Javascript, JQuery, and AJAX

CITS3403 Agile Web Develpopment

Regular Expressions

- Regular expressions are used to specify patterns in strings
- JavaScript provides two methods to use regular expressions in pattern matching
  - String methods
  - RegExp objects (not covered in the text)
- A regular expression pattern is indicated by enclosing the pattern in slashes
- The `search` method returns the position of a match, if found, or -1 if no match was found
- The `replace` method can replace a pattern with a different string.

Example Using search

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position > 0)
   console.log("'bits' appears in position",
          position, "<br />");
else
   console.log(
     "'bits' does not appear in str <br />");
```

- This uses a pattern that matches the string 'bits'
- /bits/ is a javascript object the describes a regular expression
- The output of this code is as follows:
  - 'bits' appears in position 3

Characters and Character-Classes

- *Metacharacters* have special meaning in regular expressions
  - \ | ( ) [ ] { } ^ $ * + ? .
    - May be used literally by escaping them with \
- Other characters represent themselves
- A period matches any single character
  - `/f.r/` matches `for` and `far` and `fir` but not `fr`

- A character class matches one of a specified set of characters
  - [**character set**]
  - List characters individually: [abcdef]
  - Give a range of characters: [a-z]
  - Beware of [A-z]
  - ^ at the beginning negates the class

Predefined character classes

Name	Equivalent Pattern	Matches
\d	[0-9]	A digit
\D	[^0-9]	Not a digit
\w	[A-Za-z_0-9]	A word character (alphanumeric)
\W	[^A-Za-z_0-9]	Not a word character
\s	[\r\t\n\f]	A whitespace character
\S	[^ \r\t\n\f]	Not a whitespace character

Repeated Matches

- A pattern can be repeated a fixed number of times by following it with a pair of curly braces enclosing a count
- A pattern can be repeated by following it with one of the following special characters
  - * indicates zero or more repetitions of the previous pattern
  - \+ indicates one or more of the previous pattern
  - ? indicates zero or one of the previous pattern
- Examples
  - /\(\d{3}\) *\d{4} *\d{4}/ might represent a telephone number
  - /[$_a-zA-Z][$_a-zA-Z0-9]*/ matches identifiers

Anchors

- Match positions rather than characters
  - Anchors are 0 width and may not take multiplicity modifiers
- Anchoring to the ends of a string
  - `^` at the beginning of a pattern matches the beginning of a string
  - `$` at the end of a pattern matches the end of a string
    - The `$` in `/a$b/` matches a `$` character
- Anchoring at a word boundary
  - `\b` matches the position between a word character and a non-word character or the beginning or the end of a string
  - `/\bthe\b/` will match 'the' but not 'theatre' and will also match 'the' in the string 'one of the best'

Pattern Modifiers

- Pattern modifiers are specified by characters that follow the closing / of a pattern
- Modifiers modify the way a pattern is interpreted or used
- The `x` modifier causes whitespace in the pattern to be ignored
  - This allows better formatting of the pattern
  - `\s` still retains its meaning
  - (but only in perl…. I'm not sure why this is here?)
- The javascript modifers are
  - `i`, match case insensitive
  - `g`, global match, find all matches
  - `m`, multiline macthing.
- eg

```
var str = "Rabbits are furry";
var position = str.search(/rab*/i);
```

Other Pattern Matching Methods

- The `replace` method takes a pattern parameter and a string parameter
  - replaces a match of the pattern in the target string with the second parameter
  - A `g` modifier on the pattern causes multiple replacements

```
var str = "low, lower, lowest";
str.replace(/low/g, "high");
```

- Parentheses can be used in patterns to mark sub-patterns
  - The pattern matching machinery will remember the parts of a matched string that correspond to sub-patterns
- The `match` method takes one pattern parameter
  - With a g modifier, the return is an array of all matches
  - Without a g modifier, the return is an array of the match and parameterized (parenthesised) sub-matches

```
var str="CITS3403 is the best in 2019";
var matches = str.match(/\d/g);      // [3,4,0,3,2,0,1,7]
matches = str.match(/(\d+)([^\d+])(\d+)/);
              // ["3403 is the best in 2019"]
```

jQuery

- jQuery is the most popular javascript library in the world.
- It was built by John Resig in 2006, and is available on the MIT license.
- The jQuery library has features for
  - HTML/DOM manipulation
  - CSS manipulation
  - HTML event handling
  - Effects and animations
  - AJAX message handling
  - Some utilities
  -

```
<script>
$(document).ready(function(){
  $("#btn1").click(function(){
    $("#test1").text("Hello world!");
  });
  $("#btn2").click(function(){
    $("#test2").html("<b>Hello world!</b>");
  });
  $("#btn3").click(function(){
    $("#test3").val("Dolly Duck");
  });
});
```

- jQuery, like Bootstrap, is most commonly accessed through a CDN:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
```

jQuery Syntax

- Basic jQuery syntax is *$(selector).action*(), where:
  - $ is an abbreviation for jQuery. It can be changed using $.noConflict()
  - *selector* is a query to find HTML elements, much like in CSS
  - *action* is a jQuery function to be applied to the selected elements.

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $(".test").hide();
  });
});
</script>
</head>
<body>

<h2 class="test">This is a heading</h2>

<p class="test">This is a paragraph.</p>
<p>This is another paragraph.</p>

<button>Click me</button>
```

This is a heading

This is a paragraph.

This is another paragraph.

Click me

Syntax	Description
$("*")	Selects all elements
$(this)	Selects the current HTML element
$("p.intro")	Selects all <p> elements with class="intro"
$("p:first")	Selects the first <p> element
$("ul li:first")	Selects the first element of the first
$("ul li:first-child")	Selects the first element of every
$("[href]")	Selects all elements with an href attribute
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"
$(":button")	Selects all <button> elements and <input> elements of type="button"
$("tr:even")	Selects all even <tr> elements
$("tr:odd")	Selects all odd <tr> elements

jQuery Events

- jQuery event notation assigns a javascript function to DOM events.

- For example to assign a click event to all paragraphs, we would use:

    `$("p").click(function(){//code});`

- jQuery events also include `$(document).ready()` to force jQuery to wait until the document is fully loaded.

- The on() method allows multiple events to be assigned to a given selector.

```
$("p").on({
  mouseenter: function(){
    $(this).css("background-color", "lightgray");
  },
  mouseleave: function(){
    $(this).css("background-color", "lightblue");
  },
  click: function(){
    $(this).css("background-color", "yellow");
  }
});
```

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

jQuery Effects

- jQuery implements some useful effects, including hide, show, fade, slide and animate.

```
<script>
$(document).ready(function(){
  $("#flip").click(function(){
    $("#panel").slideDown("slow");
  });
});
</script>
<style>
#panel, #flip {
  padding: 5px;
  text-align: center;
  background-color: #e5eecc;
  border: solid 1px #c3c3c3;
}

#panel {
  padding: 50px;
  display: none;
}
</style>
</head>
<body>

<div id="flip">Click to slide down panel</div>
<div id="panel">Hello world!</div>
```

Click to slide down panel

Click to slide down panel
Hello world!

These methods work by changing the display attribute of an element.

jQuery DOM

- jQuery can select elements and classes in the DOM, traverse the DOM, get and set elements and attributes of the DOM, and add or remove elements.
- You can get the text, raw html, or value of a field with the methods `text()`, `html()` and `val()`.
- You can access the attributes of an element with the method `attr(label)`
- You can modify elements with `text(newText), html(newHTML)` and `val(newVal)`
- You can modify the attributes of an element with the method `attr(label,new value)`
- You can append, prepend, insert before and after with the functions `append, prepend before` and `after`.

```
function afterText() {
  var txt1 = "<b>I </b>";                      // Create element with HTML
  var txt2 = $("<i></i>").text("love ");       // Create with jQuery
  var txt3 = document.createElement("b");      // Create with DOM
  txt3.innerHTML = "jQuery!";
  $("img").after(txt1, txt2, txt3);            // Insert new elements after <img>
}
```

jQuery Filters

- Using selectors, ids, names, and the DOM structure you can selector a number of elements.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1
/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("#first").parent().find("span").css({"color": "red",
"border": "2px solid red"});
});
</script>
</head>
<body>

<div class="descendants" style="width:500px;">div (current
element)
  <p id="first">p (child)
    <span>span (grandchild)</span>
  </p>
  <p>p (child)
    <span>span (grandchild)</span>
  </p>
</div>
```

p (child)

> span (grandchild)

p (child)

> span (grandchild)

- jQuery can also filter results, tables, lists and paragraphs:

```
<script>
$(document).ready(function(){
  $("#myInput").on("keyup", function() {
    var value = $(this).val().toLowerCase();
    $("#myList li").filter(function() {
      $(this).toggle($(this).text().toLowerCase().indexOf(value) > -1)
    });
  });
});
</script>
</head>
<body>

<h2>Filterable List</h2>
<p>Type something in the input field to search the list for specific
items:</p>
<input id="myInput" type="text" placeholder="Search..">
<br>
```
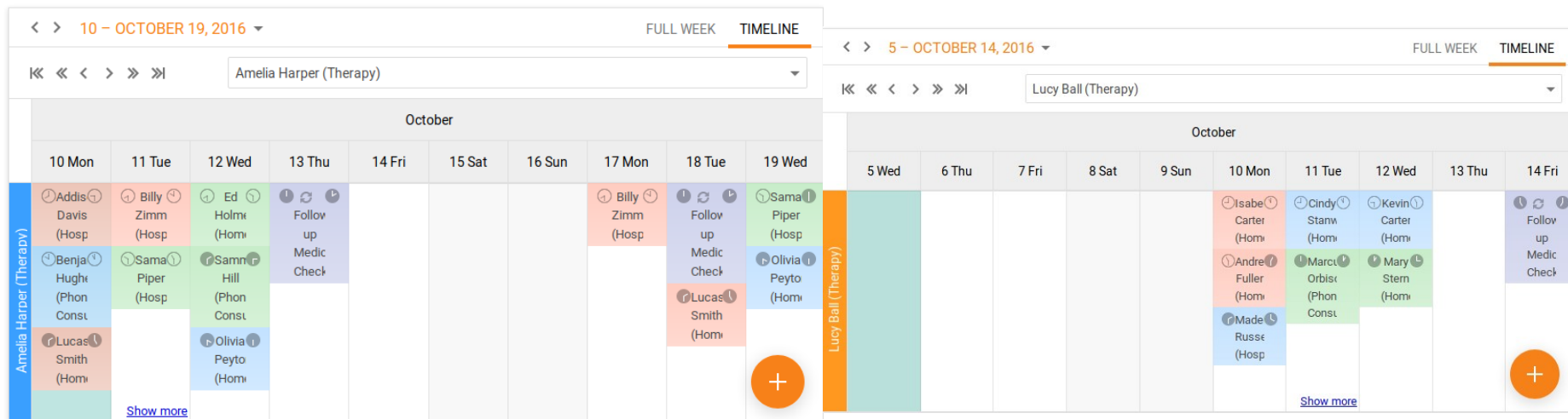
Filterable List

Type something in the input field to search the list for specific items:

> Search..

- First item
- Second item
- Third item
- Fourth

Message Passing in Javascript

- So far the javascript we have seen responds to local events in the browser, such as users clicking buttons, pages loading, mouse movements etc.

- However, we often want to respond to to remote events, such as someone sending you a message, liking a post etc.

- We also may want to dynamically respond to a local event using global information: if a user enters the 1st of April as a prefered appointment date, then we would like to immediately show them the available appointments.

- We could send the date to the server, have the server rebuild the page and send the entire page back, but we only required a few bytes of data.

AJAX vs WebSockets

- There are several technologies to solve these problems

- AJAX stands for Asynchronous Javascript And XML (eXtensible Markup Language), and is really an approach rather than a technology.

- AJAX was coined in 2005 by Jesse James Garrett, using *asynchrous http requests* to a remote server and receiving *XML data* which could be parsed using javascript and dyanmically update a webpage, using the DOM.

- Each AJAX request is a single http protocol exchange, and is done asynchronously, so that waiting for a response does not freeze the environment.

- The server will send the response as a data object (XML or JSON), which can then be factored into the current page.

- Websockets (2011) maintain an open two way connection between a program running on a broswer, and a server, allowing the continual exchange of data.

- We will focus on AJAX for now, as it is the more fundamental technology.

Asynchrony

- When we make a request to a service in javascript, we do not know when, if ever the server will respond.

- Javascript is single threaded, so it must execute each statement in order.

- The environment Javascript runs in is *not* single threaded, so we can write a function, with a function as a parameter, which will be executed when and if the server responds.

- `Object.asyncFn(parameters, callbackFunction)`

- The callback function takes parameters, for errors or the response generated, and executes them.

```
1 function big_Request(data, callback){
2   var req = request_to_server(data);
3   //register_event_listener(req,callback);
4   //when sever responds:
5     callback(req.error,req.response);
6 }
7
8 big_Request(myData, function(error,data){
9   if(error) alert(error);
10  else render(data);
11 }
```
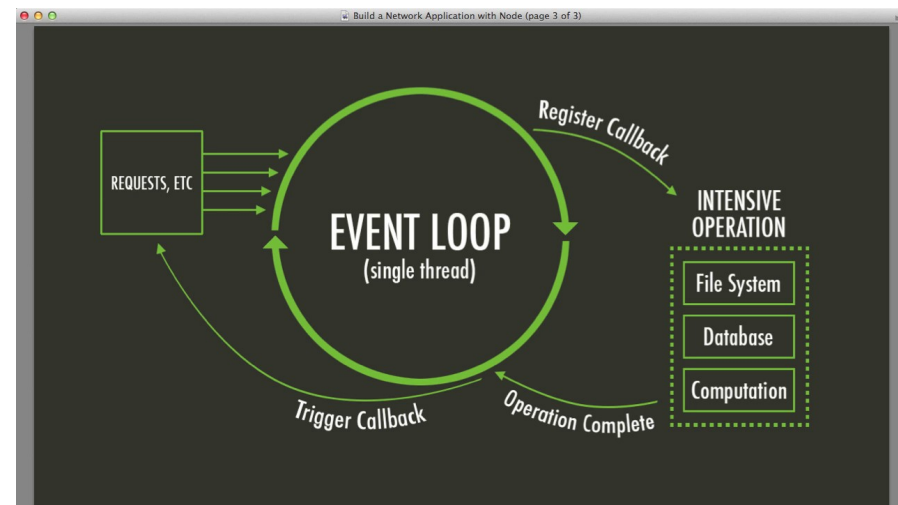
XML vs JSON

- For a javascript function to communicate directly with a server, we require a universal format to transmit data. The two most common forms are XML and JSON.

- XML is *eXtensible Markup Langauage* and is similar in form to HTML. All data is contained in a tree of named tags. It is designed to be as general as possible, it only contains data and does not execute, and is favoured by academics.

- JSON is *JavaScript Object Notation* and stores data in the syntax of javascript: specifically the structural object declaration required to create the object instance representing the data

- JSON is more succinct and can contain arrays, but should not include functions. Both XML and JSON are interpretted by parsers (*JSON.parse*)

```xml
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

```json
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

Client Server Architecture

- When Javascript running in the browser requires a service running on a server, we are using a client-server-architecture.

- The client sends a request to the server, the server recieves the request, formulates and sends a response, and then forgets every thing about the exchange (stateless).

- In the HTTP protocol, requests have a specific form , specifying the method (GET, POST, UPDATE, DELETE) and URL, come with a header and a message body.

- A response reports the status (200 OK, 404 file not found), has a header and a message body.


```
GET /doc/test.html HTTP/1.1          Request Line
Host: www.test101.com
Accept: image/gif, image/jpeg, */*                    Request
Accept-Language: en-us                                Message
Accept-Encoding: gzip, deflate     Request Headers    Header
User-Agent: Mozilla/4.0
Content-Length: 35

                                   A blank line separates header & body
bookId=12345&author=Tan+Ah+Teck    Request Message Body
```

Computer (Client) — HTTP Request / Response — Computer (Server)
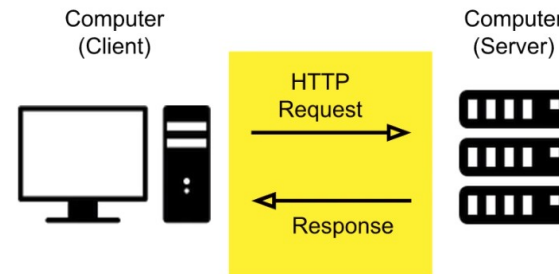
```
HTTP/1.1 200 OK                    Status Line
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)                         Response
Last-Modified: Sat, 07 Feb xxxx                       Message
ETag: "0-23-4024c3a5"              Response Headers    Header
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

                                  A blank line separates header & body
<h1>My Home page</h1>             Response Message Body
```

XMLHTTP requests

- Modern browsers have an XMLHTTP request object to handle requests to, and responses from a remote server.
- The object is initialised, and then *open*s a connection to a server. The *send* method sends the request to the server, and when the server responds, the *status* and *response* can be accessed as properties.
- Browsers only handle GET and POST requests.

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(*method,url,async,user,psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

HTTP requests continued

- Requests maybe either GET or POST. Http allows other request methods (DELETE, PUT) but these are not implemented by most browsers.

- A GET request is used to retrieve data from a server, such as loading a webpage, and a POST request is used to send data to a server, such as submitting a form.

- All requests should be done asynchronously so they do not block other scripts running.

- An asynchronous request has a *readystate* property describing the progress of the request, and an *onreadystatechange* callback function, that is executed when the *readystate* changes.

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);
xhttp.send();
```

```
xhttp.open("POST", "ajax_test.asp", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

HTTP responses

- The request changes state when the server responds, and the response is accessibile as a property of the request (*responseText* or *responseXML*)

- As with the request, the response has a status, the status text, and a header, which is a set of value key pairs.
- The header can be used to set parameters and cookies etc.
- It is accessible via the *getResponseHeader* function

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

HTTP response codes

1xx Informational

100 Continue	101 Switching Protocols	102 Processing (WebDAV)

2xx Success

★ 200 OK	★ 201 Created	202 Accepted
203 Non-Authoritative Information	★ 204 No Content	205 Reset Content
206 Partial Content	207 Multi-Status (WebDAV)	208 Already Reported (WebDAV)
226 IM Used		

3xx Redirection

300 Multiple Choices	301 Moved Permanently	302 Found
303 See Other	★ 304 Not Modified	305 Use Proxy
306 (Unused)	307 Temporary Redirect	308 Permanent Redirect (experimental)

4xx Client Error

★ 400 Bad Request	★ 401 Unauthorized	402 Payment Required
★ 403 Forbidden	★ 404 Not Found	405 Method Not Allowed
406 Not Acceptable	407 Proxy Authentication Required	408 Request Timeout
★ 409 Conflict	410 Gone	411 Length Required
412 Precondition Failed	413 Request Entity Too Large	414 Request-URI Too Long
415 Unsupported Media Type	416 Requested Range Not Satisfiable	417 Expectation Failed
418 I'm a teapot (RFC 2324)	420 Enhance Your Calm (Twitter)	422 Unprocessable Entity (WebDAV)
423 Locked (WebDAV)	424 Failed Dependency (WebDAV)	425 Reserved for WebDAV
426 Upgrade Required	428 Precondition Required	429 Too Many Requests
431 Request Header Fields Too Large	444 No Response (Nginx)	449 Retry With (Microsoft)
450 Blocked by Windows Parental Controls (Microsoft)	451 Unavailable For Legal Reasons	499 Client Closed Request (Nginx)

5xx Server Error

★ 500 Internal Server Error	501 Not Implemented	502 Bad Gateway
503 Service Unavailable	504 Gateway Timeout	505 HTTP Version Not Supported
506 Variant Also Negotiates (Experimental)	507 Insufficient Storage (WebDAV)	508 Loop Detected (WebDAV)
509 Bandwidth Limit Exceeded (Apache)	510 Not Extended	511 Network Authentication Required
598 Network read timeout error	599 Network connect timeout error	

jQuery and AJAX

- We can build XMLRequest objects directly, but jQuery provides some basic functionality for us, and pairs it directly with the DOM.
- The *load* function will send a GET request to a url, and load the data directly into an HTML element
- The *get* function will send a GET request to a url, and passes the data to a callback function.
- The post function will send a POST request, with data to a url and passes the response to a callback function.

```javascript
$("button").click(function(){
  $.get("demo_test.asp", function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

```javascript
$("button").click(function(){
  $.post("demo_test_post.asp",
  {
    name: "Donald Duck",
    city: "Duckburg"
  },
  function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

```javascript
$("button").click(function(){
  $("#div1").load("demo_test.txt", function(responseTxt,
statusTxt, xhr){
    if(statusTxt == "success")
      alert("External content loaded successfully!");
    if(statusTxt == "error")
      alert("Error: " + xhr.status + ": " +
xhr.statusText);
  });
});
```

AJAX Callbacks

- The following is a sequence diagram for an AJAX request
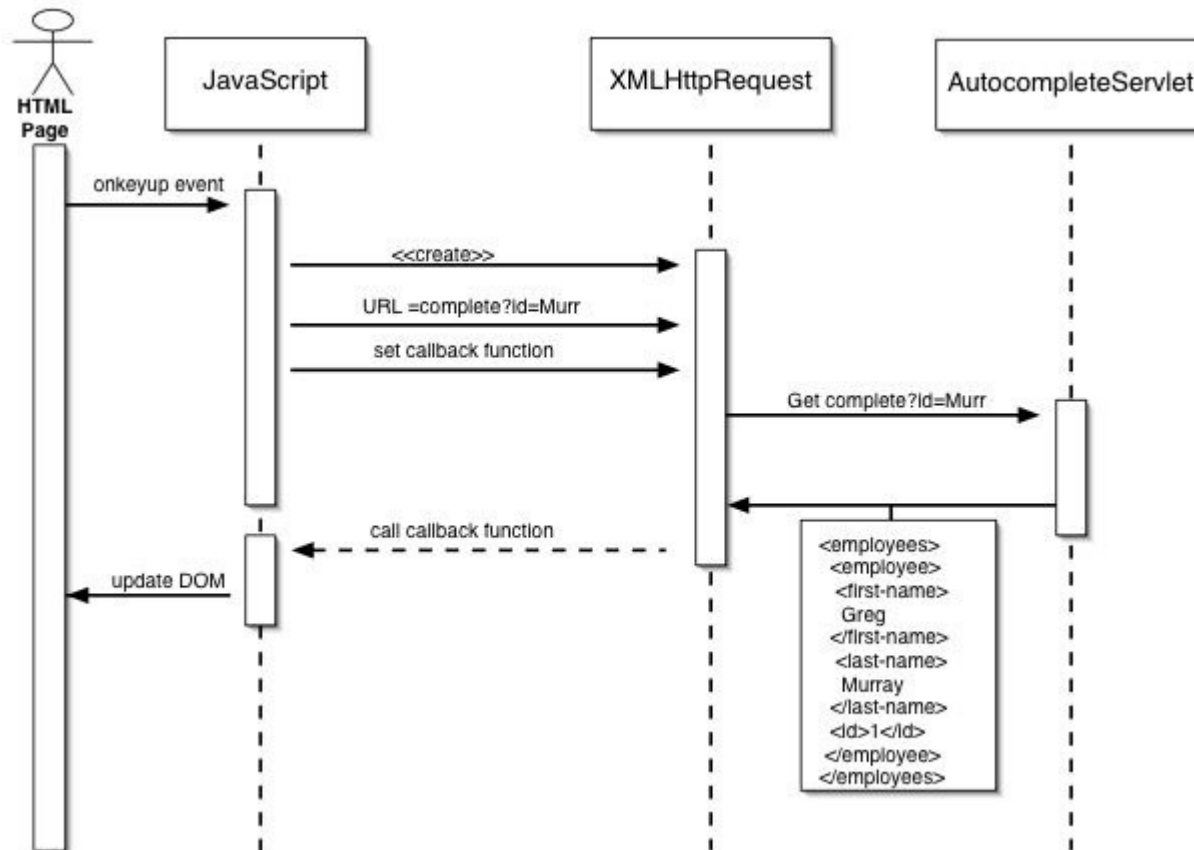
Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1
/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").load("demo_test.txt", function(responseTxt,
statusTxt, xhr){
        if(statusTxt == "success")
          alert("External content loaded successfully!");
        if(statusTxt == "error")
          alert("Error: " + xhr.status + ": " + xhr.statusText);
    });
  });
});
</script>
</head>
<body>

<div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>

<button>Get External Content</button>
```

Let jQuery AJAX Change This Text

Get External Content

Query and AJAX is FUN!

is is some text in a paragraph.

Get External Content

External content loaded successfully!

OK

Single Page Applications

- Single Page Applications have the browser/client do the heavy lifting in a web application: The server just provides the data while the client does the logic and rendering
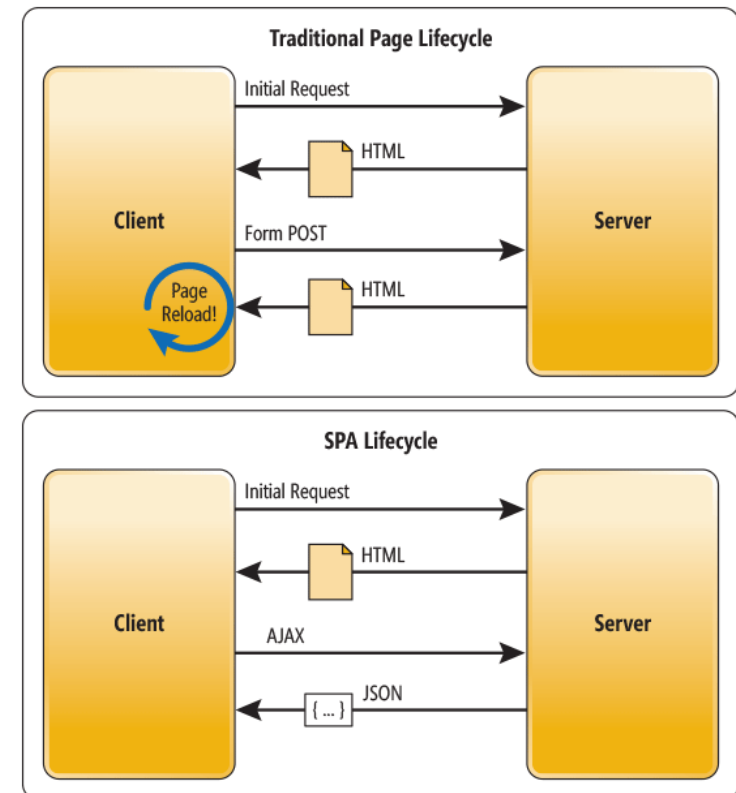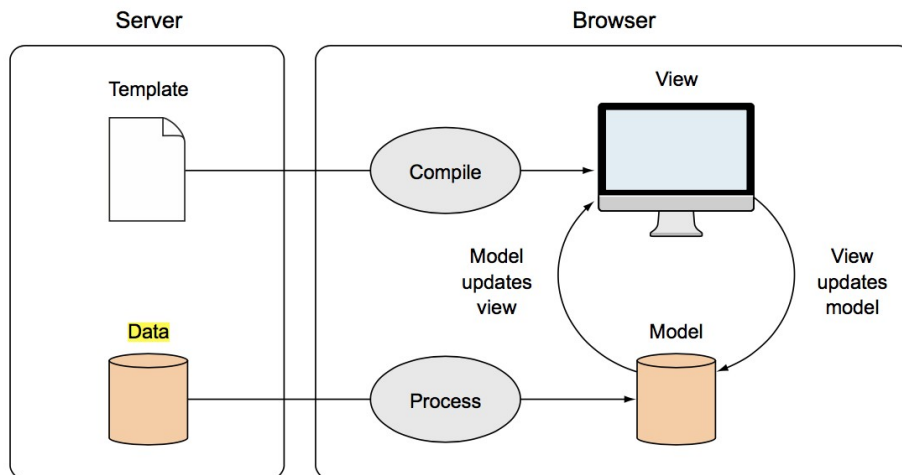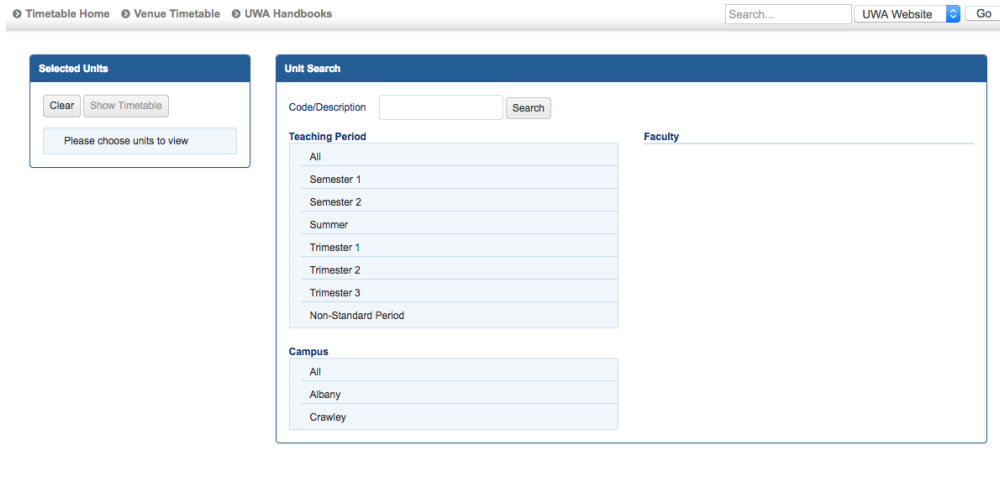
Application Architectures
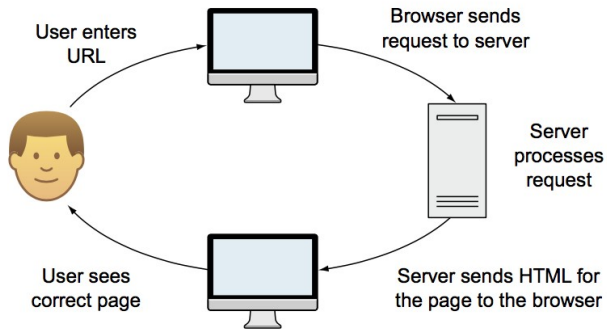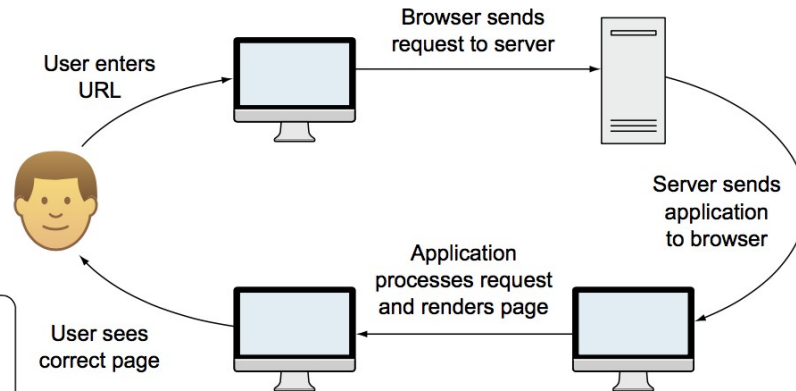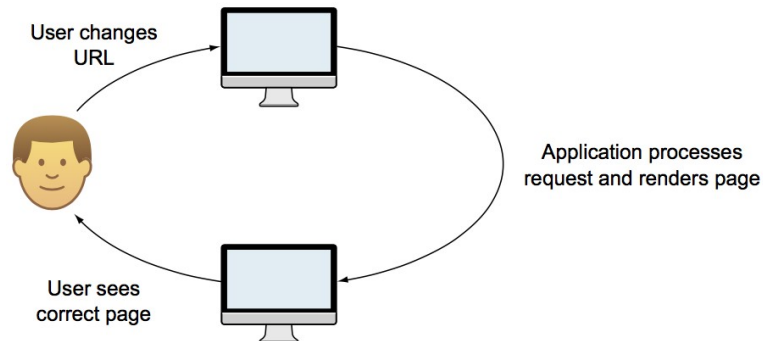
Pros and Cons

Pros

- Less load on the server, able to respond to more clients.
- A more responsive client. No need to wait for server responses.
- Genuine separation between content and presentation.

Cons

- Longer load time, first up. A lot of JS has to be transferred.
- SEO can be a problem. Robots won't crawl js.
- Navigation can be an issue.

To use AJAX or build single page applications, we need to be able to configure the server to handle different requests. So next lecture we'll move on to looking at the back end.