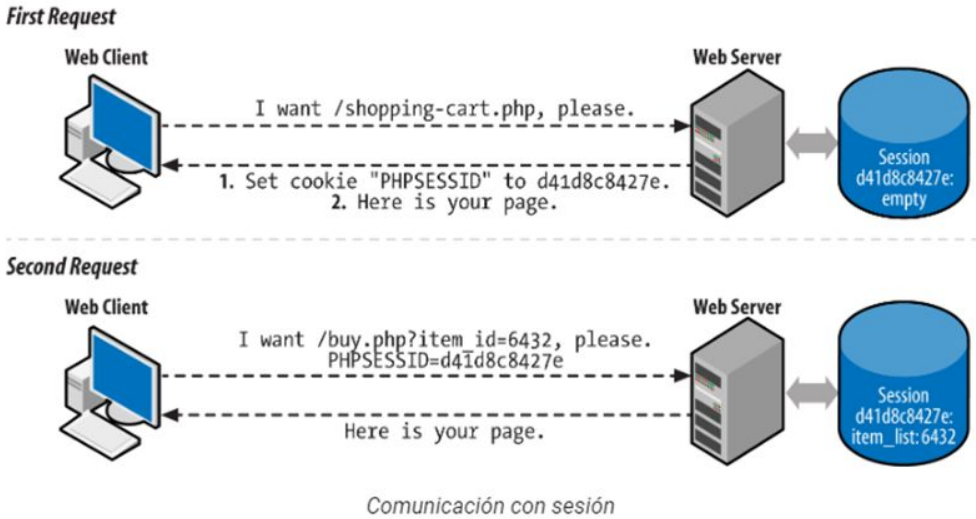
4.2 Sesión

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



Comunicación con sesión

4.2 Sesión

Las **operaciones que podemos utilizar con la sesión** son las siguientes:

`session_start();` carga la sesión

`session_id()` devuelve el id

`$_SESSION[clave] = valor;` inserción

`session_destroy();` destruye la sesión

`unset($_SESSION[clave];` borrado

4.2 Sesión

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

```php
<?php
session_start(); // inicializamos
$_SESSION["ies"] = "IES Kursaal"; // 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 `sesion2.php`

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

Actividades propuestas

- **Classroom → Unidad 5: Tarea 2**
- **Importante: en la carpeta DWEES creada anteriormente, crea una carpeta que se llame U5Tarea2**

5. Autentificación 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, preparamos 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 con un **ejemplo** de código cada paso del proceso.

5. Autentificación de usuarios

Comenzamos con el archivo `index.php`

```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>
```

5. Autentificación de usuarios

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

```php
<?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";
        }
    }
}
?>
```

5. Autentificación de usuarios

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

```php
<?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>
```

5. Autentificación de usuarios

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");
?>
```

Nota: en la actualidad la autenticación de usuario no se realiza gestionando la sesión directamente, si no que se realiza mediante algún framework que abstrae todo el proceso o la integración de mecanismos de autenticación tipo OAuth, como estudiaremos más adelante

Actividades propuestas

- **Classroom → Unidad 5: Tarea 3**
- **Importante: en la carpeta DWEES creada anteriormente, crea una carpeta que se llame U5Tarea3**