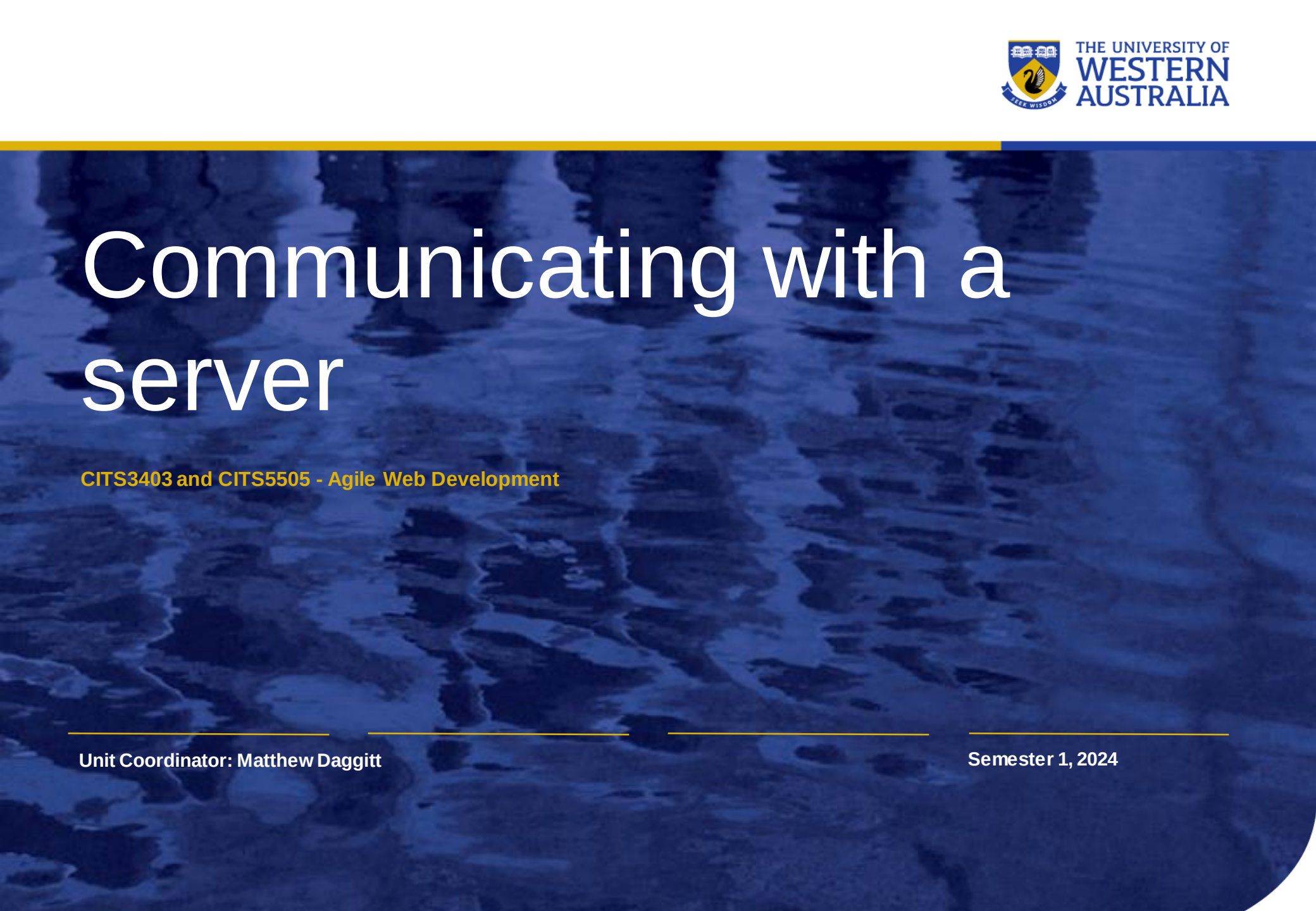

THE UNIVERSITY OF
WESTERN
AUSTRALIA
Communicating with a server
CITS3403 and CITS5505 - Agile Web Development
Unit Coordinator: Matthew Daggitt
Semester 1, 2024

Second half of the course

THE UNIVERSITY OF WESTERN AUSTRALIA

- The second half of the course will cover servers:
    - Week 7 – Communicating with a server & creating your own server
    - Week 8 – Dynamically updating webpages using a server on the fly
    - Week 9 – Adding a database to your server
    - Week 10 – Adding user accounts and making your server secure
    - Week 11 – Testing your server and webpage
    - Week 12 – Group project deadline - end of the week on May 19th.
    - Week 12 – Designing public facing APIs
    - Week 13 – Deploying your server
- Therefore, you will have at least 10 days after the Testing lecture before the project is due. Of course, if you're keen feel free to get ahead!

Message passing in JavaScript

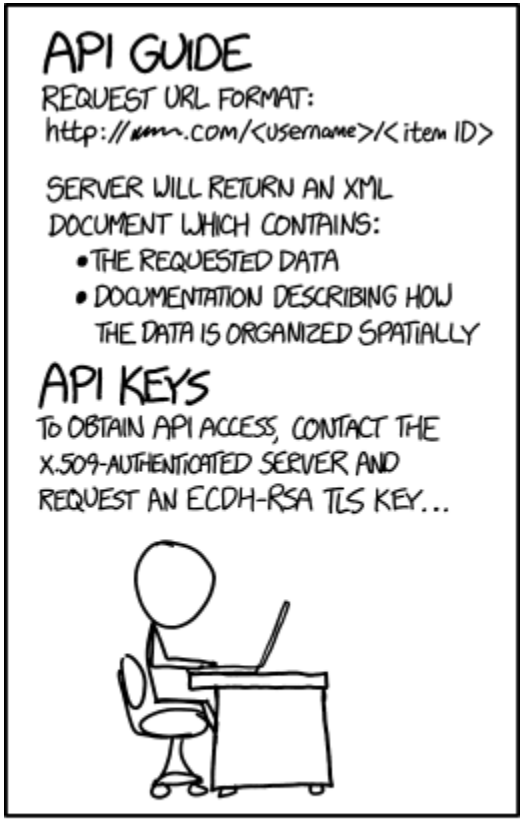
THE UNIVERSITY OF WESTERN AUSTRALIA

- So far, the dynamic pages we have constructed have used to JavaScript to respond to local browser events, e.g.
  - users clicking buttons
  - pages loading

- However, we often want to the browser to be able to communicate with the server after the page is loaded, e.g.
  - updating the page when someone sends you a message, liking a post etc.,
  - submitting a form and storing the completed information in the server.

- One way is to use the same HTTP requests that the browser uses to obtain the original page.

- Another way is to use web sockets (covered in later lectures).



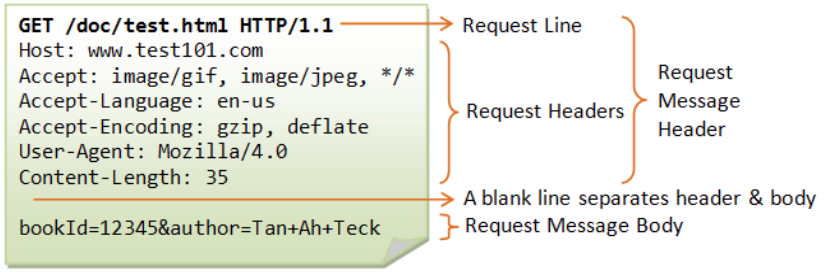
https://xkcd.com/1481/

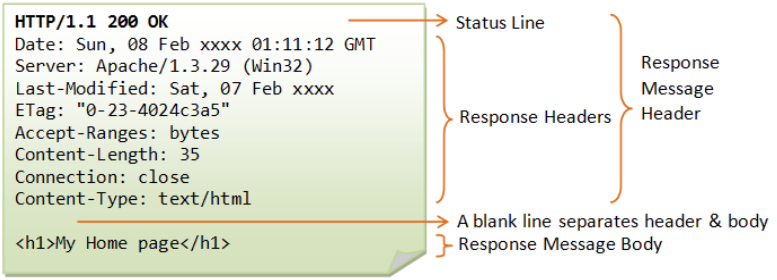
HTTP requests

Structure of HTTP requests and responses

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

- The client sends a HTTP request to the server, the server receives the request, formulates and sends a response, and then forgets everything about the exchange, i.e. the protocol is 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.

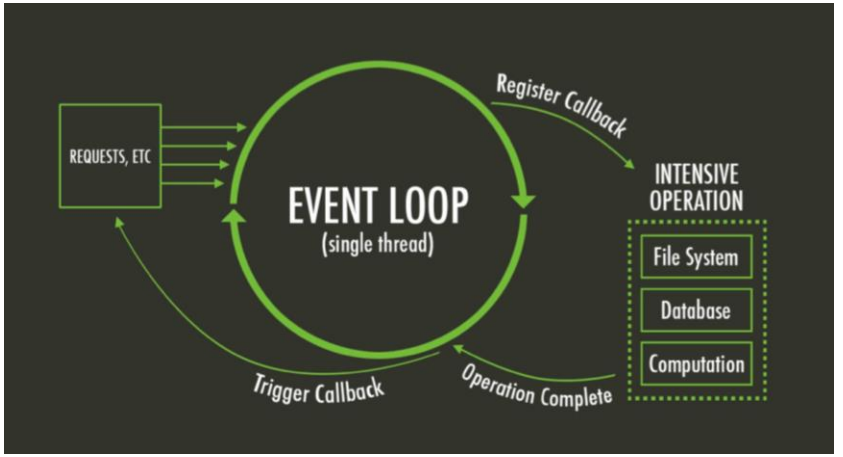
Asynchronous communication

- 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 (it must execute each statement in order), and therefore you might imagine making a request would block other scripts from running until the server responds.
- However, 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 and the response, 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 }
```



XMLHttpRequests

- Modern browsers have an `XMLHttpRequest` 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.

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

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")

Sending a HTTP request in JavaScript

THE UNIVERSITY OF WESTERN AUSTRALIA

- HTTP requests have several standard methods such as GET, POST, DELETE, PUT.
- A GET request is used to retrieve data from a server, such as loading a webpage.

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

- A POST request is used to create or alter data on the server, such as submitting a form.

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

- A PUT request is used to replace data on the server (must be idempotent!).
- The third parameter to `open` is whether the request should be asynchronous. The header can be used to set parameters and cookies etc.

Receiving a HTTP response in JavaScript

THE UNIVERSITY OF WESTERN AUSTRALIA

- The request changes state when the server responds, and the response is accessible as the `responseText` property of the request.

- 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.

- As with the request, the response has a status, the status text, and a header, which is a set of value-key pairs.

```
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();
}
```

HTTP response status codes

THE UNIVERSITY OF WESTERN AUSTRALIA

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 HTTP requests

THE UNIVERSITY OF WESTERN AUSTRALIA

- The jQuery `get` function will send a GET request to a URL and passes the data to a callback function.

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

- The jQuery `post` function will send a POST request, with data to a URL and passes the response to a callback function.

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

AJAX

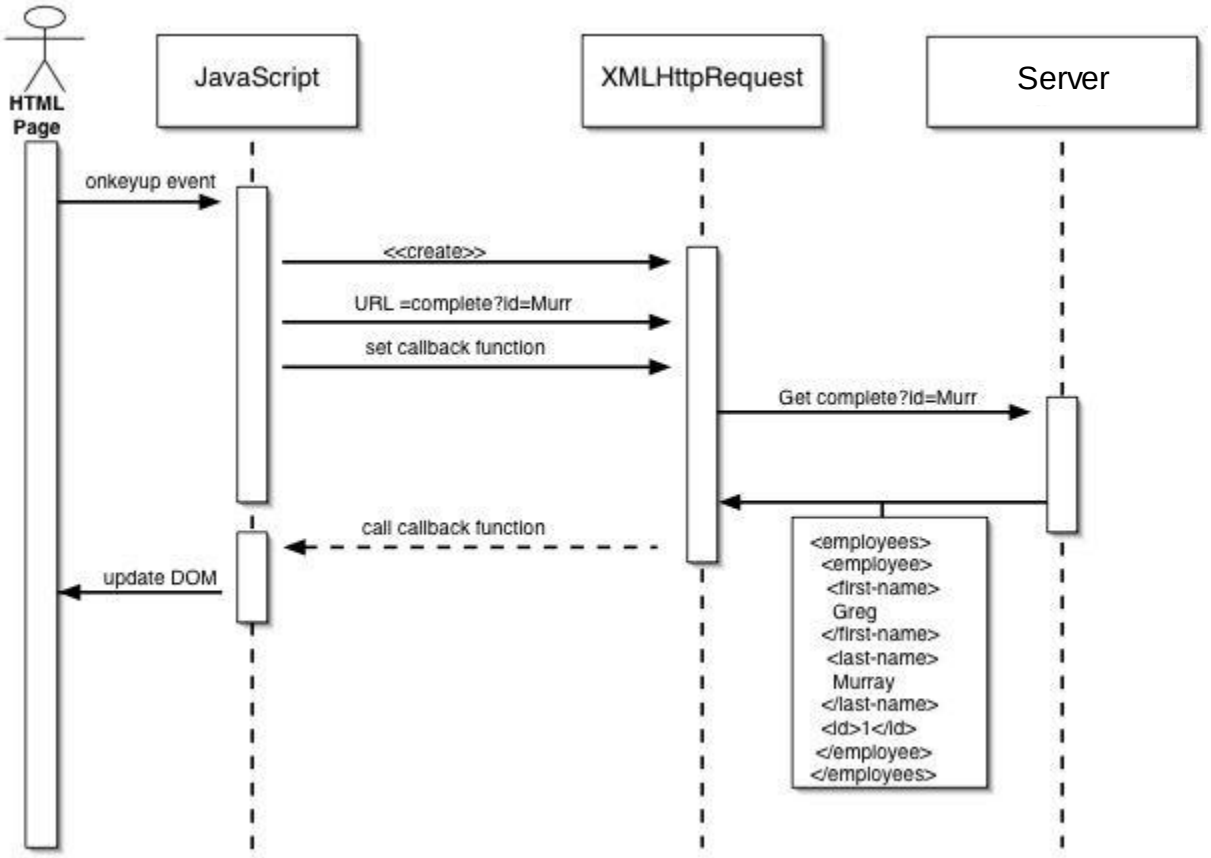
AJAX motivation

THE UNIVERSITY OF WESTERN AUSTRALIA

- HTTP requests allow us to exchange information. But what information should we exchange? We need a protocol!
- 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 - sending asynchronous http requests to a remote server and receiving structured data which could be parsed using JavaScript and dynamically 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.

AJAX callbacks

- The following is a sequence diagram for an AJAX request



XML

- For a JavaScript function to communicate directly with a server, we require a universal format to transmit data (i.e. a protocol!). The two most common formats are XML and JSON.

- XML is *eXtensible Markup Language* 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.

```
<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

- 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

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

- JSON is more succinct and can contain arrays but should not include functions.

- Both XML and JSON strings can be converted back to JavaScript objects using parsers (e.g. the function `JSON.parse`).

- From there the resulting JavaScript objects can be used to update the DOM in the various ways we have already covered.

jQuery and AJAX

THE UNIVERSITY OF WESTERN AUSTRALIA

- We can build `XMLHttpRequest` objects directly, but jQuery provides some basic functionality for us in the form of the `ajax` function.

```
$( "#dtr" ).click(function() {
     $.ajax({
        url: '{{ url('employees/profile/dtr/data?id=').$profile->fempidno }}',
        dataType: 'json',
        success: function (data) {
            console.log(data);
            $('#datatable tr').not(':first').not(':last').remove();
            var html = '';
            for(var i = 0; i < data.length; i++){
                html += '<tr>'+
                            '<td>' + data[i].famin + '</td>' +
                            '<td>' + data[i].famout + '</td>' +
                            '<td>' + data[i].fpmin + '</td>' +
                            '<td>' + data[i].fpmout + '</td>' +
                        '</tr>';
                }
            $('#datatable tr').first().after(html);
        },
        error: function (data) {
        }
    });
});
```

June ▾ 2016 ▾ Go

DATE	#	AM IN	AM OUT	PM IN	PM OUT
Wed	1	07:35	12:07	12:35	6:19
Thu	2	07:46	12:25	12:45	5:18
Fri	3	07:31	12:12	12:37	7:10

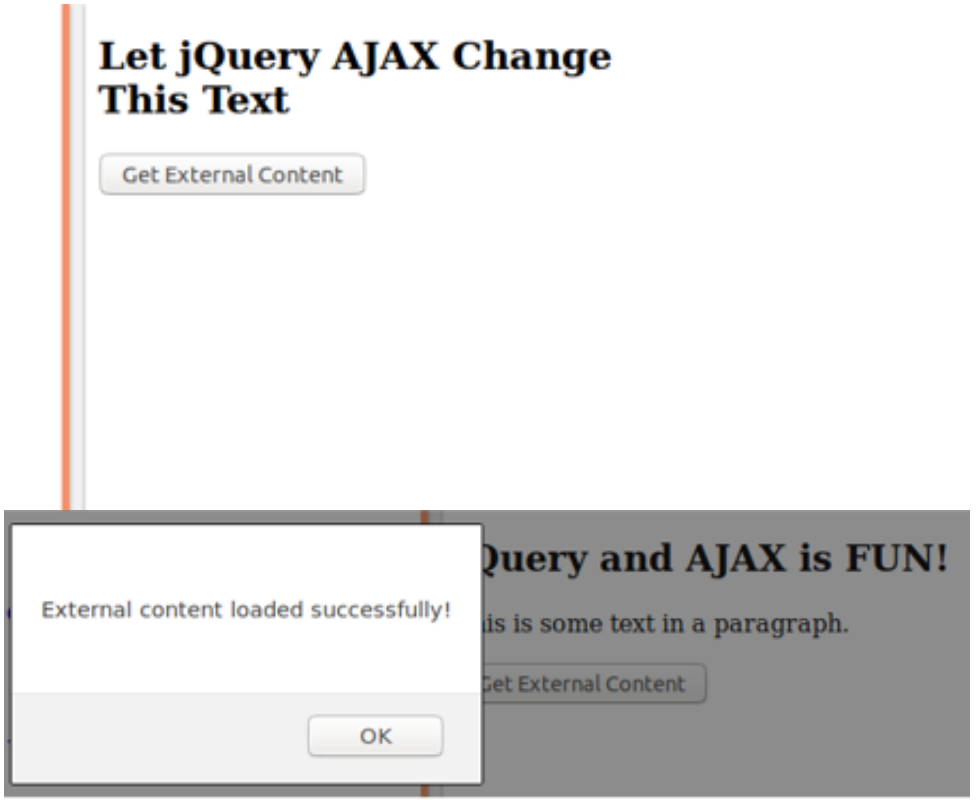
jQuery and AJAX

- jQuery also has functions that allow you to interact directly with the DOM
- For example, the `load` function will send a GET request to a URL, and load the data directly into an HTML element

```
<!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>
```



AJAX in the individual project

THE UNIVERSITY OF WESTERN AUSTRALIA

- For CITS5505 students, the individual project requires you to show off an AJAX requests.
- As we haven't yet covered how to write your own server, the easiest way is to make a call to a public API.
- You may find this list of public facing APIs easy (pick one without an API key!).

  https://github.com/public-apis/public-apis

- For example, https://zenquotes.io/ offers a simple public API to retrieve motivational phrases.