PROGRAMMING FOR MOBILE DEVICES

Data Feeds and the Cross-Origin Policy

Lecture 6

The Cross-Origin Policy

- The Cross-Origin Policy (or Same-Origin Policy)
  - Prevents a browser displaying a page at http://myDomain.com from accessing data at http://yourDomain.com
  - This is a security restriction
    - Without it, one site could access cookies or other data belonging to another (e.g. to access your online bank identity)
    - A site could use Javascript to access other browser windows
  - The restriction is built into browsers and can not be switched off

A Browser Issue

- The X-Origin Policy does not affect web-sites or native applications
  - It protects browser users
  - Website developers and native app developers can provide their own security
  - Without the X-Origin policy, browsers would only be able to exercise control for security purposes in a gross way
    - e.g. By switching off cookies and Javascript
- The main aim is to protect web users
  - Originally from cookie harvesting
  - Now from just about any scam that could harm their privacy or their computer

Mash-ups?

- A Mash-Up is a ***website*** that incorporates data from more than one internet source
  - e.g. Share Prices, Weather Data, Maps etc.
    - Often incorporated into sites to give them added value
  - Important word in the description – WEBSITE
    - Not affected by cross origin policy, because a website is defined at a ***server***
    - Website designers can access data from any Internet source and include it
      - May not be legal (copyright-wise), but technically it is ok
    - Think of the number of sites you visit that include a Google Map, Weather forecast or other syndicated material

Loopholes

- The most obvious way to get around the X-Origin policy is to create a website
  - Your site accesses web data and incorporates it in some way
    - e.g. Using a Java component – ***URLConnection*** is common
- Next possibility – use a Proxy server
  - Use a 3rd party site to provide the data feed
    - http://ajax.googleapis.com/ajax/services/feed/load does this for free
    - There is a good code library to work with this (jGFeed.js) – makes it easy to use
- Final option – use JSONP
  - How did you incorporate jQuery/jQuery Mobile etc. into your apps?
  - Browsers will allow a <script>…</script> tag to access Javascript from any source
    - Javascript can be used in a sneaky way…

JSON

- JSON – JavaScript Object Notation
  - Javascript objects can be written as name:value pairs
  - A whole data structure can be embedded in a string
    - This means it can be sent over HTTP
- In the code, the script contains one function definition and a function call
  - The function definition displays JSON data as an HTML <ul> list
  - The function call calls it, inserting data into the document
  - That's the principle a Proxy site like jGFeed's uses

```
<body>
   <script id='imported'>
      // Function definition…
      function showInfo(info){
         var i, markup = "<ul>";
         for(i=0; i<info.data.length; i+=1) {
            markup += "<li>"+info.data[i]+"</li>"
         }
         markup += "</ul>";
         return markup;
      };
   </script>
   <script>  // Could specify a src link.
      // Function call…
      showInfo({"data": [
                     {"name": "Fred"},
                     {"name": "Wilma"]
                     {"name": "Barney"}
                      ]});
   </script>
</body>
```

JSONP

- A script from a remote site can execute a function defined locally (browsers allow this)
  - e.g. showInfo() can be defined in <script> tag in your own code
  - A call to it can come from a remote site, embedded as a script tag
  - But – how does the remote site know the name of the function to call?
- JSONP – JSON with Padding gets around that
  - Simply include the name of the function to call in the script link

```
<script src="myscript.js"></script>
<script src="http://yourdomain/yourdatafeed.php?callback=showInfo"></script>
```

myscript.js defines the showInfo() function, which describes how to format the returned data

yourdatafeed.php is an active webpage that returns data encoded as JSON.
It will wrap this data in a call to showInfo()

Providers of JSONP

- For all this to work, you need to access a site that returns JSONP – some do so directly…
  - Twitter, Facebook and various weather, finance, entertainment sites etc. do this
  - e.g.
    - https://dev.twitter.com/docs/api/1/get/statuses/user_timeline
    - http://www.footytube.com/openfooty/service.php?package=League&method=getResults
    - http://www.programmableweb.com/api/met-office-datapoint
    - http://currencyfeed.com/
    - http://code.google.com/p/yahoo-finance-managed/wiki/YahooFinanceAPIs
    - http://www.apple.com/itunes/affiliates/resources/blog/introduction

An Example of JSONP

- mcm-to-do-json.appspot.com
  - Works as a to-do-list web-app
    - (not in a Mobile version)
  - Also provides access to to-do-list items via JSONP
    - Just specify a callback function (e.g. getItems())
- Since the output is JSONP, the items can be manipulated in Javascript
  - e.g. to apply formatting for a Mobile Web App
- All we need to do is execute the request (URL) to collect the data

URL Request:

```
http://mcm-to-do-json.appspot.com/json?
user=AlistairUWS@gmail.com&callback=getItems
```

JSONP Response:

```
getItems([
    {'user': 'AlistairUWS@gmail.com', 'item': 'Test', 'priority': '3',
    'description': 'A test item from a class.','due_date': 'Tuesday
    26/02/2013 13:00', 'completed': 'False'},
    {'user': 'AlistairUWS@gmail.com', 'item': 'Go home and have nervous
    breakdown', 'priority': '3', 'description': 'Long-awaited','due_date':
    'Monday 25/02/2013 17:00', 'completed': 'False'},
    {'user': 'AlistairUWS@gmail.com', 'item': 'Prepare studeets for lab
    assessment', 'priority': '2', 'description': 'Frighten the hell out of
    them','due_date': 'Monday 25/02/2013 14:00', 'completed': 'False'},
    {'user': 'AlistairUWS@gmail.com', 'item': 'Test item', 'priority': '1',
    'description': 'A first test of this to-do account','due_date': 'Monday
    25/02/2013 00:15', 'completed': 'False'}
    ]);
```

Formatting JSON

```
function formatItem( it ){
  var html = "<li>";
  if(it.completed) {
    html += "<input type='checkbox' checked/>";
  } else {
    html += "<input type='checkbox'/>";
  }
  html += "<strong>" + it.item + "</strong><br/>";
  // carry on formatting properties of the 'it' object...
}
function showList(serviceReturn){
  var i, item, strList = "";
  for (i = 0; i < serviceReturn.length; i += 1) {
    item = serviceReturn[i];
    strList += formatItem(item);
  }
  $.mobile.changePage("#listPage", {transition: "flip"});
  $("#list").html(strList).listview('refresh');
}
```

- Using raw Javascript, this could get ugly
- Using jQuery, there are several ways
  - Assume we want a listview of items
    - Markup in jQM is easy
  - Assume each item is to be optimally formatted
    - Check boxes, formatted text etc.

This call is important – it applies the jQM listview style to the new items

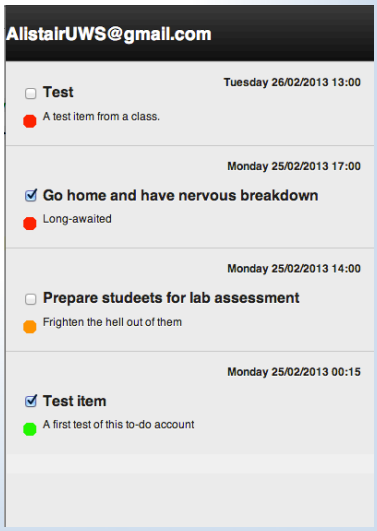
Using Ajax to update a page

- Final link in the chain…
- AJAX (Asynchronous Javascript And XML) can be used to access web data and update the 'live' HTML page
  - This gives a web-app a more 'native' look and feel
- Basic principle:
  - Call a function to make a web request for new data for the page
  - When the data is returned, use this to update the DOM of the page (asynchronous)

```
function getToDoList(listID) {
  $.ajax({
    type: 'GET',   // This is the HTTP method used.
    url: "http://mcm-to-do-json.appspot.com/json?user="
            + listID,  // The email of the list's user.
    async: true,
    jsonpCallback: 'showList', // this will be called
    contentType: "application/json",
    dataType: 'jsonp',
    success: function(json) {
      // No need – showList, the callback is called
      // on success.  We could use this for other purposes
    },
    error: function(e) {
      alert(e.message);
    }
  });
}
```

Result

- Using the Javascript/jQuery code to format the JSON data returned, we can generate a page that is easily as good as we could have drawn on a native app
- Typically, define CSS styles for the elements we want to display on the page
  - Can define table format, images, transitions etc.



So…

- Same-origin policy works to allow us to use data from sites that provide it in the JSONP format
  - In some situations, can access simple JSON data without the padding – this is browser dependent
- We can make our mobile/web apps 'live' – e.g. providing train timetable info., weather data, football scores etc.
  - Anything that provides a JSONP feed
- To access a broader range of sources, we can use a "proxy service"
- http://feed2js.org is such a service
  - Usually RSS feeds (there are loads of these)
    - e.g. look up Apple RSS feed on Google
    - Also look at iTunes RSS generator – a nice tool
  - The feed2js website provides a lot of help to get started

https://itunes.apple.com/gb/rss/topmovies/limit=10/json

The above was generated from the iTunes RSS Generator page (http://itunes.apple.com/rss?cc=GB), and enables data-feed access to much of the data behind the iTunes website and associated pages.
It returns a lot of data in XML format, but also a lot of pre-formatted web-page material – <img> tags etc.

Your project work

- Almost all this year's projects will involve the collection of information provided via external feeds, such as RSS feeds or sites which return JSON
  - One or two projects which don't do this will involve other areas of difficulty – e.g. a lot of graphics coding
- In this week's lab, you'll access data from a site which provides JSONP – so this ought to work in any browser
- Treat this as practice/preparation for your project work