Practical Session 2: Access Data from a Web Service

The Task

Your task for this lab session is to write a simple app to access data from an existing web service. You should approach building this app in three stages:

1. Use a browser or other pre-built client to access a web service so that you can get used to how you need to compose a query and how the service will send the data back to your app
2. Now build a very simple app that simply sends off the query string and displays the returned data as simply as possible. Yu can build a web app or a native app for this – see the notes on accessing web data below
3. Now change the app's user-interface so that the return data (in JSON format) is displayed in a more suitable format for a mobile app

Resources

The following software tools websites, notes and information sources will be useful to you during this exercise:

- http://www.jquery4u.com/json/jsonp-examples/ : this site provides an explanation of JSONP, and its use with jQuery
- http://auws-to-do.appspot.com/: this is a REST-based app-engine application that provides logged-in users with services to manage an online to-do list. There is no user-interface in the app (yet). We'll cover the details of how it works in subsequent weeks, but for this lab, what you need to know is:
  - To add items to a to-do list, you will need to use an HTTP POST request. There are various ways of doing this, but the most convenient is to add the PostMan plug-in to Google Chrome and use this app. See instructions at the end of this lab-sheet for using PostMan
  - To get items that are in an existing to-do list, a GET request is needed – this can be done with a normal browser, by entering the appropriate URL. For example:
    - http://auws-to-do.appspot.com/users/fred@bedrock.com

    This URL selects a specific user (fred@bedrock.com) and returns the to-do items from his to-do list. If fred@bedrock.com does not exist (he does!) an empty list is returned
  - By default, a request will return JSONP. The default function call is to a function with the name func(). For example, if we use the URL above (for fred@bedrock.com) the response is:

    ```
    func([{"description":"A second to do item from
    Fred","completed":false,"due":"2014-08-
    07T12:00:00.000000Z","priority":1,"item":"Fred's second
    item","user":"fred@bedrock.com","key":5649391675244544},{"description":"tes
    ting post() method to update Fred's to-do
    list","completed":false,"due":"2015-02-
    19T10:30:00.100000Z","priority":2,"item":"New item for
    Fred","user":"fred@bedrock.com","key":5644406560391168}])
    ```

    Examine this carefully and you'll see that it is an array ([ ]) of JSON objects wrapped in a call to func(). Assuming your client app has such a function, the returned list will be sent to it in its first (and only) parameter
  - As well as users, the REST interface provides access to a list of to-do items. Items are accessed by a key value (an integer number). For example, the first item listed above has the key 5649391675244544. To access that item directly, we can use the URL …

http://auws-to-do.appspot.com/items/5649391675244544

This will return the single item in the format:

```
func({"description":"A second to do item from
Fred","completed":false,"due":"2014-08-
07T12:00:00.000000Z","priority":1,"item":"Fred's second
item","user":"fred@bedrock.com","key":5649391675244544})
```

- The http://auws-to-do.appspot.com/items/ URL format supports the HTTP verbs GET (to retrieve a single to-do item from a user's list), PUT (to update fields in a specific item) and DELETE (to mark an existing item as 'completed'
- The http://auws-to-do.appspot.com/users/ URL format supports the verbs GET (to retrieve a user's list), POST (to add a new item for an existing or new user) and DELETE (to remove all items marked as 'completed' from a user's list)
- Rather than use PostMan or some other HTTP utility program, we can use AJAX calls (see below) to retrieve, create, update and delete lists and items. Note that the function call that is returned from the GET verb passes an array of to-do items to the function, so if I had the func() function defined as shown below, the to-do items would be displayed in the browser's log:

```
function func( itemList ) {
    for(var i=0; i<itemList.length; i += 1) {
        for( key in itemList[i] ) {
            console.log(key + " : " + itemList[i][key]);
        }
    }
}
```

- You can define the name of the callback function in GET requests in your client program (to be used instead of func()) by adding a parameter to the URL:

  http://auws-to-do.appspot.com/users/fred@bedrock.com?callback=getItems

  This would return the list of items wrapped in a call to a getItems() function

Accessing Web Data

How you access data from a web service to update a mobile app depends on a) whether you are using a native app or a web-app, and b) the 'domain' of the web service – overall there are three options:

1. A Native application (e.g. iOS, Android etc.)
   For a native application there are no in-built restrictions. You use an internet access object to call on a URL, and use the data returned from it. For example, and Android app would use the HttpURLConnection class with a URL object. The app would need to have permission to access the web, settable in the Android manifest
2. A web-app that exists at the same web domain as the web service
   Again, there is no restriction here – because the URL you access and the URL that serves the mobile app are at the same domain (the bit between http:// and the actual service function being called), there is no cross-origin policy violation
3. A web-app at a different domain
   Here we are limited in what we can do:
   - We can only call on the HTTP GET method
   - We can only call on URLs that return JSONP data, since the data returned would be valid inside a <script>…</script> element on the page (<script> elements are allowed to cross

domains – otherwise, you couldn't use jQuery, jQuery Mobile, Google Maps etc. in your web app.

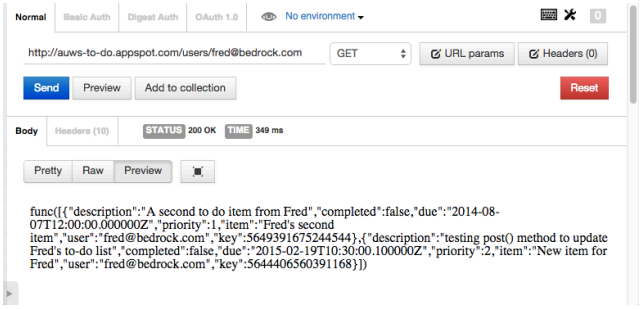
As many of you will probably use a web-app, only options 2 & 3 are available to you. Until you have hosted your own web services, you're stuck with option 3. A small demo app is on Moodle in the lab2 folder in the Week 3 block. This contains the following function definition to make an AJAX call:

```
function getToDoItems(useremail) {
    var URL = 'http://auws-to-do.appspot.com/users/';
    $.ajax({
        type: 'GET',    // This is the HTTP method that will be used.
        url: URL + useremail,   // The email of the list's user.
        async: true,
        jsonpCallback: 'getItems', // This is the callback function we want the response to call.
        contentType: "application/json", // Header for format of data expected
        dataType: 'jsonp',      // Expected return type
        error: function(e) {
            alert(e.message);
        }
    });
}
```

Using Postman

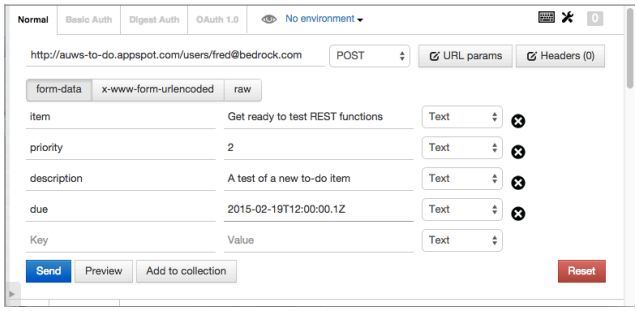
In Google Chrome, go to the Extensions page (Main menu/More Tools/Extensions). If PostMan is not an existing extension, click on Get More Extensions at the bottom of the page and enter Postman in the search box. Select the Postman entry, and install it.

You can now run Postman by clicking on the Apps shortcut in the Bookmarks bar (at the left) and clicking on its name:



This shows Postman open and having just completed a request – the URL http://auws-to-do.appspot.com/users/fred@bedrock.com is shown, the verb (GET) is to the right of it, and the response (after pressing the SEND) button is shown at the bottom.

To send a POST request, select POST from the list of verbs, and enter key and value pairs to define the list of input values – for example:



You can send any form of request from Postman – the ones needed to fully work with the auws-to-do service are: /users/ GET, POST, DELETE and /items/ GET, PUT and DELETE