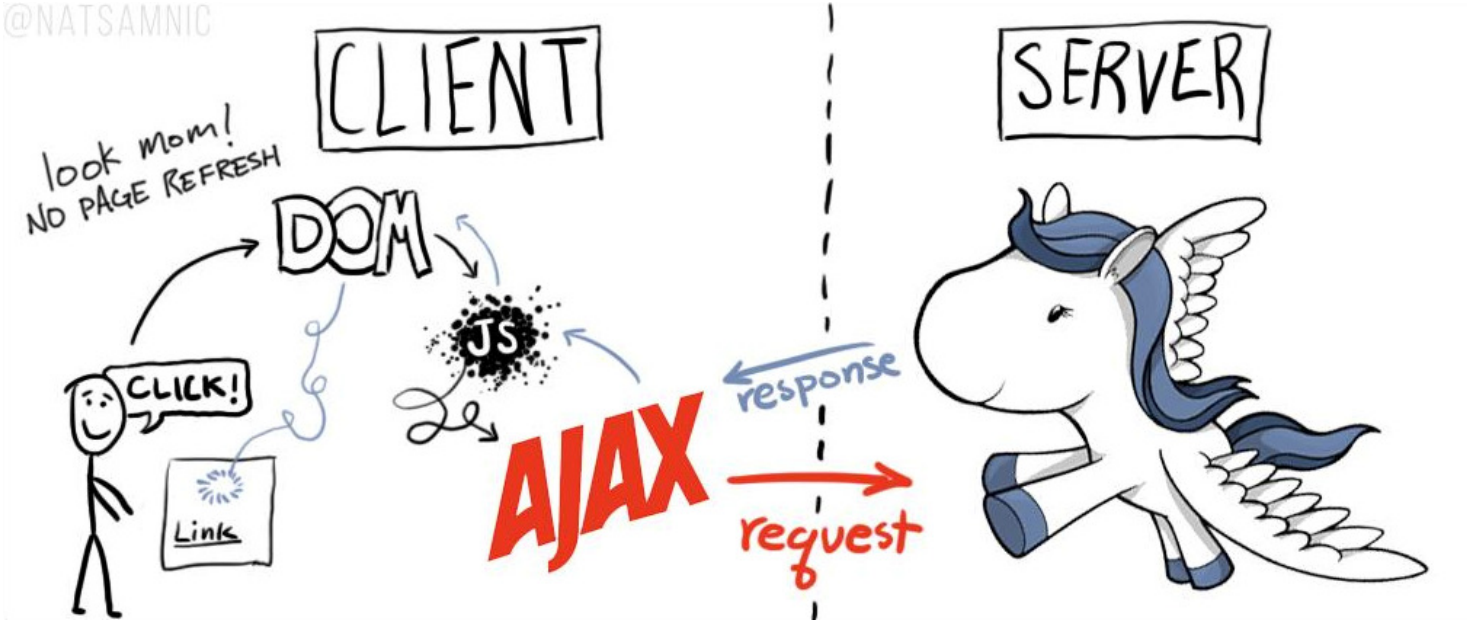
COMP721 Web Development

AUT UNIVERSITY

Week 7: Introduction to Ajax



https://realpython.com/blog/python/django-and-ajax-form-submissions-more-practice/

Agenda

- *The Ajax application model*
- *Ajax component technologies*
- *A glance at an Ajax example*

The Ajax application model

Classic Web Application Models

- are broadly based on the server-side method
  - enter your inputs
  - send the page to the server, which processes the inputs, and then sends a new page back to be rendered on the client
  - The user waits for a response before the next client action
- This is a "click, wait, and refresh" user interaction model
- It is a synchronous "request/response" communication model; the user waits idly for the server to respond
- There are some problems
  - slow performance: full page replacement; waiting synchronously
  - Limited interactivity – why?

AJAX

- Various techniques have been devised to resolve some of the issues of the classic web model
- The aim is to allow dynamic change in the interface presented to the user, and to allow the user to continue interacting with the system whilst the server is busy responding to an earlier request
- "AJAX" – Asynchronous JavaScript and XML – now refers to a collection of technologies that allow the above
  - **Now JSON is becoming more popular than XML as the data transferring format**
  - But still as HTML is a type of XML, we still need to know how to process (CRUD) XML (unless using frameworks to hide the nuts and bolts)

The Ajax Application Model

- Original web page remains displayed in the browser
- Messages are sent to server to do some processing, asynchronously (i.e. the browser does not wait for response, and can still be used for other tasks before data comes back from server – but beware : the user may do something that will interfere with the previous request!)
- Results are sent back from the server when they are ready (as XML, JSON, or plain text)
- JavaScript in the browser is essentially waiting for these, and when the data arrives back, it interprets these, and updates sections of the web page that is being displayed – triggered by receipt of the data from the server

Ajax : How

- There are several options available to implement the Ajax Model
  - **XMLHttpRequest Object – our main focus, as this is now the standard, and is designed to handle XML/JSON technology used to manage asynchronous communication with the server**
  - Hidden Frames – not popular, outdated
  - Hidden iFrames (inline frames) – not popular, outdated

The Ajax Application Model

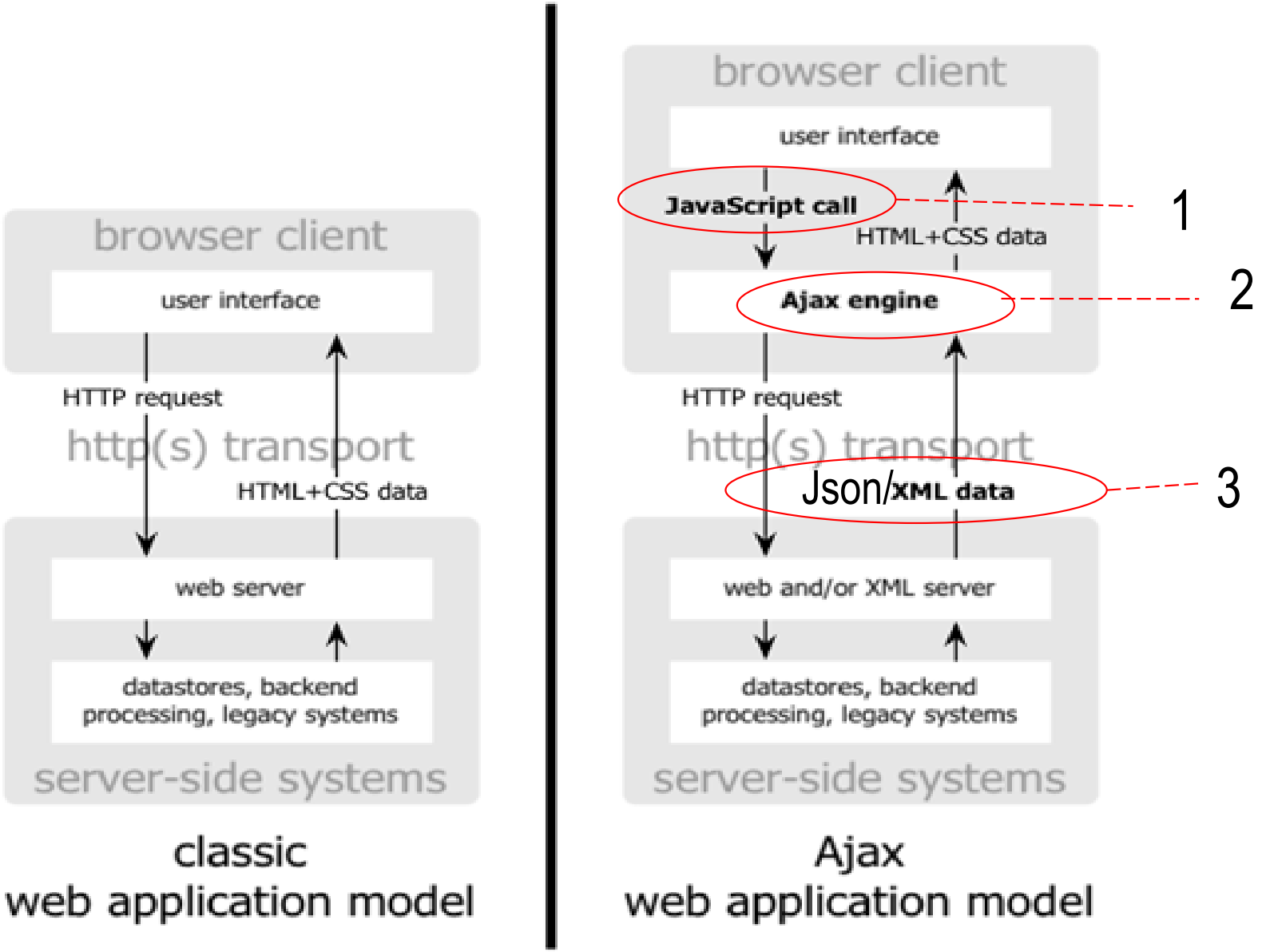
- "Partial screen update" user interaction model
  - During user interaction within an AJAX-based application, only user interface elements that contain new information are updated; the rest of the user interface remains displayed without interruption.
  - This "partial screen update" interaction model not only enables the continuous operation context, but also makes non-linear workflow possible.
  - The partial update is effected through dynamic change of the internal representation of the "document" displayed in the browser

HTML webpage/document

The Ajax Application Model

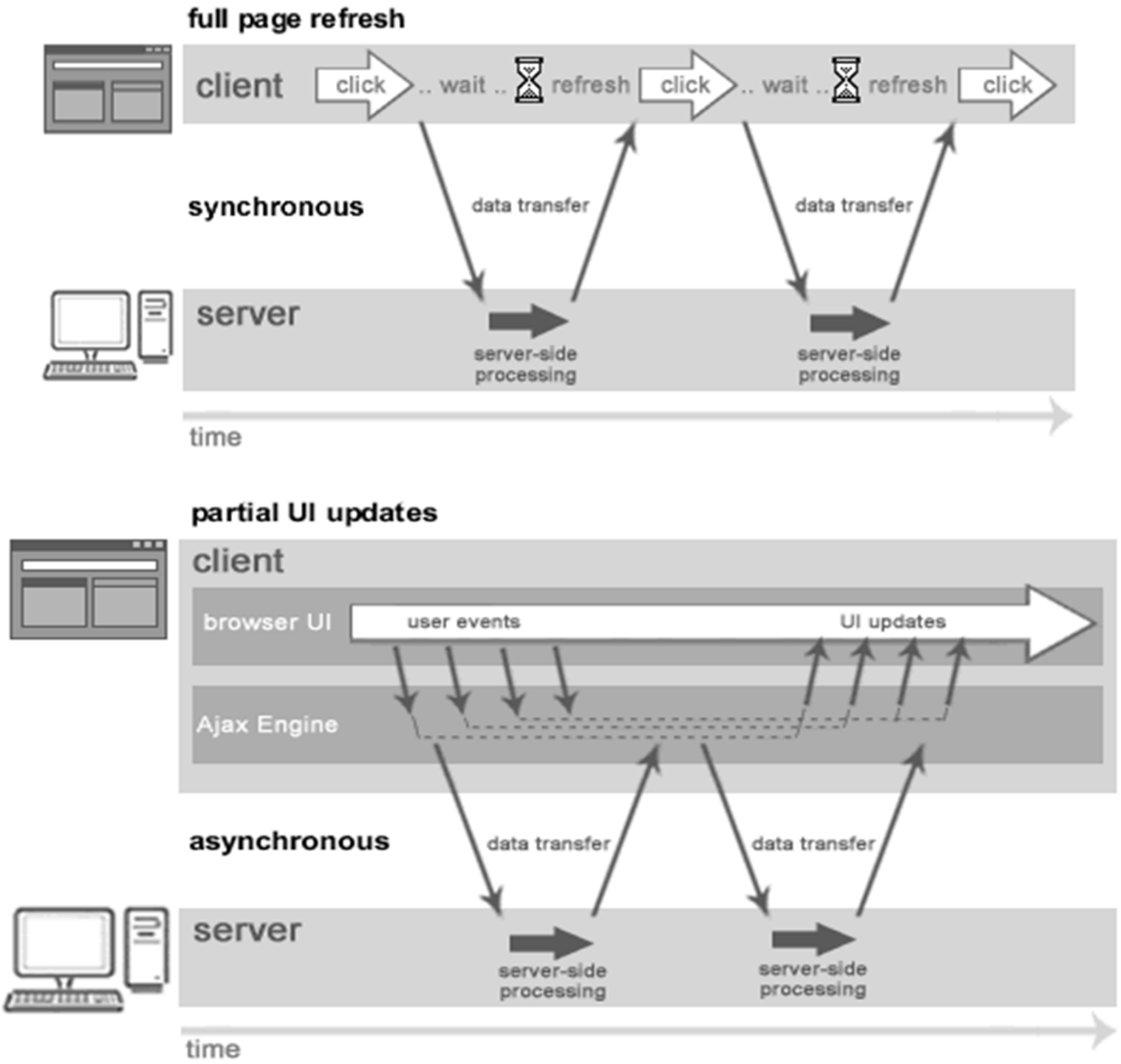
- Asynchronous communication model
  - For an AJAX-based application, the request/response can be asynchronous, decoupling user interaction from server interaction. As a result, the user can continue to use the application while the client program requests information from the server in the background.

Ajax vs. Classic Web Application Models



http://www.adaptivepath.com/publications/essays/archives/000385.php

Ajax vs. Classic Communication Models



Why Should I Use Ajax?

- Partial Page Updating (e.g., return search results)
- Invisible Data Retrieval – populating user controls (e.g., google suggest)
- Smooth Constant Updating: E.g., news column
- Interfaces – AUT homepage dynamic menu
- Simplicity and Rich Functionality
- Drag and Drop – eg via CSS to control visibility of controls

E.g., gmail, blackboard

Popular Examples using Ajax

- Flickr
  http://www.flickr.com/
- Basecamp
  http://www.basecamphq.com/
- Amazon
  http://www.a9.com/
- Google Suggest
  http://www.google.com/webhp?complete=1&h1=en
- Google Maps
  http://maps.google.co.nz/

Ajax component technologies

Ajax Technology Family

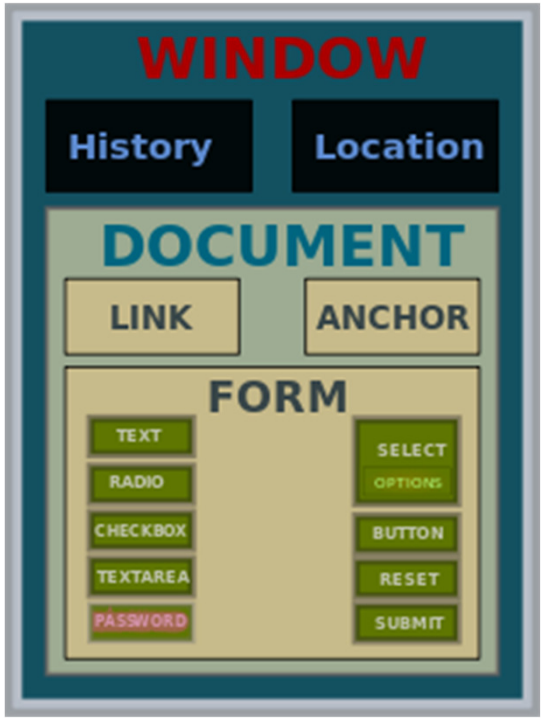
- HTML and CSS for *standards-based presentation*
- Document Object Model for *dynamic display and interaction on the client, and management of JSON/XML on both client and server*
- XML (meta language of HTML) and JSON for *data interchange and manipulation*
- XMLHttpRequest for *asynchronous data retrieval*
- JavaScript for *binding everything together*
- and also need a server-side language to handle any interaction with the server (**PHP**, ASP.NET*, or Java)
- Finally frameworks such as jquery, AngularJS greatly simplify the programming tasks

CSS

- Cascading Style Sheets - describe the presentation and layout of the text and data contained within an HTML page.
- web application design criteria - clear division between the content/structure of the page and the presentation.
- changes made to the style sheet are instantly reflected in the display of the page.
- linked into the document commonly with the HTML <link> tag,
- possible to specify style attributes for each individual HTML tag on a page.
- can also access CSS properties via the DOM.

DOM

- The Document Object Model – API
- lets developers create and modify HTML/XML documents as sets of program objects (in a hierarchy), which makes it easier to design web pages that users can manipulate.
- defines the attributes associated with each object, as well as the ways (methods) in which users can interact with objects.
- works with JavaScript, HTML, and CSS to dynamically change the appearance of Web pages, and make Ajax applications particularly responsive for users.
- DOM techniques are applicable to the client side, and as a result, the browser can update the page or sections of it instantly.



DOM example: the Element object

- Properties

https://www.w3schools.com/jsref/dom_obj_all.asp

attributes	Returns a NamedNodeMap of an element's attributes
children	Returns a collection of an element's child element
innerHTML	Sets or returns the content of an element

- Methods

hasAttribute()	Returns true if an element has the specified attribute, otherwise false
appendChild()	Adds a new child node, to an element, as the last child node

JavaScript

- interacts with HTML code and makes web pages and Ajax applications more active
- can be embedded in HTML pages
- Ajax uses asynchronous JavaScript, allows an HTML page calls the server asynchronously, retrieve new XML data, and simultaneously update the web page partially in the background, all while the user continues interacting with the program.
- JavaScript is a cross-platform scripting language → Ajax applications require no plug-ins
- more in week 8 – used throughout this unit
- Node.js: an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser

XML

- the markup language that is used to **describe** and **structure** data exchanged on the web
- allows developers to *customise* own elements (i.e., tags) to structure data and provide it meaning, that's why XML is also called meta-language
- HTML is a special XML language that has pre-defined tags and fixed tag semantics
- an XML document contains no information about how it should be displayed or searched; it needs other technologies to search and display information from it
- Ajax uses XML to encode data for transfer between a server and a browser (although plain text can also be used)

JSON

- JavaScript Object Notation
- Also used to **describe** and **structure** data exchanged on the web
- More lightweight than XML
- Works very well with JavaScript
- Great for most web applications: well supported in JavaScript, PHP and Document-based DB

XML Technologies

- XML documents – have to be well-formed
- DTD and XML Schema – define what is a “valid” document
- DOM
- XPath
- XSL (XSLT, XSL FO)
- XQuery
- Less important to modern web applications…

XSLT and Xpath (out of our scope)

- XSLT - a language for transforming XML documents into another format (e.g. XHTML, XML, or text)
- XPath - a language for selecting parts of an XML document according to criteria
- XPath is required in XSLT

XMLHttpRequest (XHR) Object

- plays a major role in Ajax applications
- allows a developer to transport the data backward and forward behind the scenes
- Can provide the asynchronous part of the Ajax application by using the *onreadystatechange* event to indicate when it has finished loading information.
- Chrome, Firefox, IE 7+ use a “native” XMLHttpRequest object
  var xhr = new XMLHttpRequest(); --- W3C standard

For some old browsers uses ActiveX control called the XMLHttp object
var xhr = new ActiveXObject(“Microsoft.XMLHTTP”);

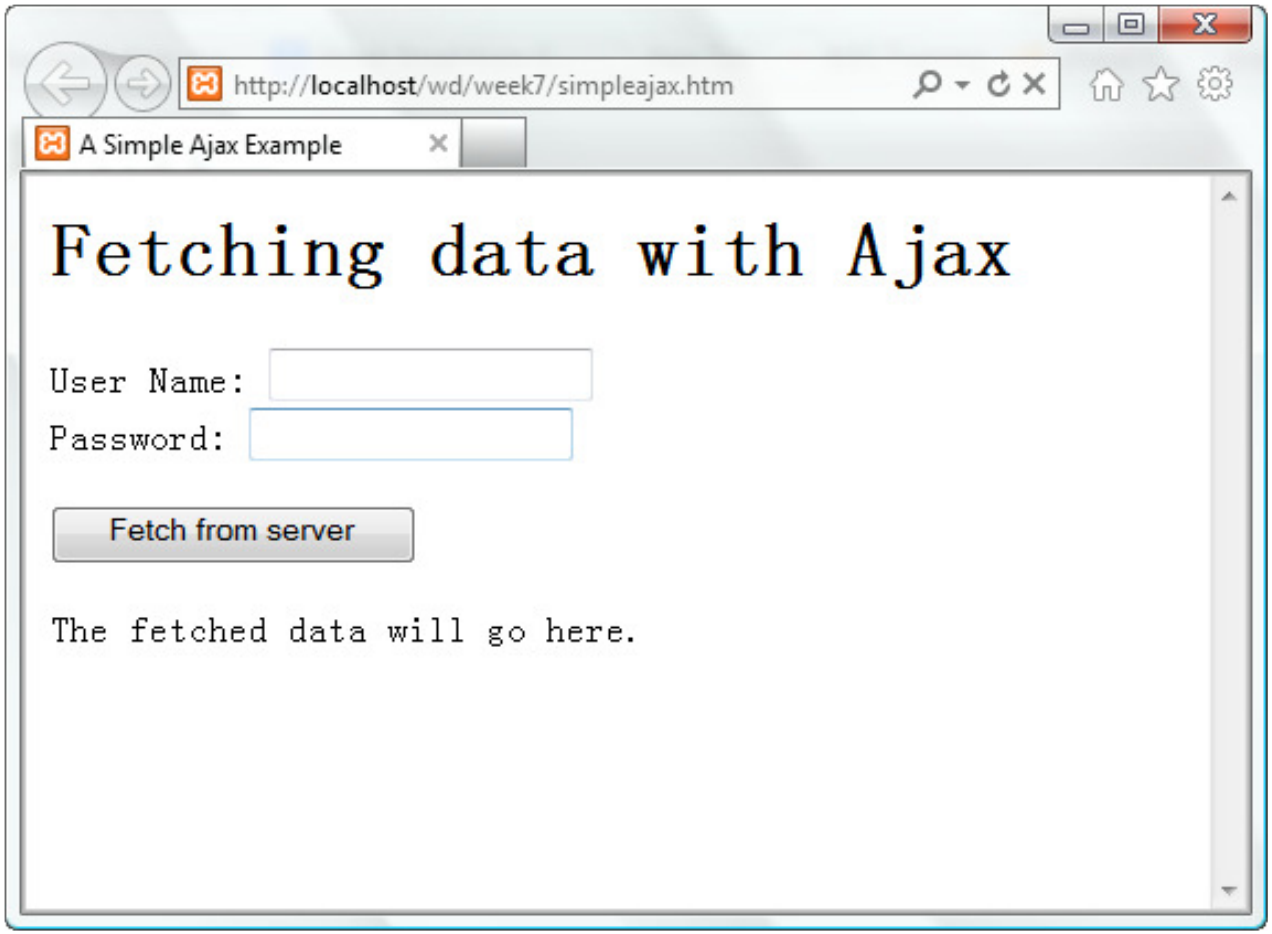
Other Technologies Examined

- DTD and XML Schemas – Describing the structure of XML documents
- Ajax Frameworks
- Web services, APIs and mashups (e.g., displaying property data on a map)
  - Reuse of existing software components (written by others)

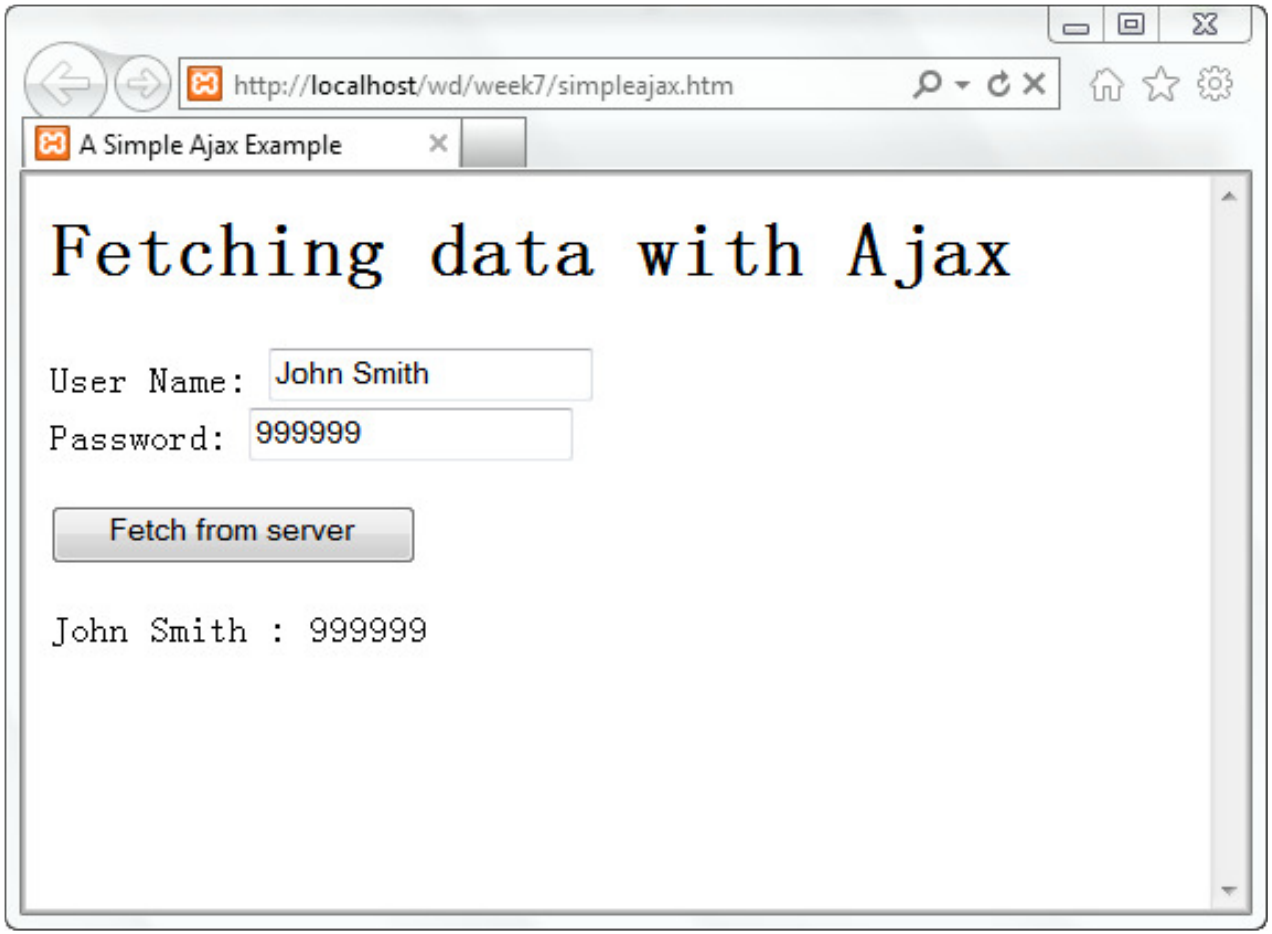
A Very Simple Example

http://jiyu.cmslamp14.aut.ac.nz/wd2021/week07/simpleajax.htm

What it Looks Like



What it Looks Like



Behind the Scenes, as the User imagines it

- User goes to URL (simpleajax.htm)
- User enters data in a simple form
- User clicks on button
- Onclick event handler for button causes data to be sent to server as “parameter” to call of the URL of the PHP file
- This PHP file “executes” on the server extract the data sent, concatenating the two data fields, and sends it back to the client
- When client has received the data, it displays the data received from the server in a “div” set aside in the browser document for that purpose – thus the document is itself changed

A Very Simple Example

- Surely, once the user id and password have been received by the server, we could do some interesting processing : eg, check that the password is ok, by querying a database (in MySQL, or in XML); by extracting relevant data from a database (eg an email address), etc
- The basic interaction, though, is fully illustrated by the simple example
- We will build on this example in the labs, gradually extending to a more complex small system

A Very Simple Example

- This example shows a number of basic techniques that will be used in much more complex examples
  - Handling communication using an XHR object – in this case we pass some data to the server, and later receive back some text created by a PHP program on the server, which processes the data sent by the client, and then sends the output back to the client
  - Using an external JavaScript library function to handle browser differences in creating an XHR object (we use a simple library with just the one function)
  - Using separate JavaScript files to store the JavaScript code that is to be loaded in the HTML document header

What Do We Have to Develop?

- Main (X)HTML file to display user interface, to handle interaction and to display outcome
- XHR object creation function (in a JavaScript "library" file)
- JavaScript file to drive the client side – react to user input, communicate with server, process data returned from server
- PHP script on server to receive data from client, process it, and send data back to client
- Note that all files reside in same directory on the server – we will see later that *data* stored on the server should be in a separate directory, for access control purposes.

What Do We Have to Develop?

- Note that in this example we do not require the following
  - CSS file to describe style formatting in the client
    - Because the example is so simple
  - A database on the server
    - Because we are not storing any persistent data
- In the exercises we will add both of these features

“Main” HTML file

```
<!– file simpleajax.htm -->
<HTML XMLns="http://www.w3.org/1999/xHTML">
 <head>
    <title>A Simple Ajax Example</title>
    <script type="text/javascript“ src=“xhr.js"></script>
    <script type="text/javascript" src="simpleajax.js"> </script>
 </head>
 <body>
```

contains XHR object creation function that caters for browser differences

JavaScript code for this example

To follow on next slide

“Main” HTML file

```
<body>
   <H1>Fetching data with Ajax  </H1>
   <form>
      <label>User Name:  <input type="text" name="namefield"> </label>
      <label><br>Password: <input type="text" name="pwdfield"> <br><br> </label>
      <input name="submit" type = "button" onClick = "getData('data.php',
            'targetDiv', namefield.value, pwdfield.value) " value = "Fetch from server">
   </form>
   <div id="targetDiv">
      <p>The fetched data will go here.</p>
   </div>
</body>
</HTML>
```

getData is in the file simpleajax.js

When button is clicked, execute function getData, telling it : which PHP file to execute, where to place the data that comes back from the server , and what data to send to PHP

Discussion : Normal Synchronous Processing

- Suppose that the system used conventional synchronous processing.
- When the button is pressed in the user-interface, the event-handler attached to the button simply has to make a direct *remote procedure call* to the server
- When the remote procedure is called, the client would then WAIT for the remote procedure to return the result of its processing
- When the procedure result was received back at the client, as a result of the procedure call terminating, it would be assigned to the appropriate place in the document (within the code of the button event-handler).

Discussion : Asynchronous Processing

- With the asynchronous model, it works in a different way!
- This time, when the button is pressed, the event-handler also make the remote procedure call to the procedure on the server, but the event-handler should then immediately terminate, so that the client may accept on-going interaction from the user whilst the server is doing its work.
- We need to specify what is to occur once the remote procedure has sent its result back. The client will accept further user interaction, but we need to specify an event-handler for the client-side event that it has received a message that the server has signalled it has done its work.

Discussion : Asynchronous Processing

- How does the client handle the returned data?
- The way we manage this is that we use an XMLHttpRequest object (XHR object) that sits in the client and serves as the communication link between the client and the server.
- We create this object, and set up the link to the file on the server that has to be "called", and include the parameters that have to be passed to it – it's a bit like plugging a device into a power outlet.
- We then specify what is to occur when the result from the server is eventually received by the XHR object. This is specified as an event-handler attached to the onreadystatechange event of the XHR object
- Finally, we initiate the remote call (xhr.send) – this is a bit like switching the device on.

Discussion : Asynchronous Processing

- Note that in the client, the browser is forever monitoring the objects that have associated events. Thus for example, button presses are picked up.
- The XHR object is also monitored. Thus the onreadystatechange event of this object is monitored, and if this event occurs (or "fires"), the event-handler for THIS event executes.
- Unfortunately this event simply signifies that the readyState property of the XHR object has changed. It does not directly signify that the data has been received from the server.
- The event-handler thus must check if the data HAS been received, and that processing has completed properly. If not, it should do nothing. If so, it should pick up the received data, and process it as required.

- IT'S TIME FOR CODE!!

Our JavaScript Code : simpleajax.js

```
var xhr = createRequest(); // from file xhr.js
function getData(dataSource, divID, aName, aPwd)  {
    if(xhr) {
        var place = document.getElementById(divID);
        var url = dataSource+“?name="+aName+“&pwd="+aPwd;
        xhr.open("GET", url, true);
        xhr.onreadystatechange = function() {
                alert(xhr.readyState); // to monitor progress
                if (xhr.readyState == 4 && xhr.status == 200) {
                        place.innerHTML = xhr.responseText;
                } // end if
        } // end anonymous call-back function
        xhr.send(null);
    } // end if
```

Only do this if xhr was successfully created. If it was not, then in fact nothing happens! There is no connection to the server, no data returned, and the user will see nothing! To be honest, we should include an else alternative, that alerts the user to an error.

Use GET protocol. The “true” means that asynchrono us access is requested.

Alternative, with named call-back function

```
var xhr = createRequest(); // from file xhr.js
function getData(dataSource, divID, aName, aPwd)  {
    if(xhr) {
        var place = document.getElementById(divID);
        var url = dataSource+“?name="+aName+“&pwd="+aPwd;
        xhr.open("GET", url, true);
        xhr.onreadystatechange = placeData(place);
        xhr.send(null);
     } // end if
} // end function getData()

function placeData(location) {
    alert(xhr.readyState); // to monitor progress
    if (xhr.readyState == 4 && xhr.status == 200) {
        location.innerHTML = xhr.responseText;
```

Here we provide a named function as the call-back, and supply a parameter as we do need to pass the location for the data display in to this function

We only ever use this function for the one purpose, and it really does not need to be named. So the typical programming practice is to use the anonymous function approach of the previous slide.

Creating the XHR Object – Use of External File

- Note that xhr is just our *chosen* name for the XHR object that communicates between client and server; any name could be chosen, as in later examples
- We define the object via a call to an external function, that handles browser differences (see later slide)
- The function createRequest, which returns a valid XHR object, is defined in the JavaScript file xhr.js. That file is indicated as an included script in our main HTML file.

XHR Object creation

```
// file xhr.js
// return a valid XHR object
 function createRequest() {
    var xhrObj = false;
    if (window.XMLHttpRequest) {
        xhrObj = new XMLHttpRequest();
    }
    else if (window.ActiveXObject) {
        xhrObj = new ActiveXObject("Microsoft.XMLHTTP");
    }
    return xhrObj;
} // end function createRequest()
```

Check for native XHR object first, as most contemporary browsers have this. Use the native object in IE where possible.

Older versions of IR do not have the native XHR object, so we then create an appropriate ActiveXObject that has similar functionality

Defining the XHR Object – Browser Differences

- Our implementation is very simple, and just covers essential differences between Firefox and older versions of IE. For IE7+, it will use the native XMLHttpRequest object available rather than the ActiveX object.
- A more sophisticated implementation is needed to give wider browser availability.
- If neither condition in the code is satisfied, then no XHR object is created, and the function returns false.
- The function call does not fail, but the possibility of false being returned should be handled in the calling code.

NB : Client-Server Communication Protocol

- We have used the GET protocol in this example
- It is simpler
- But it exposes the transferred data in the URL that is linked to, and so is not secure
- GET should NOT be used for id/password
- Also, the length of the URL is limited, so only small amount of data can be transmitted
- POST should really be used here. The lab exercises ask you to change to using POST.

Server Script : data.php

```
<!--file data.php -->
<?php
    // get name and password passed from client
    $name = $_GET['name'];
    $pwd = $_GET['pwd'];
    sleep(10); // simulate delay at server to make async obvious
    // write back the password concatenated to end of the name
    echo ($name." : ".$pwd)
?>
```

What Happens?

- We start by loading “simpleajax.htm” in the browser
- When this loads, the JavaScript files in the header load
  - xhr.js just makes available the function createRequest() that creates an XHR object
  - simpleajax.js is executable code, and when it is loaded
    - The XHR object named xhr is created via a call to function createRequest() {xhr.js is loaded first to enable this}
    - The function getData() is made available – it does not execute until it is called
- The body of the HTML now loads

What Happens?

- This basically displays the form for the user interface, and prepares the location for the returned data (an HTML div). Nothing happens – it is waiting for us to enter the data fields and press the button
- When we press the button, the registered “onclick” event causes the JavaScript function getData() to be called
- This function executes; it sets up variable “place”, opens up the connection of xhr to the server, specifying that the “GET” protocol will be used, specifies the PHP file on the server that will load to xhr, specifies that processing is to be asynchronous (via the use of “true” as the third parameter) - and defines the (anonymous) call-back function that will execute when the ready state of xhr changes

What Happens?

- Next, start the client – server communication via xhr by “send” request; this sends the data as part of the URL in the GET protocol, and it activates the execution of the PHP script on the server, which in this case simply takes the data passed as parameters, concatenates them, sleeps for 10 seconds (to simulate heavy load on the server – don’t do this in a real system!!) and echos the result. This essentially writes the resulting string to xhr’s responseText property. Whilst the user waits for the response (s)he can interact with the system.

What Happens?

- As the communication continues, the server updates properties of xhr, keeping xhr aware of the state of the communication; every time the state changes, the onreadystatechange event is fired on xhr, and the anonymous call-back function (mentioned on the previous slide) executes – each time it executes we get an alert box that shows the new value of the readyState property of xhr
- Don't have this alert in a real system either – it is here to show how the event fires several times as the state changes, before finally indicating that the communication has ended)

What Happens?

- The conditional code in the call-back function checks the readyState and status properties of xhr
- If the former is 4, and the latter is 200 (ie, the server has completed sending its response, and the status of xhr is “OK”), then the innerHTML property of the location represented by the variable place is assigned the value of the responseText property of xhr (which is the concatenated string sent from the server)
- Note that the state of xhr changes several times and the call-back function executes several times prior to the completion of the communication