Session 7 exercises

Advanced JavaScript for Web Sites and Web Applications

Exercise 1

Download the workshop 7 files from Moodle.

Extract them in your workspace and open `test.js` in your editor. You should also open `test.html` in your browser.

Note, there is a copy of jQuery within the *assets* folder, without which, this exercise will not work, so make sure you have extracted it!

A link to the jQuery file has already been added to `test.html` for you.

Open this api URL in your browser:

```
http://gerardluskin.com/city/service/people/?callback=&name=dave
```

What you see displayed is the JSON data your script will receive from the api.

Change the *name* parameter value from "dave" to either "tom", "jean" or "alex" and reload the page. E.g.:

```
http://gerardluskin.com/city/service/people/?callback=&name=tom
```

Your task:

- In `test.js` , make an AJAX call to the api URL for each of the "names" listed above (loop through an array of names if you like)
- In your callback function, console.log the first name, last name and age of the person

You should see that there is a problem with *alex*. Load his URL in your browser again and examine the data that is displayed.

This is something that we regularly face when dealing with Ajax.

The service API is informing you that you have passed an invalid parameter. It has not produced a server/HTTP error, so the *failure* callbacks are not triggered.

In other words, the request was successful, it just didn't return the data we expected!

How might you handle this kind of response in your script?

Exercise 2

Open `workshop7.html` in your browser.

You should also open this URL in your browser:

```
http://gerardluskin.com/city/service/products/?callback=&category=all
```

Change the `category` parameter from *all* to either *phone*, *toy*, *laptop* or *book* and reload the page.

You can also load the URL with an invalid category to see what happens (e.g. category=television)

Your task

You are going to connect the HTML in `workshop7.html` with the data from the product api.

- When a button is clicked, you will make an AJAX request to the api, passing the relevant *category* value as part of the query string (the category is stored in each button's `data-category` attribute).
- You will then display the returned data as HTML within the page

How it will work

This is how the application needs to work:

1. When a user clicks on one of the filter buttons, make an ajax call to the api, passing the button's `data-category` value as the *category* parameter.
2. When the API response is received, process the data, transforming it to HTML
3. Add the HTML to the page

The code

Here is one way to structure the code:

- Use the revealing module pattern with:
  - an init function that adds an event listener for the filter buttons (using delegation!)
  - a function to make the request (this is the event listener for the filter buttons)
  - a function to convert the returned data to HTML
  - a function that adds the HTML to the DOM

Only the init function needs to be public. All other functions will be used internally.

Getting started

In `exercise7.js`, a *revealing module* has been defined for you, which you will add to in this exercise.

The module functions have also been defined for you, but they do nothing currently.

Some variables have been declared, using the *single var* pattern we looked at previously. However, you will need to declare/define a few more variables for the module to be able to carry out its job.

First, you need a reference to the element that holds the buttons (which has the id: `filters`). You can call this variable `filters` (or similar)

You will also need a reference to the element into which you will place the new HTML (which has the id: `products-list`). You can call this variable `productsList` (or similar).

Add these 2 variables to the single var pattern and store the element reference in them (using `getElementById`):

```
var init,
   callAjax,
   jsonToHTML,
   addHTMLToDom,
   filters = document.getElementById('filters'),
   productsList = document.getElementById('products-list');
```

Event handler

The event handler will be added to the element stored in the `filters` variable.

Within the `init()` function, add an event handler that listens for the *click* event on the `filters` element.

The event handler will call the internal function `callAjax`. It does not need to do anything else.

The `callAjax()` function

In the `callAjax` function, the first thing you will need to do is to check that the element that was clicked is actually a filter button.

Assuming it is, you will then use the button's `data-category` attribute in your AJAX call.

Remember, `event.target` will be a reference to the clicked element.

As we saw previously (when covering events), there are several ways you can check that the element is a button.

In this case, it makes sense to test for the `data-category` attribute as we need it to make the Ajax request!

E.g.

```
// Inside callAjax()...
var categoryValue = event.target.getAttribute('data-category');
if (categoryValue != null) {
    // Clicked element was a button
    // The value we need for AJAX is in "categoryValue"
}
// No need for an "else" as we do nothing if element is not a button
```

Once you have this in place, you can fill in the body of the `if` statement.

You will need to:

- Construct the api URL, using the element's `data-category` value
- Remove any existing HTML from the `productsList` element (you can use `innerHTML` for this)
- Make the AJAX call using jQuery's `.ajax()` method.
  - The *url* will be the api url with the appropriate category appended to it.
  - The *dataType* will be *jsonp*.

We will also need to register a *callback* for when the request completes successfully. You can use `.done()` or the `success` property, whichever you prefer.

Successful completion (done/success)

When the request completes successfully, we will want to pass the data to the `jsonToHTML` function.

If you check the api output in your browser, you will notice that the JSON is actually...

- an *object*, with a single property: *products* ...
- ... which is an array of objects...
- ... each of which represents a product

We are only interested in the array of product objects, so we should extract this from the data object and pass it to `jsonToHTML`

E.g.

```
// Inside the done/success callback...
// "ajaxData" is the entire object but we only
// want the array stored under "products"
var productArray = ajaxData.products;
jsonToHTML(productArray);
```

The `jsonToHTML` function

Now you will write the `jsonToHTML` function.

This function receives one argument (an array of objects). You will need to loop through this array (using a `for` loop)

```
var i, currentItem, total = productData.length;

for (var i = 0; i < total; i++) {
    // Copy product object to currentItem (easier to type!)
    currentItem = productData[i];
}
```

For each product object in the array, you need to build a HTML string like this:

```
<li class="phone">
    <a href="url">Item name</a> -
    <span>Price: Item price</span>
</li>
```

Note, if you want the pretty colours to be applied to the displayed products, make sure you use the product's *category* as the list item's *class* attribute.

Remember, you can use standard *dot notation* to access the object properties.

E.g.:

```
// The product category
currentItem.category;
```

You will also need to add a `<ul>` with a class of *products* around all the `<li>` tags.

```
<ul class="products">
    <!-- product LI tags -->
</ul>
```

You can build the HTML any way you like, as long as you end up with a single string, containing the entire HTML list.

Finally, you will call the `addHTMLtoDOM` function, passing it the string containing the HTML list.

The `addHTMLtoDOM` function

Now you can complete the `addHTMLtoDOM` function.

This function receives one argument (the html string from `jsonToHTML`)

It will add it to the div with the id of *products-list*, which you should have a reference to already (`productsList`).

You can use `insertAdjacentHTML` for this.

Test the script

Now you can test the script in your browser and you should see products after you click on the buttons.

Extras 1 - feedback

You will notice that when we click on one of the filters, we receive no feedback telling us what is happening.

The html is being updated eventually, but the user does not know that this is happening behind the scenes.

To rectify this, we can do the following:

- Add a small loading image to indicate that something is happening.
  - You can do this by adding a class of *loading* to the *products-list* div (the CSS is already in place for this to have an effect on the page). You can do this within the `callAjax` function, immediately after you remove existing HTML from the div.
- Just before you add the new html to the page, remove the image
  - by removing the class *loading* from the div

Extras 2 - Error handling

Currently, if the api returns an error, the script will not handle it.

Two typical errors that may occur:

- the category isn't passed correctly to the api (although it shouldn't happen...)
- the ajax calls fails and returns a 404 or time out (network issues)

For the first case, we can check the data object returned in the success method. If it contains an *error* property, the category passed was corrupt/invalid/non-existent

For the second case, we can register a callback method to run if the the AJAX call fails. (e.g. using `.done()`)

Test the error handling

1. Change the api URL to an invalid one, e.g. http://gerardluskin.com/wrong/service/products. This should produce a 404 error, so your error callback should run
2. Add another button to the page, and put a non-existent category in its data-category attribute (e.g. "television"). This should trigger your custom error handling (i.e data.error will have a value)

```html
<p class="filter" data-category="television">
    Television
</p>
```