Writing multi-page apps

- In this module, we've looked at using Javascript to build single-page applications
  - One web page = 1 user-interface (usually)
- Application programs often need to define multiple user-interface elements
  - Different views, utility pages (e.g. to select data, print, manage storage etc.), more than one component type
  - These typically need to access the same data-set – e.g.
    - Appointment form for adding, editing
    - Appointment list to select (for edit or delete) an appointment
  - Ideally, we'd use separate user-interfaces for each since users will find that less confusing, it looks neater, operations are compartmentalized etc.

19/11/15

Javascript Applications: transferring data between pages

- All of the code we've done in labs so far has been on a single HTML page
  - Could have many Javascript files, but moving from page to page causes a synch problem
    - A page-load resets script elements
      - Variables are cleared, scripts are removed (possibly to be re-imported by the next page)
      - We can't pass data in variables on one page into another page
  - This gives us a problem for many types of application
    - Different pages could be used to do different things – data input, data view, configuration
    - As a workaround, we can use localStorage, or…
  - We can put data into sessionStorage (similar to localStorage) by one page and retrieve it from another. i.e.
    - On first page, execute…

```
sessionStorage.transferData = "Some data to transfer to another page.";
```

    - On second page, something like…

```
document.write(sessionStorage.transferData);
```

- However, if we had a web server to work with..
  - We could use form data to pass information between pages (a POST operation)
  - We could pass data to a second page in the URL (a GET operation)
  - ***These mechanisms are not available from client side script.***

19/11/15

Passing data between pages in a website/domain

- Web servers have access to a wide range of ways of moving application data from page to page
  - Embed data in the URL – GET/QueryString (e.g. Google)
  - Pass data in HTTP request body or headers – POST/form-data (e.g. any page with a form)
  - Use public property values
  - Use Cookies
- These have different characteristics that make them more or less useful in a given situation
  - QueryStrings are fine for small amounts of data – a user ID or a word or two (hence their use in Google search)
  - POSTdata is perfect for transferring data from a web form – no coding required at the client end
  - Public properties are useful but only if the server supports them (e.g. IIS)
  - Cookies can persist beyond a session – they are stored on the client can be retrieved by any page delivered form the same host
- However, none of this is directly available to the browser

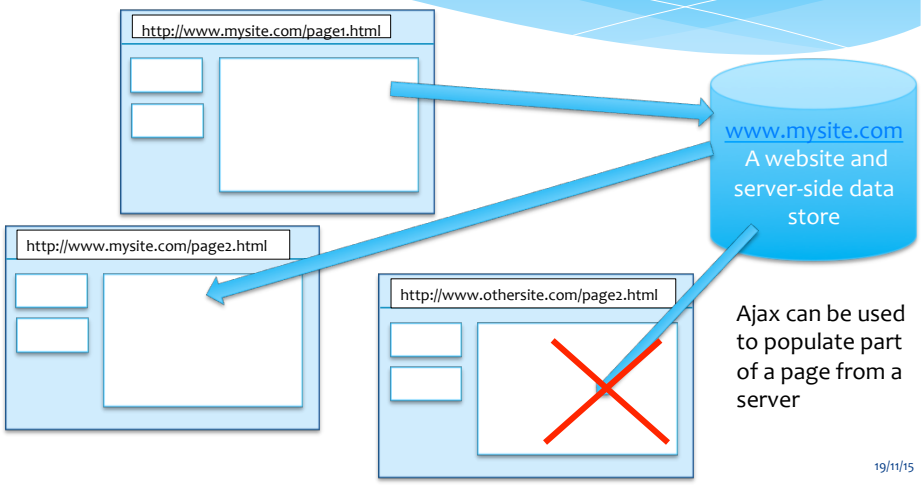
19/11/15

Why are browsers so limited?

- All browsers implement a Same Origin Policy (SOP)
  - A document loaded from one domain can not get or set properties of a document loaded from another: e.g.
    - http://myDomain.com/myPage.html can interact with any other page at http://myDomain.com/
    - It can not interact with a page at http://yourDomain.com/
  - Same origin refers to same domain (myDomain) AND same protocol (http://) AND same port (typically 80)
- This is a fundamental security aspect of the web
- We've met this in the labs – in Lab5 you had to use the WebStorm server or view your app in Firefox instead of Chrome to see the effects of geo-location
  - Chrome takes a stricter view of the "origin" of a page
    - It doesn't accept the file:// protocol as a web domain (rightly)
  - Firefox is more sensible about this instance (geo-location), but still upholds the single origin policy strictly in other respects

19/11/15

Same Origin Policy



Ajax can be used to populate part of a page from a server

19/11/15

Javascript is special

- Javascript can be loaded from ANY domain, port, protocol
  - SOP does not affect Javascript
  - This works on a page in http://myDomain.com:
    <script src="http://yourDomain.com/somecode.js"></script>
    <script> /* Can call a function in somecode.js here */ </script>
- This lets us use online libraries like Google-maps & jQuery
- You may wonder why Javascript is not affected by SOP when everything else is
  - It is a known vulnerability – so care is needed
  - NEVER include a script from a source you don't know and trust – we can probably trust Google, Yahoo, Amazon etc.

19/11/15

Javascript & Browsers

- It's pretty clear that Javascript is designed to be a client-side language, intended to run inside web browsers
  - Browsers contain not only the language, but all of the libraries that the language uses (DOM, Navigator, Screen etc.)
- Javascript is also widely used, so many developers know it to some extent
  - Even developers who would never dream of "programming" in Javascript, use it for simple web operations
    - Validating data entered to web forms
    - Adding some 'fizz' to a web page
- This makes Javascript an ideal scripting language for a wider range of operations – desktop apps, shell scripts etc.

19/11/15

Can we use Javascript as a desktop development language?

- All operating systems contain some scripting options
  - Windows has .bat (batch) and .cmd (command) files
  - OS X and Linux have shell scripts (same thing really)
- These are used to automate simple operating system tasks
  - creating sets of folders
  - batch processing simple jobs like print runs
- They generally have access to operating system facilities – files, devices, user settings etc.
  - Javascript in a browser can not access these areas
- They are also usually very primitive, based on a 1960's programming model
  - No data structures, limited control options etc.
- Javascript is a more capable language with better facilities

19/11/15

Node.js

- Created by Ryan Dahl, of Joyent, SF
  - See his Node.js presentation at http://www.youtube.com/watch?v=jo_B4LTHi3I
- Node.js
  - Developed as a ***server-side*** scripting system…
    - …meaning, it was designed to respond to web requests from browsers (or web-enabled applications)
    - To do its job, it must be able to access operating system entities – files, network nodes etc.
  - This makes it an ideal host for executing Javascript code that runs on the server, but can use a browser as a user-interface
    - A desktop machine or laptop can be the server
    - This is a "best of both worlds scenario"
      - Use operating system resources directly
      - Use the browser code environment for user-interactions

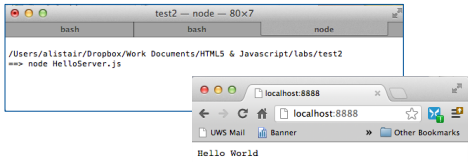
19/11/15

Building Node.js projects

- Node.js runs as a program on a computer system
  - It exists in versions for Windows, Linux, Mac OS X
  - It is built to interact with the network system built into operating systems
  - It contains loads of code modules for doing useful stuff
- It is almost trivial to create a Node web server
  - Create an event handler to respond to an incoming request
    - Respond by writing out an html header followed by document content

```
var http = require("http");

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8888);
```



19/11/15

A 'proper' Web Server

- Node was not built to be a full-featured web-server
  - Apache, IIS, Tomcat etc. do that job perfectly well
- Node is a 'lightweight' server
  - It is best employed sending simple responses to simple requests
    - Typically Ajax requests, which can be used to update web pages in browsers without doing a page-refresh
- This is ideal for "single-purpose" web components
  - Returning data to clients
  - Aggregating data from many clients
  - Providing remote access to sensors, cameras etc.

19/11/15

Writing to a file in Node.js

- Node uses require() to import library code
- Typically start by creating a server
  - This has two parameters – the request object and the response object
    - You read the request and write to the response
  - createServer() needs to say which Port to listen to
    - A port is just a number – effectively a sub-address
    - Web is port 80, other ports have specific purposes, other have set services (e.g. 21 is FTP)
    - Anything from 8080 up is free for custom use
    - Need to be sure no to clash with other ports that might be used
- See http://en.wikipedia.org/wiki/Port_(computer_networking)

```
var http = require("http");

var fs = require('fs');

http.createServer(function(request, response) {
   var stream = fs.createWriteStream("my_file.txt");
   stream.once('open', function(fd) {
      stream.write("My first row\n");
      stream.write("My second row\n");
   });
   response.writeHead(200, {"Content-Type": "text/html"});
   response.write("ok");
   response.end();
}).listen(8890);
```

19/11/15

A Web Server in Node.js

- Node.js is well supported with library code files
- 'express' is a full web server that is easy to use
- Using this, you can create server apps that use a browser for the UI
  - Written of course in HTML5/JS

```
var express = require('express');
var app = express();
app.configure(function () {
   app.use(
      "/",  // the end of the URL of your app
      express.static(__dirname) //your app's file locations
   );
});
app.listen(3000); //the port you want to use
```

19/11/15

So – you can use a server to remove restrictions on your app

- It rationalizes the single origin restriction for all browsers
  - You can transfer data between pages using GET (QueryString) or POST (form data)
  - Chrome does not impose the geo-location restriction
  - You can write server-side code in Node (Javascript) to access databases like MySQL (see http://nodejsdb.org/)
  - Server-side code can read and write files in the local file-system and access local machine features
  - Can use iframes to display content from separate html files (no SOP problems)
  - Can use Ajax to update a page dynamically

19/11/15

An alternative – spoof a multi-page app in a single html page

- jQuery Mobile uses CSS and a little Javascript/Ajax to make a single HTML file contain several "pages"
  - Each page is a element that can be shown (style="display:block) or hidden (style="display:none")
  - AJAX code is used to select the page to be displayed
  - jQM can use CSS animations to make more impressive page transitions
- We can use this same principle is an app without jQM
  - Create 'page styles' to define the show/hide properties
  - Use a little JS code to make the changes
  - See Multi-Page-App.zip on Moodle for code that does this
    - A combination of HTML markup using s, a CSS file and a small amount of Javascript
  - No page refresh, no variables reset

19/11/15