FORMULARIOS: RESUMEN

Ya hemos visto muchas características básicas de PHP pero la verdadera potencia del lenguaje reside en la comunicación con el usuario que utilice nuestras aplicaciones, para ello PHP puede acceder a las variables de los formularios o a través de la barra de dirección del navegador.

PHP cuenta con dos arrays **$_POST y $_GET** para acceder a las variables de formulario enviadas a través del método **POST y GET** respectivamente

Formularios XHTML

Antes de empezar a utilizar PHP para manipular las variables de formularios vamos a hacer un repaso de las etiquetas XHTML usadas para crear los formularios y los distintos campos.

La etiqueta **<form>** cuenta con una serie de atributos a destacar:

- **method**: Este atributo puede ser POST o GET e indica el método con el que se pasaran los datos al servidor. Si utilizamos el método GET los datos se pasaran por la URL y serán visibles por todos los usuarios. Si utilizamos el método POST los datos van incluidos en el cuerpo de la petición HTTP al servidor.
- **action**: Este atributo indica el archivo PHP que procesará la información. Puede ser una ruta relativa o absoluta.

Dentro de la etiqueta **<form>** podemos encerrar distintos campos de texto, de password, checkbuttons, radiobuttons, listas, etc. De estos campos cabe destacar el atributo name ya que accederemos a los datos de cada campo en el servidor utilizando el valor de ese atributo.

```
<form method="post" action="contacto.php">
  <label for="nombre">Nombre:</label>
  <input type="text" name="nombre" id="nombre" />
  <br />
  <label for="asunto">Asunto:</label>
  <input type="text" name="asunto" id="asunto" />
  <br />
  <label for="cuerpo">Cuerpo:</label>
  <br />
  <textarea name="cuerpo" id="cuerpo" rows="10" />
</form>
</body>
</html>
```

En nuestro ejemplo tenemos tres campos, nombre, asunto y cuerpo. Además enviamos los datos a través del método POST. Vamos a ver como acceder a estos datos a través de PHP.

```php
<?php
        $nombre = $_POST['nombre'];
        $asunto = $_POST['asunto'];
        $cuerpo = $_POST['cuerpo'];
?>
```

De esta forma accedemos a los datos del formulario a través del array super- global $_POST. Hemos declarado tres variables y hemos introducido el valor de cada campo en ellas, ahora podemos manipular esas variables como queramos. Si hubiéramos enviado el formulario a través del método GET no cambiaría mucho la cosa, solo hay que tener en cuenta que no podemos enviar todo lo que queramos por GET porque tiene una limitación por caracteres y además es visible en la barra de direcciones.

Variables Super-Globales (Variables de Servidor)

PHP cuenta con una serie de arrays super-globales:

- **$_GET**: Almacena las variables que se pasan desde un formulario mediante el método GET.
- **$_POST:** Almacena las variables pasadas por POST.
- **$_COOKIE**: Guarda los valores que están almacenados en cookies.
- **$_SESSION:** Guarda las variables que se pasan entre sesiones.
- **$_SERVER:** Contiene numerosos valores relativos al servidor.
- **$_FILES:** Los archivos que enviemos a través de un formulario serán recogidos en este array.

FORMULARIOS: MÉTODOS GET Y POST

Cuando un usuario rellena un formulario en una página web los datos hay que enviarlos de alguna manera. Vamos a considerar las **dos formas de envío de datos posibles**: usando el método POST o usando el método GET.

Por ejemplo:

```
<form action="http://www.aprenderaprogramar.com/prog/newuser" method ="get">
```

En el ejemplo anterior la **acción que se ejecutará cuando el usuario pulse el botón "Enviar" (submit)** será el envío de los datos a la url especificada usando el método get.

Se suele prestar a confusión ya que tanto GET como POST son **métodos del protocolo HTTP** el cual esta compuesto por un envío al servidor conocido como petición (**request**) y una respuesta a dicha solicitud (**response**).

La diferencia entre los métodos GET y POST radica en la forma de enviar los datos a la página cuando se pulsa el botón "Enviar". Mientras que el método **GET** envía los datos usando la URL, el método **POST** los envía de forma que no podemos verlos (en un segundo plano u "ocultos" al usuario).

Método GET

Un resultado usando el **método GET**, a modo de ejemplo, podría ser el siguiente:

```
http://www.aprenderaprogramar.com/newuser.php?
nombre=Pepe&apellido=Flores&email=h52turam%40uco.es&sexo=Mujer
```

En esta URL podemos distinguir varias partes:

- **http://www.aprenderaprogramar.com/newuser.php** es la dirección web en sí.
- El símbolo **?** indica dónde empiezan los parámetros que se reciben desde el formulario que ha enviado los datos a la página.
- Después del símbolo **?** aparecen parejas de datos con su nombre y valor separadas por el símbolo **&.** Las parejas dato1=valor1, dato2=valor2, dato3=valor3… reflejan el **nombre y el valor de los campos** enviados por el formulario.

Por ejemplo: nombre=Pepe, apellidos=Flores, etc. nos dice que el campo del formulario que se denomina nombre llega con valor "Pepe" mientras que el campo del formulario que se denomina apellidos llega con valor "Flores".

Tener en cuenta que para separar la primera pareja de la dirección web en sí se usa el símbolo '?' y para separar las restantes parejas entre sí se usa el símbolo '&'.

Nota:

Otro aspecto a tener en cuenta es que determinados caracteres no son recibidos en la URL de la misma forma exactamente en que fueron escritos en el formulario. Por ejemplo, el valor del campo email que se recibe en la URL es h52turam%40uco.es, mientras que el usuario en el formulario habrá introducido con toda seguridad h52turam@uco.es. Como vemos, **el carácter @ ha sido sustituido por los caracteres %40.** Estas equivalencias se introducen automáticamente en la transmisión de datos debido a que las URLs no admiten determinados caracteres como letras con tildes, arrobas y otros. No debes preocuparte por esta codificación, ya que si posteriormente rescatamos los valores mediante otros mecanismos volveremos a obtener el texto original. Simplemente, conviene conocer esta circunstancia para no pensar que están ocurriendo cosas extrañas o errores.

Método POST

En el caso de un envío de datos usando el **método POST**, aunque estos datos también serán enviados (de una forma que podemos denominar "oculta"), no los podremos ver en la URL. Para poder recuperar los valores de los campos en el caso de un envío con el método POST necesitaríamos del lenguaje PHP para recuperar el valor de esos campos).

El resultado final con ambos métodos podemos decir que es el mismo: la información se transmite de un lado a otro.

Por ejemplo:

```
<form action="http://www.aprenderaprogramar.com/prog/newuser" method ="post">
```

FORMULARIOS: MÉTODOS GET Y POST

ENTENDIENDO EL CONCEPTO

Tendemos a entender que cuando **doy click a un link eso es GET** y cuando **envío un formulario es POST**. Mucho peor, solemos pensar que enviando peticiones POST los datos viajan seguros por no ir como parte de la URL como lo hace GET. Debemos darnos cuenta que GET y POST no son la diferencia entre links y formularios.

El concepto GET es *obtener información* del servidor. Traer datos que están en el servidor, ya sea en un archivo o base de datos, al cliente. Independientemente de que para eso tengamos que enviar (request) algún dato que será procesado para luego devolver la respuesta (response) que esperamos, como por ejemplo un identificador para obtener una noticia de la base de datos.

POST sin embargo es *enviar información* desde el cliente para que sea procesada y actualice o agregue información en el servidor, como sería la carga o actualización en sí de una noticia. Cuando enviamos (request) datos a través de un formulario, estos son procesados y luego a través de una redirección por ejemplo devolvemos (response) alguna página con información.

Ambos métodos solicitan una respuesta del servidor y ahí es donde parecen que los conceptos son iguales ya que con ambos se podría lograr los mismos objetivos.

Yo podría, aunque estaría mal, enviar por GET ciertos datos en la URL y "actualizar o insertar" información en mi base de datos, pero eso le correspondería al método POST.

De la misma manera podría solicitar una página diferente por medio de POST y simplemente mostrarla como respuesta aunque eso debería ser a través de una llamada GET.

Las llamadas **GET pueden ser cacheadas** (historial del navegador), indexadas por buscadores, agregar los enlaces a nuestros favoritos o hasta pasar una url completa a otra persona para que directamente ingrese a esa página. Con el método **POST sin embargo no se puede hacer esto.**

Generalmente usamos **links para ejecutar llamadas GET** ya que la idea del link es simplemente "solicitar" una información (página) al servidor y que sea devuelta como una respuesta.

Mientras usamos **formularios** para actualizar datos de productos, clientes, noticias, etc, también teniendo en cuenta que **por el método POST** también se puede enviar mucha más cantidad de datos que por GET.

EJEMPLOS DE MAL USO

GET:

Para entender finalmente la diferencia voy a darles un caso de análisis. Supongamos que tenemos en nuestro sitio un listado de productos con un link que agregue ese producto al carrito de compras. Si hacemos que ese link ejecute el método GET, como generalmente lo usamos, usaríamos una URL parecida a esta:
www.misitio.com/agregar_item_carrito.php?id=1. Al ser una llamada GET, Google podría indexar esa URL y podría aparecer en el buscador al buscar la palabra carrito. Cuando una persona le diera click automáticamente se ejecutaría esa página y agregaría el item con id 1 al carrito del sitio, lo cual les puedo asegurar que no es la idea ya que el visitante al buscar carrito debería querer simplemente entrar al sitio y no agregar un ítem que ni siquiera sabe cual es. Por lo tanto vemos que **para este caso**, por más que usemos un link, **deberíamos de usar** una llamada al método **POST.**

POST:

Otro ejemplo sencillo sería cuando en un administrador de noticias tenemos un listado de las noticias con un link "eliminar" para borrarlas una por una (situación muy común en sistemas Web). Deberíamos de hacer esta petición usando POST para no permitir, por seguridad, que esa URL creada sea indexada, enviada a otra persona, guardada en favoritos, ni mucho menos ejecutada por culpa del botón atrás del navegador ya que quedaría cacheada en el historial.

Demos un ejemplo opuesto. Hay casos en los que los formularios de búsquedas al ser enviados por POST van a la página de resultados pero como estas llamadas no son cacheadas en el historial del navegador, no podemos volver usando la tecla atrás del navegador por lo que se suele dejar como llamadas GET a fin de ser cacheadas.

En la siguiente tabla mostramos un resumen de las diferencias entre GET y POST:

MÉTODO	CONCEPTO	OBSERVACIONES
GET	GET lleva los datos de forma "visible" al cliente (navegador web). El medio de envío es la URL. Los datos los puede ver cualquiera.	Los datos son visibles por la URL, por ejemplo: www.aprenderaprogramar.com/action.php?nombre=pedro&apellidos1=gomez
	El concepto **GET** es *obtener información* del servidor. Traer datos que están en el servidor, ya sea en un archivo o base de datos, al cliente. Independientemente de que para eso tengamos que enviar (request) algún dato que será procesado para luego devolver la respuesta (response) que esperamos, como por ejemplo un identificador para obtener una noticia de la base de datos.	Pueden ser cacheadas Generalmente usamos links para ejecutar llamadas GET
POST	POST consiste en datos "ocultos" (porque el cliente no los ve) enviados por un formulario cuyo método de envío es post. Es adecuado para formularios. Los datos no son visibles.	La ventaja de usar POST es que estos datos no son visibles al usuario de la web. En el caso de usar get, el propio usuario podría modificar la URL escribiendo diferentes parámetros a los reales en su navegador, dando lugar a que la información tratada no sea la prevista.
	POST sin embargo es *enviar información* desde el cliente para que sea procesada y actualice o agregue información en el servidor, como sería la carga o actualización en sí de una noticia. Cuando enviamos (request) datos a través de un formulario, estos son procesados y luego a través de una redirección por ejemplo devolvemos (response) alguna página con información.	No pueden ser cacheadas Generalmente usamos formularios para ejecutar llamadas POST