

---

---

# File upload and management

— DWES UD6 —

---

---

# 1.- Upload files to the server

The way to send files to the server from the web application is by using forms. In order to send files, the following aspects must be taken into account:

- The form submission method must always be **post**
- The form must have the attribute **enctype="multipart/form-data"**.
- The server has a maximum file size configured.
- The files are stored in a temporary directory and when the execution of the script that receives the data ends, these files are deleted.

# 1.- Upload files to the server

To send files, the **file** type **inputs** are used in the form:

```
<form action="subida.php" method="post" enctype="multipart/form-data">  
  Selecciona el archivo a subir:  
  <input type="file" name="archivo" id="archivo">  
  <input type="submit" value="Enviar">  
</form>
```

A form requires one input file field per file. So, to send more than one file, it's needed one input field per file.

# 1.- Upload files to the server

To change the maximum upload size you can choose two options:

- In the configuration file **php.ini**:  
upload\_max\_filesize = 2M

- Adding a hidden field to the form indicating the size in bytes:

```
<input type="hidden" name="MAX_FILE_SIZE" value="1000000">
```

The value of MAX\_FILE\_SIZE cannot be greater than that specified in upload\_max\_filesize

# 1.- Upload files to the server

Once one or more files have been sent to the server, the script that receives the form data can use the **`$_FILES`** superglobal array to access the information about the uploaded files.

The `$_FILES` array will have as many elements as there are files uploaded from the form. If you upload 5 files, the array will have 5 elements.

Each element of the array will be an array with the information of a file:

- **tmp\_name**: temporary file path on server.
- **name**: original file name (on the client)
- **size**: size in bytes of the file.
- **type**: file MIME type (image/gif, text/plain...)
- **error**: error code (UPLOAD\_ERR\_OK if it has been uploaded correctly)

# 1.- Upload files to the server

Given the following form:

```
<form action="subida.php" method="post" enctype="multipart/form-data">  
    Selecciona el archivo a subir:  
    <input type="file" name="archivo" id="archivo">  
    <input type="submit" value="Enviar">  
</form>
```

In the **subida.php** file you can access the submitted data as follows:

```
echo $_FILES['archivo']['tmp_name'];
```

## 2.- Common problems

- Do not have permission on the route indicated in the policy **upload\_tmp\_dir** in php.ini.
- The **memory\_limit** directive in php.ini has a very small value or less than **upload\_max\_filesize**.
- The **max\_execution\_time** directive in php.ini has a low value and the script execution including the file upload exceeds this time.
- The **post\_max\_size** directive in php.ini has a very low value, its value must be greater than upload\_max\_filesize (maximum file size + rest of the data sent in the form).

## 3.- Security

Receiving files from the client can involve security risks, so some checks must be made. Check **`$_FILES['upfile']['error']`** in order to continue only if the file has been uploaded correctly.

```
switch ($_FILES['upfile']['error']) {  
    case UPLOAD_ERR_OK:  
        break;  
    case UPLOAD_ERR_NO_FILE:  
        throw new RuntimeException('No file sent.');
```

  

```
    case UPLOAD_ERR_INI_SIZE:  
    case UPLOAD_ERR_FORM_SIZE:  
        throw new RuntimeException('Exceeded filesize limit.');
```

  

```
    default:  
        throw new RuntimeException('Unknown errors.');
```

  

```
}
```

## 3.- Security

Check that the type of received file is the expected using **`$_FILES['upfile']['type']`**

```
// Se comprueba que sea del tipo esperado
if ($_FILES['imagen']['type'] != 'image/gif') {
    echo 'Error: No se trata de un fichero .GIF.';
    exit();
}
```

## 3.- Security

Or check the mime type of received file by yourself:

```
$finfo = new finfo(FILEINFO_MIME_TYPE);  
if (false === $ext = array_search(  
    $finfo->file($_FILES['upfile']['tmp_name']),  
    array(  
        'jpg' => 'image/jpeg',  
        'png' => 'image/png',  
        'gif' => 'image/gif',  
    ),  
    true  
) {  
    throw new RuntimeException('Invalid file format.');
```

## 3.- Security

Check that the file has really been uploaded using [is\\_uploaded\\_file](#)

```
<?php
```

```
if (is_uploaded_file($_FILES['archivo_usuario']['tmp_name'])) {  
    echo "Archivo ". $_FILES['archivo_usuario']['name'] ." subido con éxito.\n";  
    echo "Mostrar contenido\n";  
    readfile($_FILES['archivo_usuario']['tmp_name']);  
} else {  
    echo "Posible ataque del archivo subido: ";  
    echo "nombre del archivo '". $_FILES['archivo_usuario']['tmp_name'] . "'.";  
}
```

```
?>
```

# 3.- Security

Move the file using [move\\_uploaded\\_file](#)

```
<?php
$uploads_dir = '/uploads';
foreach ($_FILES["pictures"]["error"] as $key => $error) {
    if ($error == UPLOAD_ERR_OK) {
        $tmp_name = $_FILES["pictures"]["tmp_name"][$key];
        // basename\(\) puede evitar ataques de denegación de sistema de ficheros;
        // podría ser apropiada más validación/saneamiento del nombre del fichero
        $name = basename($_FILES["pictures"]["name"][$key]);
        move_uploaded_file($tmp_name, "$uploads_dir/$name");
    }
}
?>
```

## 4.- File system functions

When working with files on the server, it is a good help to know the functions that allow you to work with the file system ([documentation](#)).

Some useful functions:

- delete
- realpath(\_ \_FILE\_ \_)
- dirname
- is\_dir
- rename
- mkdir
- rmdir

# 5.- Image processing

Any of the image processing functions can be applied to images uploaded through forms, such as:

- Add [watermark](#).
- Create different versions of images with:
  - [imagescale](#)
  - [Imagecopyresized](#)

These modifications can be made to both the temporary file and the final file moved to the corresponding directory.

There are many functions for [image](#) processing.

## 6.- Store files on the server

Just as the user may be required to send a photo to the server, they may be interested in sending any other type of files.

We have seen how to store the files in directories but there is the possibility of saving the files in a database field as binary data (BLOB).

Disadvantages:

- Slowness in file queries.
- Difficulty when recovering files.
- Low compatibility between Database Management Systems.
- Server overload.

Advantages:

- Security.
- If you do not have permissions on the file system

The usual thing is to store files in directories. If you choose to store the files in the database, you should try to keep them small or use a specific system.

# Exercise

Create an application with a 'Register user' form including a profile image that must be uploaded. Include a profile page for an authenticated user (you will need a login page) where he can see all of his data. The application must:

- Check that the image is of an allowed type: png or jpg.
  - Check the image size, maximum (360x480px).
  - Save two versions of the image:
    - 360x480px - will show on profile page
    - 72x96px - will be displayed next to the username when logged in
- For example, image names can be:  
idUserBig.png and idUserSmall.png
- The directory where the images will be saved will be: /img/users/\$username
  - In the database users table, the path to each image must be saved in a different field.