

## FUNDAMENTOS DE PHP - 03

### **Formularios y Tipos de peticiones, Sesiones y Cookies.**

Los formularios se utilizan para que un sitio web pueda solicitar información al usuario. Formulario HTML puede contener diferentes campos de entrada de información (textual, numérico, rango de valores, selección de opciones, etc.)

<https://developer.mozilla.org/es/docs/Web/HTML/Elemento/form>

La etiqueta raíz del formulario <form> debemos especificar el atributo method (método) que indica si la información introducida se va a enviar en una petición de tipo post o get, también tenemos el atributo action que indica a qué página se va a enviar la petición (puede ser a la página actual o a otra).

### **Formularios y Tipos de peticiones**

A continuación, se muestra un ejemplo de un formulario básico en HTML. Solicita 3 datos al usuario y envía el formulario a través de una petición GET a la página: controlget.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form method="get" action="controlget.php">
        Usuario: <input type="text" name="usuario"> <br>
        Password: <input type="password" name="password"> <br>
        <input type="checkbox" name="contabilizar"> Contabilizar <br>
        <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

Al completar los datos del formulario y enviarlos vemos que el valor de cada parámetro viaja en la URL de la petición http, así es como funcionan los parámetros GET.



El parámetro definido con el input de tipo checkbox es un poco especial, solo se incluye en la respuesta si el checkbox está marcado, si no está marcado el parámetro será nulo (no aparece, no tiene ningún valor).

En la página controlget.php procesamos los parámetros que esperamos recuperar de la petición GET. Los parámetros de la posición get viajan en un array asociativo, para recuperar un parámetro concreto utilizamos `$_GET["nombre del parámetro"]`

Sí no estamos seguro de que la petición vaya a contener ese parámetro lo mejor es no utilizarlo directamente y comprobar que el parámetro realmente existe, para ello podemos usar varias funciones por ejemplo **`$isset(variable)`** que comprueba que una variable este definida y no sea nula o **`empty()`**, En el archivo: `controlget.php` se muestra como recuperar los parámetros GET enviados desde el formulario.

```
<?php
    // isset: comprobar que recibe el parámetro
    if ( isset( $_GET["usuario"] ) ){
        $usuario = $_GET["usuario"];
        echo "usuario: ".$usuario."</br>";
    }
    if ( isset( $_GET["password"] ) ){
        $password = $_GET["password"];
        echo "password: ".$password."</br>";
    }
    // lo recibe únicamente si el usuario marca el check.
    if ( isset( $_GET["contabilizar"] ) ){
        $contabilizar = $_GET["contabilizar"];
        echo "contabilizar: ".$contabilizar."</br>";
    }
?
>
```

Sí en lugar de especificar un **method** de tipo: GET en el formulario especificamos el tipo **POST** los parámetros van a viajar en el cuerpo de la petición.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form method="post" action="controlpost.php">
        Usuario: <input type="text" name="usuario"> <br>
        Password: <input type="password" name="password"> <br>
        <input type="checkbox" name="contabilizar"> Contabilizar <br>
        <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

Al pulsar el botón de enviar se realiza una petición POST, la información enviada en el cuerpo de la petición post no se puede ver la URL, pero sí que se puede ver si utilizamos un analizador de tráfico de red (Por ejemplo: del navegador Chrome, pulsado F12 y abriendo la pestaña Network), estos analizadores muestran todos los detalles de la petición incluyendo la cabecera y el cuerpo.

DevTools - localhost/1803183/controlpost.php

Elements Console Sources Network Performance Memory

Preserve log Disable cache Online ▾ ⌂ ⌃ ⌄

Filter Hide data URLs All XHR JS CSS Img Media

50 ms	100 ms	150 ms
-------	--------	--------

Name: controlpost.php

Headers Preview Response Initiator

Referer: http://localhost/1803183/  
 Sec-Fetch-Dest: document  
 Sec-Fetch-Mode: navigate  
 Sec-Fetch-Site: same-origin  
 Sec-Fetch-User: ?1  
 Upgrade-Insecure-Requests: 1  
 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.122 Safari/537.36

Form Data view source view URL encoding

usuario: Antonio
password: secreto
contabilizar: on

2 requests 634 B transferred 128 E

La forma de recuperar los datos en: controlpost.php será casi la misma de antes, pero en lugar de leer el array `$_GET` tendremos que leer el array `$_POST`

```
<?php
    // isset: comprobar que recibe el parámetro
    if ( isset( $_POST["usuario"] ) ){
        $usuario = $_POST["usuario"];
        echo "usuario: ".$usuario."</br>";
    }

    if ( isset( $_POST["password"] ) ){
        $password = $_POST["password"];
        echo "password: ".$password."</br>";
    }

    // lo recibe únicamente si el usuario marca el check.
    if ( isset( $_POST["contabilizar"] ) ){
        $contabilizar = $_POST["contabilizar"];
        echo "contabilizar: ".$contabilizar."</br>";
    }
?>
```

La variable `$_REQUEST`, contiene los parámetros de ambas: `$_POST` y `$_GET` además también contiene los parámetros almacenados en las Cookies (`$_COOKIE`).

## Sesiones

Una sesión es conjunto de valores almacenados que se asigna a un cliente durante un tiempo determinado, este conjunto de valores está asociado de forma única a ese cliente concreto y puede ser utilizado desde diferentes páginas.

Toda la información de la sesión se almacena en el Servidor, pero cada petición del cliente incluye un parámetro ID que se permite identificar al Servidor a que sesión está haciendo referencia, normalmente esa ID la almacena el cliente en una Cookie. Las sesiones se destruyen o caducan cuando pasa un cierto periodo de tiempo sin actividad, esto es algo que podemos apreciar cuando estamos navegando en muchos sitios web "su sesión ha caducado vuelva a identificarse".

La sesión se utiliza de forma muy recurrente en las aplicaciones web para guardar información del usuario, por ejemplo: si está identificado en el sitio, su nombre de usuario, sus preferencias y otros datos de carácter temporal dependientes de la ejecución del sitio.

Para que el servidor cree (o reanude) una sesión al usuario hace falta utilizar la función `session_start()`, a partir de que la sesión ha sido creada podemos manejar la sesión con el array asociativo `$_SESSION`. Antes de acceder a un elemento del array `$_SESSION` también es muy recomendable comprobar si el elemento existe (de la misma forma que hacíamos con GET y POST). El siguiente es el archivo: `index.php`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form method="post" action="index.php">
        Usuario: <input type="text" name="usuario"> <br>
        <input type="submit" value="Enviar">
    </form>
</body>
<?php
    session_start();
    if ( isset($_POST["usuario"] ) ){
        if( $_POST[usuario] == "admin" ){
            $_SESSION["validado"] = 1;
            // redirección a controlsession.php
            header("Location: controlsession.php");
            die();
        }
    }
?>
</html>
```

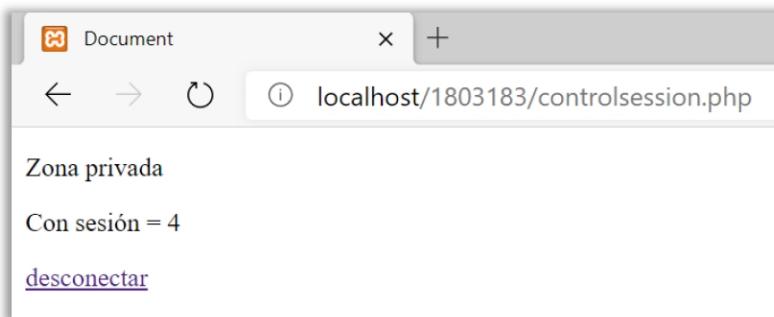
En este ejemplo tenemos un formulario POST de identificación, se espera que el usuario introduzca el nombre "admin", si lo introduce se guarda el valor "validado" = 1 en `$_SESSION` y se redirección al usuario a la página: `controlsession.php`

El siguiente es el archivo: controlsession.php

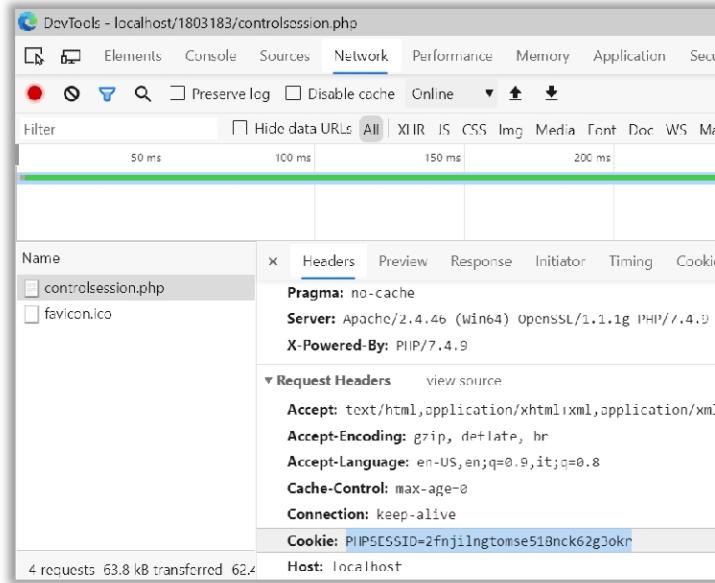
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<?php
    session_start();
    if ( isset($_SESSION["validado"] ) &&
        $_SESSION["validado"] == 1 ){
        if ( isset($_SESSION["contador"]) ) {
            // Si hay contador , contador + 1
            $_SESSION["contador"] = $_SESSION["contador"] + 1;
        } else {
            // Si no hay contador lo inicializamos a 1
            $_SESSION["contador"] = 1;
        }
    }else {
        header("Location: index.php");
        die();
    }
?>
<body>
    <p>Zona privada</p>
    <p>Con sesión = <?php echo $_SESSION["contador"] ?> </p>
    <a href="cerrar.php"> desconectar </a>
</body>
</html>
```

En esta página se accede a la `$_SESSION` para comprobar si esta almacenado el valor "validado" = 1, en ese caso quiere decir que el usuario se ha identificado y comenzamos a contar el número de veces que accede a la página. El contador de accesos se guarda en `$_SESSION["contador"]`, la primera vez que usamos el contador lo tenemos que inicializar con el valor 1. Sí por el contrario la `$_SESSION["validado"]` no ha sido establecido significa que el usuario está intentando acceder a: `controlsession.php` sin haberse identificado previamente, redirigimos al usuario a: `index.php`.

Al realizar varias peticiones (actualizando el navegador con la tecla F5) en la web: `controlsession.php` vemos como el contador va aumentando.



Sí abrimos un monitor de tráfico (Chrome F12 -> Network) y analizamos las peticiones vemos que cada petición envía el identificador de sesión bajo el parámetro: PHPSESSID=2fnjil... Ese código es el que le permite al servidor acceder a los datos guardados en la sesión de ese usuario concreto.



The screenshot shows the Network tab in Chrome DevTools. A request for 'controlsession.php' is selected. In the Headers section, the 'Cookie' header is expanded, showing the value 'PHPSESSID=2fnjilngtome518nck62g3okr'. Other headers listed include Pragma, Server, X-Powered-By, Accept, Accept-Encoding, Accept-Language, Cache-Control, Connection, and Host.

Para forzar la destrucción de una sesión y borrar todo lo que contenía utilizamos la función **session\_destroy()**. En la siguiente página: `cerrar.php` destruye la sesión actual, una vez destruida si intentamos acceder a `controlsession.php` no nos dejará ya que todos los datos de la sesión fueron eliminados (`$_SESSION["validado"]` ya no existe).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p><b>Sesión cerrada! !</b></p>
</body>
<?php
    session_start();
    session_destroy();
?>
</html>
```

La llamada a **session\_start()** en todas las páginas que utilizan la sesión puede no ser necesaria, depende de la configuración de php (fichero C:\xampp\php\php.ini). Cambiando el valor del atributo `session.auto_start` de 0 a 1 deja de ser necesaria en todos los casos.

```
1456 ; Initialize session on request startup.
1457 ; http://php.net/session.auto-start
1458 session.auto_start=1
1459
```

Desde la configuración de PHP podemos modificar muchos aspectos relativos al funcionamiento de la sesión.

## Cookies

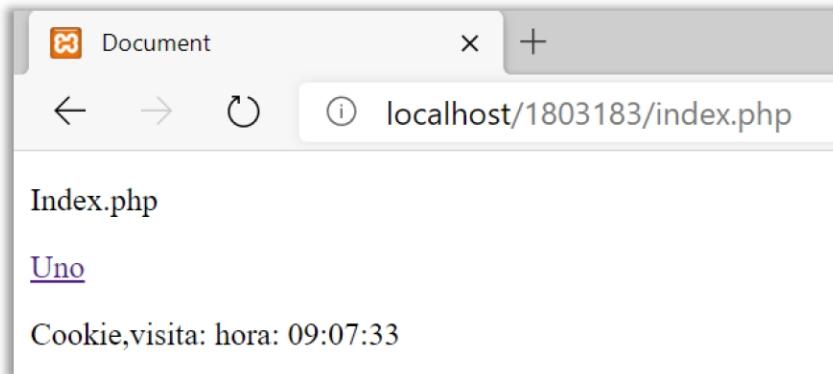
Desde el punto de vista del desarrollador las cookies funcionan de forma similar a la sesión, que nos ofrece un array. Al igual que en la sesión la información que se almacena es única para cada cliente, la diferencia reside en que la información se guarda en un fichero en el propio del cliente y no en el servidor.

No es recomendable utilizar las cookies para guardar ningún dato crítico (ya que pueden ser editadas por el usuario), normalmente se guardan datos sobre la navegación o preferencias del usuario poco importantes, datos del navegador, etc. Por ejemplo, muchas tiendas online guardan en las cookies artículos que el usuario ha mirado para sugerírselos en futuras visitas. En cada petición que el cliente realiza al sitio web envía la Cookie para que el servidor pueda utilizar los datos almacenados en ella.

PHP tiene una función para la definición del funcionamiento de las cookies, puede recibir hasta 6 parámetros siendo obligatorios los dos primeros **setcookie(nombre, valor, tiempo de expiración, ruta, dominio del sitio web, seguridad https, accesible solo desde http)**

- Nombre: nombre del parámetro a guardar.
- Valor: valor que le queremos asignar al parámetro.
- Tiempo de expiración: normalmente se obtiene el milisegundo actual y se suma el número de segundos que queremos que la cookie permanezca viva.
- \*Ruta: especificar una ruta donde se almacenará la Cookie, en lugar de la ruta por defecto.
- \*Dominio: especificar el dominio de la aplicación web.
- \*Seguridad http: [0 ó 1] especificar si la cookie será válida únicamente para protocolos https.
- \*Accesible solo desde http: [0 ó 1] especificar si la cookie será válida únicamente para protocolos http.

En el siguiente ejemplo se crea una cookie con la clave "visita\_index" donde se guarda la hora a la que el usuario accede a `index.php`. Una vez la cookie ha sido almacenada se accede a los valores utilizando el array `$_COOKIE` y la clave establecida. Al igual que ocurría anteriormente es importante comprobar si los elementos del array realmente existen antes de utilizarlos, podemos hacerlo utilizando `isset(variable)`.



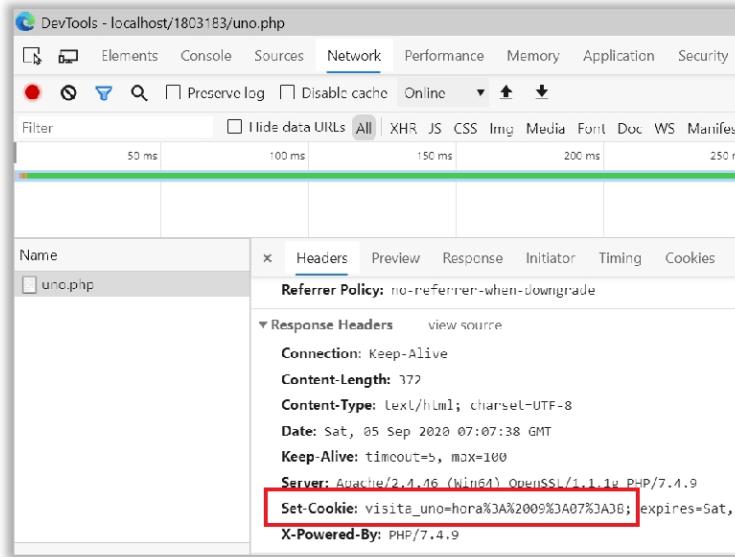
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>Index.php</p>
    <p><a href="uno.php">Uno</a><br></p>
</body>
<?php
    setcookie( "visita_index", "hora: ".date("H:i:s") , time()+1500);
    if( isset( $_COOKIE["visita_index"])){
        echo "Cookie, visita: ".$_COOKIE["visita_index"]."<br>";
    } else {
        echo "Sin Cookie <br>";
    }
?
</html>
```

La página uno.php guarda un nuevo valor en la cookie, "visita\_ultimo", a continuación, recorre todo el array e imprime las claves y sus valores, de esta forma podemos ver todo lo que contiene la cookie



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <p>uno.php</p>
    <a href="index.php">Index</a><br></p>
</body>
<?php
    setcookie( "visita_ultimo", "hora: ".date("H:i:s") , time()+1500);
    foreach($_COOKIE as $clave => $valor) {
        echo "Clave =".$clave.", Valor =".$valor."<br>";
    }
?
</html>
```

Utilizando el analizado de tráfico podemos ver la cookie que se envía en cada petición al servidor.



DevTools - localhost/1803183/uno.php

Network

Name: uno.php

Headers

Response

Initiator

Timing

Cookies

Referrer Policy: no-referrer-when-downgrade

Connection: Keep-Alive

Content-Length: 377

Content-Type: text/html; charset=UTF-8

Date: Sat, 05 Sep 2020 07:07:38 GMT

Keep-Alive: timeout=5, max=100

Server: Apache/2.4.46 (Win64) OpenSSL/1.1.1w PHP/7.4.9

**Set-Cookie: visita\_uno-hora%3A%2009%3A07%3A30; expires=Sat**

X-Powered-By: PHP/7.4.9

Como podemos apreciar ejecutando el ejemplo anterior sí volvemos a escribir un valor con la misma clave el valor anterior se sobrescribe. Es posible que nos interese destruir una cookie antes de que se cumpla su tiempo de expiración, para ello debemos utilizar una función, para ello lo que se suele hacer es volver a llamar al setcookie() y darle un valor de tiempo anterior al tiempo actual. Por ejemplo, para destruir las cookies anteriores se podría hacer lo siguiente:

```
<?php
    setcookie( "visita_index", "" , time()-1500);
    setcookie( "visita_uno", "" , time()-1500);
?>
```