Unidad 5
Formularios y sistema de archivos

Índice

1. Introducción a los formularios 2
2. Acceso a formularios desde PHP 3
2.1. Acceso a diferentes tipos de elementos 6
2.2. Agrupando variables en un array 8
3. Envío de archivos al servidor 9
4. Cabeceras HTTP 12
4.1. Función header() 13
4.2. Redirigir a otra página. 15
4.3. Otros ejemplos de uso de header() 15
5. Formulario y procesamiento en un solo fichero 16
6. Validación y saneamiento de formularios 17
7. Sistema de archivos 19
7.1. Ficheros de texto 19
7.2 Manipular directorios 21
7.3 Ficheros XML 21

1. Introducción a los formularios

Los formularios son la forma más habitual de enviar datos a un servidor. Permiten que el usuario rellene varios campos mediante diferentes tipos de controles y lo envíe al servidor al pulsar un botón. El servidor proceso el formulario y genera la respuesta.

Los formularios no forman parte de PHP, sino de HTML.

Hasta el momento sólo hemos enviado la información por la URL y siempre los datos están visibles. Pero cuando usamos un formulario lo deseable es que los datos no se visualicen en la URL.

Cuando describimos un formulario especificamos:

- Mediante el atributo *action* la ruta del script al que se enviará el formulario para que lo procese.
- Y con el atributo *method* el método HTTP que se usará para la petición. El tipo de envío influye en la forma en que PHP captura los datos que se envían con el formulario. Lo más habitual es hacer una petición a través de HTTP usando el método POST, pero también es posible usar GET.

Algunas consideraciones sobre GET:

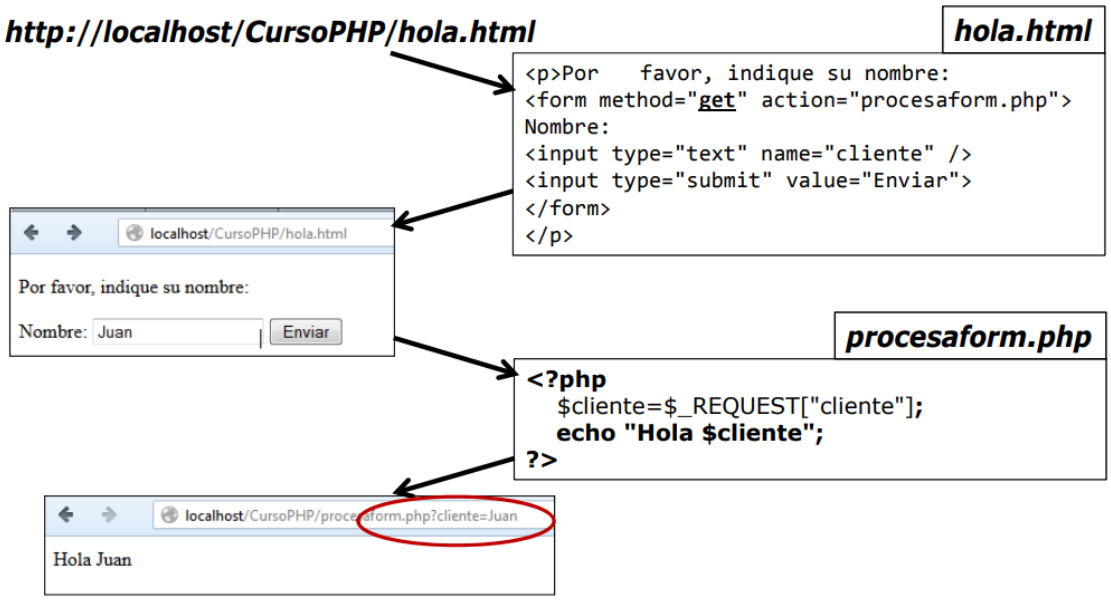
- Recordad, que cuando se usa el método GET, a la ruta normal para acceder a una página se le añade el carácter "?" como indicador de que empieza la lista de parámetros. Siendo el siguiente formato:

  Fichero.php?param1=val1&param2=val2…

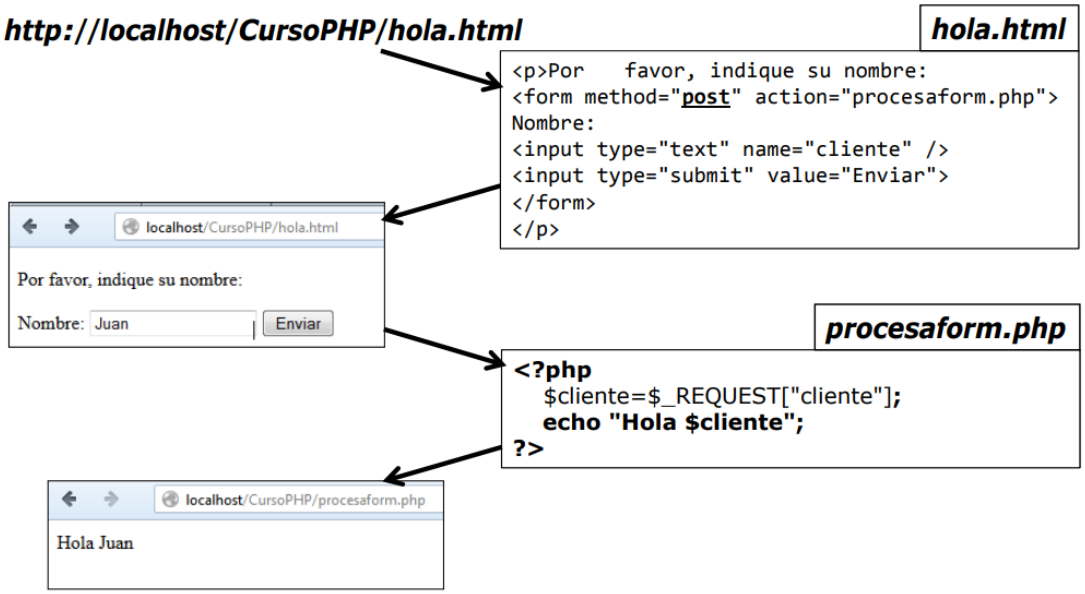
Ejemplo:
http://localhost/unidad3/saludo.php?nombre=Ana&apellido=Prieto

- Debemos evitar enviar información susceptible a través de GET como password o datos personales
- Debemos utilizarlo, en general, solo cuando estemos proporcionando información al servidor relativa al contenido que queremos recuperar (orden, página, cantidad, etc.).
- No debemos utilizarlo cuando estemos enviando información para que sea almacenada en el servidor.
- GET mejora el SEO de nuestra página web y evita el típico mensaje de error de ¿desea reenviar el formulario?
- GET proporciona al usuario la capacidad de alterar las consultas en la URL, lo cual puede generar errores para los que nuestro script debe estar preparado.

Escenario típico de interacción (con GET)



Escenario típico de interacción (con POST)



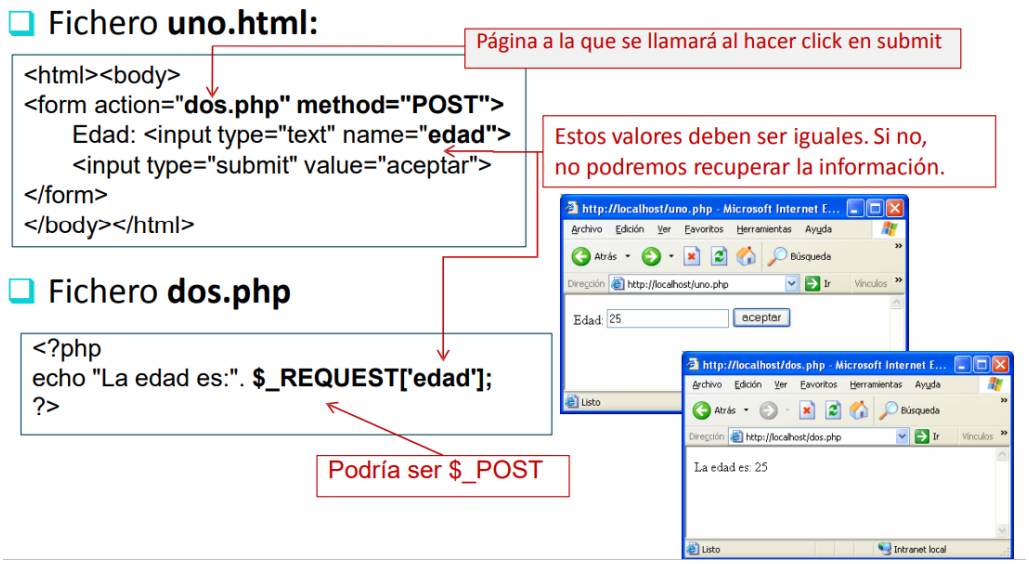
2. Acceso a formularios desde PHP

Desde PHP podemos obtener la información que hemos enviado en el formulario usando las variables globales:

- $_GET : Es un array que contiene toda la información enviada por un formulario o una URL (QUERY_STRING) . Esta variable tiene un elemento por cada argumento presente en la URL. El nombre del argumento será la clave del elemento del array.

- $_POST: Contiene toda la información enviada por un formulario o una URL utilizando el método POST. La clave de cada argumento dentro del array es el atributo name del elemento correspondiente en el formulario.

- $_REQUEST. Contiene toda la información enviada por un formulario o una URL, ya sea con el método POST o GET. También incluye información relacionada con las cookies.



Como puedes observar, dentro del elemento ***form*** se especifican los campos que rellenará el usuario, así como los botones de envío y reseteo de la información.

Ejemplo de formulario en HTML:

```
<form method="POST" action="nuevoAmigo.php">
      <label for="nombre">Nombre</label>
      <input type="text" name="nombre" >

      <label for="apellido">Apellido</label>
      <input type="text" name="apellido" >

      <input type="submit" value="enviar datos">
  </form>
```

Para que un control envíe información es necesario:

- que el control esté incluido en un formulario (<form>).

- que el formulario tenga establecido el atributo action, con la dirección absoluta o relativa del fichero PHP que procesará la información
- que el control tenga establecido el atributo name[1].
- que el formulario contenga un botón de tipo submit

El fichero que recibe la información "nuevoAmigo.php" podría ser el siguiente:

```
<?php
echo "<h1>Datos recibidos:</h1>";
echo "<h2> Nombre: ".$_POST['nombre']."</h2>";
echo "<h2> Apellidos: ".$_POST['apellido']."</h2>";
?>
```

Si se ejecuta podremos comprobar que los datos se han capturado.

En realidad, podemos incluir el código de PHP en nuestro HTML y mostrar los datos. A continuación se muestra un ejemplo de cómo podríamos modificar el archivo anterior:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Formulario</title>
</head>
<body>
      <h1>Datos recibidos:</h1>
      Nombre: <?= $_POST['nombre']?>
      <br>
      Apellidos: <?= $_POST['apellido']?>
</body>
</html>
```

Por supuesto,quedaría comprobar si realmente los datos han llegado por el método deseado como veremos más adelante. Aquí tienes un ejemplo de cómo se puede hacer esa comprobación:

```
if($_SERVER["REQUEST_METHOD"] == "POST"){
            if(!empty($_POST['nombre']) and (!empty($_POST['apellido']){

            …
            }
```

[1] el atributo name puede contener cualquier carácter (números, acentos, guiones, etc.), pero si contiene espacios, PHP sustituye los espacios por guiones bajos (_) al procesar los datos enviados

2.1. Acceso a diferentes tipos de elementos

Veremos algunos ejemplos para acceder a diferentes tipos de elementos de entrada de formulario:

- Text

```
<form method="POST" action="tipoText.php">
           <label for="cadena">Introduzca la cadena buscar: </label>
           <input type="text" name="cadena" >

           <input type="submit" value="enviar">
       </form>
```

Podemos acceder desde PHP a este elemento desde la variable $_POST o si el método de envío es GET usamos $_GET. En ambos caso también $_REQUEST:

```
Cadena recibida: <?= $_REQUEST['cadena']?>
```

- Textarea, Hidden y Password: Igual que un tipo text
  Por ejemplo :

  Si en el formulario se pide este tipo de dato:

```
Contraseña: <input type="password" name="clave">
```

  En PHP accedemos:

```
<?php
$clave = $_REQUEST['clave'];
print ($clave);
?>
```

- Radio

```
<form method="POST" action="tipoRadio.php">
       Sexo:
       <input type="radio" name="sexo" value="M" CHECKED>Mujer
       <input type="radio" name="sexo" value="H">Hombre

       <input type="submit" value="enviar">
</form>
```

El acceso desde PHP a través de las variables $_POST,$_GET o $_REQUEST:

```
<?php
$sexo = $_REQUEST['sexo'];
print ($sexo);
?>
```

- Chexbox

```
<form method="POST" action="tipoChexbox.php">
  <input TYPE="checkbox" name="extras[]" value="garaje"
  CHECKED>Garaje
  <input TYPE="checkbox" name="extras[]" value="piscina">Piscina
  <input TYPE="checkbox" name="extras[]" value="jardin">Jardín
  <input type="submit" value="enviar">
</form>
```

En este caso la variable $_GET , $_POST o $_REQUEST contiene un array con los diferentes valores recogidos y por lo tanto hay que recorrerlo como tal para obtener los valores:

```
<?php
$extras = $_REQUEST['extras'];
// los valores recogidos están en un array
foreach ($extras as $extra)
print ("$extra<br>\n");
```

- Select
  En caso de un select simple como el del ejemplo:

```
<form method="POST" action="tipoSelectSimple.php">
      <select name="color">
          <option value="rojo">rojo</option>
          <option value="verde" selected>verde</option>
          <option value="azul">azul</option>
      </select>

      <input type="submit" value="enviar">
</form>
```

  Los datos pueden obtenerse igual que en el Text

```
<?php
 $color = $_REQUEST['color'];
 print ($color);
?>
```

Sin embargo, hay que tener cuidado si el select es múltiple. En este caso para obtener los datos hay que recordar que estarán en un array.

```
<form method="POST" action="tipoSelectMultiple.php">
      <select multiple size="3" name="idiomas[]">
          <option value="francés">Francés</option>
          <option value="inglés" selected>Inglés</option>
          <option value="alemán">Alemán</option>
      </select>

      <input type="submit" value="enviar">
  </form>
```

Los datos se obtienen desde el array:

```
<?php
 $idiomas = $_REQUEST['idiomas'];
 foreach ($idiomas as $idioma)
    print ("$idioma<br>\n");

?>
```

- Submit
  Ejemplo con más de un submit (no es lo habitual):

```
<form method="POST" action="tipoSubmit.php">
 <p>¿Blanco o negro?</p>
     <p>
         <input type="submit" name="respuesta" value="Negro">
         <input type="submit" name="respuesta" value="Blanco">
     </p>

 </form>
```

  Acceso desde PHP

```
<?php

   $respuesta = $_REQUEST['respuesta'];
   print ($respuesta);
?>
```

- Button: Si en el formulario tenemos definido un button

```
<button type="submit" name="boton1"
value="enviado">Enviar</button>
```

  Acceso a este dato en PHP:

```
<?php
    $respuesta = $_REQUEST['boton1'];
    print ($respuesta);
 ?>
```

- File : Este elemento lo veremos en detalle en el apartado 3 de la unidad.

2.2. Agrupando variables en un array

Podemos agrupar las variables definidas como campos de los formularios para que PHP pueda procesarlas como arrays. Veremos que esto es útil cuando trabajemos con bases de datos para relacionar lo campos de una tabla con los campos del formulario y agruparlos según la tabla de la base de datos.

En realidad, ya hemos visto que también se usa cuando tenemos un campo de formulario con múltiples valores como chexbox.

Ejemplo:

```
<label for="nombre" >Nombre</label>
<input type="text"  id="nombre" name="contacto[nombre]" >

<label for="apellidos" >Apellidos</label>
<input type="apellidos"  id="apellidos" name="contacto[apellidos]">
```

Y se accede como a cualquier array en PHP:

```
<?php
      $contacto = $_POST["contacto"];?>
      Nuevo Contacto: <?=$contacto["nombre"] . " " .
      $contacto["apellidos"]."<br>"?>
?>
```

3. Envío de archivos al servidor

En este apartado veremos cómo subir archivos a un servidor desde un formulario.

Hay que tener en cuenta una serie de consideraciones importantes:

- Realizaremos una petición a través de HTTP usando el método POST.
- Para subir un fichero al servidor se utiliza el elemento de entrada FILE, como ya se indicó anteriormente. Es decir, en el formulario, utilizaremos una etiqueta con un atributo de tipo file: `<input type="file">`. Con este control se abre una ventana para que el usuario pueda escoger el archivo de su equipo.
- El elemento FORM debe tener el atributo `ENCTYPE="multipart/form-data"`.

Antes de trabajar con los ficheros deberemos de verificar algunos ajustes de configuración de PHP para asegurarnos de que se suben correctamente.

a. El fichero tiene un límite en cuanto a su tamaño. Este límite se fija de dos formas diferentes y complementarias:

- En el fichero de configuración php.ini

php.ini

```
;;;;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;;;;
; Si se permite o no subir archivos mediante HTTP
file_uploads = On
;
; Tamaño máximo de cada archivo subido.
upload_max_filesize = 2M
; Tamaño máximo de los datos mandados por POST
;(incluidos los que no sean archivos)
post_max_size = 8M
```

Algunas de las directivas que puedes modificar en ese archivo son:

- file_uploads : se debe establecer en On para permitir la carga de archivos.
- upload_max_filesize : permite configurar el tamaño máximo del archivo cargado. De forma predeterminada, se establece en 2 MB
- post_max_size : para configurar el tamaño máximo de los datos POST. Dado que los archivos se cargan con solicitudes POST, este valor debe ser mayor que lo que has establecido para la directiva upload_max_filesize.
- max_file_uploads: permite establecer el número máximo de archivos que se pueden cargar a la vez. El valor predeterminado es 20.

- En el propio formulario

```
<form enctype="multipart/form-data" action="/" method="POST">
 <input type="hidden" name="MAX_FILE_SIZE" value="2097152" />
 <input name="file_input_name" type="file" /><br /><br />
 <input type="submit" value="Send File" />
</form>
```

b. Usarmos la variable superglobal $_FILES en el script que recibe el formulario, ya que esta contiene información sobre el archivo que se está subiendo. Se trata de un array bidimensional en el que la primera dimensión identifica el fichero, según el atributo name en el formulario, y la segunda tiene como claves: name, size, type, tmp_name y error.

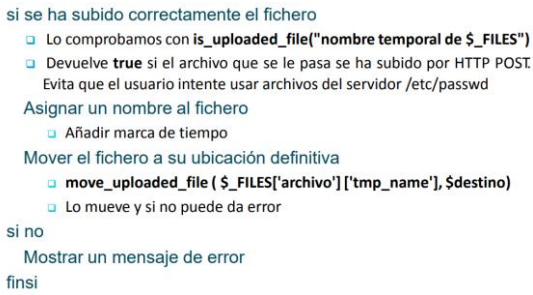
$_FILES['imagen']['name']
Nombre original del fichero en la máquina cliente
$_FILES['imagen']['type']
Tipo mime del fichero. Por ejemplo, "image/gif"
$_FILES['imagen']['size']
Tamaño en bytes del fichero subido
$_FILES['imagen']['tmp_name']
Nombre del fichero temporal en el que se almacena el fichero subido en el servidor
$_FILES['imagen']['error']
Código de error asociado al fichero subido

c. El fichero se almacena en principio en el directorio temporal del servidor y se puede mover al directorio que se desee con la función:

```
bool move_uploaded_file($fichero, $destino).
```

Si el fichero no se mueve del directorio temporal, el servidor se encargará de eliminarlo.

d. Debemos reforzar la seguridad comprobando que el archivo recibido y que se desea mover es realmente un archivo enviado mediante HTTP POST de PHP. El procedimiento:



e. Debemos tener cuidado al nombrar los ficheros evitando coincidencias con ficheros ya subidos. Por ello, y como norma general, debe descartarse el nombre original del fichero y crear uno nuevo que sea único, por ejemplo añadiéndole la fecha y hora.

f. Deberemos tener algunas precauciones con:
   - Permisos de escritura en el directorio temporal
   - Permisos de escritura en el directorio de destino
   - Atención con los ficheros que puedan subir los usuarios
   - Troyanos, scripts, ejecutables, etc.

Ejemplo de formulario con subida de archivo:

```
<form  action = "nuevoContactoConImagen.php" method = "post"
enctype="multipart/form-data">

    <label for="foto" >Imagen</label>
    <input type="file" name ="foto" placeholder="Añada una imagen"/>

    <button type="submit" >Enviar</button>
 </form>
```

Ejemplo de cómo obtener, a través de la variable superglobal $_FILES, información sobre el archivo que hemos subido en el formulario anterior.

```
<body>
    Fichero recibido:
    Nombre:    <?= $_FILES[ "foto" ][ "name" ]."<br>"?>
    Tamaño:    <?= $_FILES[ "foto" ][ "size" ]." bytes"."<br>"?>
    Temporal: <?= $_FILES[ "foto" ][ "tmp_name" ]."<br>"?>
    Tipo:      <?= $_FILES[ "foto" ][ "type" ]."<br>"?>
    Error:     <?= $_FILES[ "foto" ][ "error" ]."<br>"?>
</body>
```

Siguiendo con el ejemplo anterior, podemos añadir al código la función que nos permite mover desde el almacén temporal a otro directorio el archivo que hemos subido. Recuerda que desaparecerá del almacenamiento temporal :

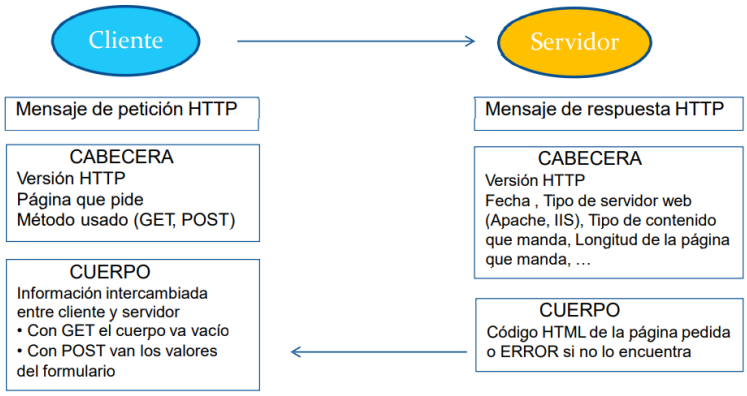
```
move_uploaded_file ($_FILES["foto"] ["tmp_name"], "miaplicacion/ficheros");
```

Si añadimos comprobaciones el código quedaría así:

```
<?php
// comprobando y moviendo a un directorio

if (is_uploaded_file ($_FILES[ "foto" ][ "tmp_name"]) ){
  $nombreDirectorio = "img/";
  $nombreFichero = $_FILES[ "foto" ][ "name" ];

  $nombreCompleto = $nombreDirectorio.$nombreFichero;
 if ( is_file($nombreCompleto)) {
  $idUnico = time();
  $nombreFichero = $idUnico."-".$nombreFichero;

 }
 move_uploaded_file ($_FILES["foto"][ "tmp_name"], $nombreDirectorio.$nombreFichero);
}
else
  print ("No se ha podido subir el fichero\n");
?>
```

4. Cabeceras HTTP

HTTP es el protocolo, sin estado, que usamos para enviar información de páginas web, entre el servidor web y el cliente(navegador). El cliente es el que inicia la comunicación.



La respuesta o HTTP response es el mensaje que envía el servidor al cliente tras haber recibido una petición o HTTP request.

Cuando un servidor envía una página web al navegador, no sólo envía la página web, sino también información adicional (el estado y los campos de cabecera). Tanto el estado como los campos de cabecera *se envían antes de la página web.*

Ejemplo de cabeceras:

```
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:104.0)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Origin: http://localhost
Connection: keep-alive
Referer: http://localhost/ejerClases/formularios/3_FormularioP
Cookie: tinymceToolbarTogglerPreference=visibleToolbars
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

Petición

```
Origin: http://localhost
Connection: keep-alive
Cookie: tinymceToolbarTogglerPreference=visibleToolbars
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
nombre=mara&apellido=mara@mara.es
POST: HTTP/1.1 200 OK
Date: Sun, 11 Sep 2022 15:54:09 GMT
Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/8.1.6
X-Powered-By: PHP/8.1.6
Content-Length: 126
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Respuesta

Desde PHP podemos generar contenido HTML, imágenes, documentos planos o PDF, un objeto, un fichero zip, un objeto JSON, etc. En la cabecera irá información acerca del recurso que se está sirviendo. Por ejemplo, la fecha de modificación, el tipo de contenido, etc.

4.1. Función header()

Normalmente, un programa PHP sólo genera la página web y es el servidor el que genera automáticamente la información de estado y los campos de cabecera y los envía antes de enviar el contenido generado por el programa.

Pero un programa PHP también puede generar la información de estado y los campos de cabecera, mediante la función **header()**.

Su sintaxis:

```
header ( string $header [, bool $replace = TRUE [, int $http_response_code ]] ) : void
```

Algunos headers, de un HTTP response, que usaremos:

- Content-type
- Content-length
- Date
- Status
- Location
- Set-cookie

Ejemplos de cabeceras:

- header("location: http://www.upm.es");
- header ('Content-Type: text/html; charset=UTF-8');
- header("HTTP/1.0 404 Not Found");

Es importante recordar que hay que enviar las cabeceras antes de empezar con el cuerpo de la respuesta. Esto implica que hay que utilizar la función header() antes de que se empiece a escribir la salida. Si se intenta llamar a header() después de haber realizado, por ejemplo, echo, se producirá un error. El motivo es que en cuanto un programa genera contenido HTML, el servidor genera automáticamente la información de estado y los campos de cabecera y a continuación envía el contenido generado. Si después el programa contiene una instrucción header(), se produce un error porque las cabeceras ya se han enviado.

Se pueden ver los headers enviados, o que serán enviados, con la función _headers_list() , y con con headers_sent()_ Se puede saber si los headers ya se han enviado.

Para extraer cabeceras HTTP del cliente en PHP usaremos *apache_request_headers():*

```
<?php
var_dump(apache_request_headers());
?>
```

También podemos usar una extensión de Firefox para visualizar las cabeceras de una manera más cómoda, Live HTTP Headers.

4.2. Redirigir a otra página.

La función **header()** se puede utilizar para redirigir automáticamente a otra página, enviando como argumento la cadena *Location:* seguida de la dirección absoluta o relativa de la página a la que queremos redirigir. Ejemplo: `header("Location: index.php");`

Hay varias ocasiones en los que podemos querer redireccionar a los usuarios a otra URL. Por ejemplo:

- Hemos actualizado la web y el recurso solicitado ahora está en otra dirección (para no perder posicionamiento, por ejemplo).
- Ponemos enlaces intermedios para saber cuándo un usuario accede a un enlace externo (cuando un usuario hace clic, por ejemplo, de esa forma podemos redirigir al usuario antes a otra página nuestra y luego desde esa a la que el desea).

También podemos mostrar un mensaje antes de redirigir:

```
header ('Refresh: 5; url=http://www.google.es');
echo 'Lo que busca no existe, le redirigiremos a Google en 5 segundos';
```

4.3. Otros ejemplos de uso de header()

- Ocultar la versión de nuestro intérprete PHP

```
<?php header("X-Powered-By: adivina-adivinanza"); ?>
```

- Ofrecer la descarga de un archivo desde PHP.
  Por ejemplo, queremos que se descargue un PDF llamado «Hipercubo»

```
<?php

header("Content-disposition: attachment; filename=Hipercubo.pdf");
header("Content-type: application/pdf");
readfile("Hipercubo.pdf");
?>
```

  Una vez hayamos definido el nombre y la extensión, guardamos el archivo, lo subimos al hosting y enlazamos la descarga en HTML creando un hipervínculo que llame al archivo PHP.
- Generar contenidos diferentes a páginas HTML con PHP. Imágenes, documentos PDF, películas en tiempo real , etc.

```
header("Content-type:image/jpeg");
header("Content-Disposition:inline ; filename=captcha.jpg");
```

El fichero se abre en el navegador en lugar de descargarse (attachment)

- Otro ejemplo de uso de header() es cuando creamos una página de error y queremos asegurarnos de que se emita el código de estado HTTP correcto. Para lello, podemos poner el siguiente código:

```
<?php header("HTTP/1.1 404 Not Found"); ?>
```

5. Formulario y procesamiento en un solo fichero

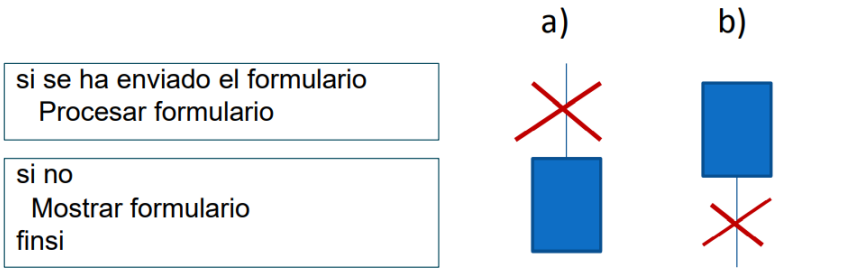
Una forma de trabajar con formularios en PHP es utilizar un único programa que procese el formulario o lo muestre según haya sido o no enviado, respectivamente.

En este caso el formulario HTML y el bloque PHP que lo procesa se integran en un solo fichero, en el que hay que distinguir entre:

- Cuando se accede para rellenarlo
- Cuando se envía para procesarlo

Cuando se accede a la página usando el método GET, es decir, introduciendo la dirección en el navegador se muestra el formulario. En cambio, si se accede mediate POST quiere decir que el cliente está enviando el formulario.
Podemos diferenciar entre los dos métodos de acceso consultando la variable `$_SERVER["REQUEST_METHOD"]`.

Esquema de funcionamiento:



- La 1ª vez que se carga la página se muestra el formulario (a)
- La 2ª vez se procesa el formulario (b)

Cuando el formulario llama al mismo fichero se recomienda usar en lugar del propio nombre del fichero:

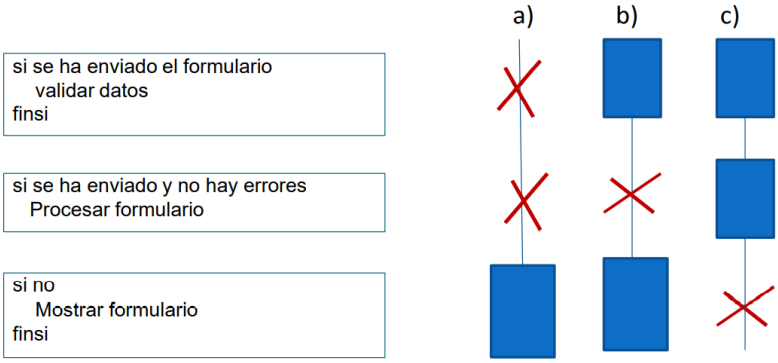
```
action = "<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"
```

La variable superglobal `$_SERVER["PHP_SELF"]` contiene el nombre del fichero y la función htmlspecialchars() sirve para filtrar los caracteres por seguridad. Esta función garantiza que cualquier carácter que sea especial en HTML se codifique adecuadamente, de manera que nadie pueda inyectar etiquetas HTML o Javascript en la página.

6. Validación y saneamiento de formularios

La validación de formularios es algo fundamental ya que previene posibles ataques de intrusos, además de asegurar que los datos que se reciben son realmente del tipo deseado.

Esquema de funcionamiento:



- La 1ª vez que se carga la página se muestra el formulario (a)
- La 2ª y sucesivas veces se validan los datos
  - Si hay errores, se muestra de nuevo el formulario con los errores (b)
  - Si no hay, se procesa el formulario (c)

Existen **dos formas de validación de formularios**: en el lado del cliente y en el lado del servidor.

Antes de enviar datos al servidor es importante asegurarse de que se completan todos los controles de formulario requerido y en el formato correcto. En el lado del cliente la validación suele ser mediante JavaScript y ayuda a garantizar que los datos que se envían coinciden con los requisitos establecidos en los diversos controles de formulario. Esta validación es más rápida y evita enviar más trabajo al servidor.

La validación en el lado del cliente[2] es una verificación inicial y una característica importante para garantizar una buena experiencia de usuario. Mediante la detección de datos no válidos en el lado del cliente,

[2] Aquí puedes consultar acerca de la validación en el lado cliente.

el usuario puede corregirlos de inmediato. Si el servidor lo recibe y, a continuación, lo rechaza; se produce un retraso considerable en la comunicación entre el servidor y el cliente que insta al usuario a corregir sus datos. Si sólo hacemos esta validación un usuario malintencionado podría enviar datos malintencionados que comprometan la seguridad de la página.

Una vez superada esta validación se realizará el proceso de validación en el back-end. En el lado del servidor se emplea PHP para verificar que se envían valores correctos.

Si sólo se realiza esta validación, sin hacer la validación en el front-end, cuando se produce un error al procesar el formulario puede que el usuario tenga que volver a rellenar todos los campos de nuevo.

La doble validación conlleva más trabajo, pero es el sistema recomendado, puesto que es más estricto y sobre todo más seguro.

Aunque en el servidor se haya comprobado que hemos recibido todos los campos requeridos, también tendremos que comprobar si los tipos de datos de dichos valores se corresponden con los esperados. Por ejemplo, si esperamos un valor numérico hay que comprobar que no haya letras u otros símbolos.

El proceso de limpiar los datos que nos llegan, eliminando caracteres no deseados, se llama saneamiento. Para corroborar que la información cumple con los requisitos establecidos, se utiliza el proceso conocido de validación, pero desde el lado del servidor que contiene PHP.

PHP tiene una extensión especial que facilita realizar estos procesos de una forma más rápida. Su nombre es Filter y esta activa por defecto.

Existen varias funciones de filtrado entre las que destacamos:

Filter_var que nos permite filtrar una variable con el filtro especificado. Y por tanto, con ella podemos realizar el saneado y la validación de los datos.

Aplicando filtros de saneamiento ,a los datos que nos llegan, se eliminan los caracteres que no se ajustan al tipo de filtro que se le pasa como parámetro.

Una vez que hemos limpiado la información comprobamos que cumple los requisitos establecidos. Es aquí cuando podemos aplicar sobre ellos los filtros de validación.

Los requisitos pueden ser muy variados, desde que el campo exista obligatoriamente hasta que sea numérico o una dirección de correo electrónica correcta.

Ejemplo en el que nos aseguramos de que la información recibida es un email:

```
$email = $_POST["email"];
// Primero eliminamos cualquier carácter que pueda dar problemas
```

```
$email = filter_var($email, FILTER_SANITIZE_EMAIL);
// Luego validamos el email
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo("ERROR: $email NO es una dirección de email válida");
}
```

7. Sistema de archivos

Existen varias librerías para manejar ficheros.

La más importante es Filesystem que nos permite acceder a ellos y manejar el sistema de archivos.

7.1. Ficheros de texto

En el manual de PHP podremos consultar numerosas funciones para trabajar con ficheros.

Por ejemplo, para abrir un archivo usaremos la función fopen() en la que indicamos tanto el nombre como los permisos (tipo de acceso: lectura, escritura…). Si ocurre error la función devuelve FALSE o en caso contrario un puntero para manejar el archivo.

Estos son algunos tipos de acceso:

Modo	Descripción
r	Solo lectura. Si el fichero no existe, devuelve FALSE
r+	Lectura y escritura. Si el fichero no existe, devuelve FALSE
w	Solo escritura. Si el fichero no existe, se crea; si existe, se trunca (es decir, se borra el contenido anterior). Si no puede crearlo, devuelve FALSE
w+	Lectura y escritura. Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE
a	Solo escritura. Las escrituras se realizan siempre al final de fichero (append). Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE
a+	Lectura y escritura. Las escrituras se realizan siempre al final de fichero. Si el fichero no existe, se crea; si existe, se trunca. Si no puede crearlo, devuelve FALSE

Si lo que queremos es cerrar un fichero que tenemos abierto usamos la función fclose() junto con el nombre del archivo.

Para leer el contenido del archivo usaremos la función fgets(). Ten en cuenta que, esta función sólo nos muestra una línea. Si deseamos mostrar todas las líneas del fichero necesitamos un bucle y la función feof() que nos devuelve TRUE si hemos llegado al final del archivo.

Si lo que deseamos es leer el fichero carácter a carácter usaremos la función fgetc(), que devuelve el siguiente carácter a partir del indicador de posición.

Para escribir en un archivo utilizamos la función fwrite. Recuerda que, debemos de tener permiso a+ (leer y escribir).

En el siguiente cuadro hay un resumen de algunas de las funciones más usadas:

Función	Descripción
fgets($fich)	Devuelve una cadena con los caracteres desde el indicador de posición
fputs($fich, $cad)	Escribe una cadena en el fichero
fseek($fich, $pos)	Sitúa el cursor del fichero en la posición indicada
ftell($fich)	Devuelve el indicador de posición del fichero
rewind($fich)	Sitúa el indicador de posición al principio del fichero
fopen($ruta, modo)	Abre un fichero
fclose($fich)	Cierra un fichero
fread($fich, $longitud)	Lee $longitud *bytes*
fwrite($fich, $cadena)	Escritura una cadena en un fichero
copy($origen, $destino)	Copia un fichero
unlink($borra)	Borra un fichero
move($actual, $nuevo)	Mueve un fichero
filesize($ruta)	Devuelve el tamaño del fichero en *bytes*
Filetype($ruta)	Devuelve el tipo de fichero
is_file($ruta) is_dir($ruta)	Para comprobar si la ruta corresponde a un fichero Para comprobar si la ruta corresponde a un directorio
rename($actual,$nuevo)	Cambia el nombre a un fichero

Para leer un fichero que sigue un formato determinado se puede usar la función fscanf(), que lee una línea del fichero y le aplica un formato. Hay dos maneras de usarla:

- En la primera, se le pasa el fichero y el formato y devuelve un array con los valores leídos.

  $valores=fscanf($fichero, $formato);

- También se le pueden pasar variables adicionales para que almacene en ellas los valores leídos en lugar de devolverlos. En este caso devuelve el número de valores leídos correctamente.

  $valores= fscanf($fichero,$formato,$var1,…);

Cuando queremos volver al principio de un fichero podemos usar la función rewind().

También pueden ser útiles las funciones file get contents() y file put contents(). La primera devuelve una cadena con el contenido de un fichero. La segunda hace lo contrario, escribe datos en un fichero. La ventaja que tienen es que funcionan directamente a partir de la ruta del fichero, así que no hace falta llamar a fopen() y fclose().

Para copiar un archivo usamos la función copy() indicando la fuente y el destino. Podemos usar la función die() para mostrar un mensaje de error si falla.

Para renombrar un fichero usamos la función rename()

Y para eliminar un fichero tenemos la función unlink()

También es importante comprobar si un archivo o un directorio existe, para ello utilizamos la función file_exits

7.2 Manipular directorios

Para crear un directorio mkdir() indicando los permisos. Es conveniente asegurarse antes de que la carpeta no existe para lo que usamos if_dir()

Borramos un directorio con rmdir()

Si queremos recorrer el contenido de un directorio usamos:

- Opendir abre un gestor de directorios. Es decir, que podemos abrir la carpeta que deseemos.
- Con un bucle y la función readdir() podemos ir mostrando los archivos que hay dentro. Es posible indicar que queremos evitar que nos muestre el . y el .. a la hora de presentar estos archivos.

7.3 Ficheros XML

Existen librerías para trabajar con XML. La librería SimpleXML viene activada por defecto.

La función simplexml_load_file() lee un fichero XML y devuelve un objeto de clase simpleXMLElement. El fichero se manipula a través de este objeto.

Por ejemplo Xpath() nos permite seleccionar elementos.

Para validar con esquemas XSD usaremos la clase DOMDocument y el método schemaValidate().

También existe la posibilidad de hacer transformaciones XSLT con esta librería.