Session 7 - AJAX and JSON

Today we will cover the following topics:

- JavaScript and DOM - revision
- AJAX - Revision
- Using JSON in AJAX applications

JavaScript Revision

If you would like some general revision on JavaScript please see the notes here.

JavaScript and the Document Object Model (DOM)

The ***Document Object Model (DOM)*** is an addressing system for web pages or XML documents. The DOM represents the elements of a web page or XML document as a series of hierarchically-nested ***objects*** (see above), in which objects representing child tags are contained within objects representing the parent. Using the DOM, we can access and manipulate properties of a web page using JavaScript. This allows us to dynamically update the state of the page, for example change the colour of a ***div*** or the text within it. Each page element is treated as a JavaScript object. Each object then has a series of attributes which we can manipulate.

Probably the most important feature of the DOM is the getElementById() function of the ***document*** object (an object representing the entire web page). This function takes the ID of a page element, and gives us back a JavaScript object representing that page element. For example:

```
<html>
<head>
<title>Basic DOM example</title>

<style type='text/css'>
#div1 { background-color: black; color: yellow; width:400px; height:200px;
</style>

<script type='text/javascript'>

function changecolour()
{
    document.getElementById('div1').style.backgroundColor='red';
    document.getElementById('div1').style.color='white';
}

</script>
</head>
<body>

<h1>Basic DOM example</h1>

<input type='button' value='Go!' onclick='changecolour()'/>
<div id='div1'> This is the test div </div>

</body>
</html>
```

This page contains a ***div*** with an ID of div1. The JavaScript function ***changecolour()*** uses ***document.getElementById()*** to get hold of that element, and then changes the backgroundColor (background colour) and color (text colour) properties of its style to red and white respectively.

Changing the contents of a page element

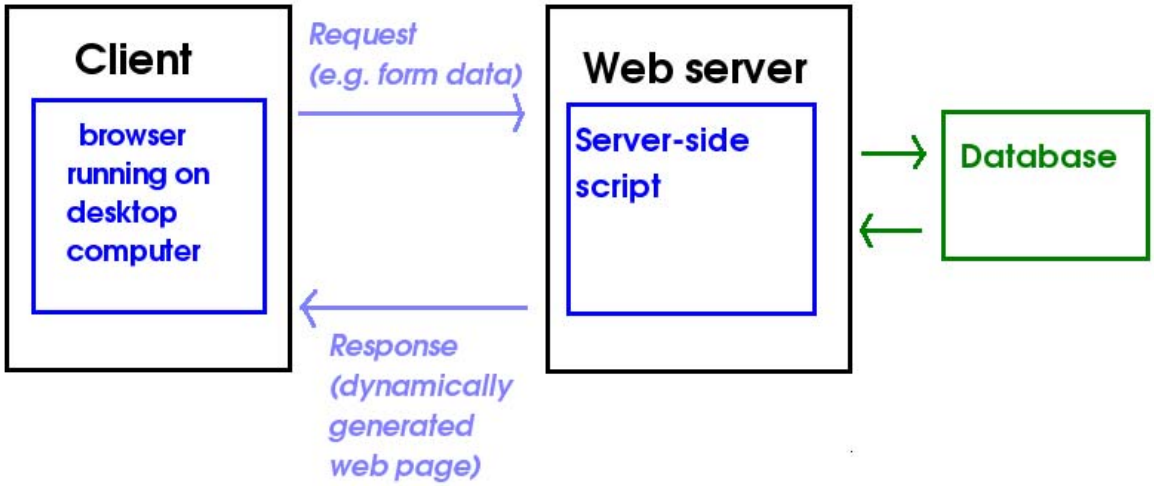
Another feature (not an official DOM feature, but an unofficial extension supported by most browsers) is the ability to change the contents of a page element using ***innerHTML***. This is quite simple to use, for example:

```
document.getElementById("mydiv").innerHTML = "Some new text!";
```

AJAX

We have covered AJAX before in DFTI, but some of you might need revision so here is a refresher.

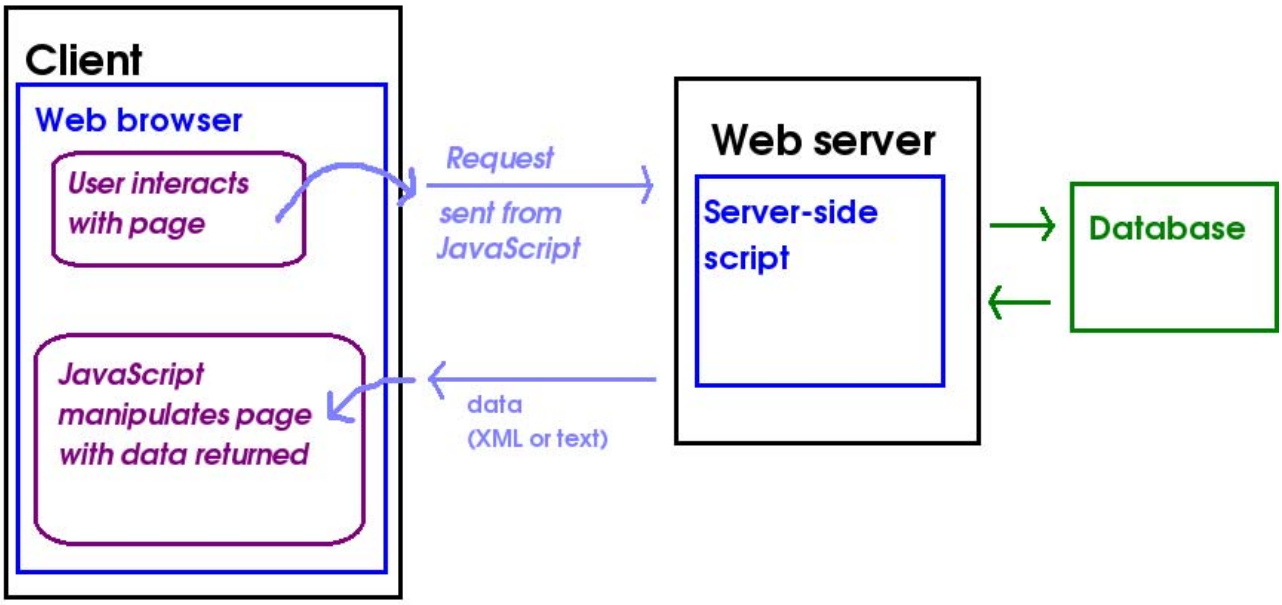
The problem with "classic" web applications



Classic web applications give no ***instant*** feedback to the user: when the web application's state changes (e.g. when the user enters something in a form), the page is reloaded in the browser.

This is because when a form is submitted, the server sends back a ***whole new HTML page*** as a response. So, web applications feel less responsive than desktop applications, because the user has to wait for the page to be sent back. They cannot interact with the page in the meantime.

A different approach - AJAX



AJAX stands for ***Asynchronous JavaScript and XML***. It is a technique which allows browsers to communicate with a web server ***without having to reload the page***.

A user might enter a search term in a form and click Go, and then the search results would be sent back from the server and ***one small part of the page only*** (as opposed to the entire page) updated with the search results.

Furthermore, requests to the server are sent, and responses received, ***in the background without any interruption to the user's interaction with the website...*** so, unlike in a traditional web application where the user has to wait for a response from the server, the user can ***continue to interact with the page*** while waiting for the response to come back.

AJAX is not an actual language, but a combination of technologies used to produce the effect above. An AJAX application typically consists of three components:

- the client side application, usually JavaScript running in the browser which sends HTTP requests to a server and processes the response;
- a server side script which receives the HTTP request sent from the JavaScript, above, and delivers the response;
- the data sent back from the server to the client, which can be either plain text or a data description format such as XML (which we will cover later)

How AJAX Works - Part 1 - Sending the Request

- User interacts with the page in some way e.g. clicks a button
- JavaScript runs in response to this event, and sends a request to a server in the background
- Server-side script processes the request, and sends back data

How AJAX Works - Part 2 - Receiving the Response

- JavaScript on the client receives the response
- The response is used to change the state of the page in some way: for example, the server could send back some text, and JavaScript on the client could replace the text in a paragraph with the text returned
- In this way, pages dynamically update with data from a server without having to reload

Typical structure of JavaScript code in an AJAX application

An AJAX application typically contains two JavaScript functions in addition to a server-side component:

1. The first function gathers the information from the user and sends the request to a server-side script (e.g. a PHP script);

2. The second - the ***callback*** - runs when the response is received from the server, and updates the page with the response.

The server-side script involved is no different to an ordinary server-side script: it simply receives the requests, acts upon it (e.g. looks up data in a database) and delivers a response. For example, if it's PHP, it uses $_GET or $_POST (depending on whether we're sending a GET request or a POST request) to read the data, then queries a database and sends back the results.

Developing an AJAX application

Here is an example of an AJAX application offered by Solent Airways, to allow users to search for flights to a particular destination on a given date.

To write applications, ***AJAX libraries*** formerly had to be used, as there were differences in AJAX implementation between Internet Explorer and other browsers. However, nowadays a more consistent native approach is available via ***XMLHttpRequest2*** which is an improved version of the original browser AJAX implementation. This has worked on Firefox, Chrome, Opera and Safari for several versions as well as version 10+ of Internet Explorer (ref: caniuse.com).

AJAX example

```
<html><head>
<title>AJAX Example</title>


<script type='text/javascript'>

function ajaxrequest()
{
    var xhr2 = new XMLHttpRequest();

    xhr2.addEventListener ("load", responseReceived);

    var a = document.getElementById("destination").value;
    var b = document.getElementById("date").value;

    xhr2.open("GET" , "/wad/flights.php?destination=" + a + "&date=" + b);
    xhr2.send();
}

// The callback function
function responseReceived(e)
{

    var output = ""; // initialise output to blank text
    var data = JSON.parse(e.target.responseText);
    for(var i=0; i<data.length; i++)
    {
        // add the details of the current flight to the "output" variable
        output = output + 'Flight number: ' + data[i].flightnumber + ' Dep
    }
    // Put the HTML output into the results div (up to you to do!)
}
</script>

</head>


<body>

<h1>Solent Airways!</h1>
```

```
<h2>Get the cheapest flights ever!!!!!!!!!!</h2>
<p>Whether it's New York, LA, Montreal or Denver, you can be sure to find
the best deals on the web right here!!!!!!!!!!!!!!!!!!!!</p>
<p>
Destination: <input id="destination" /> <br/>
Date: <input id="date" /> <br/>

<input type="button" value="Go!" onclick="ajaxrequest()" />
</p>

<div id="response"></div>

</body></html>
```

You can see an example similar to this running here. How does this work?

- The ajaxrequest() function first retrieves the destination and date that the user entered and stores them in the variables ***a*** and ***b*** respectively.
- We specify what function to run when the response has been received (the ***callback function***) with this line:

  ```
  xhr2.addEventListener ("load", responseReceived);
  ```

  This means that when the response has been received (the ***load*** event has occurred), the ***responseReceived()*** function will run.
- We then ***open the connection to the server*** with this line:

  ```
  xhr2.open("GET", "/wad/flights.php?destination=" + a + "&date=" + b);
  ```

  Note how we are dynamically creating the query string using the form data, and specifying a ***GET*** request. The URL ***omits the server name*** (e.g. edward2.solent.ac.uk) because with AJAX applications, the back-end usually comes from the same server that delivered the front-end (see the same origin policy, below)
- Finally we actually ***send*** the request:

  ```
  xhr2.send();
  ```

- The callback function, ***responseReceived()***, is discussed below.

Using JSON in an AJAX application

We will now discuss what the callback function, ***responseReceived()***, does, but before we do so we need to discuss how JSON parsing works in JavaScript.

JSON is most commonly used in conjunction with AJAX. An AJAX application can make a request to a server side script which generates JSON from the contents of a database, and then interpret the JSON returned. We can then convert the JSON into JavaScript ***arrays*** and ***objects***. For example, imagine that this JSON code is returned from the server in the above example if the user searched for flights from Gatwick to Rome.

```
[
    {
        "origin" : "Gatwick",
        "destination": "Rome",
        "flightnumber" : 1,
        "depart"  : "0900",
        "arrive" : "1200"
    } ,

    {
        "origin" : "Gatwick",
        "destination": "Rome",
        "flightnumber" : 3,
```

```
        "depart"  : "1100",
        "arrive" : "1400"
    } ,

    {
        "origin" : "Gatwick",
        "destination": "Rome",
        "flightnumber" : 5,
        "depart"  : "1400",
        "arrive" : "1700"
    }
]
```

This JSON code, an array of three objects, can be converted into the JavaScript equivalent, a ***JavaScript array of three JavaScript objects***, using the ***JSON.parse()*** function. A ***JavaScript object*** is similar to a PHP associative array - we will look at objects in more detail later in the unit.

We use the ***JSON.parse()*** function to convert JSON into the corresponding JavaScript arrays and objects. So the code below in an AJAX callback will parse the JSON above, loop through and add each line in turn to the variable ***output***:

```
function responseReceived(e)
{
    var output = ""; // initialise output to blank text
    var data = JSON.parse(e.target.responseText);
    for(var i=0; i<data.length; i++)
    {
        // add the details of the current flight to the "output" variable
        output = output + 'Flight number: ' + data[i].flightnumber + ' Depa
    }
    // Put the HTML into the results div (up to you to do!)
}
```

In this example, the JSON code is loaded into the variable ***data***. The variable ***data*** will contain whatever data structure is described in the JSON. So if we load the JSON in the above example into the variable ***data***, the variable ***data*** will contain an array of 3 flight objects. We can access each object just like ordinary JavaScript objects, e.g.

```
data[0].depart
```

represents the departure time of the first flight (0900) whereas

```
data[1].depart
```

represents the departure time of the second flight (1100).

The same-origin policy

AJAX applications are normally subject to the ***same-origin policy***. This means that the back-end (the server-side script that the JavaScript talks to) must be delivered from the same exact domain as the front-end. The reason for this is security: the ability for an AJAX front end to talk to a third-party server opens up the possibility of cross-site scripting. According to the W3C on their same origin policy document, with the same-origin policy "the overarching intent is to let users visit untrusted web sites without those web sites interfering with the user's session with honest web sites" - in other words, it prevents the possibility of a third-party AJAX-based site using AJAX to communicate with a legitimate site which the user might be currently logged on to, which could lead to stealing of session-specific information.

The domain must be exact; subdomains of a top-level domain are treated as different. For example ***booking.solentholidays.com*** and ***hotels.solentholidays.com*** are treated as different. This is because several users with different subdomains might be sharing the same top-level domain of a hosting company, e.g. ***fredsmith.solenthosting.com*** and ***timjones.solenthosting.com***.

Circumventing the same-origin policy with CORS

There is, however, a way in which server-side developers can circumvent the same origin policy in certain cases. This is done by explicitly allowing, on the server side, certain AJAX clients to connect. A common case is where one person owns two domains, and would like the two domains to communicate with each other over AJAX. For example, Solent Holidays might have two domains, ***hotels.solentholidays.com*** and ***booking.solentholidays.com*** - and wishes to be able to send users' booking details from ***booking.solentholidays.com*** to a hotels web service on ***hotels.solentholidays.com***. To do this, they would have to add code to the server-side scripts on ***hotels.solentholidays.com*** to ***allow booking.solentholidays.com to connect***.

This is done by using the technique of ***Cross-Origin Resource Sharing (CORS)***. An ***Access-Control-Allow-Origin*** line is added to the HTTP response from the PHP script using the ***header()*** function, e.g. PHP scripts on ***hotels.solentholidays.com*** could include the following:

```
<?php
header("Access-Control-Allow-Origin: booking.solentholidays.com");
... rest of script ...
```

Arrow functions

Arrow functions are a new feature of the recent JavaScript standard, ECMAScript 6. They are anonymous, nameless functions which can be used as parameters where a callback is expected. Rather than specifying the name of the callback, we write the function in full within the block of code where the callback is expected. Here is the AJAX example above rewritten so that the callback is an arrow function. Rather than specifying a separate function ***responseReceived()*** to handle the AJAX response, note how we specify an anonymous arrow function as the 2nd parameter to ***addEventListener()***. Note also how we do not use the word "function" with arrow functions, but separate the parameter list and function body with the arrow operator (**=>**):

```
function ajaxrequest()
{
    var xhr2 = new XMLHttpRequest();

    xhr2.addEventListener ("load",  (e) =>
                {
                    var output = "";
                    var data = JSON.parse(e.target.responseText);
                       for(var i=0; i<data.length; i++)
                    {
                        output = output + 'Flight number: ' + data[i].flig
                    }
                } );

    var a = document.getElementById("destination").value;
    var b = document.getElementById("date").value;

    xhr2.open("GET" , "/wad/flights.php?destination=" + a + "&date=" + b);
    xhr2.send();
}
```

Arrow functions have other features, such as preserving "this" scope, but we have not covered these topics yet, so we will return to this later.

AJAX and JSON Parsing - Exercise

The main exercise is to write a website for HitTastic! including an AJAX front end which ***parses*** (interprets) the JSON that your web service from the first week produced. The Standard Questions are the most important to complete.

Standard Questions

1. Ensure that your HitTastic! page (which you should have done as homework) includes a form field to allow the user to enter an artist, and a button, as in the flights example, above.
2. Write an AJAX request function which connects to your web service from session 2.
3. Write an AJAX callback function to ***parse*** the JSON returned. Place each matching song on a separate line within a .
4. Rewrite your callback as an arrow function.

More advanced questions

1. Display the results in a table.
2. Move on to the next topic. If you begin this now, you will have more chance of completing all the advanced questions in that topic.