Manejo de Ficheros en PHP

En el desarrollo web del lado servidor, la gestión de ficheros es una parte fundamental porque muchas aplicaciones necesitan almacenar, intercambiar o registrar información en el sistema de archivos del servidor.

PHP, por su naturaleza y su integración con el sistema operativo, permite manipular ficheros de forma muy sencilla y potente. Sin embargo, dado que las operaciones con ficheros pueden afectar a la seguridad y la integridad de la información, deben manejarse con cuidado.

Existen tres grandes situaciones prácticas en las que un desarrollador web suele necesitar trabajar con ficheros:

1. Subida de ficheros

Casi todas las aplicaciones web modernas permiten subir archivos:

- Una red social permite subir una foto de perfil.
- Un sistema de gestión documental permite subir informes, facturas o contratos.
- Una intranet permite a los empleados enviar solicitudes o documentos PDF.

La subida de ficheros es, por tanto, una necesidad real y común. En el lado del servidor, PHP proporciona herramientas para recibir, validar y almacenar los archivos enviados por los usuarios a través de formularios HTML.

Funcionamiento técnico:

1. El navegador envía el archivo mediante un formulario con el atributo enctype="multipart/form-data".
2. El servidor PHP recibe el archivo de forma temporal en una carpeta del sistema (ruta temporal).
3. El programador decide qué hacer con él: validarlo, moverlo a otra carpeta definitiva, renombrarlo o rechazarlo.

El archivo recibido se representa en PHP mediante la variable superglobal $_FILES, que contiene toda la información del archivo subido: nombre original, tamaño, tipo, ruta temporal, y posibles errores.

Ejemplo práctico de subir un fichero:

En la parte del formulario, pondremos lo siguiente:

```
<form action="subir.php" method="POST" enctype="multipart/form-data">
   <label>Selecciona un archivo:</label>
   <input type="file" name="archivo">
   <button type="submit">Subir</button>
</form>
```

- enctype="multipart/form-data" es obligatorio para enviar archivos.

- El input debe ser type="file".
- El archivo se envía al script subir.php.

En la parte que recibe el envío del formulario, podríamos gestionarlo así:

```php
<?php
if (isset($_FILES['archivo'])) {
   $nombre = $_FILES['archivo']['name'];     // Nombre original
   $tmp = $_FILES['archivo']['tmp_name'];    // Ruta temporal
   $error = $_FILES['archivo']['error'];     // Código de error
   $size = $_FILES['archivo']['size'];       // Tamaño del archivo

   // Validaciones básicas
   $permitidos = ['jpg','png','pdf'];
   $extension = strtolower(pathinfo($nombre, PATHINFO_EXTENSION));

   if ($error === 0 && in_array($extension, $permitidos) && $size < 2*1024*1024) {
      $destino = "uploads/". basename($nombre);

      if (move_uploaded_file($tmp, $destino)) {
         echo "Archivo subido correctamente: $destino";
      } else {
         echo "Error al mover el archivo.";
      }
   } else {
      echo "Archivo no permitido o demasiado grande.";
   }
}
?>
```

- $_FILES['archivo'] contiene los datos del fichero subido.
- pathinfo() extrae la extensión del nombre. La función pathinfo() de PHP analiza una ruta de archivo o nombre completo y puede devolver varias partes:
    - el nombre del archivo,

    - el directorio,
    - la extensión,
    - o toda la información a la vez.

  En este caso, se le pasa el segundo parámetro PATHINFO_EXTENSION, que le dice que queremos solo la extensión del archivo.

- in_array() comprueba si la extensión es válida.
- move_uploaded_file() mueve el archivo desde la ruta temporal hasta la carpeta uploads/.
- basename extrae únicamente el nombre del archivo, por si alguien mete rutas maliciosas. Además, dependiendo del navegador y del sistema operativo, $_FILES['archivo']['name'] puede traer la ruta completa local del usuario.

2. Descarga de ficheros

Permitir que el usuario descargue ficheros es otra operación fundamental:

- Descarga de informes o facturas generadas.
- Recuperación de documentos previamente subidos.
- Envío de archivos procesados por el servidor.

La descarga se controla con PHP para poder aplicar seguridad y control de acceso, evitando que los usuarios descarguen archivos no autorizados. El navegador por sí solo no puede “leer” los archivos del servidor directamente (por seguridad).

Por eso, PHP actúa como intermediario: recibe el nombre del archivo, busca el archivo en el servidor, y envía los datos al navegador con las cabeceras adecuadas para que el navegador entienda que debe descargarlo.

Ejemplo práctico de descargar un fichero:

En un sistema real, los usuarios no escriben el nombre del archivo que quieren descargar. En su lugar, el sistema debe leer los archivos disponibles en una carpeta del servidor (por ejemplo /uploads), y generar un listado dinámico con enlaces clicables para descargar cada archivo.

Esto se puede gestionar como se muestra en la siguiente página.

```php
<?php
    $carpeta = 'uploads';
    if (is_dir($carpeta)) {
        $archivos = scandir($carpeta);
        $lista = array_diff($archivos, array('.', '..'));
        if (!empty($lista)) {
            echo "<ul>";
            foreach ($lista as $archivo) {
                echo '<li><a href="descargar.php?nombre=' . urlencode($archivo) . '">' .
htmlspecialchars($archivo) . '</a></li>';
            }
            echo "</ul>";
        } else {
            echo "<p>No hay archivos disponibles.</p>";
        }
    } else {
        echo "<p>La carpeta uploads no existe.</p>";
    }
    ?>
```

No hace falta que dominemos el código por completo, pero sí que tengamos claras las operaciones esenciales que contiene:

1. PHP comprueba si la carpeta /uploads existe.
2. Si existe, lee todos los archivos.
3. Quita los elementos especiales (. y ..).
4. Si hay archivos, genera una lista HTML <ul> con enlaces <a> a descargar.php?nombre=archivo. **YA SABEMOS QUE ESTO MANDA POR GET dicha información.**
5. Si no hay archivos, muestra un mensaje: “No hay archivos disponibles.”
6. Si la carpeta ni siquiera existe, muestra un mensaje de error.

El resultado final: **una lista clicable de archivos que el usuario puede descargar sin escribir nada manualmente**.

En la parte que recibe el envío de la petición, podríamos gestionarlo así:

```php
<?php
if (isset($_GET['nombre'])) {
    $archivo = "uploads/" . basename($_GET['nombre']);

    if (file_exists($archivo)) {
        header('Content-Description: File Transfer');
        header('Content-Type: application/octet-stream');
        header('Content-Disposition: attachment; filename="'.basename($archivo).'"');
        header('Expires: 0');
        header('Cache-Control: must-revalidate');
        header('Pragma: public');
        header('Content-Length: ' . filesize($archivo));

        readfile($archivo);
        exit;
    } else {
        echo "Archivo no encontrado.";
    }
}
?>
```

if (isset($_GET['nombre']))

Comprueba si se ha pasado el parámetro nombre por la URL. Ojo, hacemos todas estas comprobaciones porque no nos podemos fiar de que el archivo en el que hemos clickado exista (Otro usuario o proceso podría borrar o mover el archivo después de que se generó la lista, etc)

$archivo = "uploads/" . basename($_GET['nombre']);

Esto construye la ruta completa del archivo que queremos descargar, teniendo en cuenta lo que ya comentamos de que basename() es una función de seguridad que elimina posibles rutas peligrosas.

Si un atacante pusiera en la URL algo como: descargar.php?nombre=../../etc/ejemplo

- Sin basename(), PHP intentaría acceder a un archivo fuera de tu carpeta.

- Con basename(), eso se convierte en ejemplo, anulando el intento de escape.

if (file_exists($archivo))

Verifica que el archivo realmente existe en el servidor antes de intentar enviarlo.

Las cabeceras header()

Son instrucciones que PHP envía al navegador antes de mandar el archivo. Indican cómo debe interpretarse la respuesta.

header('Content-Description: File Transfer');

Solo añade una pequeña descripción del tipo de operación. Es informativo.

header('Content-Type: application/octet-stream');

Indica que lo que viene a continuación son bytes binarios, no HTML, texto ni imagen.

header('Content-Disposition: attachment; filename="'.basename($archivo).'"');

Dice al navegador: "Voy a enviar un archivo adjunto, y el nombre sugerido para guardarlo es el que está dentro de basename($archivo)."

header('Expires: 0'); header('Cache-Control: must-revalidate');header('Pragma: public');

Estas tres cabeceras gestionan la caché (almacenamiento temporal). Se usan para asegurar que el archivo se descarga siempre de nuevo, y no una copia vieja.

header('Content-Length: ' . filesize($archivo));

Informa al navegador del tamaño total del archivo en bytes, lo que permite mostrar una barra de progreso de descarga.

readfile($archivo)

Lee el archivo y envía su contenido al navegador. Como ya hemos configurado las cabeceras, el navegador no intentará mostrarlo, sino que abrirá el cuadro de descarga.

3. Gestión de logs

El desarrollador de backend es lógico que desarrolle un mecanismo para dejar registro logs tanto de los eventos normales que van sucediendo en la aplicación web, como de los errores.

Hasta ahora, lo único que hemos hecho es utilizar la función echo en nuestras aplicaciones, que simplemente envía texto al navegador.

Esto es genial porque permite mostrar mensajes informativos al usuario, y podemos depurar cosas (por ejemplo, echo var_dump($row); para ver los datos).

Pero ojo, también lo hemos usado de momento para mostrar errores técnicos, y en este caso el usuario ve información que no debería ver (por ejemplo, el nombre de la tabla o un mensaje del servidor).

Además, los mensajes no quedan registrados en ningún sitio.

Por eso, aunque echo es genial para aprender y para pruebas, no es un sistema de registro ni de control de errores.

Un log es un *registro histórico* que guarda lo que ha pasado dentro de tu aplicación:
una especie de "diario de actividad".

Gran ventaja: aunque el usuario no vea nada, tú puedes mirar el fichero de log y saber exactamente qué ha pasado, incluso días después.

En PHP, el log, que típicamente se crea en una carpeta logs que hemos generado antes, se escribe normalmente así:

```
file_put_contents(
    'logs/eventos.log',
    date("Y-m-d H:i:s") . " - Usuario filtró productos por categoría Ropa\n",
    FILE_APPEND
);
```

file_put_contents() escribe texto en un archivo.

FILE_APPEND hace que añada al final en lugar de sobrescribirlo.

date() pone la fecha y hora exactas.

\n salta de línea.

De esta forma puedes ir registrando acciones de usuarios, errores de conexión o de consulta, etc.

Pero ojo… ¡los logs no evitan los errores!

Los logs registran lo que pasa, pero no evitan ni controlan los errores.

Si ocurre un error en tu código, la aplicación puede seguir rompiéndose igual — solo que el fallo quedará anotado en un archivo.
Por tanto, los logs sirven para saber qué pasó, pero no para impedir que pase.

Para controlar los errores: las excepciones

Cuando quieres que tu aplicación no se rompa al producirse un error, y que tú tengas el control total sobre la reacción, entran en juego las excepciones.

Las excepciones permiten que un error:

1. Interrumpa el bloque de código donde ocurre (para no seguir ejecutando cosas incorrectas),
2. Salte a un bloque preparado para capturarlo (catch),
3. Y desde ahí puedas decidir qué mostrar al usuario y qué registrar en el log.

Ejemplo:

```
try {
    $conexion = new mysqli("localhost", "usuario", "clave", "base");
    if ($conexion->connect_error) {
        throw new Exception("Error de conexión: " . $conexion->connect_error);
    }
    $resultado = $conexion->query("SELECT * FROM Productos");
    if (!$resultado) {
        throw new Exception("Error en la consulta: " . $conexion->error);
    }
    echo "Consulta ejecutada correctamente.";
} catch (Exception $e) {
    // Registrar en log
    file_put_contents('logs/errores.log',
        date("Y-m-d H:i:s") . " - " . $e->getMessage() . "\n",
        FILE_APPEND
    );
    // Mostrar mensaje genérico al usuario
    echo "Ha ocurrido un error. Inténtelo más tarde.";
}
```

Aquí tienes el sistema completo:

- Si algo falla, se lanza (throw) una excepción con un mensaje detallado.
- El bloque catch la captura sin romper la página.
- Se guarda la información en un log de errores.
- El usuario no ve información técnica, solo un aviso amable.

EJERCICIO: INTENTA INCLUIR ESTA GESTIÓN DE LOG Y EXCEPCIONES EN ALGUNO DE TUS CÓDIGOS, Y HAZ UNA CONEXIÓN ERRÓNEA A LA BASE DE DATOS PARA VER SI FUNCIONA CORRECTAMENTE