# JavaScript and Ajax

## EXERCISE 2: PHOTO ALBUM CLIENT-RENDERED VERSION B

### Photo Album Client-Rendered Version B

Just as the first PHP photalbum we created had the photographs hard coded in an array, the first version of the JavaScript version also hard coded the photographs, in the form of a JSON string. The second PHP version deleted the array and got the list of photographs from the database instead. Now we will do something similar with the JavaScript version. Make a copy of ***photo-album-client-rendered-a.html*** and call it ***photo-album-client-rendered-b.html*** then delete the json string declaration at the top:

|  |
| --- |
| var json = '[' +  '{ "image": "b1.png", "name": "John", "description": "<p>This is John Lennon. He was one of the Beatles.</p><p>He was one of the two main song-writers.</p>" },' +  '{ "image": "b2.png", "name": "Paul", "description": "<p>This is Paul McCartney. He was also one of the Beatles.</p><p>He was the other main song-writers.</p>" },' +  '{ "image": "b3.png", "name": "George", "description": "<p>This is George Harrison. He was also one of the Beatles.</p>" },' +  '{ "image": "b4.png", "name": "Ringo", "description": "<p>This is Ringo Starr. He was also one of the Beatles.</p>" }' +  ']';  **DELETE THIS CODE** |

KEEP the ***createPhotoHTML*** function, but delete the existing function that displays the photos as shown below:

|  |
| --- |
| $( document).ready(  function() {  var photos = $.parseJSON( json);  for (i = 0; i < photos.length; i += 3) {  var rowDiv = $( "<div class='w3-row'>");  $('#album').append( rowDiv);  for( j = i; j < i + 3; j++ ) {  if ( j < photos.length) {  rowDiv.append( createPhotoHTML( photos[j]));  }  }  }  }  );  **DELETE THIS CODE** |

Now add the following code, a function which loads the photo data from the server and generates the HTML to display them, and a ***$(document).ready*** function to trigger it.

|  |
| --- |
| function loadPhotos() {  $.getJSON( "json.txt",  function( photos) {  $('#album').empty();  for (i = 0; i < photos.length; i += 3) {  var rowDiv = $( "<div class='w3-row'>");  $('#album').append( rowDiv);  for( j = i; j < i + 3; j++ ) {  if ( j < photos.length) {  rowDiv.append( createPhotoHTML( photos[j]));  }  }  }  }  );  }    $( document).ready(  function() {  loadPhotos();  }  ); |

In the previous version the ***$(document).ready*** event invoked an anonymous function which parse the JSON and then looped through the photographs adding them to the DOM.

In this version the ***$(document).ready*** event invokes an anonymous function that simply calls the declared function ***loadPhotos().***

***loadPhotos*** uses the JQuery helper function **getJSON**. **getJSON** requires two parameters: the first is a URL which provides a JSON data object. The second is an anonymous function which will be invoked when the JSON has been retrieved and parsed into a JavaScript object. That object will be passed to the function. This anonymous function is basically the same as the code that was originally in the ***$(document).ready*** function,and loops through the photographs adding them to the DOM.

Paste the following into a new file and save it with the name json.txt, alongside your new photoalbun script. With this is place the script should display two photographs based on this JSON data.

|  |
| --- |
| [  {  "image": "b3.png",  "name": "FAKED",  "description": "<p>This is JSON data read from a plain text file.</p>"  },  {  "image": "b4.png",  "name": "JSON",  "description": "<p>A better idea would be to read it from a dynamic process.</p>"  }  ] |

This just reads the JSON from a static file, which is not much use. We want to read it from a dynamic PHP script. This means writing a script which can emit JSON data. We also want to read the data for the JSON from a database.

We already have functions in the ***dbutils.php*** script which read a table from the database and provide it as an HTML table or as a raw variable dump. We need a similar function that converts it into a JSON data structure. Add the following function to your existing ***dbutils.php***:

|  |
| --- |
| function jsonTable( $pdo, $table, $whereOrder = "") {  $sql = "SELECT \* FROM `".$table."`";  if ( $whereOrder != "") {  $sql .= " ".$whereOrder;  }  // specify only an associative array to be returned  $stmt = $pdo->query( $sql, PDO::FETCH\_ASSOC);  print json\_encode( $stmt->fetchAll(PDO::FETCH\_ASSOC));  } |

Note how all the work of converting the database result into a JSON object is achieved by a single built-in function ***json\_encode***.

To create a script that returns JSON data requires another detail: we need to set the content-type header of the HTTP response to the value application/json. This can be accomplished using the PHP header command as the first line in the script – see below.

Create a new file called ***json-photoalbum-photos.php*** and put the following into it:

|  |
| --- |
| <?php  header("content-type: application/json");  require\_once("lib/dbutils.php");  $pdo = connect();  print jsonTable( $pdo, "photographs", "ORDER BY `photoid`");  ?> |

You should have no difficulty understanding what this code does.

Now change the JavaScript in so that it invokes this script instead of loading “json.txt”, upload them all to the server and make sure it works (you see the photographs you have defined in your database).

|  |
| --- |
| **EXERCISE 2: PHOTO ALBUM CLIENT-RENDERED VERSION B** |
| ***Required in your report (basic exercise).*** |
| * A clickable link to photoalbum-client-rendered-b.html. * A description in your own words of how ***json-photoalbum-photos.php*** works. * 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.* |