AJAX – LAB

INTRODUCTION

AJAX allows developers to send requests for data to the webserver without the need to refresh the browser. This can be used to improve user interactions.

GETTING STARTED

In these examples we'll use jQuery's AJAX methods `$_get`, `$_post` and `$_ajax`.

We'll also look at using third party APIs including Google Maps.

In the ZIP for this lab you'll find that the current jQuery 3.x library has been included in the scripts folder.

BEFORE YOU START

Ensure you can reference your **conn.inc.php** file from the PHP labs as we'll be using PHP to query the database.

SHOWING AND HIDING EXTRA CONTENT WITH $_GET

Open the file **index.php**. This file queries the **movies** table to return a list of film names. We are going to use AJAX to make a call to the database to reveal more information about each film when the user clicks on the `<a>` enclosing the text **Preview**.

This example might help you understand the techniques used in this example.

http://www.mustbebuilt.co.uk/SHU/WAD/demo/javascript/jq-data-get.html

Open the file *getPreview.php* in the **data** folder. Notice that the file contains a `prepare` statement to query the database by **filmID**. The file also returns a header to indicate the content is JSON.

Add the following:

```
$returnAr = array("filmDesc" => $filmDescription);
echo json_encode($returnAr);
```

This code will create an array from the data extracted by the query. The array is then displayed via the method `json_encode`. This will render the PHP array as JSON, that can then be called by an AJAX call.

Notice that there is a link to the *listing.js* file in the *index.php*. Open *listing.js* and add the following Javascript.

```
var myVars = {'filmID': '1'};
$.get('data/getPreview.php', myVars, function(myData) {
     console.dir(myData);
}, 'json');
```

This code makes an AJAX GET call using jQuery's `$.get` method. The jQuery's `$.get` method takes the following parameters.

$.get(url, variablesToSend, callBackFunction)

This code sends a name/value pair of **filmID = 1** to the **getPreview.php** file.

- Look at the values return, as displayed in the console. What is displayed when you `console.info` the value `myData.filmDesc`?
- To make this more dynamic place the AJAX logic inside of a click event attached to all the links of class **getPreview**. Use `ev.preventDefault()` to stop the `<a>` tags behaving as native links.
- Notice how each `<a>` tag has a custom attribute named `data-id`. Use the jQuery `attr()` method to extract this value and then send it as a parameter to the PHP script.

```
$('.getPreview').on('click', function(ev){
          ev.preventDefault();
          var myVars = {'filmID': $(this).attr('data-id')};
          $.get('data/getPreview.php', myVars,
function(myData) {
     console.dir(myData);
     }, 'json');
});
```

There is a `<div>` node of class **fullDetails** that is a sibling of the `<div>` of class **preview**. It is the **fullDetails** node that we would like to hold the film description returned by the AJAX call.

Add an event to the *listing.js* attached to **getPreview** as follows:

```
$('.getPreview').on('click', function(ev){
          ev.preventDefault();
          var myVars = {'filmID': $(this).attr('data-id')};
          var currentNode = $(this);
          var detailsNode =
     $(this).parents('.grid').find('.fullDetails');
          $.get('data/getPreview.php', myVars,
function(myData) {
        console.info(detailsNode);
     }, 'json');
});
```

The jQuery here traverses the DOM to locate the **fullDetails** node by locating the parent of the clicked element of class **grid** and then finding the an element of class **fullDetails** inside of the **grid**.

Once we have the correct DOM element we can add the data from the JSON into it and use jQuery's `slideDown()` method to reveal the text and change the text of the **preview** node to 'Hide Preview'.

```
$.get('data/getPreview.php', myVars, function(myData) {
          $(detailsNode).html(myData.filmDesc).slideDown(500);
          $(currentNode).html('Hide Preview');
}, 'json');
```

We'll also add a if/else condition such that if the `html()` value of **getPreview** is 'Preview' the data will be loaded - else it will be hidden with a jQuery `hide()`.

So the final code for *listing.js* will be:

```
$('.getPreview').on('click', function(ev){
     var currentNode = $(this);
     var detailsNode =
$(this).parents('.grid').find('.fullDetails');
     if($(this).html() === 'Preview'){
          ev.preventDefault();
          var myVars = {'filmID': $(this).attr('data-id')};
          $.get('data/getPreview.php', myVars,
function(myData) {

     $(detailsNode).html(myData.filmDesc).slideDown(500);
               $(currentNode).html('Hide Preview');
     }, 'json');
     }else{
          $(detailsNode).html('').hide();
          $(currentNode).html('Preview');
     }
});
```

UPDATING DATA WITHOUT A PAGE REFRESH WITH $_POST

Open the file *fullDetails.php*. This page uses a PHP query to display 'drilling deeper' information about a particular film. We want the user of the page to be able to add to a score rating for film via the radio buttons already included in the page. We'll do this with jQuery's AJAX method `$_POST`.

To do this we'll add a new field to the database. Open phpMyAdmin at http://homepages.shu.ac.uk/mysql and add a field called **userRating** as a `varchar` of 30 characters in length. Allow this field to be 'null'.

We'll be storing a JSON string in this field like:

```
{"Reviews":59,"Score":149}
```

Here **Reviews** is the number of user reviews, **Score** the total score. By dividing **Score** by **Reviews** we'll get an average user score.

Open the file *updateReview.php*. This PHP script is designed to run a SQL query to update the **userRating** field in the movies table with a value received via HTTP POST.

Add the following to the *updateReview.php* so that it returns JSON to the AJAX call:

```
echo $backToJson;
```

Add a jQuery change event to the radio buttons in the form such that when they are changed a value is dispatched to the AJAX. This is the basic event:

```
$('input[name=filmReview]').on('change', function(){
     console.info('Fired');
});
```

We can create a text string in standard URL-encoded notation to send to the PHP script using jQuery's `serialize()` method. If you look at the form already included inside the **fullDetails.php** page it includes a hidden field that contains the **filmID** value. This will be send along with the **filmReview** value so that the PHP script in **updateReview.php** knows which record to update. To serialize both form values use:

```
var sendVals = $('form').serialize();
```

- Create the necessary `$_post()` method to update the database when the radio buttons are changed.

The jQuery `$post()` method takes the same form as the `$_get()` but sends data via HTTP POST as opposed to HTTP GET.

$.post(url, variablesToSend, callBackFunction)

Add a switch/case to the callback function to display the average user rating.

```
$.post("data/updateReview.php", sendVals, function(myData){

     console.dir(myData);
     var noOfRatings = myData.Reviews;
     var currentScore = myData.Score;
     var newDisplayScore = (currentScore/noOfRatings);
     newDisplayScore = Math.round(100*newDisplayScore)/100;
     $('.currentRating').html(newDisplayScore);

}, "json");
```

CROSS DOMAIN CALLS

The last two AJAX calls were on the same domain. We can also use AJAX to call data from other domains although because of the security issues involved in cross domain calls we need to call JSONP (JSON-Padded).

- Take a look at http://www.tastekid.com/
- Have a look at their API document here http://www.tastekid.com/page/api
- Register as a developer so you can use their API

When you register you will be given a 'key' value. The URL to call with your AJAX call could take the form of:

```
http://www.tastekid.com/api/similar?q=full+monty&type=movie&jsonp=tasteKid&k=<YOURKEY>
```

Tip: The tastekid API document gives you lots of variants on this to experiment with.

As this AJAX call is by its nature more complex we need to deploy jQuery's `$_ajax()` as this gives us a full range of options to control the AJAX call and return.

Firstly retrieve the film name value from the page with:

```
var myFilmName = $('h2').html();
```

We'll also create an array to receive the suggested film titles returned by tasteKid.

```
var suggestedFilms = new Array();
```

We can then build the AJAX call:

```
$.ajax({
type: 'GET',
url:
'http://www.tastekid.com/api/similar?q='+myFilmName+'&type=mov
ie&callback=handleTcReturn&k=YOURKEY',
crossDomain: true,
contentType: 'application/json',
dataType: 'jsonp'
});
```

This requests a JSONP response from tastekid and as such we need to create a callback function to handle the response named **handleTcReturn**.

Add the **handleTcReturn** function and add consoles to see the values returned:

```
function handleTcReturn(myData){
     console.dir(myData);
     console.dir(myData.Similar.Results);
}
```

The **myData.Similar.Results** value is a Javascript object that contains the names of the films suggested by tastekid. We could use a `$.each()` to loop around the values in the object to add them to the array we created earlier and then add them to the DOM.

```
$.each( myData.Similar.Results, function( key, value ) {
     suggestedFilms.push('<li>' + value.Name + '</li>');
});
$(".suggestions").html(suggestedFilms.join("\n"));
```

The above loops the Javascript object, and pushes the values retrieved into the **suggestedFilms** array, each value been enclosed in a `<li>` as we'll be adding them to the `<ul>` in the DOM. The **suggestedFilms** array is then added to the DOM via the jQuery `html()` method.

The 'vanilla'Javascript `join()` method is used to extract the array values as a string - the `\n` is just used to separate the values adding a new line in the resultant HTML code to make it more readable.

ADDING A GOOGLE MAP TO SHOW CINEMA LOCATION

Finally to the right of the suggested films we would like a Google Map to display the location of a nearby cinema.

For a quick start guide to adding a Google Maps take a look at:

http://www.mustbebuilt.co.uk/2012/04/21/getting-starting-with-google-maps/

and you'll find code snippets at:

http://www.mustbebuilt.co.uk/SHU/WAD/demo/javascript/index.html

Firstly you'll need an API key. Sign up for one at:

https://developers.google.com/maps/documentation/javascript/tutorial#api_key

We have a `<div>` with an ID of **myMap**.

- Add your map to the **myMap** `<div>`.
- The lat / long of Cineworld at Centertainment is 53.4015201, -1.4149660000000495. Add

this as a `marker` on your map.

- Add an `infoWindow` to the Map to appear when the `marker` for CineWorld is clicked.

USING THE CINEWORLD API

STOP PRESS – DOCUMENTATION RELATING TO THIS API SEEMS TO HAVE BEEN REMOVED ALTHOUGH THE API IS STILL ACCEPTING REQUESTS

CineWorld offers an API for developers so that other sites can use their listing data. You'll find information about the API at:

https://www.cineworld.co.uk/developer/api/films

To register to use the API key visit:

https://www.cineworld.co.uk/developer/register

The documentation gives example API calls that use jQuery.

Take a look at the files *sheffield-listing.php* and *showings.php*.

Given that Sheffield Cineworld has a code of 54 and assuming you have registered to get an API key can you amend the jQuery `$.ajax()` method to retrieve the data.