Guion para la Unidad 3

Sitio: Centros - Granada
Curso: Desarrollo web en entorno servidor
Libro: Guion para la Unidad 3

Imprimido por: Aguilera Aguilera, Javier
Día: miércoles, 17 de noviembre de 2021, 08:42

Tabla de contenidos

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.

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.

El atributo *action* del formulario especifica la ruta del script al que se enviará el formulario para que lo procese.

Para especificar el método HTTP que se usará para la petición se usa el atributo *method*. Lo más habitual es utilizar POST (ya que los parámetros no aparecen en la URL), pero también es posible usar 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.

Se puede acceder a los argumentos de la URL desde un bloque PHP usando el array superglogal $_GET, que tiene un elemento por cada argumento presente en la URL. El nombre del argumento será la clave del elemento del array.

Al usar el método POST los parámetros están disponibles en el array superglobla $_POST. La clave de cada argumento dentro del array es el atributo name del elemento correspondiente en el formulario.

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

Un ejemplo de formulario:

```
<form action="ejemplo.php" method="post">
  <p><input type="text" name="nombre"></p>
  <p><input type="submit" value="Enviar"></p>
</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
- que el formulario contenga un botón de tipo submit

Nota: 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. Función header()

HTTP como sabemos es el protocolo que usamos para enviar información de páginas web, entre el servidor web y el cliente(navegador).

Las respuestas HTTP tienen dos bloques: contenido y cabecera.

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.

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

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.

Ejemplos de cabeceras:

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

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.

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. Nosotros la usaremos de esta forma en algunos ejercicios de clase.

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)
- Cuando en un formulario se logea un usuario y queremos redirigirlo a su perfil, etc.

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 lo cual podemos poner el siguiente código:
<?php header("HTTP/1.1 404 Not Found"); ?>

3. Validación 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.

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 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, 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.

Aquí puedes consultar acerca de la validación en el lado cliente

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 que no haya letras u otros símbolos.

Para poder validar los campos de una forma más precisa podemos utilizar las expresiones regulares mediante la función preg_match().

También, debemos modificar los inputs de entrada para que se eliminen algunos caracteres indeseables y de esta forma se normalicen los datos para emplearlos o guardarlos de forma segura. Incluso para prevenir ataques.

```
// Definimos la codificación en el HTTP header:

    header('Content-Type: text/html; charset=UTF-8');

// Definimos la codificación empleada al convertir caracteres:

        echo htmlspecialchars($_GET['query'], ENT_QUOTES, 'UTF-8');
```

Usando la extensión de filtrado de PHP conseguiremos de forma más rápida tanto la validación como el saneamiento de los datos.

Aquí puedes consultar las funciones de validación.

4. 1. Sistema de archivos y envío de archivos

Hay varias librerías para manejar ficheros. La más importante es Filesystem que nos permite acceder a ellos y manejar el sistema de archivos.

4.1. Ficheros de texto

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, para saber más consultar el manual.

Modo	Descripción
r	Solo lectura. Si el fichero no existe, devuelve FALSE
r+	Lectura y escritura. Si el fichero no existe, devuelve FALSE
w	Sólo 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(). Sin embargo, 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. Para poder hacerlo debemos de tener permiso a+ (leer y escribir).

4.2. Manipular ficheros y directorios

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,…);

Si deseamos volver al principio del 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().

4.3. Copiar

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

4.4. 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()

Borrar un directorio rmdir()

Recorrer el contenido de un directorio:

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. También podemos evitar que nos muestre el . y el .. a la hora de mostrar estos archivos

4.5. Envío de archivos

El propósito es subir archivos a un servidor desde un formulario en el que se permite seleccionar el archivo que deseamos cargar desde nuestro disco duro.

Para ello usaremos el método POST y configuraremos el atributo *enctype="multipart/form-data"*. Además, 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.

También será necesario usar 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 bidimiensional en el que la primera dimensión identifica el fichero según el atributo name en el formulario y en la segunda las claves son

- name: nombre del fichero en el cliente
- size: tamaño del fichero en bytes
- type: tipo MIME del fichero
- tmp_name: nombre temporal con el que se ha subido al servidor
- error: código de error asociado a la subida

El fichero se almacena en principio en el directorio temporal del servidor y se puede move al directorio que se desee con la función

move_uploaded_file($fichero, $destino)

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

También debemos de tener en cuenta que antes de trabajar con los ficheros deberemos de verificar algunos ajustes de configuración de PHP para asegurarnos de que se suben correctamente.

Estos ajustes se pueden hacer directamente en WampServer en la pestaña PHP> configuración de PHP . Aquí modificaremos las directivas upload_mad_filesize y post_max_size para que se adecuen a nuestras necesidades . Y por último, para que las horas se procesen bien modificaremos date_timezone.

Todas estas opciones se pueden configurar en el archivo **php.ini**. Estas son algunas de las directivas que puedes modificar en ese archivo:

- file_uploads : El valor de esta directiva 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 megabytes
- post_max_size : permite 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.

Por ejemplo, si tu upload_max_filesize es 16MB , es posible que desee establecer post_max_size en 20MB.

- max_file_uploads: permite establecer el número máximo de archivos que se pueden cargar a la vez. El valor predeterminado es 20.

En el manual de PHP puedes consultar más información sobre subida de archivos.

4.6. Ejemplo 1

Descarga aquí el ejemplo

4.7. Ficheros XML

Existen librerías para trabajar con XML.

- La librería SimpleXML viene activada por defecto. Proporciona un conjunto de herramientas para **convertir XML a un objeto** que pueda ser procesado con selectores de propiedades e iteradores de arrays.
  - **Clases**: SimpleXMLElement, SimpleXMLIterator.
  - Functions: simplexml_import_dom, simplexml_load_file, simplexml_load_string ...

  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.

- Se puede validar con esquemas XSD utilizando la clase DOMDocument
- También es posible utilizar transformaciones XSLT con la misma libreria.

En el manual de PHP puedes encontrar más información sobre manipulación de XML.