

# Communicating with a server

CITS3403 and CITS5505 - Agile Web Development

Unit Coordinator: Matthew Daggitt

Semester 1, 2024

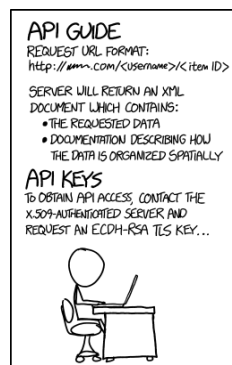
## Second half of the course

- 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!

**DON'T PANIC**

## Message passing in JavaScript

- 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).



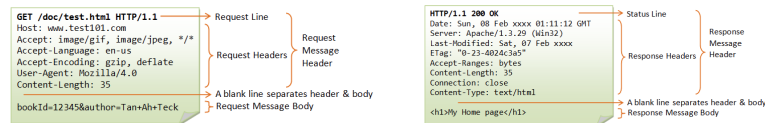
IF YOU DO THINGS RIGHT, IT CAN TAKE PEOPLE A WHILE TO REALIZE THAT YOUR "API DOCUMENTATION" IS JUST INSTRUCTIONS FOR HOW TO LOOK AT YOUR WEBSITE.

<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.

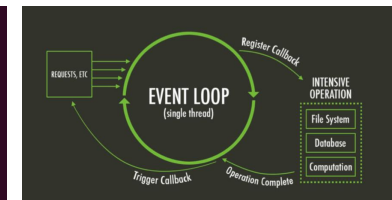
## Asynchronous communication

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

Property	Description
<code>onreadystatechange</code>	Defines a function to be called when the readyState property changes
<code>readyState</code>	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
<code>responseText</code>	Returns the response data as a string
<code>responseXML</code>	Returns the response data as XML data
<code>status</code>	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the <a href="#">Http Messages Reference</a>
<code>statusText</code>	Returns the status-text (e.g. "OK" or "Not Found")

## Sending a HTTP request in JavaScript

- 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 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



### 1xx Informational

100 Continue

### 2xx Success

★ 200 OK  
203 Non-Authoritative Information  
206 Partial Content  
226 IM Used

### 3xx Redirection

300 Multiple Choices  
303 See Other  
306 (Unused)

### 4xx Client Error

★ 400 Bad Request  
★ 403 Forbidden  
406 Not Acceptable  
★ 409 Conflict  
412 Precondition Failed  
415 Unsupported Media Type  
418 I'm a teapot (RFC 2324)  
423 Locked (WebDAV)  
426 Upgrade Required  
431 Request Header Fields Too Large  
450 Blocked by Windows Parental Controls (Microsoft)

### 5xx Server Error

★ 500 Internal Server Error  
503 Service Unavailable  
506 Variant Also Negotiates (Experimental)  
509 Bandwidth Limit Exceeded (Apache)  
598 Network read timeout error

101 Switching Protocols

★ 201 Created  
★ 204 No Content  
207 Multi-Status (WebDAV)

301 Moved Permanently  
★ 304 Not Modified  
307 Temporary Redirect

★ 401 Unauthorized  
★ 404 Not Found  
407 Proxy Authentication Required  
410 Gone  
413 Request Entity Too Large  
416 Requested Range Not Satisfiable  
420 Enhance Your Calm (Twitter)  
424 Failed Dependency (WebDAV)  
426 Precondition Required  
444 No Response (Nginx)  
451 Unavailable For Legal Reasons

501 Not Implemented  
504 Gateway Timeout  
507 Insufficient Storage (WebDAV)  
510 Not Extended  
599 Network connect timeout error

102 Processing (WebDAV)

202 Accepted  
205 Reset Content  
208 Already Reported (WebDAV)

302 Found  
305 Use Proxy  
308 Permanent Redirect (experimental)

402 Payment Required  
405 Method Not Allowed  
408 Request Timeout  
411 Length Required  
414 Request-URI Too Long  
417 Expectation Failed  
422 Unprocessable Entity (WebDAV)  
425 Reserved for WebDAV  
429 Too Many Requests  
449 Retry With (Microsoft)  
499 Client Closed Request (Nginx)

502 Bad Gateway  
505 HTTP Version Not Supported  
508 Loop Detected (WebDAV)  
511 Network Authentication Required

## jQuery and HTTP requests



- 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

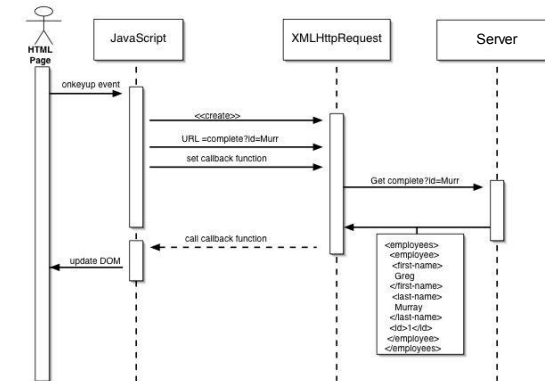


## AJAX motivation

- 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>
    <first-name>John</first-name> <last-name>Doe</last-name>
  </employee>
  <employee>
    <first-name>Anna</first-name> <last-name>Smith</last-name>
  </employee>
  <employee>
    <first-name>Peter</first-name> <last-name>Jones</last-name>
  </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

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

```
1 $( "#dtr" ).click(function() {
2   $.ajax({
3     url: 'url('employees/profile/dtr/data?id=').$profile->fempidno }',
4     dataType: 'json',
5     success: function (data) {
6       console.log(data);
7       $('#datatable tr').not(':first').not(':last').remove();
8       var html = '';
9       for(var i = 0; i < data.length; i++){
10        html += '<tr>'+
11          '<td>' + data[i].famin + '</td>' +
12          '<td>' + data[i].famout + '</td>' +
13          '<td>' + data[i].fmin + '</td>' +
14          '<td>' + data[i].fpmout + '</td>' +
15          '</tr>';
16      }
17      $('#datatable tr').first().after(html);
18    },
19    error: function (data) {
20    }
21  });
22 });
```

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>
```

### Let jQuery AJAX Change This Text

Get External Content

External content loaded successfully!

is some text in a paragraph.

Get External Content

OK

## AJAX in the individual project

- 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.

### Sample Requests

<https://zenquotes.io/api/quotes> - Generate a JSON array of 50 random quotes on each request

<https://zenquotes.io/api/today> - Generate the quote of the day on each request

<https://zenquotes.io/api/random> - Generate a random quote on each request

**NEW!** <https://zenquotes.io/api/image> - Generate a random inspirational image on each request.