# PHP and MySQL

## EXERCISE 4: PHOTO ALBUM SERVER VERSION D

### Photo Album Server Version D

Save a copy of **photoalbum-server-version-c.php** and call it **photoalbum-server-version-d.php**. Add the following HTML code after the end PHP tag ?>.

|  |
| --- |
| <div class='w3-container w3-center'>  <a href="#id01" class='link'>Add images</a>  </div>  <div id="id01" class="w3-modal">  <div class="w3-modal-dialog">  <div class="w3-modal-content w3-card-4">  <header class="w3-container w3-teal">  <a href="#" class="w3-closebtn">&times;</a>  <h2>Upload photograph</h2>  </header>  <div class="w3-container">  <form action='?' method='post' enctype='multipart/form-data' id='photoform' class='w3-container'>  <div class="w3-group">  <input type='file' name='uploadfile' class='w3-input'>  <label class="w3-label">Image file</label>  </div>  <div class="w3-group">  <input type='text' name='name' class='w3-input' required>  <label class="w3-label">Name</label>  </div>  <div class="w3-group">  <textarea name='description' class='w3-input' required></textarea>  <label class="w3-label">Description</label>  </div>  <input type='hidden' name='formid' value='fileupload'>  <input type='submit' value='Upload' id='uploadbutton' class='w3-btn'>  </form>  </div>  <footer class="w3-container w3-teal">  <p></p>  </footer>  </div>  </div>  </div> |

This defines a form to upload an image which is initially hidden by the w3css classes used. The first <div> in this code contains a link which points to the hidden form and uses w3css classes to make the form appear when it is clicked.

The form action is set to a ‘?’. This will cause the form to call the script it is in (**photoalbum-server-version-d.php**) but will make sure that there are no values attached to the URL. This is because if we have just deleted a picture there would be a deletionid attached to the URL and we want to get rid of that.

The form has four values in it: a file upload whose name is ‘uploadfile’, a text field whose name is ‘name’, a text area whose name is ‘description’ and a hidden field whose name is ‘formid’. These are all highlighted in yellow in the code above. Because the form uses the method ‘post’ the values will be accessible to the script in the $\_POST array.

The hidden ‘formid’ field is so that the script can tell whether it is receiving a form submission or not, and therefore whether to try and add the photograph. The ‘name’ and ‘description’ fields are simple text to be inserted into the database. The file upload field is a bit more complicated. This is because PHP handles file uploads using another special array, this time called $\_FILES. It can handle any number of files being uploaded in the same form, although we are allowing only one. The $\_FILES array will have a named entry for each file upload field in the form and the name is the name given to the form element. In our case this is ‘uploadfile’. Thus there will be an element in the $\_FILES array with the name ‘uploadfile’, that is $\_FILES[‘uploadfile’]. The value of this element is actually itself an array with a number of named elements. The most important ones are ‘name’ and ‘tmp\_name’. If we assign the value of $\_FILES[‘uploadfile’] to a variable called $fileinfo with the statement $fileinfo = $\_FILES[‘uploadfile’]; then

* $fileinfo[‘name’] will contain the name of the uploaded file.
* $fileinfo[‘tmp\_name’] will contain the name PHP has saved the uploaded file as on the server.

We need to do two things: create a new row in the database table and move the uploaded file into the **images** folder. We actually want to give the uploaded file a unique name, because otherwise if two different pictures called “mypicture.jpg” were uploaded the second would overwrite the first. Fortunately, when we add the database row we get a unique id number. We could just name the file with the id number but it makes it easier to see what it going on in the images folder if we keep the file’s original name and stick the id number onto the front of it.

Here is a function that does all of that. Add it to your **photoalbum-common.php** file.

|  |
| --- |
| function addPhotograph( $pdo, $name, $description, $user, $fileinfo) {  // create DB entry to get an ID  $stmt = $pdo->prepare("INSERT INTO photographs ( `name`, `description`, `user`) VALUES (?, ?, ?)");  $stmt->execute( array( $name, $description, $user));  $photoid = $pdo->lastInsertId();    // receive the files, I only have one  $tmp\_name = $fileinfo["tmp\_name"];  // create final image name from unique DB id  $name = $photoid."\_".$fileinfo["name"];  // update the database with the final image name  $stmt = $pdo->prepare("UPDATE photographs SET `image`=? WHERE `photoid`=?");  $stmt->execute( array( $name, $photoid));  // move the uploaded file to the final location  $result = move\_uploaded\_file($tmp\_name, "images/$name");  return $photoid;  } |

Once this is done the code to add to **photoalbum-server-version-d.php** is quite simple. Add the following after the code which does the deleting and before the line which selects the photographs from the database (both shown in light grey here):

|  |
| --- |
| } else {  print "<div class='message'>Image deleted.</div>\n";  }  }  if ( isset( $\_POST['formid']) && $\_POST['formid'] == 'fileupload') {  $photoid = addPhotograph( $pdo, $\_POST['name'], $\_POST['description'], 'self', $\_FILES['uploadfile']);  }  $sql = "SELECT \* FROM `photographs` ORDER BY `photoid`";  $stmt = $pdo->query( $sql); |

This code checks to see if the request contains the field ‘formid’ and if it does, if that field has the value ‘fileupload’. This is to test whether the form is being submitted. If this is the case it calls the ***addPhotograph*** function in **photoalbum-common.php** with the database connection object, the name of the photograph as entered into the form, the description of the photograph as entered into the form, the value ‘self’, and the information about the uploaded file from the $\_FILES array.

The ‘self’ value is inserted into the database as the value for the owner of the photograph. This field has not yet been used for anything but is there for future development.

View **photoalbum-server-version-d.php** in a browser and upload a photograph. Check the contents of the photographs table using PHPMyAdmin to see that it has been added, and examine the **images** folder to see the uploaded file is there.

|  |
| --- |
| **EXERCISE 4: SERVER PHOTO ALBUM VERSION D** |
| ***Required in your report (basic exercise).*** |
| * A clickable link to photo-album-server-version-d.php * A complete listing of the source code of photo-album-server-version-d.php. * Reflection on the tasks and all activities involved. |
| ***Extended Tasks*** |
| * Create a means of uploading multiple photographs with a single form submission. This isn’t that easy to do because PHP delivers the information to you in a complex array structure. To do this, create a form in a separate page of HTML and create a dedicated PHP script for it to upload to. The PHP script should include the ***dbutils.php*** and ***photoalbum-common.php*** files. Start by using a file upload tag with the *multiple* attribute and a name with empty square brackets on it like this: <input type='file' name='uploads[]' multiple>. Then read <http://php.net/manual/en/features.file-upload.multiple.php> and especially the first comment from “phpuser at gmail dot com” who provides a function for rearranging the files array into a better form. If you use the function he provides in the comment, you can then use a loop to feed each file array to the addPhotographs function. You won’t have names and descriptions for each file, so just feed the file name (or even an empty string) into the function for those values. The script should output a confirmation message and offer a link to the normal photoalbum page so the uploaded pictures can be seen. |
| *Original additional work:*  *Investigate and experiment with any related subject matter that interests you.* |