A Connected Web-App

The previous practical took you through some of the fundamentals of creating web-apps using HTML 5, jQuery and jQuery mobile. In this lab we will consolidate this by developing a coherent, useful application that makes use of an on-line data source.

Working methods

As previously, "test continuously" is the abiding principle you should stick with while working through this practical. However, the code in this application is more extensive and therefore more likely to go wrong (Javascript is very unforgiving of seemingly trivial errors in punctuation etc.), and so as the code is developed, we will need to be careful to test each function as we create it (and certainly before moving on to the next one).

The App

The application will be a currency conversion app. If you took the HTML5 and Javascript module you may recall a lab to build a similar app. However, in this app, we will make use of a JSON-based web service to return currency exchange rates, and will use AJAX as the mechanism to collect the JSON.

For a simple test of the web service, open any web browser and enter the URL:

http://api.fixer.io/latest?base=GBP

The response will be data in pure JSON format, which when converted into a Javascript object will be similar to:

{ base: "GBP", date: "<today's date>", rates: { "AUD": 1.9139, "BGN": 2.4867 … }}

Currently the service returns 32 different exchange rates. The file **fixer-symbols.js** from the Moodle page lists the symbols, the countries they represent and the name of the currencies.

1. Create the basic jQuery Mobile page structure

Our app will be a SPA (Single Page Application), although a little jQuery Mobile mark-up will support a slightly more complex page structure:

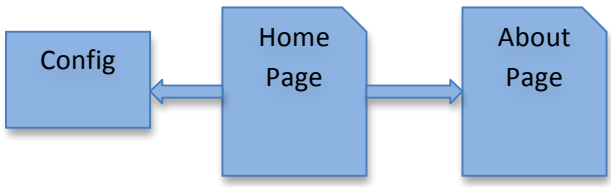

Figure 1: The app structure

To build the HTML file for the app:

a) Using WebStorm, create a new project in a folder on your H: drive or a memory stick. Call this folder Lab 2
b) Add an HTML file to the project, giving it the name lab2.html

c) In the <head> section of the HTML file, place the cursor on a new line (below the <title> tag-pair) and type ***jqm+TAB***. Assuming that WebStorm has been pre-configured with the appropriate settings file, this should expand to the full set of <meta>, <link> and <script> tags required by a jQuery Mobile project. If it does not, you can find the settings file in the Moodle folder for this lab and import it using ***File→Import Settings*** from the WebStorm menus. If you are using a different editor than WebStorm, the jQuery Mobile headings can be found at http://jquerymobile.com/download/ (the Copy+Paste snippet)

d) Place the cursor inside the HTML <body> tag and type ***jqmpage+TAB*** to add the definition of a jQuery Mobile page (a element with included mark-up) to the file. At the top of the page definition, change the page's ID property to "home"

e) At the bottom of the HTML file (just above the </body> closing tag) add a second page definition, and give this the ID "about". Both page definitions are in listing 1

The <body> section of the HTML file should appear similar to that shown below, but note that in addition to adding the ID properties to each page, each page has a header ("X-Currency", and "About X-Currency") and home page's "back" button and the entire <nav> section of the About page footer has been removed.

```
<div id="home" data-role="page">
    <div data-role="header">
        <h1>X-Currency</h1>
    </div>
    <div data-role="content">
    </div>
    <div data-role="footer" data-position="fixed" class="ui-bar">
        <nav>
            <div data-role="navbar">
                <a href="#about">About...</a>
            </div>
        </nav>
    </div>
</div>
<div id="about" data-role="page">
    <div data-role="header">
        <h1>About X-Currency</h1>
        <a href="#" data-rel="back" data-icon="back">Back</a>
    </div>
    <div data-role="content">
    </div>
    <div data-role="footer" data-position="fixed" class="ui-bar">
    </div>
</div>
```

Listing 1: The basic HTML mark-up for the app

You should now test the mark-up by opening the page in a browser (remember that you can click on one of the small browser icons at the top-right of the editor area to open the current page in that browser.

2. Adding Content to the pages

In the home page, we will need a text input field (for the amount of currency to calculate an exchange for) and a select menu so that the user can choose which currency to translate to. We will also need to add somewhere for the result to be displayed.

a) Add content as follows to the home page "content" area:

```
<div data-role="content">
    <label for="amount">Base Currency = <span id="basesymbol"></span>
    </label>
    <input type="number" id="amount"
           placeholder="Amount in base currency"/>
    <label for="currency">Convert to:</label>
    <select name="select_currency" id="currency" data-mini="true"
            data-native-menu="false">
    </select>
    <input type="number" id="selected"
           placeholder="Amount in selected currency"/>
    <br/>
    <a href="#" id="convert" data-role="button"
       data-mini="true">Convert base to selected</a>
    <br/>
    <a href="#" id="convert-back" data-role="button"
       data-mini="true">Convert selected to base</a>
</div>
```

Listing 2: The content of the home page

b) Add some "about" text to the About page. Generally you would expect to add a heading (e.g. Copyright © with a name and date) and a description of the app's purpose. You could also include an email address so that users could contact you to congratulate you on your magnificent work (a mailto: anchor link is the best for this)

c) Test that the pages display properly, and that the About button on the home page takes you to the about page.

3. Getting the Conversion Rates

The format of the conversion rates provided by the Fixer.io service make it an easy matter to attach the data set to the application. Since the service supports the JSONP format, there will be no cross-origin problems in getting the data, and the will be no need to process it in any way. Simple AJAX code to get the rate data is described in the Fixer.IO website – we can upgrade this to make sure that we can minimize the data retrieved from the Internet (which may be important to users who may have to pay a lot for data downloads on a mobile device).

a) Add a new directory to the project and call it "**js**" – this will house the Javascript files

b) Add a new Javascript file to the folder – call it **lab2.js** – and copy the code in listing 3 into this file:

```
var ratesURL = "http://api.fixer.io/latest?base=",
    rateList = {};

function getRates(symbol) {
    // Get the exchange rates from the service.
    $.ajax({
        dataType: 'jsonp',
        url: ratesURL + symbol,
        success: function(data) {
            // Get a list of exchange rates from the service...
            rateList = data;
            // Stash these rates for offline use...
            localStorage.setItem("rateList",
                                 JSON.stringify(data));
            localStorage.setItem("baseRate", rateList.base);
            // and update the U-I...
            $("#baseList").val(rateList.base)
```

```
                            .selectmenu('refresh');
                $("#basesymbol").text(symbol);
            },
            error: function(err) {
                // Retrieve the most recently saved list of
                // exchange rates...
                rateList = JSON.parse(localStorage.getItem(
                                                "rateList", null));
                if(rateList) {
                    alert("Unable to get current rates.\n"+
                          "Using rates from " + rateList.date);
                    $("#basesymbol").text(symbol);
                } else {
                   alert("Unable to get exchange rates.\n" +
                   "Please check network connection.");
                }
            }
        });
}
```

Listing 3: Code for the currency conversion

c) Add a <script> reference to **lab2.js** to the HTML file (this should come after the existing <script> tags

4. Displaying a currency list

The user has 32 different currencies to choose from (or 31, since we shouldn't count the base currency), so these need to be displayed as options for selection. The standard HTML <select> control is ideal for this (and jQuery Mobile styles that nicely for a Mobile phone):

```
function getSymbolList() {
    var options = "";
    for(symbol in symbolList) {
        options += "<option value='" + symbol + "'>" +
                    symbolList[symbol] + "</option>"
     }
    $('select').html(options).selectmenu('refresh');
}
```

Listing 4: Displaying the list of currencies

a) Add the code in listing 4 to the **lab2.js** file

b) The **getSymbolList()** function in listing 4 extracts data from a variable called **symbolList**. This is already defined in a file **fixer-symbols.js** that you can get from the Moodle page for this lab. Download **fixer-symbols.js** and add it to the **js** folder of the project

c) Add a <script> reference to **fixer-symbols.js** to the HTML file – make sure that the <script> tag is placed *before* the **lab2.js** file

All that is needed now is to execute the two functions (**getRates()** and **getSymbolList()**) as the app starts up. The standard way of doing this in a jQuery app is to add a **$(document).ready()** function:

```
$(document).ready(function() {
    getSymbolList();
    getRates("GBP");
});
```

Listing 5: Initializing the app

Note that we pass the symbol for the base currency (GBP = Great British Pound) in the call to the **getRates()** function – this is appended to the URL as the base currency.

You should now test the app to make sure that a list of currencies is displayed on the home page. It is worth examining the **fixer-symbols.js** file to see how it is structured. Basically, the variable **symbolist** refers to an Object definition, where each symbol name (e.g. "USD") is a property, whose value is the associated country and currency (e.g. "United States (Dollar)").

5. Doing Currency Conversions

To make our app do its job, we now need to add three event handlers, to deal with the user's interactions. These are:

- When the user enters a value into the Base Currency <input> box
- When the user enters a value into the Selected Currency <input> box
- When the user changes the currency selection in the <select> menu

The event handlers for these are shown in listing 6:

```
function baseToSelected() {
    var amount = parseFloat($("#amount").val(), 10),
        convertedAmount = 0.0;
    if(isNaN(amount)){
        showError($("#amount"));
    } else {
        $("#amount").val(amount.toFixed(2));
        convertedAmount =
                 amount * rateList.rates[$("#currency").val()];
        $("#selected").val(convertedAmount.toFixed(2));
    }
}
function selectedToBase() {
    var amount = parseFloat($("#selected").val(), 10),
        baseAmount = 0.0;
    if(isNaN(amount)){
        showError($("#selected"));
    } else {
        $("#selected").val(amount.toFixed(2));
        baseAmount =
                amount / rateList.rates[$("#currency").val()];
        $("#amount").val(baseAmount.toFixed(2));
    }
}

function changeSelected() {
    setRecentCurrencySelection($("#currency").val());
    baseToSelected();
}
```

Listing 6: Converting between currencies.

a) Add the code from listing6 to **lab2.js** (place the code *before* the initialization routine – this does not affect how it works, but it is a more logical organisation of the code)

b) Add the following code to the initialization routine ($(document).ready()):

```
$(document).ready(function() {
    getSymbolList();
    getRates("GBP");
    $("#convert").on('click', baseToSelected);
    $("#convert-back").on('click', selectedToBase);
    $("#currency").on('change', baseToSelected);
});
```

Lsting 7: Binding the event handlers to the User-Interface controls

6. Exercises

We have a working application that, if installed as a set of files on a web server, can be navigated to on a connected device (any web-browser connected to the Internet, but in particular, a browser on a SmartPhone device) and used to do currency calculations using up-to-date exchange rates. However it is not fully polished. The following enhancements would be useful:

- Instead of having to press the 'convert' buttons to initiate a conversion, the "blur" event of both <input> controls could be used for the same purpose – this would save the user having to press a button to make a conversion. To make this happen, all that would be needed would be to add the code to connect the appropriate events to the conversion functions
- It would be much more convenient for the user if the selected currency could be made to stick. This can be done by adding code to the changeSelected() routine to store the selected conversion symbol in localStorage, and retrieving and applying that value in the initialization code (note that you can set a <select> control's selection using the .val() method - e.g. **$("#currency").val("EUR");** would make the selection change to the Euro setting
- Some error handling code ought to be added, so that when the user entered some non-numeric value into either <input> control, the error would be highlighted
- (Biggie) At the moment, the base currency is hard-wired to "GBP". It would be very useful to make this user-configurable. A new configuration page could be added to the app (or a Panel off the home page) with a duplicate <select> list of currencies to allow the user to set the base currency.