INDICE

Programación Web

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 vía upload

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

PHP_SELF: nombre del script ejecutado, relativo al document root (p.ej: /tienda/carrito.php)

SERVER_SOFTWARE: (p.ej: Apache)

SERVER_NAME: dominio, alias DNS (p.ej: www.elche.es)

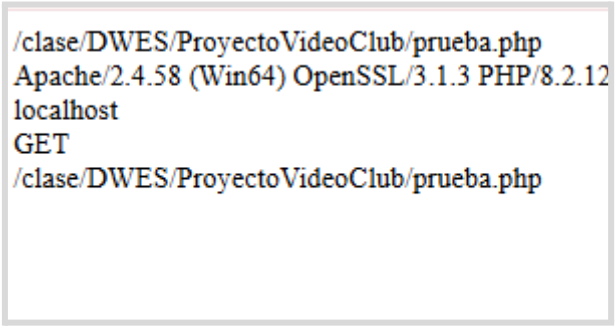
REQUEST_METHOD: GET

REQUEST_URI: URI, sin el dominio

QUERY_STRING: todo lo que va después de ? en la URL (p.ej: heroe=Batman&nombre=Bruce)

pruebaServer.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?heroe=Batman
echo $_SERVER["QUERY_STRING"]."<br>"; // heroe=Batman
?>
```



Otras propiedades relacionadas:

- PATH_INFO: ruta extra tras la petición. 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
?>
```

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

FORMULARIO DE EJEMPLO

```
<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>
```

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.

Librerías de validación

Existen diversas librerías que facilitan la validación de los formularios, como son r*espect/validation* o *particle/validator*. Cuando estudiemos Laravel profundizaremos en la validación de forma declarativa.

1. Respect/Validation

Respect/Validation es una librería potente y flexible para validar datos en PHP.--> Permite combinar reglas de validación de manera fluida, lo que hace que el código sea legible y declarativo.

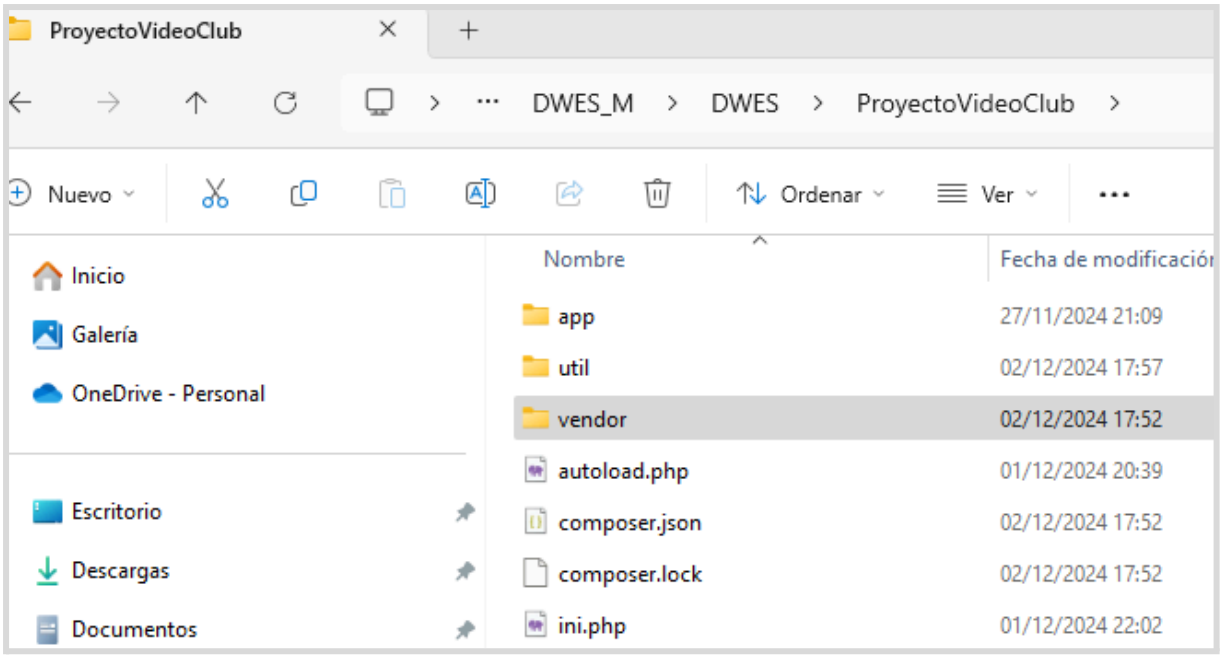
Instalación → en la terminal de comando (y aprovechando que ya tenemos composer instalado en el ordenador).

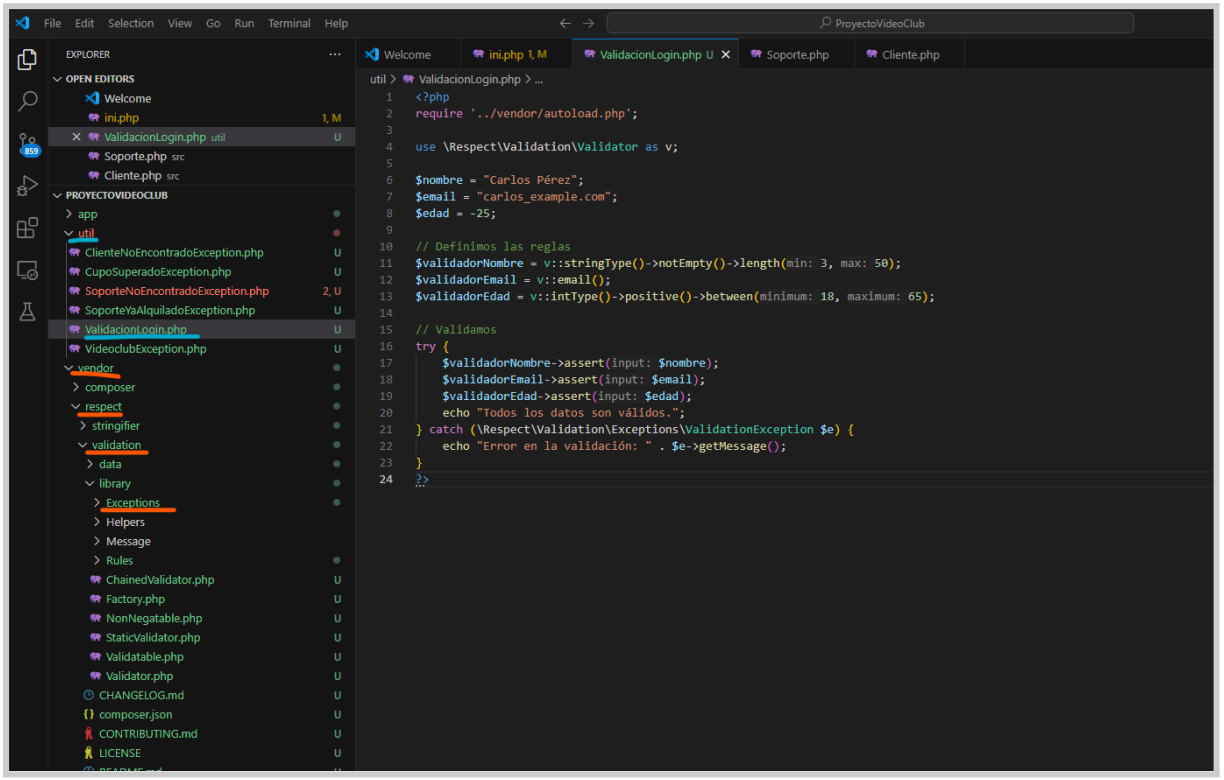
```
PS C:\Users\PC Maria\Documents\DWES\ProyectoVideoClub>composer require respect/validation
```

Ejemplo Práctico (tiene en cuenta la distribución de carpetas de mi proyecto)

Validación de los datos de un formulario de registro: nombre, email y edad.

```php
<?php
require '../vendor/autoload.php';

use \Respect\Validation\Validator as v;

$nombre = "Carlos Pérez";
$email = "carlos@example.com";
$edad = 25;

// Definimos las reglas
$validadorNombre = v::stringType()->notEmpty()->length(3, 50);
$validadorEmail = v::email();
$validadorEdad = v::intType()->positive()->between(18, 65);

// Validamos
try {
    $validadorNombre->assert($nombre);
    $validadorEmail->assert($email);
    $validadorEdad->assert($edad);
    echo "Todos los datos son válidos.";
} catch (\Respect\Validation\Exceptions\ValidationException $e) {
    echo "Error en la validación: " . $e->getMessage();
}
?>
```





PARÁMETROS MULTIVALOR

Existen elementos HTML que envían varios valores:

- select multiple
- checkbox

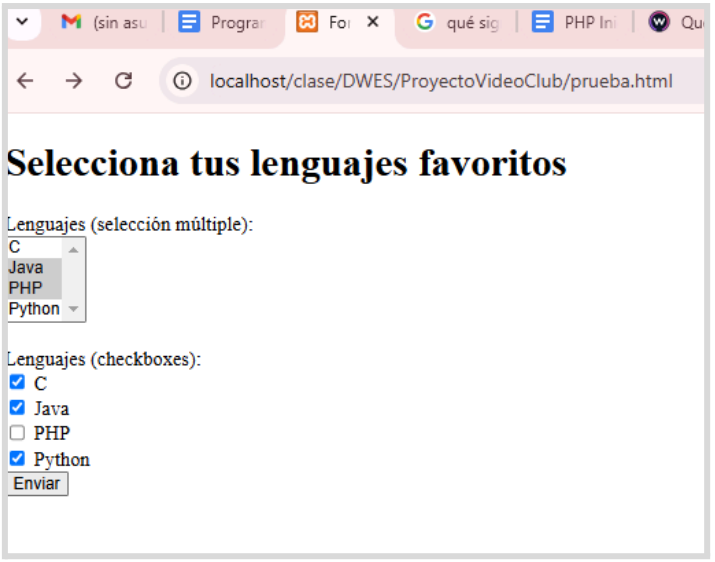
Para recoger los datos, el nombre del elemento debe ser un array.

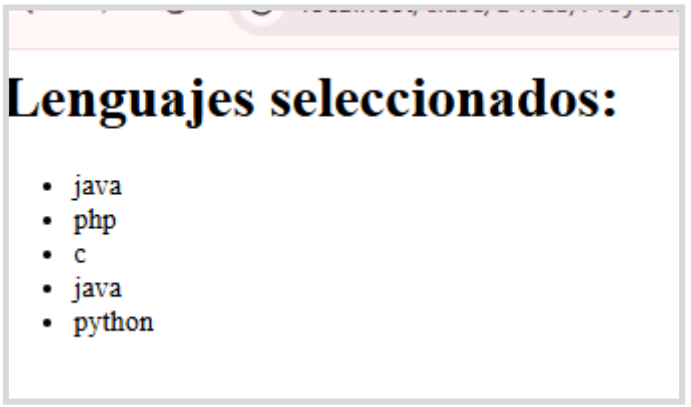
prueba.html

```html
<!DOCTYPE html>

<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulario de Lenguajes</title>
</head>
    <h1>Selecciona tus lenguajes favoritos</h1>
    <form action="prueba.php" method="POST">
        <label for="lenguajes">Lenguajes (selección múltiple):</label><br>
        <select name="lenguajes[]" id="lenguajes" multiple="true">
            <option value="c">C</option>
            <option value="java">Java</option>
            <option value="php">PHP</option>
            <option value="python">Python</option>
        </select><br><br>
        <label>Lenguajes (checkboxes):</label><br>
        <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 />
        <button type="submit">Enviar</button>
    </form>
</body>
</html>
```

prueba.php

```php
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Verificar si se enviaron datos desde el formulario
    if (isset($_POST['lenguajes'])) {
        $lenguajesSeleccionados = $_POST['lenguajes'];

        // Imprimir los lenguajes seleccionados
        echo "<h1>Lenguajes seleccionados:</h1>";
        echo "<ul>";
        foreach ($lenguajesSeleccionados as $lenguaje) {
            echo "<li>" . htmlspecialchars($lenguaje) . "</li>";
        }
        echo "</ul>";
    } else {
        echo "<h1>No se seleccionó ningún lenguaje.</h1>";
    }
} else {
    echo "<h1>Acceso no permitido.</h1>";
}
?>
```



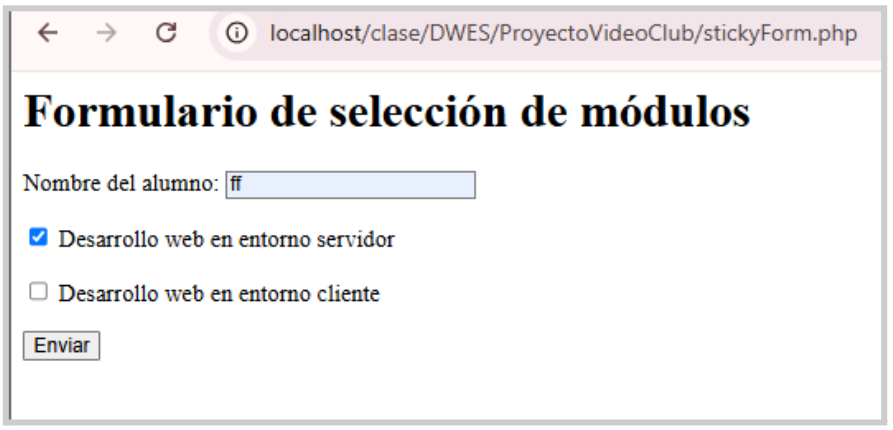


VOLVIENDO A RELLENAR UN FORMULARIO

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
    $nombre = htmlspecialchars($_POST['nombre']);
    $modulos = $_POST['modulos'];

    echo "<h1>Datos recibidos</h1>";
    echo "<p><strong>Nombre del alumno:</strong> $nombre</p>";
    echo "<p><strong>Módulos seleccionados:</strong></p>";
    echo "<ul>";
    foreach ($modulos as $modulo) {
        echo "<li>" . htmlspecialchars($modulo) . "</li>";
    }
    echo "</ul>";
} else {
    // Generamos el formulario
    $nombre = $_POST['nombre'] ?? "";
    $modulos = $_POST['modulos'] ?? [];
    ?>

    <h1>Formulario de selección de módulos</h1>
    <?php if ($_SERVER['REQUEST_METHOD'] === 'POST'): ?>
        <p style="color: red;">Por favor, completa todos los campos antes de enviar el formulario.</p>
    <?php endif; ?>
    <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="<?= htmlspecialchars($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 } ?>
```



SUBIENDO ARCHIVOS

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

Configuración en php.ini

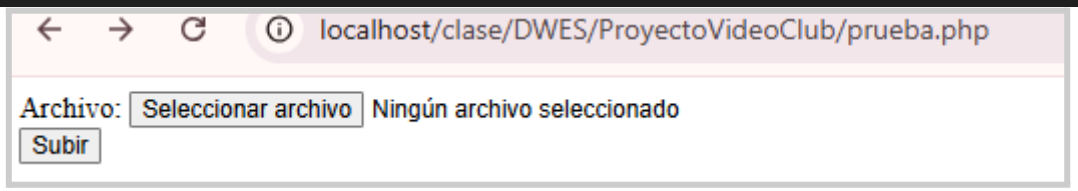
- file_uploads: on / off
- upload_max_filesize: 2M
- upload_tmp_dir: directorio temporal. No es necesario configurarlo, cogerá el predeterminado del sistema

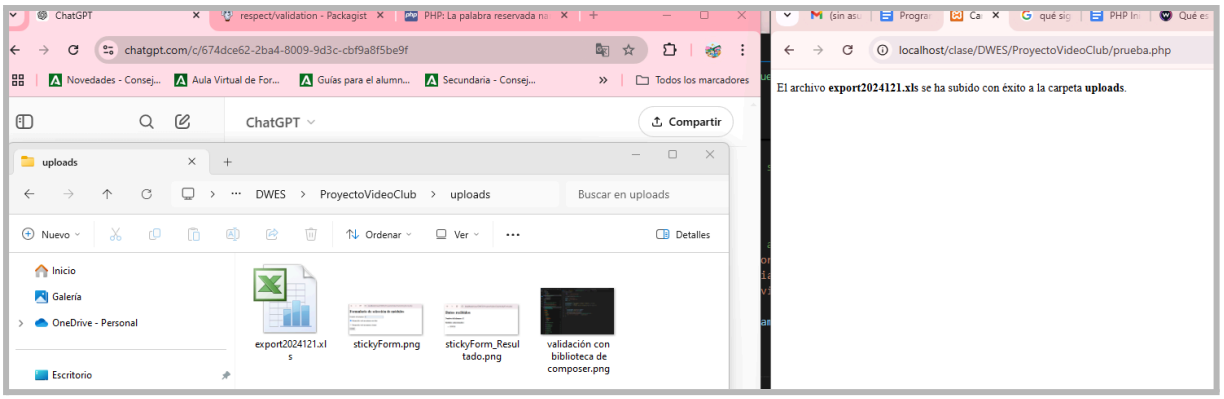
- post_max_size: tamaño máximo de los datos POST. Debe ser mayor a upload_max_filesize.
- max_file_uploads: número máximo de archivos que se pueden cargar a la vez.
- max_input_time: tiempo máximo empleado en la carga (GET/POST y upload → normalmente se configura en 60)
- memory_limit: 128M
- max_execution_time: tiempo de ejecución de un script (no tiene en cuenta el upload)

Para cargar los archivos, accedemos al array $_FILES:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
                <meta    name="viewport"    content="width=device-width,
initial-scale=1.0">
    <title>Carga de Archivos</title>
</head>
<body>
    <?php
       $mostrarFormulario  =  true;  //  Control  para  mostrar  o  no  el
formulario
                if    ($_SERVER['REQUEST_METHOD']    ===    'POST'    &&
isset($_POST['btnSubir']) && $_POST['btnSubir'] === 'Subir') {
        // Comprobamos si el archivo fue subido
        if (is_uploaded_file($_FILES['archivoEnviado']['tmp_name'])) {
            $nombre = $_FILES['archivoEnviado']['name'];
            $destino = __DIR__ . "/uploads/{$nombre}";

            // Intentamos mover el archivo al destino
                                                                    if
(move_uploaded_file($_FILES['archivoEnviado']['tmp_name'], $destino)) {
                    echo "<p>El archivo <strong>$nombre</strong> se ha
subido con éxito a la carpeta <strong>uploads</strong>.</p>";
                $mostrarFormulario = false; // Ocultamos el formulario
tras la subida exitosa
            } else {
                    echo "<p>Error: No se pudo mover el archivo al
destino.</p>";
            }
        } else {
            echo "<p>Error: No se ha recibido ningún archivo.</p>";
```

```
        }
    }
    // Mostramos el formulario solo si es necesario
    if ($mostrarFormulario):

    ?>

    <!-- Formulario para subir archivos -->
        <form enctype="multipart/form-data" action="<?php echo
htmlspecialchars($_SERVER['PHP_SELF']); ?>" method="POST">
        <label for="archivoEnviado">Archivo:</label>
        <input name="archivoEnviado" id="archivoEnviado" type="file" />
        <br />
        <input type="submit" name="btnSubir" value="Subir" />
    </form>
        <?php endif; ?>
</body>
</html>
```





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.

CABECERAS DE RESPUESTAS

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.

index.php

```
<?php
// Incluir el archivo de cabeceras para control de caché
require 'cache_headers.php';

// Ahora puedes incluir el resto del contenido
?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
                <meta    name="viewport"    content="width=device-width,
initial-scale=1.0">
    <title>Mi Proyecto</title>
</head>
<body>
    <h1>Bienvenido a mi proyecto</h1>
      <p>Este contenido no será almacenado en la caché del navegador
gracias al control de cabeceras.</p>
</body>
</html>
```

`cache_headers.php` ->debe ser incluido antes de cualquier contenido HTML o texto. NOTA, Este código debe probarse en diferentes navegadores y revisar las herramientas de desarrollo (como la pestaña "Red" en el navegador) para verificar que las cabeceras se están enviando correctamente.

```
<?php
// Configuración de las cabeceras para control de caché

// Expiración fija en el pasado (enero de 2021)
header("Expires: Sun, 31 Jan 2021 23:59:59 GMT");

// Expiración en tres horas
$now = new DateTime('now', new DateTimeZone('UTC'));
```

```php
$horas3 = $now->add(new DateInterval('PT3H'))->format('D, d M Y H:i:s')
. ' GMT';
header("Expires: {$horas3}");

// Expiración en un año
$now = new DateTime('now', new DateTimeZone('UTC'));
$anyo1 = $now->add(new DateInterval('P1Y'))->format('D, d M Y H:i:s') .
' GMT';
header("Expires: {$anyo1}");

// Fecha en el pasado para marcar como expirado
$pasado = (new DateTime('now', new DateTimeZone('UTC')))
    ->sub(new DateInterval('P1D'))
    ->format('D, d M Y H:i:s') . ' GMT';
header("Expires: {$pasado}");

// Evitar caché del 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");
?>
```

GESTIÓN DEL ESTADO

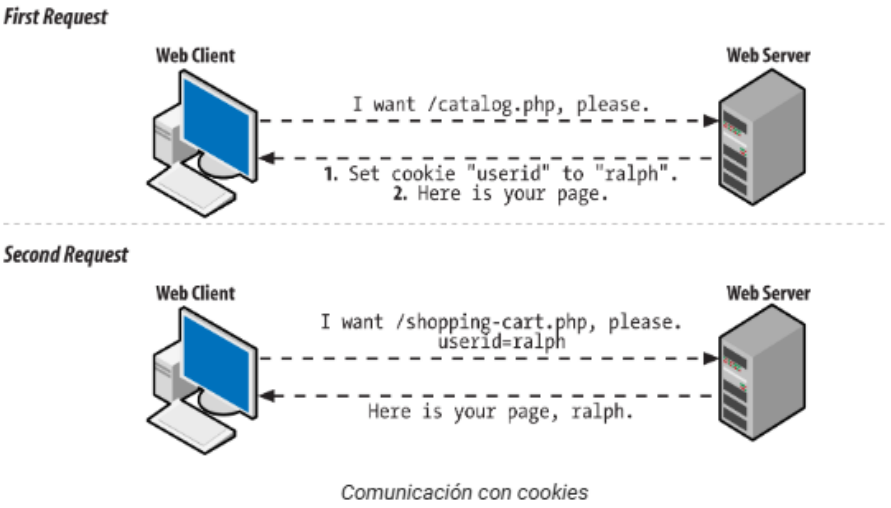
HTTP es un *protocolo stateless,* sin estado. Por ello, *se simula el estado mediante el uso de cookies, tokens o la sesión*.

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

La sesión en el servidor web.



Comunicación con cookies

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.

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:

index.php

```
<?php
```

```php
// Incluir el control de cabeceras para evitar caché
//require 'cache_headers.php'; // Asegúrate de que este archivo existe y contiene el código de control de cabeceras

// Inicializar el contador de accesos
$accesosPagina = 1;

// Si existe la cookie, recuperamos su valor
if (isset($_COOKIE['accesos'])) {
    $accesosPagina = (int)$_COOKIE['accesos'] + 1; // Incrementamos el valor
}

// Guardamos el nuevo valor en la cookie con una expiración de 1 año
setcookie('accesos', $accesosPagina, time() + (365 * 24 * 60 * 60), '/');

// Mostrar un mensaje al usuario
?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Contador de Accesos</title>
</head>
<body>
```

```
    <h1>Bienvenido a nuestra página</h1>

        <p>Has accedido a esta página <strong><?= $accesosPagina
?></strong> veces.</p>

    <p>Si deseas reiniciar el contador, <a href="reset_cookie.php">haz
clic aquí</a>.</p>

</body>

</html>
```

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

```
setcookie('accesos', '', time() - 3600, '/'); // Fecha en el pasado
para borrarla
```

O que caduquen dentro de un periodo de tiempo determinado:

```
setcookie('accesos', '', time() +3600); // Para que caduque dentro de
una hora
```

reset_cookie.php (para borrar usamos caducidad en el pasado)

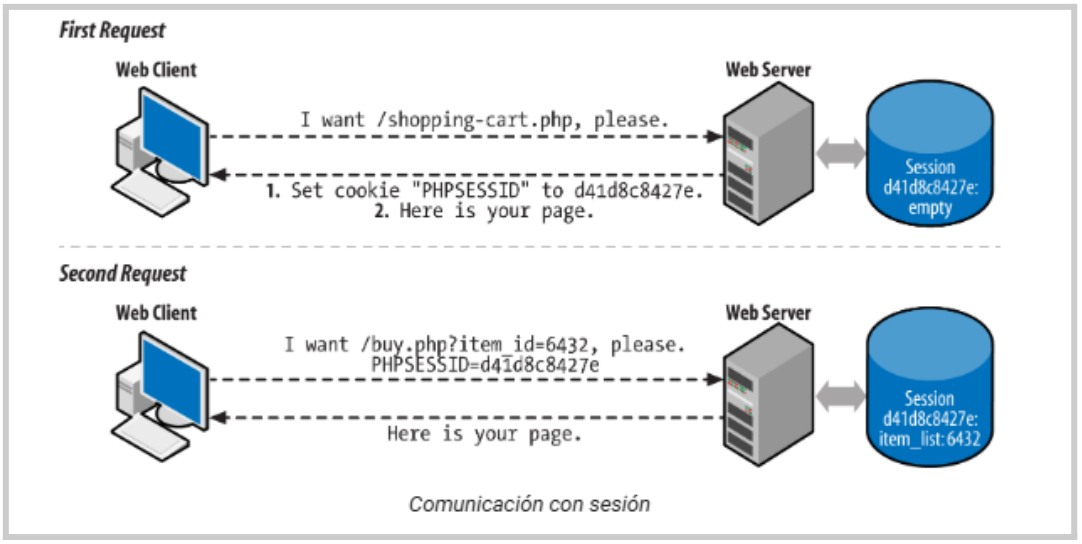
```
<?php

// Eliminar la cookie 'accesos'

setcookie('accesos', '', time() - 3600, '/'); // Fecha en el pasado
para borrarla



// Redirigir al usuario de vuelta a la página principal

header("Location: prueba2.php");

exit;
```

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

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

NOTA,

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

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 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.

index.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
                <meta    name="viewport"    content="width=device-width,
initial-scale=1.0">
    <title>Contador de Accesos</title>
</head>
<body>
<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>
</body>
</html>
```

login.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";
        }
    }
}
```

main.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'])) {
```

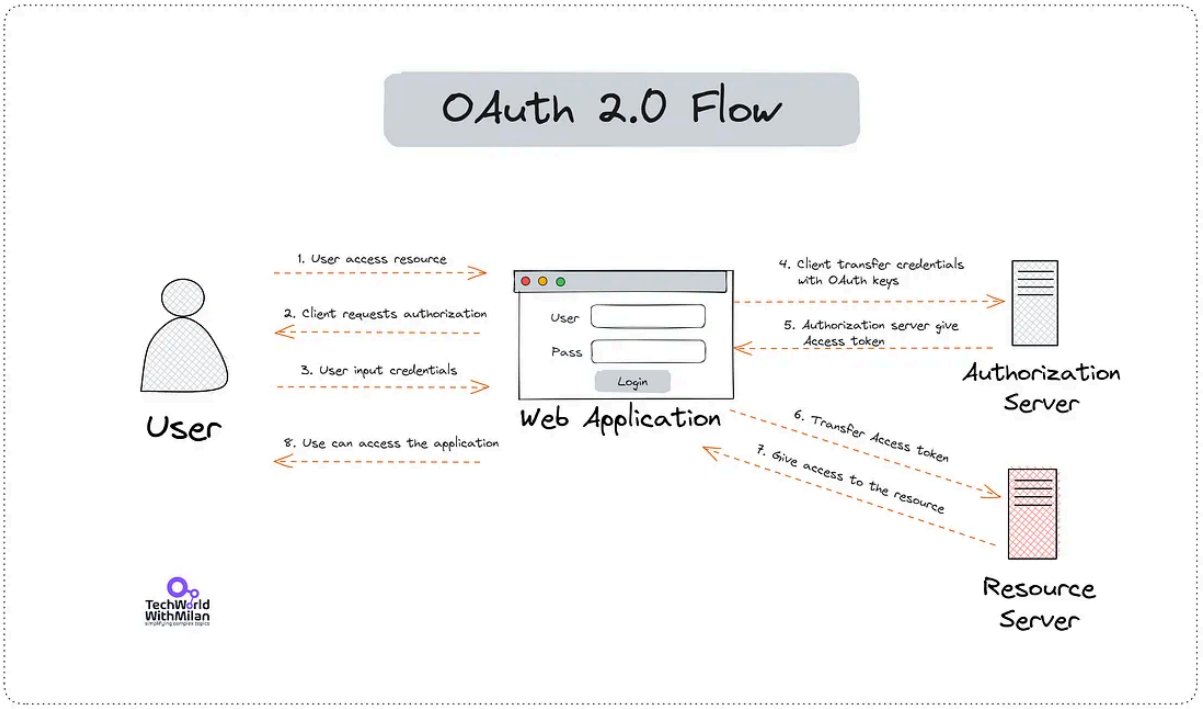
```
            die("Error - debe <a href='index.php'>identificarse</a>.<br
/>");
    }
?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
                <meta    name="viewport"    content="width=device-width,
initial-scale=1.0">
    <title>Listado de productos</title>
</head>
<body>
    <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>
```

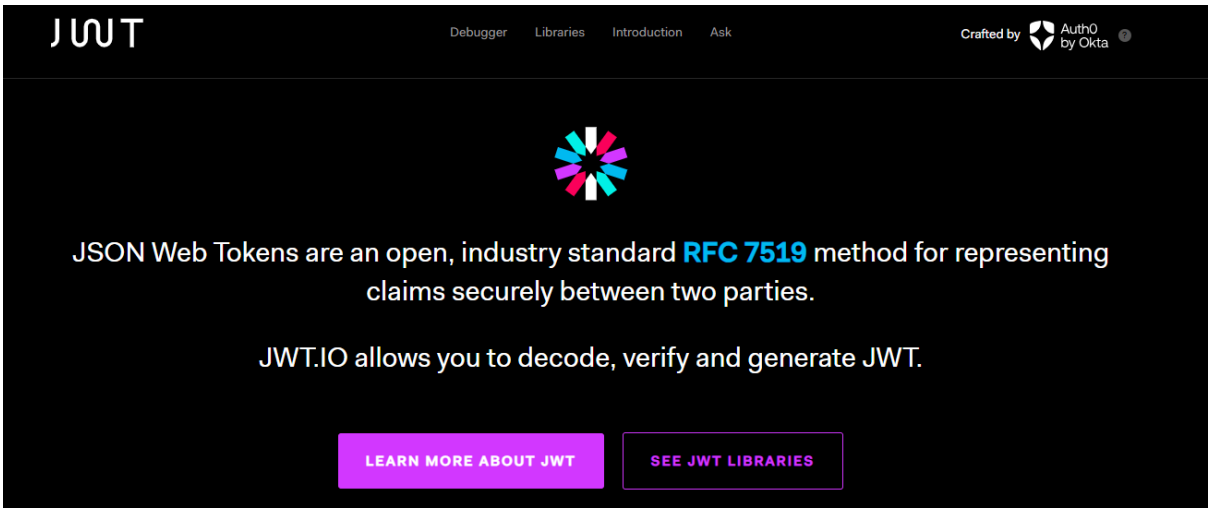
logout.php

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

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

IMPORTANTE, (como se ha indicado al principio del tema) *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 / JWT, como estudiaremos en la última unidad mediante Laravel.*

https://frontegg.com/blog/oauth-vs-jwt


OAuth 2.0 Flow
User
1. User access resource
2. Client requests authorization
3. User input credentials
8. Use can access the application
User
Pass
Login
Web Application
4. Client transfer credentials with OAuth keys
5. Authorization server give Access token
Authorization Server
6. Transfer Access token
7. Give access to the resource
Resource Server
TechWorld WithMilan


JWT
Debugger
Libraries
Introduction
Ask
Crafted by Auth0 by Okta
JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.
JWT.IO allows you to decode, verify and generate JWT.
LEARN MORE ABOUT JWT
SEE JWT LIBRARIES