1

CSC 443: Web Programming

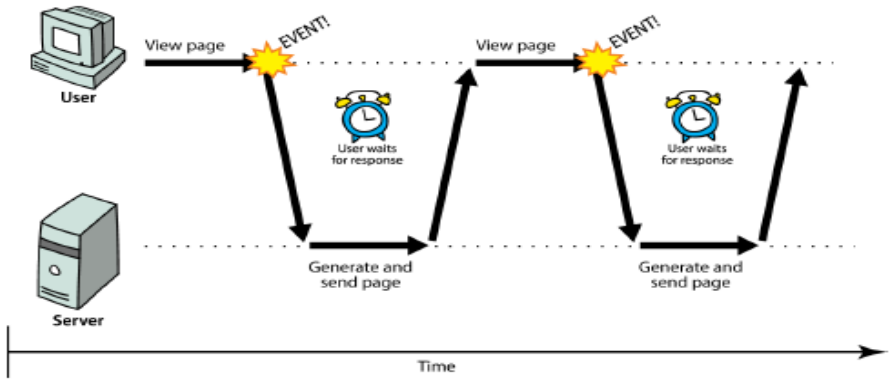
Haidar Harmanani

Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

AJAX

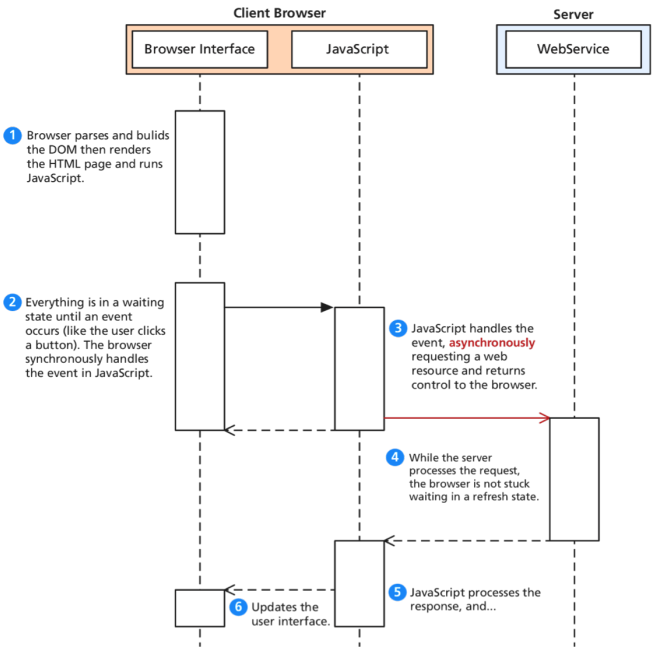
- Asynchronous JavaScript and XML
  - First mentioned by Jesse James Garrett in 2005
- Based on several technologies
  - Standards-based presentation
    - XHTML and CSS
  - Dynamic display and interaction
    - DOM
  - Data interchange and manipulation
    - XML and XSLT
  - Asynchronous data retrieval
    - XMLHttpRequest
  - Binding everything together
    - JavaScript

Synchronous web communication



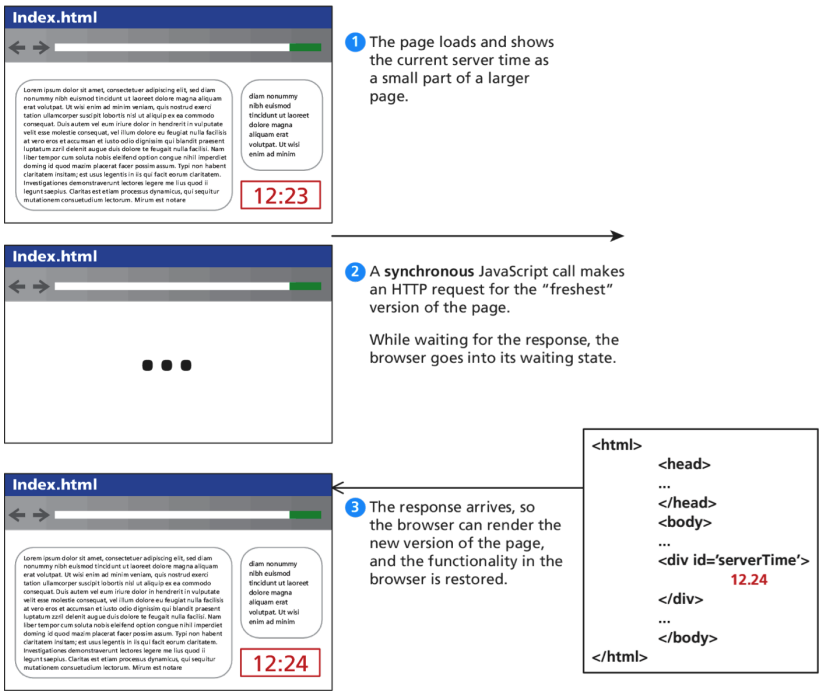
- Synchronous: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)

AJAX



AJAX

Consider a webpage that displays the server's time



AJAX

Consider a webpage that displays the server's time

Web applications and Ajax

- **web application**: a dynamic web site that mimics the feel of a desktop app
  - presents a continuous user experience rather than disjoint pages
  - examples: Gmail, Google Maps, Google Docs and Spreadsheets, Flickr, A9

XMLHttpRequest (and why we won't use it)

- JavaScript includes an XMLHttpRequest object that can fetch files from a web server
  - Supported in IE5+, Safari, Firefox, Opera, Chrome, etc. (with minor compatibilities)
- It can do this asynchronously (in the background, transparent to user)
- The contents of the fetched file can be put into current web page using the DOM

XMLHttpRequest methods

- The core JavaScript object that makes Ajax possible

Method	Description
open(*method*, *url*, *async*)	specifies the URL and HTTP request method
send() send(*postData*)	sends the HTTP request to the server, with optional POST parameters
abort()	stops the request
getAllResponseHeaders() getResponseHeader(*name*) setRequestHeader(*name*,*value*)	for getting/setting raw HTTP headers

XMLHttpRequest properties

Property	Description
responseText	the entire text of the fetched page, as a string
responseXML	the entire contents of the fetched page, as an XML document tree (seen later)
status	the request's HTTP status code (200 = OK, etc.)
statusText	HTTP status code text (e.g. "Bad Request" for 400)
timeout	how many MS to wait before giving up and aborting the request (default 0 = wait forever)
readyState	request's current state *(0 = not initialized, 1 = set up, 2 = sent, 3 = in progress, 4 = complete)*

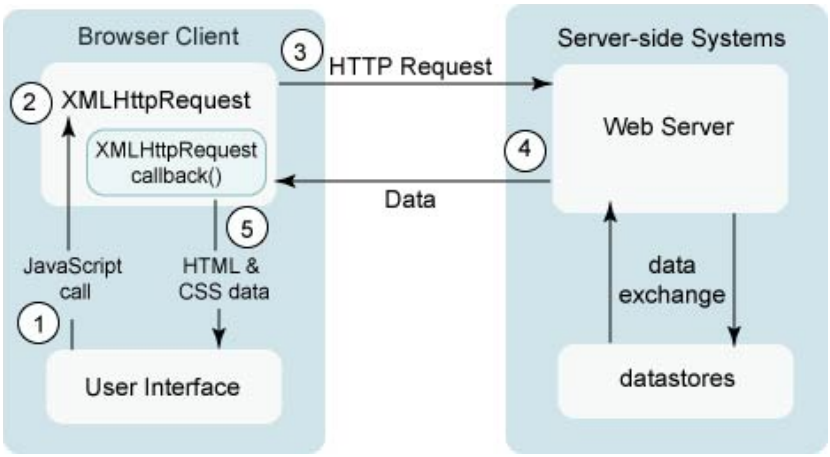
XMLHttpRequest events

Event	Description
load	occurs when the request is completed
error	occurs when the request fails
timeout	occurs when the request times out
abort	occurs when the request is aborted by calling abort()
Loadstart loadend progress readystate change	progress events to track a request in progress

A typical Ajax request

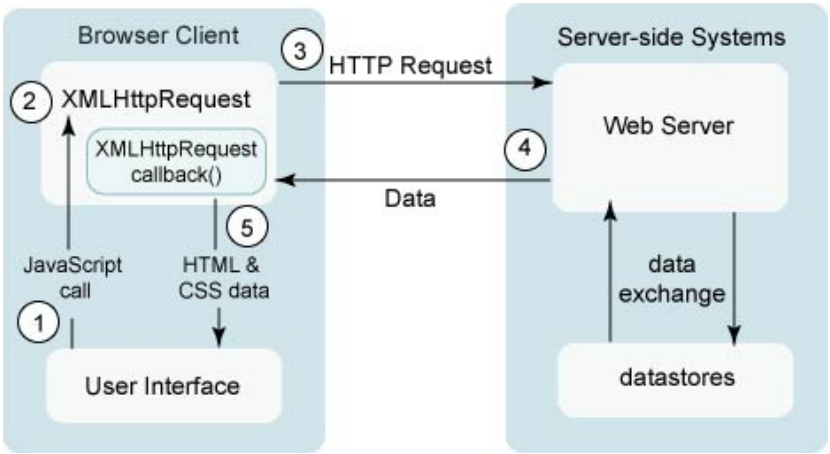
1. user clicks, invoking an event handler
2. handler's code creates an `XMLHttpRequest` object
3. `XMLHttpRequest` object requests page from server
4. server retrieves appropriate data, sends it back
5. `XMLHttpRequest` fires an event when data arrives
   - this is often called a callback
   - you can attach a handler function to this event
6. your callback event handler processes the data and displays it

A typical Ajax request



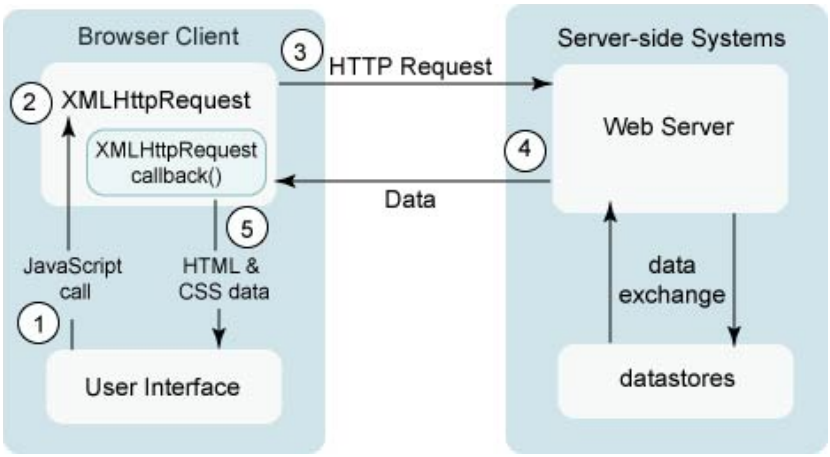
1. user clicks, invoking an event handler

A typical Ajax request



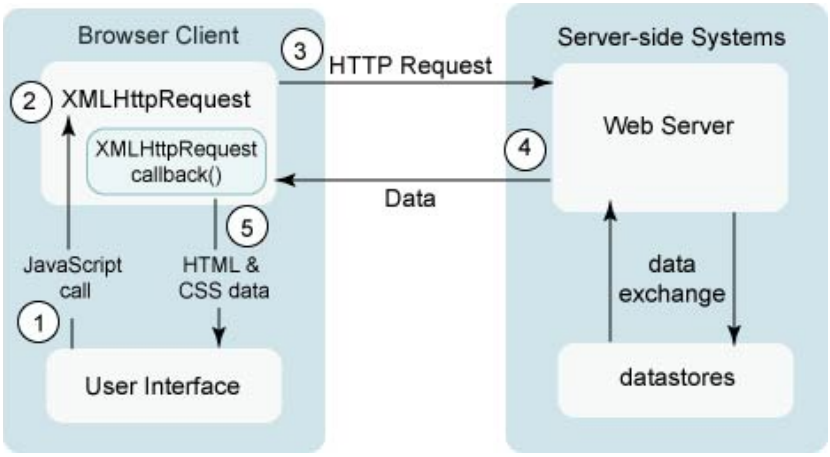
2. handler's code creates an `XMLHttpRequest` object

A typical Ajax request



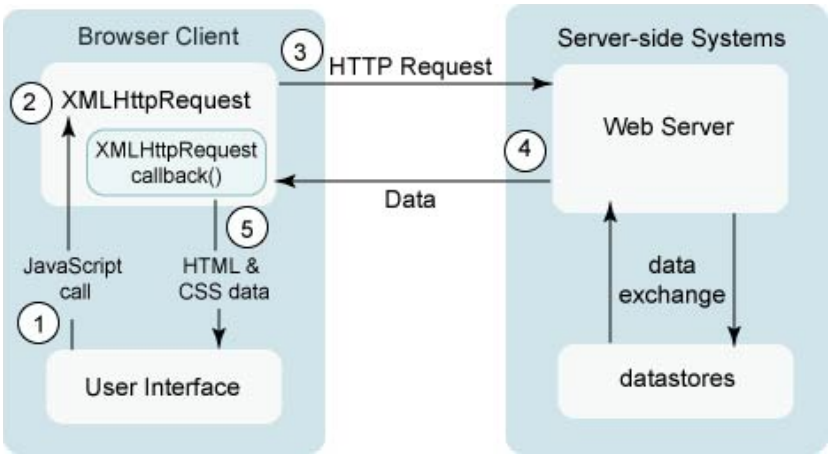
3. `XMLHttpRequest` object requests page from server

A typical Ajax request



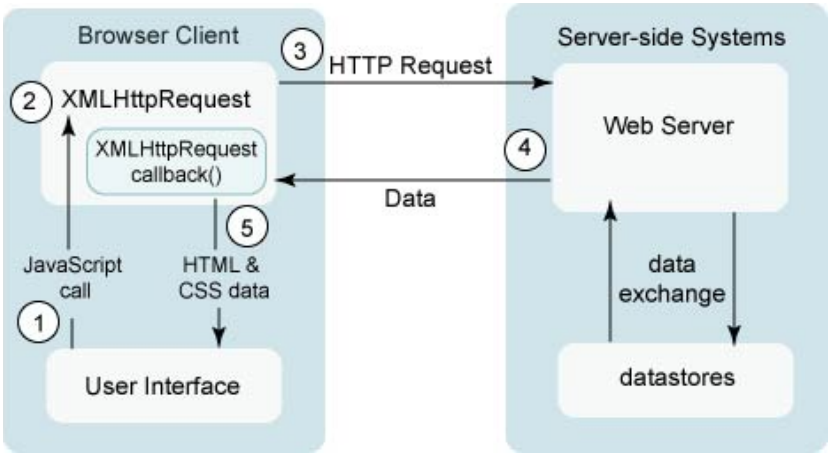
4. server retrieves appropriate data, sends it back

A typical Ajax request



5. `XMLHttpRequest` fires an event when data arrives. A callback to which you can attach a handler function

A typical Ajax request



6. your callback event handler processes the data and displays it

XMLHttpRequest (and why we won't use it)

- Sounds great!...
- ... but it is clunky to use, and has various browser incompatibilities
- jQuery provides a better wrapper for Ajax, so we will use that instead

AJAX: Making Asynchronous requests

- jQuery provides a family of methods to make asynchronous requests
- Consider the very simple server time example that we saw earlier
  - If `currentTime.php` returns a single string and you want to load that value asynchronously into the `<div id="timeDiv">` element, you could write:
    - `$("#timeDiv").load("currentTime.php");`

Ajax: The jQuery Way!

- Simplified
- `$.ajax(url, [settings])`
  - `url` : a string containing the url - optional
  - `settings` : key-value pairs
  - Returns `jqXHR` object (superset of `XMLHTTPRequest` object)

jQuery Ajax utilities

- `$.ajax({options})`
  - Makes an Ajax request.
  - Example:
    - `$.ajax({ url: "address", success: responseHandler});`
    - The response handler is passed the response text, not the response object.
    - Don't forget the "." before "ajax"!
- `load(url)`
  - Makes an AJAX call to update the selected element's data from a server
    - `$("#some-id").load("address");`
    - A data string or object is an optional second arg
  - Example
    - `$("#theTable tr").load("url");`
      - Will get the file from 'url' (which could be the result of a script) and load it into the selected table row
      - We can also put in a callback function but it is not required

jQuery Ajax utilities

- Shortcuts
  - `$.get, $.post, $.getJSON`
  - Slightly simpler forms of $.ajax. However, they support fewer options, so many developers just use $.ajax.
- `.get(url [,data] [,callback] [,datatype])`
- `.post(url [,data] [,callback] [,datatype])`
  - Make AJAX requests with the stated default way of passing parameters
  - Both can supply options to the server

Options for $.ajax({...})

- url: A string containing the URL to which the request is sent
- type: The type of request to make, which can be either “POST” or “GET”
- data: The data to send to the server when performing the Ajax request
- success: A function to be called if the request succeeds
- accepts: The content type sent in the request header that tells the server what kind of response it will accept in return
- dataType: The type of data expected back from the server
- error: A function to be called if the request fails
- async: Set this options to false to perform a synchronous request

Options for $.ajax({…})

- cache: Set this options to false to force requested pages not to be cached by the browser
- complete: A function to be called when the request finishes (after success and error callbacks are executed)
- contents: An object that determines how the library will parse the response
- contentType: The content type of the data sent to the server
- password: A password to be used with XMLHttpRequest in response to an HTTP access authentication request
- statusCode: An object of numeric HTTP codes and functions to be called when the response has the corresponding code
- timeout: A number that specifies a timeout (in milliseconds) for the request

"success" Handler

- Simplest form
  - `function someHandler(text) { … }`
    - text
      - Response data from server, not the response object
      - "text" can really be XML or JSON, depending on the dataType option
- Full form
  - `function someHandler(text, status, request) { … }`
    - text
      - Response data from server
    - status
      - String describing the status: "success" or "notmodified". Rarely useful. In error handlers, the status string is more meaningful.
  - request
    - The raw XMLHttpRequest object.

AJAX: Without jQuery


```
function getRequestObject() {
   if (window.XMLHttpRequest) {
      return(new XMLHttpRequest());
   } else if (window.ActiveXObject) {
      return(new ActiveXObject("Microsoft.XMLHTTP"));
   } else { return(null); }
}

function sendRequest() {
   var request = getRequestObject();
   request.onreadystatechange = function() { someFunct(request); };
   request.open("GET", "some-url", true);
   request.send(null);
}
```


AJAX: With jQuery


```
$.ajax({
   url: "someURL.php",
   type: "POST",
   data: {},          // data to be sent to the server
   dataType: "xml"
   }).done(function(data) {
       // Do stuff with data
   }).fail(function(xhr, status) {
      // Respond to an error
});
```

AJAX: With jQuery

- Example:

```
$.ajax({
  type: "POST",
  url: "some.php",
  data: { name: "John", location: "Boston" }
});
```

AJAX: With jQuery


```
function showTime1() {
        $.ajax({ url: "show-time.php",
        success: showAlert,
        cache: false});
}

function showAlert(text) {
alert(text);
}
```

The cache option is not required but is a convenient option when using GET and the same URL (including query data) yields different responses. This way, you don't have to send Cache-Control and Pragma headers from server.

This is the response text, not the response object. Use three-argument version if you need the raw `XMLHttpRequest` object.

AJAX: With jQuery


```
$.ajax(
{
    url: “process.php",
    type: "POST",
    data: “class=name=Tim",
    success: function(msg){
      alert(“Data:" + msg );
  }
 }
);
```

```
$.post(
    "test.php",
    {
     func: "getNameAndTime"
    },
    function(data){
        alert(data.name);
        alert(data.time);
    },
    "json“
};
```


AJAX in JQuery

- $.get(url [, data] [, success(data,textStatus, jqXHR){})

```
$.get( "ajax/test.html", function( data ) {
  $( ".result" ).html( data );
  alert( "Load was performed." );
});
```

- $.post(url [, data] [, success(data,textStatus, jqXHR){})

```
$.post( "ajax/test.html", postdata, function( data ) {
  $( ".result" ).html( data );
});
```

- $.getJSON(url [, data] [, success(data,textStatus, jqXHR){})
  - Use an AJAX get request to get JSON data

AJAX: With jQuery


```
$.getJSON(
  "http://api.flickr.com/services/feeds/photos_public.gne?tags=doggy&format=json&jsoncallback=?",
  function(data){
  $.each(data.items, function(i,item){
          $("<img/>").attr("src",item.media.m).appendTo("#images");
          if ( i == 2 ) return false;
    }
  );
 }
);
```


AJAX: With jQuery

- Ajax Events:
  - `ajaxComplete(callback), ajaxStart(callback), ajaxStop(callback), ajaxSend(callback), ajaxError(callback), ajaxSuccess(callback)`

```
$('<div id="loading">Loading...</div>')
    .insertBefore("#images")
    .ajaxStart(
       function(){
         $(this).show();
       }
).ajaxStop(
    function(){
       $(this).hide();
    }
);
```

Example 1: HTML Code


```
<head><title>jQuery and Ajax</title>...
<script src="./scripts/jquery.js" type="text/javascript"></script>
<script src="./scripts/jquery-ajax.js" type="text/javascript"></script>
</head>
<body>...

<fieldset>
<legend>$.ajax: Basics (Using onclick handler in HTML)</legend>
<input type="button" value="Show Time" onclick='showTime1()'/>
</fieldset>
```


Example 1: PHP Code


```
<?php
$msg = date('d/m/Y h:i:s');
echo $msg;
?>
```

show-time.php

jQuery and AJAX: AjaxEvents

- .ajaxComplete(): Register a handler to be called when Ajax requests complete.
- .ajaxError(): Register a handler to be called when Ajax requests complete with an error.
- .ajaxSend(): Attach a function to be executed before an Ajax request is sent
- .ajaxStart(): Register a handler to be called when the first Ajax request begins
- .ajaxStop(): Register a handler to be called when all Ajax requests have completed.
- .ajaxSuccess(): Attach a function to be executed whenever an Ajax request completes successfully

jQuery and AJAX: Methods

- jQuery.ajax(): Perform an asynchronous HTTP (Ajax) request.
  - .load(): Load data from the server and place the returned HTML into the matched element
  - jQuery.get(): Load data from the server using a HTTP GET request.
  - jQuery.post(): Load data from the server using a HTTP POST request.
  - jQuery.getJSON(): Load JSON-encoded data from the server using a GET HTTP request.
  - jQuery.getScript(): Load a JavaScript file from the server using a GET HTTP request, then execute it.
  - .serialize(): Encode a set of form elements as a string for submission.
  - .serializeArray(): Encode a set of form elements as an array of names and values.

Example2


```
<scripttype="text/javascript"src="jquery-1.4.1.min.js"></script>
<scripttype="text/javascript">

$(document).ready(function(){
function update(){
    $.ajax({
       type:'POST',
       url:'datetime.php',
       timeout:1000,
       success: function(data){
          $("#timer").html(data);
          window.setTimeout(update,1000);
       },
    });
  }
  update();
});

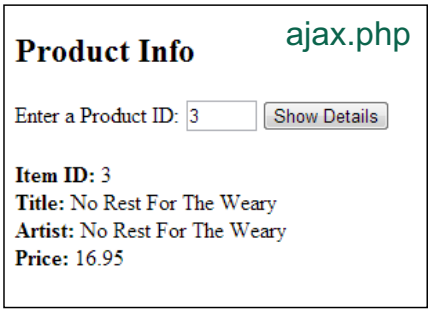
</script><divid="timer"></div>
```


Example 3

- The jQuery $.post() method loads data from the server using an HTTP POST request.
- Syntax
  - `$.post(URL, {data}, function(data){…});`
  - ```
    $.post("myScript.php", {name:txt}, function(result) {
        $("span").html(result);
      });
    ```

Parameter	Description
URL	Required. Specifies the url to send the request to.
data	Optional. Specifies data to send to the server along with the request.
function(data)	•Optional. Specifies a function to run if the request succeeds. *data* - contains the resulting data from the request

http://www.w3schools.com/jquery/ajax_post.asp


    ```

Example 3

```php
<?php

 $id = $_POST['id'];

 mysql_connect("localhost", "omuser", "omuser") or die("Error connecting");
 mysql_select_db("om") or die("Error selecting DB");
 $query = "SELECT * FROM items WHERE item_id = $id";
 $result = mysql_query($query);

 if (mysql_num_rows($result) == 0) {
   echo "Product ID $id not found.";
   return;
 }
 $row = mysql_fetch_array($result);

 echo "<strong>Item ID:</strong> {$row['item_id']}<br>";
 echo "<strong>Title:</strong> {$row['title']}<br>";
 echo "<strong>Artist:</strong> {$row['artist']}<br>";
 echo "<strong>Price:</strong> {$row['unit_price']}<br>";
```

Get this from the Ajax call

show_product.php

Example 3

```
$('#show').click(function(){

        var prodId = $('#id').val();

        $.post(

          "show_product.php",

             {id:prodId},

             function(result) {
             $('#detail').html(result);
          }
    );
});
```

When the button is clicked

Get the text box value

Ajax POST

Name of the PHP script

The key/value to be passed

Update the "detail" div
With the output of the PHP script

ajax.php

43

More about AJAX and jQuery…

Example 4


```
$('#my_input').autocomplete({
        'source': 'http://foo.com/webservice.php'
});
```

```
if (!isset($_GET['term'])) {
        header('HTTP/1.1 400 Invalid Request -No term parameter provided');
        die('No term parameter provided.');
}
$term = $_GET['term'];
$results = getCompleterResults($term);
// an array() return value print
json_encode($results);
```

- when you have too many choices to hold them all in an array, you can instead fetch subsets of choices from a server using AJAX
- instead of passing choices as an array, pass a URL from which to fetch them
  - the AJAX call is made with a term parameter
  - the choices are sent back from the server as a JSON array of strings or array of objects with label and valuefields

Ajax: GET and POST

- Similar to Prototype
  - `$.get(URL, callback);`
  - `$.post(URL, data, callback);`

```
var myURL = “someScript.php”; // or some server-side script
$.post(
myURL, // URL of script
{ // data to submit in the form of an object
name: “John Smith”,
age: 433
},
function(data, status) { ... } // callback function
);
```


jQuery and AJAX


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

```
$(document).ajaxSuccess(function(){
    alert("AJAX request successfully completed");
});
```

jQuery and AJAX


```
$.ajax({
   url: 'http://api.joind.in/v2.1/talks/10889',
   data: {
      format: 'json'
   },
   error: function() {
      $('#info').html('<p>An error has occurred</p>');
   },
   dataType: 'jsonp',
   success: function(data) {
      var $title = $('<h1>').text(data.talks[0].talk_title);
      var $description = $('<p>').text(data.talks[0].talk_description);
      $('#info')
         .append($title)
         .append($description);
   },
   type: 'GET'
});
```


AJAX

- $(selector).load(URL, data, callback)

```
var myURL = "http://www.mysite.com/myFile.txt";
$("#myButton").click( function() {
   // Pass in the URL and a callback function.
   // xhr is the XMLHttpRequest object.
   $("#myDiv").load(myURL, function(response, status, xhr)
{
         if(status == "success")
           alert(response);
         else if(status == "error")
           alert("Error: " + xhr.statusText);
   });
});
```

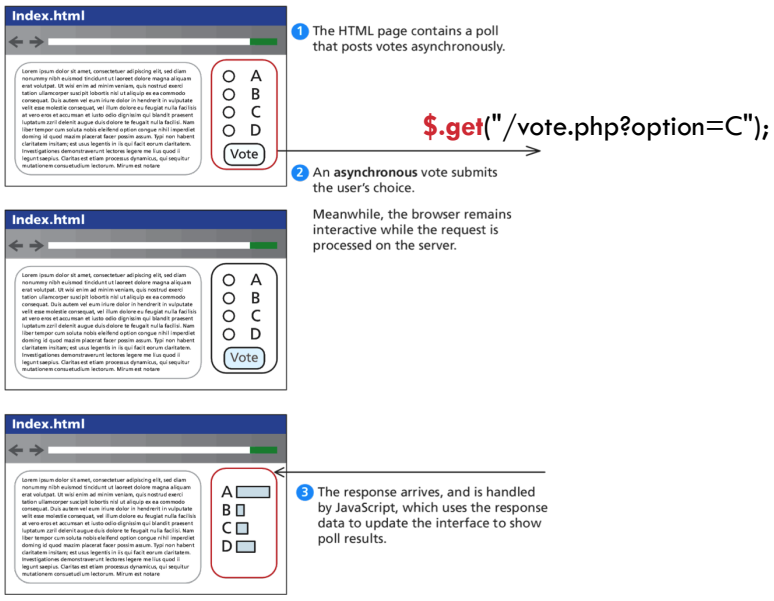
CORS: Cross Origin Resource Sharing

- Since modern browsers prevent cross-origin requests by default (which is good for security), sharing content legitimately between two domains becomes harder.
- **Cross-origin resource sharing (CORS)** uses new headers in the HTML5 standard implemented in most new browsers.
- If a site wants to allow any domain to access its content through JavaScript, it would add the following header to all of its responses.
- **Access-Control-Allow-Origin:** *

CORS: Cross Origin Resource Sharing

- A better usage is to specify specific domains that are allowed, rather than cast the gates open to each and every domain. To allow our domain to make cross site requests we would add the header:
- Access-Control-Allow-Origin: www.funwebdev.com
- Rather than the wildcard *.

AJAX
GET Requests



AJAX
GET Requests – formal definition

jQuery.get (**url** [, **data**] [, **success(data, textStatus, jqXHR)**] [, **dataType**])

- **url** is a string that holds the location to send the request.
- **data** is an optional parameter that is a query string or a *Plain Object*.
- **success(data,textStatus,jqXHR)** is an optional *callback* function that executes when the response is received.
  - **data** holding the body of the response as a string.
  - **textStatus** holding the status of the request (i.e., "success").
  - **jqXHR** holding a jqXHR object, described shortly.
- **dataType** is an optional parameter to hold the type of data expected from the server.

AJAX
GET Requests – an example

```
$.get("/vote.php?option=C", function(data,textStatus,jsXHR) {
  if (textStatus=="success") {
     console.log("success! response is:" + data);
  }
  else {
     console.log("There was an error code"+jsXHR.status);
  }
  console.log("all done");
});
```

LISTING 15.13 jQuery to asynchronously get a URL and outputs when the response arrives

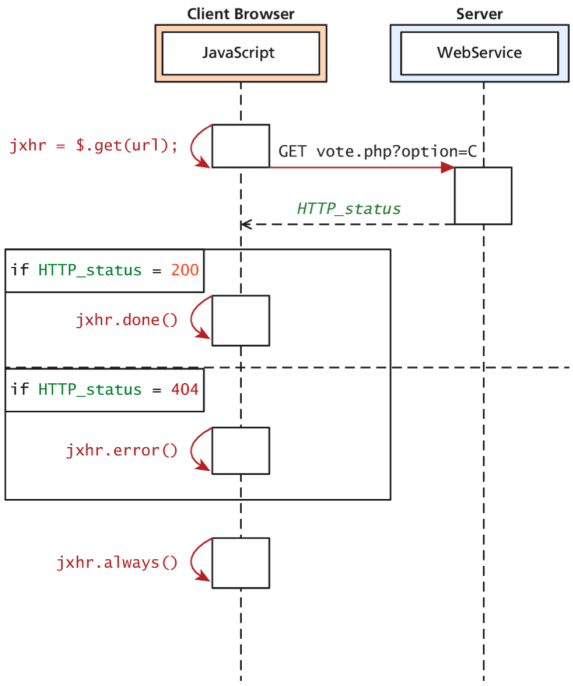
- All of the $.get() requests made by jQuery return a **jqXHR** object to encapsulate the response from the server.
- In practice that means the data being referred to in the callback from Listing 15.13 is actually an object with backward compatibility with `XMLHttpRequest`.

AJAX

jqXHR - XMLHttpRequest compatibility

- **abort()** stops execution and prevents any callback or handlers from receiving the trigger to execute.
- **getResponseHeader()** takes a parameter and gets the current value of that header.
- **readyState** is an integer from 1 to 4 representing the state of the request. The values include 1: sending, 3: response being processed, and 4: completed.
- **responseXML** and/or **responseText** the main response to the request.
- **setRequestHeader(name, value)** when used before actually instantiating the request allows headers to be changed for the request.
- **status** is the HTTP request status codes (200 = ok)
- **statusText** is the associated description of the status code.

jqXHR: Actually quite easy to use



- jqXHR objects have methods
  - done()
  - fail()
  - always()
- which allow us to structure our code in a more modular way than the inline callback

jqXHR: Very modular code

```
var jqxhr = $.get("/vote.php?option=C");

jqxhr.done(function(data) { console.log(data);});
jqxhr.fail(function(jqXHR) { console.log("Error: "+jqXHR.status); })
jqxhr.always(function() { console.log("all done"); });
```

LISTING 15.14 Modular jQuery code using the jqXHR object

POST requests: Via jQuery AJAX

- POST requests are often preferred to GET requests because one can post an unlimited amount of data, and because they do not generate viewable URLs for each action.
- GET requests are typically not used when we have forms because of the messy URLs and that limitation on how much data we can transmit.
- With POST it is possible to transmit files, something which is not possible with GET.

POST requests: Via jQuery AJAX

- The HTTP 1.1 definition describes GET as a **safe method** meaning that they should not change anything, and should only read data.
- POSTs on the other hand are not safe, and should be used whenever we are changing the state of our system (like casting a vote). get() method.
- POST syntax is almost identical to GET.
- jQuery.**post** (url [, data] [, success(data, textStatus, jqXHR)] [, dataType])

POST Requests: Via jQuery AJAX

- If we were to convert our vote casting code it would simply change the first line from
  - **var jqxhr = $.get("/vote.php?option=C");**
- to
  - **var jqxhr = $.post("/vote.php", "option=C");**

POST requests: `serialize()` will seriously help

- serialize() can be called on any **form** object to return its current key-value pairing as an & separated string, suitable for use with post().
  - **var postData = $("#voteForm").serialize();**
  - **$.post("vote.php", postData);**

Ajax: You Have Complete Control

- It turns out both the `$.get()` and `$.post()` methods are actually shorthand forms for the `jQuery().ajax()` method
  - The `ajax()` method has two versions. In the first it takes two parameters: a URL and a Plain Object, containing any of over 30 fields.
  - A second version with only one parameter is more commonly used, where the URL is but one of the key-value pairs in the Plain Object.

Ajax: More Verbose

- The one line required to post our form using get() becomes the more verbose code

```
$.ajax({ url: "vote.php",
         data: $("#voteForm").serialize(),
         async: true,
         type: post
});
```

LISTING 15.15 A raw AJAX method code to make a post

Ajax: You Have Complete Control

- To pass HTTP headers to the ajax() method, you enclose as many as you would like in a Plain Object. To illustrate how you could override User-Agent and Referer headers in the POST

```
$.ajax({ url: "vote.php",
  data: $("#voteForm").serialize(),
  async: true,
  type: post,
  headers: {"User-Agent" : "Homebrew JavaScript Vote Engine agent",
            "Referer": "http://funwebdev.com"
           }
});
```

LISTING 15.16 Adding headers to an AJAX post in jQuery