# JavaScript and Ajax

## EXERCISE 4: PHOTO ALBUM CLIENT-RENDERED VERSION D

### Photo Album Client-Rendered Version D

The next version of the photoalbum adds the ability to upload new photographs to the album. As it happens file uploads are difficult to manage from the client side (it does not seem to be possible to transmit both the file data and the fields of the form at the same time, meaning that two separate Ajax messages are necessary). To accomplish this we will use some ready made code which adds functionality to JQuery.

The code and an explanation of it can be found at this site:

<http://abandon.ie/notebook/simple-file-uploads-using-jquery-ajax>

However, all the files you need are provided on moodle in the file ***ajaxupload.zip***. Download this file and extract it so that you have a folder called **ajaxupload**, then put this folder inside the **lib** folder on the webserver.

As mentioned, this code adds functionality to JQuery. Once you have it installed, there is a new JQuery function called ***fileupload***. This function basically allows us to set up a whole bunch of handler functions and attach them to a form that includes file upload fields.

Make a copy of ***photoalbum-client-rendered-c.html*** and call it ***photoalbum-client-rendered-d.html***.

The first thing we need to add to this version is the upload form and the “Add images” link.

To do this, add exactly the same code as was given in PHP & MySQL Exercise 4. However, change the action attribute of the form from **?** to **json-photoalbum-upload.php**.

The next thing we need to do is add a line to import the fileupload javascript. Assuming you have put it in the **lib** folder as described above, this means adding a script import to the head of the file. Add the following line immediately after the line which imports the JQuery library itself:

<script src='lib/ajaxupload/jquery-fileupload.min.js'></script>

This makes the fileupload function available for us to use.

As mentioned above, the fileupload function allows us to set up handler functions on a given form. The functions we set up actually get used when the form is submitted. We need to set them up ready when the page is loaded, which means inside the ***$(document).ready*** function. Change the existing version to that as shown below (new code is bold):

|  |
| --- |
| $( document).ready(  function() {  loadPhotos();  **$('#photoform').fileUpload({**  **beforeSubmit :**  **function(uploadData){ console.log(uploadData); return true; },**  **// access the data returned by the upload return false to stop the submit ajax call**  **success :**  **function(data, textStatus, jqXHR){ },**  **// Callback for the submit success ajax call**  **error :**  **function(jqXHR, textStatus, errorThrown){ alert( "error"); console.log(jqXHR); },**  **// Callback if an error happens with your upload call or the submit call**  **complete :**  **function(jqXHR, textStatus){ $('#photoform').get(0).reset(); location.href='#'; loadPhotos(); }**  **// Callback on completion**  **});**  }  ); |

The upload form has the id ***photoform***, so ‘#photoform’ selects that form. We then add four separate functions to it which run at various times. The first, labelled **beforeSubmit** simply dumps some information on the browser console log, its not necessary but lets us see what is being sent for debugging purposes. The second, labelled **success**, runs when the first AJAX request returns. Success means the request has been replied to, not that it did what it was supposed to. In this case, the PHP script at the other end will reply with a JSON object that either contains a success flag, or an error flag and message which this function then displays in a dialog. The third labelled **error** will run if a communication error occurs, again it displays an alert with the error that occurred displayed. The last is labelled **complete**. This will be run when everything has completed and it does three things:

1. Clears all the fields of the upload form. (***$('#photoform').get(0).reset();***)
2. Clears the value of the targeted link causing the form to be hidden again. (***location.href='#';***)
3. Runs loadPhotos() again, to rebuild the gallery with the newly uploaded image included.

With these changes everything is in place in the client code, however, the form will be sent to the PHP script **json-photoalbum-upload.php**, which does not exist yet.

Create this script and put the following code in it:

|  |
| --- |
| <?php  require\_once("lib/dbutils.php");  require\_once("photoalbum-common.php");  header("content-type: application/json");  // This is called twice, once with the form value file\_upload\_incoming set to the number of files,  // and the file data then again with values of the form fields. Note that the file data DOES NOT  // come in the normal format $\_FILES['formfieldname'] but as a numbered array $\_FILES[0] etc.  $pdo = connect();  if ( isset( $\_POST['file\_upload\_incoming'])) {  // FIRST AJAX INVOCATION  // create a databse record with dummy name and description but pointing to the uploaded photo  $photoid = addPhotograph( $pdo, 'incoming', 'pending', 'self', $\_FILES[0]);  // send the photoid back to the javascript process on the client  print "{\"photoid\": \"".$photoid."\"}";  } else {  // SECOND AJAX INVOCATION    // now update the database with the other fields  $stmt = $pdo->prepare("UPDATE photographs SET `name`=?, `description`=? WHERE `photoid`=?");  $stmt->execute( array( $\_POST['name'], $\_POST['description'], $\_POST['photoid']));  print "{\"completed photoid\": \"".$\_POST['photoid']."\"}";  }  ?> |

Note that this includes ***dbutils.php*** in order to access the database and ***photoalbum-common.php*** in order to make use of the **addPhotograph** function we already used for the upload using PHP.

This script is called twice by the upload code: first with the file data, then with the form field data. The presence of the value ‘file\_upload\_incoming’ indicates that it is the first invocation, and its absence indicates the second invocation, so the script uses this with an if statement to decide what to do.

First time it calls addPhotograph to create an entry in the database for the new picture and copy the uploaded file into the images folder. Note that it passes dummy values of ‘incoming’ and ‘pending’ for the name and description fields.

The second time it updates the database to fill in the actual name and description values the user typed in to the form.

|  |
| --- |
| **EXERCISE 4: PHOTO ALBUM CLIENT-RENDERED VERSION D** |
| ***Required in your report (basic exercise).*** |
| * A clickable link to photoalbum-client-rendered-d.html. * Reflection on the tasks and all activities involved. |
| ***Extended Tasks*** |
| * No extended tasks are defined for this exercise. |
| *Original additional work:*  *Investigate and experiment with any related subject matter that interests you.* |