Web Programming

Lecture 21 - AJAX

Shamsa Abid

Agenda

- Intro to AJAX
- Hands-on

AJAX

- AJAX is not a programming language.
- AJAX is a technique for accessing web servers from a web page.
- AJAX stands for Asynchronous JavaScript And XML.

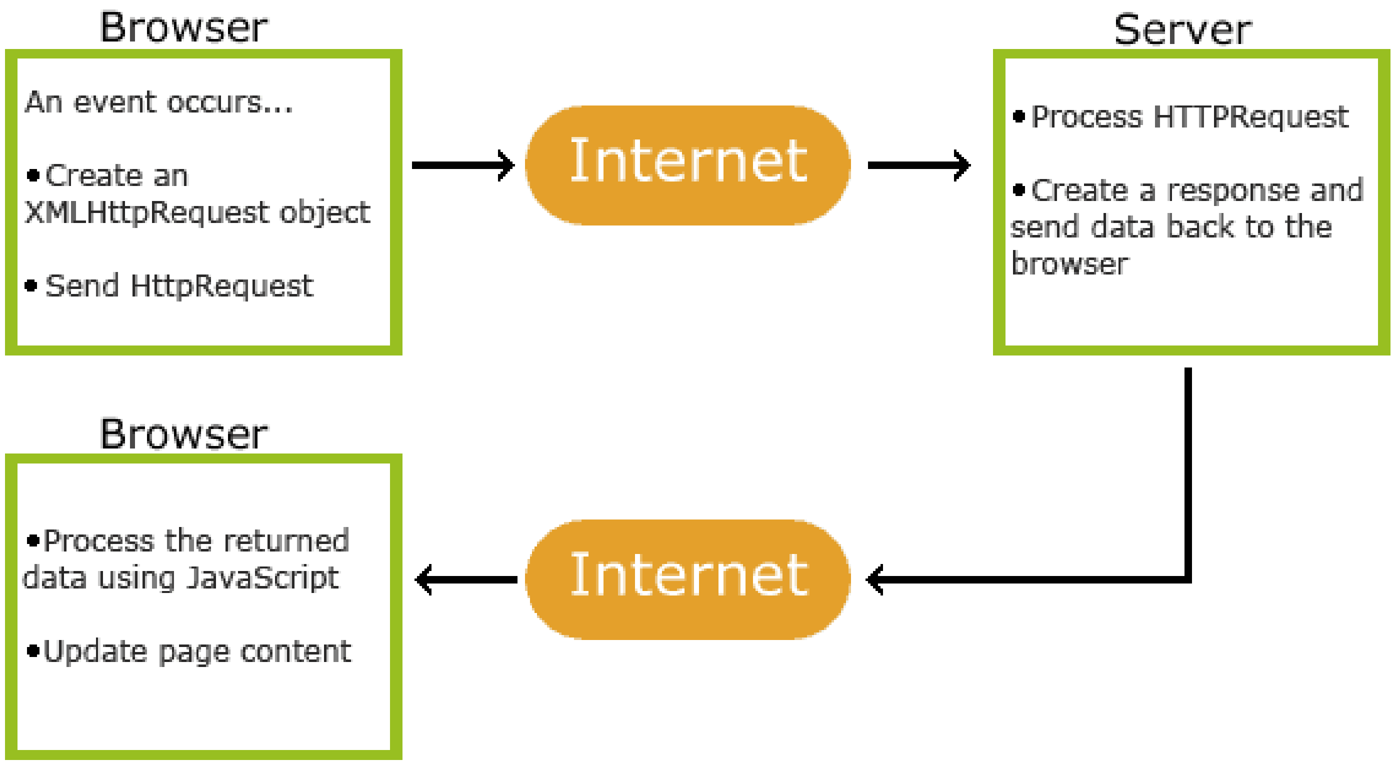
AJAX

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

AJAX

- AJAX just uses a combination of:
  - A browser built-in XMLHttpRequest object (to request data from a web server)
  - JavaScript and HTML DOM (to display or use the data)
- AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.
- AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

AJAX


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Internet
Browser
• Process the returned data using JavaScript
• Update page content

AJAX - The XMLHttpRequest Object

- All modern browsers support the XMLHttpRequest object.
- The XMLHttpRequest object can be used to exchange data with a server behind the scenes.
- This means that it is possible to update parts of a web page, without reloading the whole page.

Create an XMLHttpRequest Object

- var xhttp = new XMLHttpRequest();

Access Across Domains

- For security reasons, modern browsers do not allow access across domains.
- This means that both the web page and the XML file it tries to load, must be located on the same server.

XMLHttpRequest Object Methods

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

XMLHttpRequest Object Methods

Method	Description
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

XMLHttpRequest Object Properties

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

XMLHttpRequest Object Properties

Property	Description
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found"
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Send a Request To a Server

- To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:
-
```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Send a Request To a Server

Method	Description
open(*method, url, async*)	Specifies the type of request *method*: the type of request: GET or POST *url*: the server (file) location *async*: true (asynchronous) or false (synchronous)
send()	Sends the request to the server (used for GET)
send(*string*)	Sends the request to the server (used for POST)

GET or POST?

- GET is simpler and faster than POST, and can be used in most cases.
- However, always use POST requests when:
  - A cached file is not an option (update a file or database on the server).
  - Sending a large amount of data to the server (POST has no size limitations).
  - Sending user input (which can contain unknown characters), POST is more robust and secure than GET.

GET Requests

- A simple GET request:
- Example
- xhttp.open("GET", "demo_get.asp", true);
  xhttp.send();

GET Requests

- If you want to send information with the GET method, add the information to the URL:
- Example
  - xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);
    xhttp.send();

POST Requests

- A simple POST request:
- Example

-
  ```
  xhttp.open("POST", "demo_post.asp", true);
  xhttp.send();
  ```

POST Requests

- To POST data like an HTML form, add an HTTP header with setRequestHeader().
- Specify the data you want to send in the send() method:
- Example
  - xhttp.open("POST", "demo_post2.asp", true);
    xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhttp.send("fname=Henry&lname=Ford");

POST Requests

Method	Description
setRequestHeader(*header, value*)	Adds HTTP headers to the request *header*: specifies the header name *value*: specifies the header value

The url - A File On a Server

- The url parameter of the open() method, is an address to a file on a server:
- `xhttp.open("GET", "ajax_test.asp", true);`
- The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

Asynchronous - True or False?

- Server requests should be sent asynchronously.
- The async parameter of the open() method should be set to true:
- `xhttp.open("GET", "ajax_test.asp", true);`
- By sending asynchronously, the JavaScript does not have to wait for the server response, but can instead:
  - execute other scripts while waiting for server response
  - deal with the response after the response is ready

The onreadystatechange Property

- With the XMLHttpRequest object you can define a function to be executed when the request receives an answer.
- The function is defined in the **onreadystatechange** property of the XMLHttpResponse object:
- Example
-
```
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("demo").innerHTML = this.responseTe
xt;
  }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Synchronous Request

- To execute a synchronous request, change the third parameter in the open() method to false:
- xhttp.open("GET", "ajax_info.txt", false);
- Sometimes async = false are used for quick testing. You will also find synchronous requests in older JavaScript code.
- Since the code will wait for server completion, there is no need for an onreadystatechange function:
- Example

  -
    ```
    xhttp.open("GET", "ajax_info.txt", false);
    xhttp.send();
    document.getElementById("demo").innerHTML =
    xhttp.responseText;
    ```

Synchronous Request

- Synchronous XMLHttpRequest (async = false) is not recommended because the JavaScript will stop executing until the server response is ready. If the server is busy or slow, the application will hang or stop.
- Synchronous XMLHttpRequest is in the process of being removed from the web standard, but this process can take many years.
- Modern developer tools are encouraged to warn about using synchronous requests and may throw an InvalidAccessError exception when it occurs.

Server Response

- The onreadystatechange Property
  - The **readyState** property holds the status of the XMLHttpRequest.
  - The **onreadystatechange** property defines a function to be executed when the readyState changes.
  - The **status** property and the **statusText** property holds the status of the XMLHttpRequest object.

Server Response

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")

Server Response

- The onreadystatechange function is called every time the readyState changes.
- When readyState is 4 and status is 200, the response is ready:
- Example

  -
    ```
    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();
    }
    ```

Using a Callback Function

- A callback function is a function passed as a parameter to another function.
- If you have more than one AJAX task in a website, you should create one function for executing the XMLHttpRequest object, and one callback function for each AJAX task.
- The function call should contain the URL and what function to call when the response is ready.

Using a Callback Function

-
  ```
  loadDoc("url-1", myFunction1); // AJAX Task 1

  loadDoc("url-2", myFunction2); // AJAX Task 2
  function loadDoc(url, cFunction) {
    var xhttp;
    xhttp=new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        cFunction(this);
      }
   };
    xhttp.open("GET", url, true);
    xhttp.send();
  }

  function myFunction1(xhttp) {
    // action goes here
  }
  function myFunction2(xhttp) {
    // action goes here
  }
  ```

Server Response Properties

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

Server Response Methods

Method	Description
getResponseHeader()	Returns specific header information from the server resource
getAllResponseHeaders()	Returns all the header information from the server resource

The responseText Property

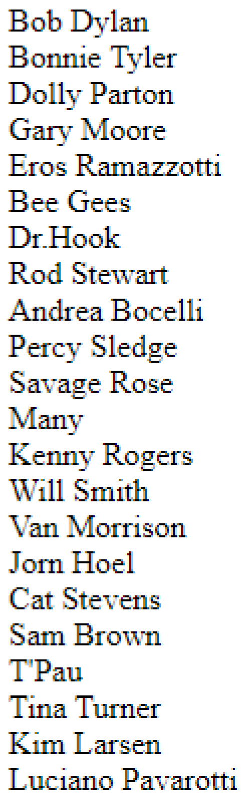
- The **responseText** property returns the server response as a JavaScript string, and you can use it accordingly:
- Example
  - document.getElementById("demo").innerHTML = xhttp.responseText;

The responseXML Property

- The XML HttpRequest object has an in-built XML parser.
- The **responseXML** property returns the server response as an XML DOM object.
- Using this property you can parse the response as an XML DOM object:
- Example
  - Request the file cd_catalog.xml and parse the response:

  -
    ```
    xmlDoc = xhttp.responseXML;
    txt = "";
    x = xmlDoc.getElementsByTagName("ARTIST");
    for (i = 0; i < x.length; i++) {
      txt += x[i].childNodes[0].nodeValue + "<br>";
      }
    document.getElementById("demo").innerHTML = txt;
    xhttp.open("GET", "cd_catalog.xml", true);
    xhttp.send();
    ```

output



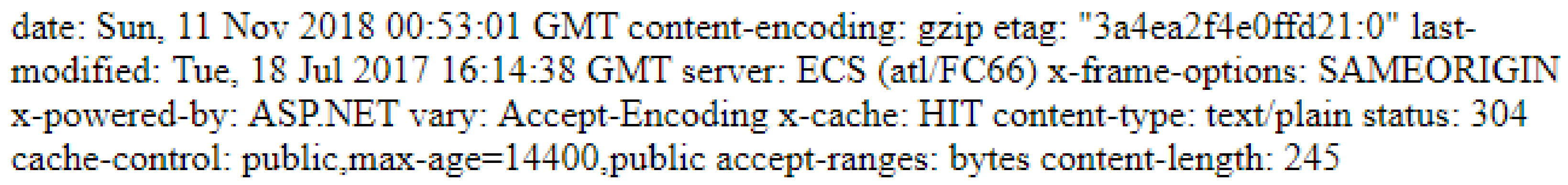
```
Bob Dylan
Bonnie Tyler
Dolly Parton
Gary Moore
Eros Ramazzotti
Bee Gees
Dr.Hook
Rod Stewart
Andrea Bocelli
Percy Sledge
Savage Rose
Many
Kenny Rogers
Will Smith
Van Morrison
Jorn Hoel
Cat Stevens
Sam Brown
T'Pau
Tina Turner
Kim Larsen
Luciano Pavarotti
```

The getAllResponseHeaders() Method

- The **getAllResponseHeaders()** method returns all header information from the server response.
- Example

  -
    ```
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("demo").innerHTML =
        this.getAllResponseHeaders();
      }
    };
    ```

The getAllResponseHeaders() Method



```
date: Sun, 11 Nov 2018 00:53:01 GMT content-encoding: gzip etag: "3a4ea2f4e0ffd21:0" last-
modified: Tue, 18 Jul 2017 16:14:38 GMT server: ECS (atl/FC66) x-frame-options: SAMEORIGIN
x-powered-by: ASP.NET vary: Accept-Encoding x-cache: HIT content-type: text/plain status: 304
cache-control: public,max-age=14400,public accept-ranges: bytes content-length: 245
```

The getResponseHeader() Method

- this.getResponseHeader("Last-Modified");

HANDS-ON

The Purpose of Ajax

- Consider a simple registration form.
- You have very likely experienced the frustration of having to try multiple usernames when registering for some new website.
- You fill out the entire form, hit the submit button, wait for a second or so, and then get the same form right back with a message saying that the username you have chosen is not available.
- You try another easy-to-remember username and find it is also not available.
- You repeat this several times until finally you pick some obscure username.
- This process wouldn't be nearly as bad if you didn't have to wait for the entire page to refresh each time you tried a new username.

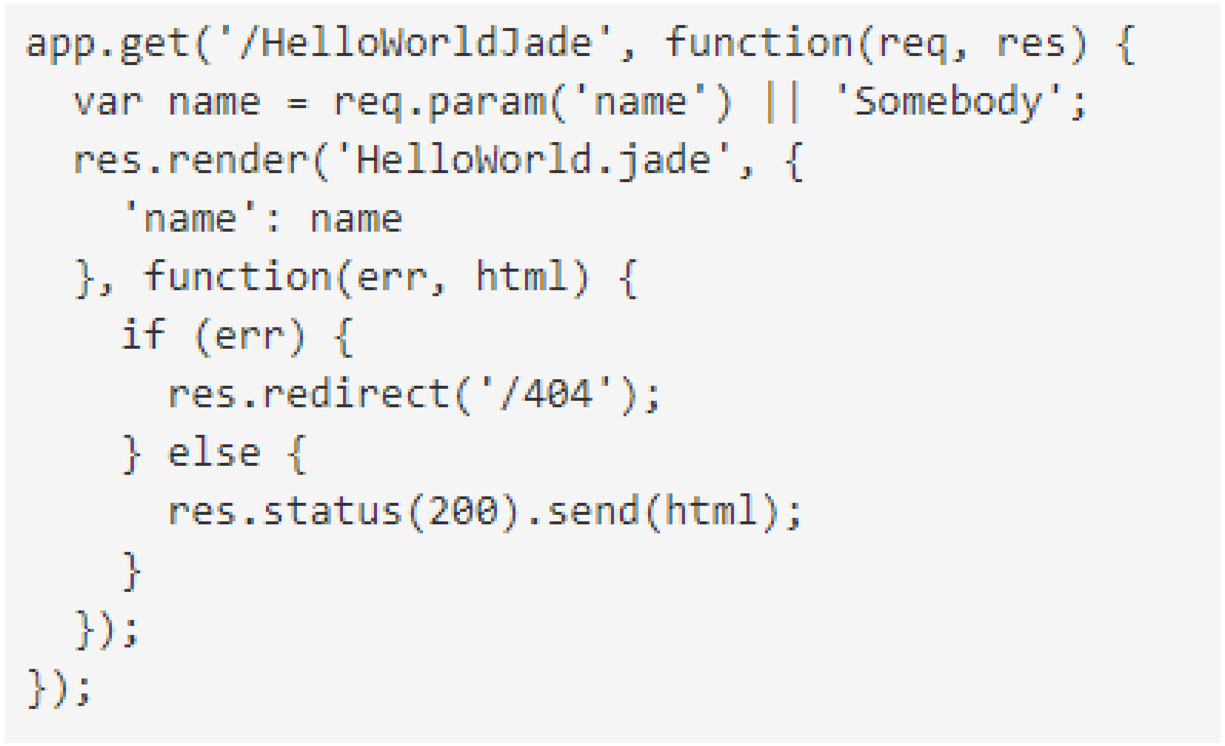
A First Node.js Application & Server

- IntroAjaxNodeJsServer/Demos/
- Npm install
- Npm start
- browse to http://localhost:8080 to view files served by and response routes defined by this application.
- http://localhost:8080/HelloWorld
- http://localhost:8080/HelloWorld?name=Jane

A First Node.js Application & Server

```
app.get('/HelloWorld', function(req, res) {
  var name = req.param('name') || 'Somebody';
  var respondWith = '<?xml version="1.0" encoding="UTF-8"?>';
  respondWith += "<h1>Hello " + name + "!</h1>";
  res.status(200);
  res.setHeader('Content-type', 'text/xml');
  return res.send(respondWith);
});
```

We send back XML, encoded as UTF-8, with a single `h1` tag containing Hello [name].
The very last line, return res.send(respondWith); sends the response back to the browser.

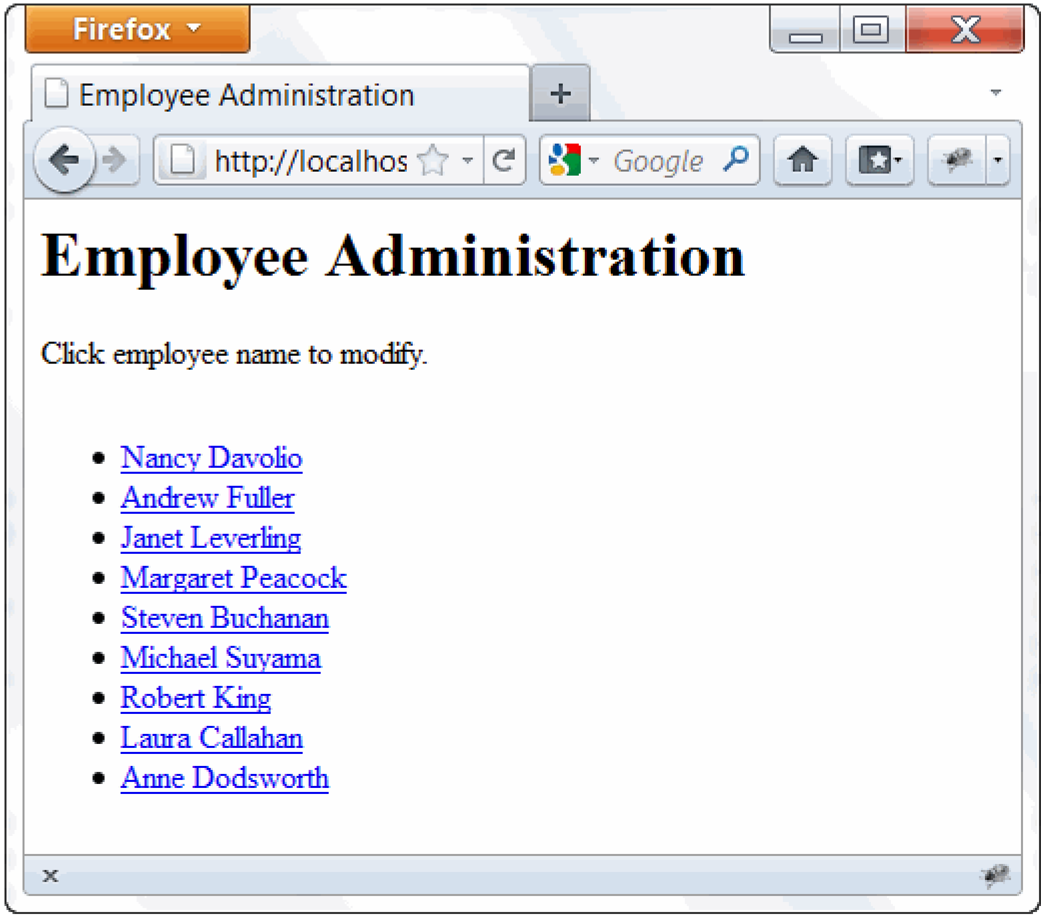


```
app.get('/HelloWorldJade', function(req, res) {
  var name = req.param('name') || 'Somebody';
  res.render('HelloWorld.jade', {
    'name': name
  }, function(err, html) {
    if (err) {
      res.redirect('/404');
    } else {
      res.status(200).send(html);
    }
  });
});
```

Browse to http://localhost:8080/HelloWorldJade

An Ajax Web Application

- AjaxBasics/Solutions/
- Npm install
- Npm start
- Browse to http://localhost:8080 /EmployeeAdmin.html



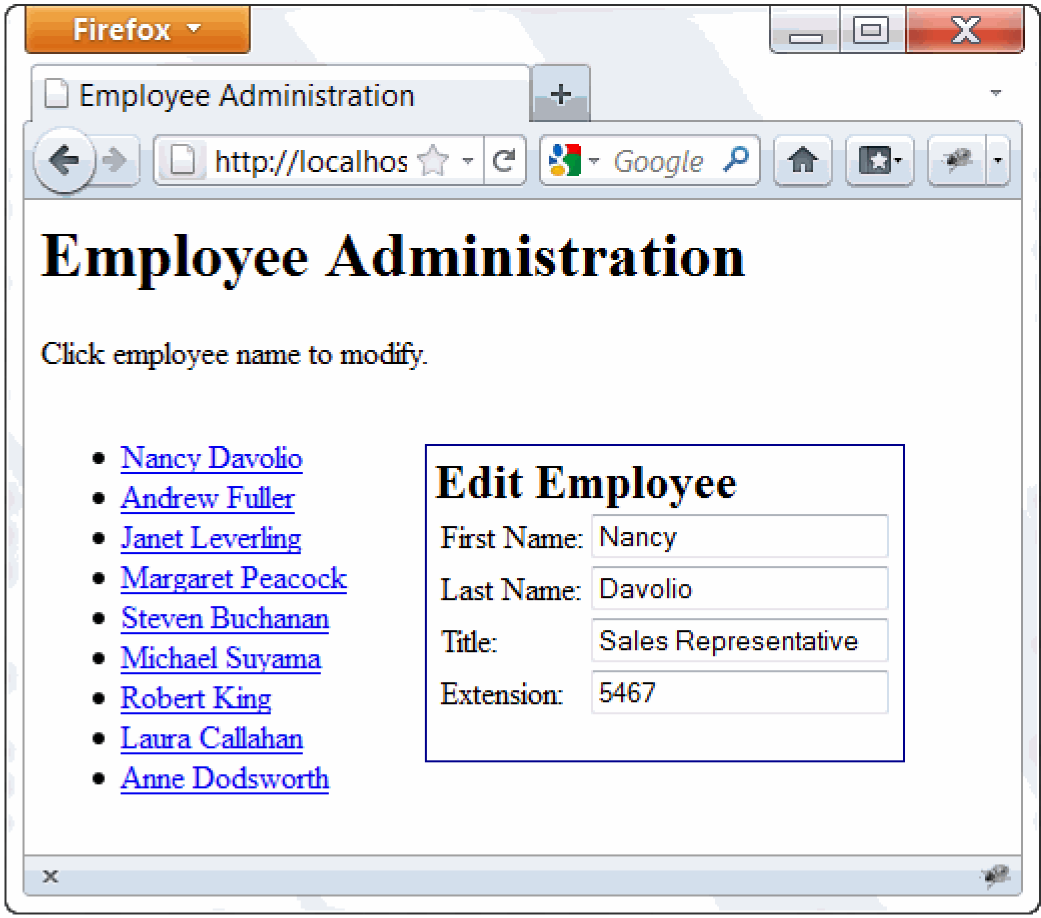
Employee Administration

Click employee name to modify.

- Nancy Davolio
- Andrew Fuller
- Janet Leverling
- Margaret Peacock
- Steven Buchanan
- Michael Suyama
- Robert King
- Laura Callahan
- Anne Dodsworth

An Ajax Web Application

- When the user chooses an employee, the page doesn't reload.
- Instead, an Ajax call to the server is made, the server returns the HTML form and JavaScript is used to display the form on the page



Employee Administration

Click employee name to modify.

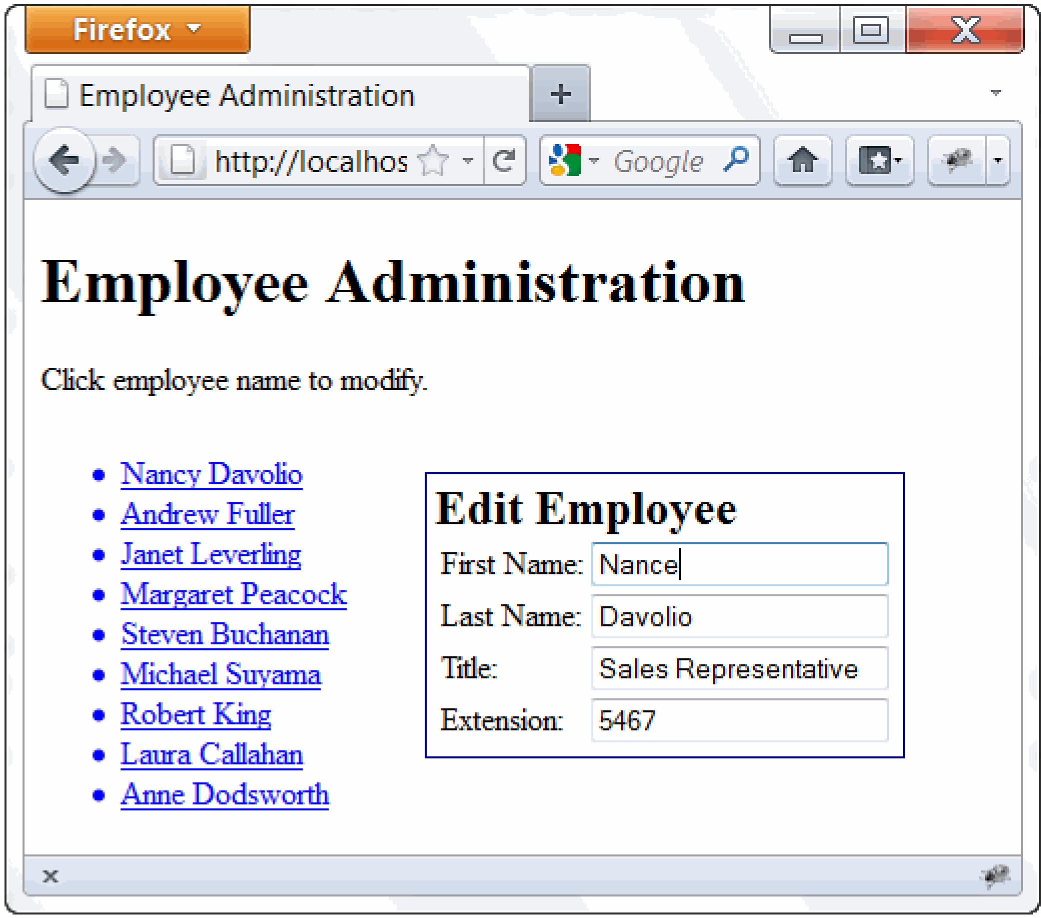
- Nancy Davolio
- Andrew Fuller
- Janet Leverling
- Margaret Peacock
- Steven Buchanan
- Michael Suyama
- Robert King
- Laura Callahan
- Anne Dodsworth

Edit Employee

First Name:	Nancy
Last Name:	Davolio
Title:	Sales Representative
Extension:	5467

An Ajax Web Application

- The user can then edit the form fields.
- Each time a changed form field loses focus the change is sent to the server with Ajax, the server-side code updates the data in the database and sends back the updated list of employees to the browser.
- JavaScript is again used to display the updated list on the page.
- The screenshot shows the state after the user has changed the field but before she has tabbed out



Employee Administration

Click employee name to modify.

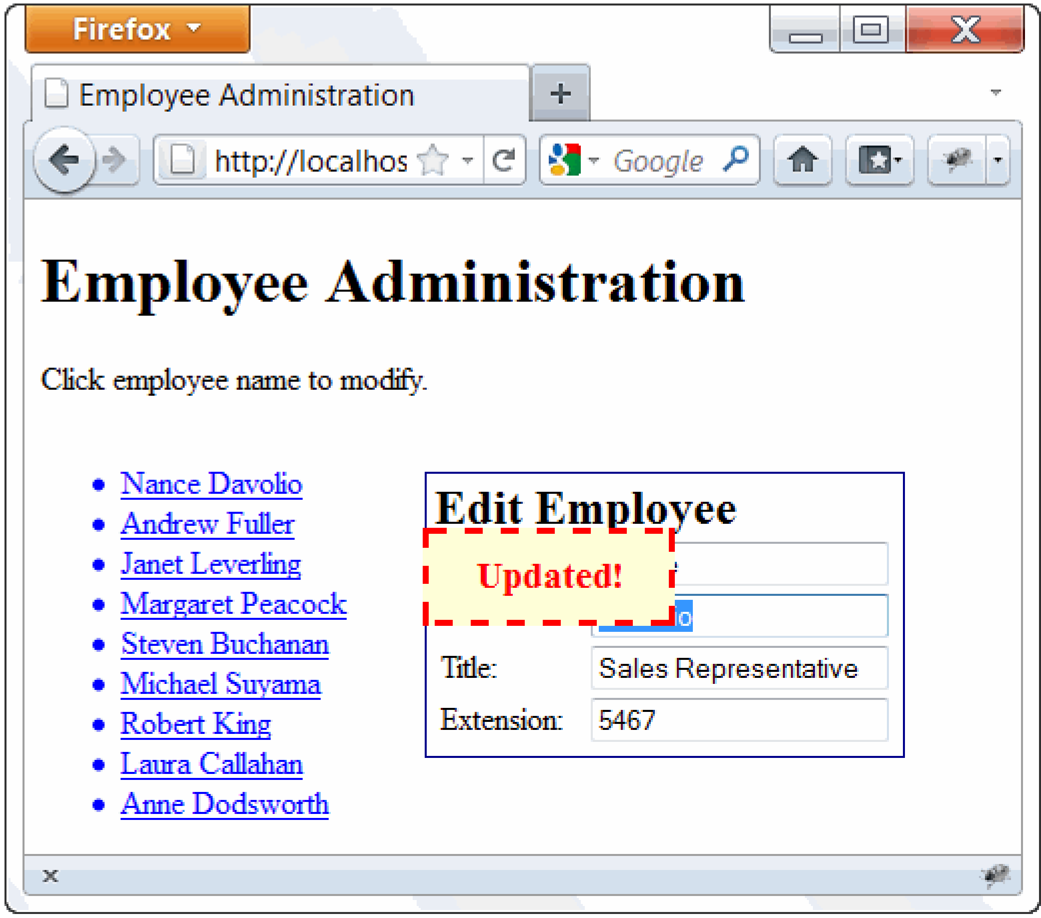
- Nancy Davolio
- Andrew Fuller
- Janet Leverling
- Margaret Peacock
- Steven Buchanan
- Michael Suyama
- Robert King
- Laura Callahan
- Anne Dodsworth

Edit Employee

First Name:	Nance
Last Name:	Davolio
Title:	Sales Representative
Extension:	5467

An Ajax Web Application

- The employee list is updated to reflect the changes made. Notice "Nancy" has been changed to "Nance" in the list. A message is displayed letting the user know that the database was updated
- All this is accomplished without having to do even one complete refresh. This is the major benefit of Ajax.



Employee Administration

Click employee name to modify.

- Nance Davolio
- Andrew Fuller
- Janet Leverling
- Margaret Peacock
- Steven Buchanan
- Michael Suyama
- Robert King
- Laura Callahan
- Anne Dodsworth

Edit Employee

Updated!

Title:	Sales Representative
Extension:	5467

The XMLHttpRequest Object

- AjaxBasics/Demos/CreateXMLHttpRequest.html
- This code attempts to create an XMLHttpRequest object using the XMLHttpRequest() constructor.
- If it succeeds, it writes out "Using XMLHttpRequest Object" to the body of the page. If it fails, it writes out "XMLHttp cannot be created!"

Using an XMLHttpRequest Object

- xmlhttp.open("GET","Demo.xml",true);

Request Types

- The most commonly supported (and used) methods are GET, POST and HEAD
- HEAD
  - The HEAD method is the least commonly used of the three; however, for simple requests, it can be all you need.
  - It simply returns the meta-information contained in the HTTP headers.
  - The call would look like this:
    - xmlhttp.open("HEAD","Demo",true);
  - And the response might look like this:
    - Date: Wed, 11 May 2011 15:46:30 GMT X-Powered-By: ASP.NET Content-Length: 63 Last-Modified: Tue, 10 May 2011 19:12:27 GMT Server: Microsoft-IIS/7.5 ETag: "712b13346fcc1:0" Content-Type: text/xml Accept-Ranges: bytes
  - The XMLHttpRequest request is sent as follows:
    - xmlhttp.send(null);

Request Types

- GET
  - The GET method is used to send information to the server as part of the URL. The server returns the same header information that the HEAD method returns, but it also returns the body of the message (i.e, the content of the page).
  - Any name-value pairs to be processed by the receiving page should be passed along the querystring.
  - The call would look like this:
    - `xmlhttp.open("GET","Demo?FirstName=Nat&LastName=Dunn",true);`
  - The response would be the same as the response shown for the HEAD method followed by the message body, which would typically be simple text, JSON, HTML or XML.
  - Again, the XMLHttpRequest request is sent as follows:
    - `xmlhttp.send(null);`

Request Types

- POST
  - The POST method is used to send information as an enclosed entity.
  - The call would look like this:
    - xmlhttp.open("POST","Demo",true);
  - The response header is somewhat different in that it specifies that the returned content is not cacheable. Like with GET, the message body would typically be plain text, HTML or XML.
  - The XMLHttpRequest request is sent as follows:
    - xmlhttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded;charset=UTF-8"); xmlhttp.send("FirstName=Nat&LastName=Dunn");
  - As you can see, with POST, we first need to set the content type to "application/x-www-form-urlencoded;charset=UTF-8". This tells the server to expect form data.
  - In the send() method, we include name-value pairs. These name-value pairs are available to the receiving page for processing.
  - Data cannot be sent in this manner with the HEAD and GET methods, which is why null was passed in the previous examples.

Handling the Response

- When using asynchronous calls, we cannot be sure when the response will come, so we must write code that waits for the response and handles it when it arrives. We do this with a *callback function*.
- Callback functions are functions that are triggered by some event. In our case, the event we are looking for is a change in the state of the xmlhttp response.
- The xmlhttp object's readyState property holds the current state of the response.
- There are five possible states (0-4).
- Browsers do not necessarily inform you of all states; states 0 and 3 in particular may not appear when you run the demo file.

Values of the readyState Property

State	Description
0	uninitialized
1	loading
2	loaded
3	interactive
4	complete

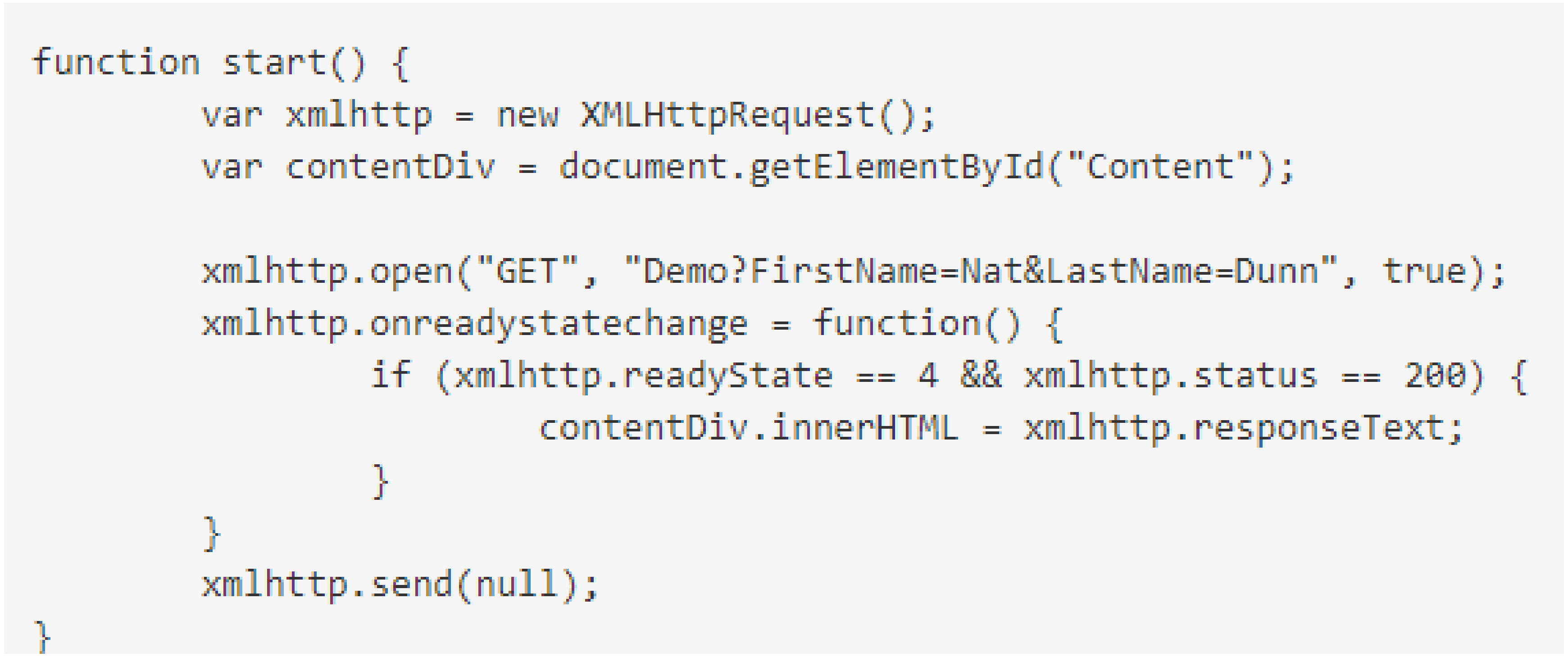
Example

- AjaxBasics/Demos/ReadyStateChange.html
- In practice, before doing anything with the xmlhttp response data, we want to make sure the readyState is complete (4), so we put a condition inside our function to check for this:

```
xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4) {
                //Do something here
        }
}
```

Example

- A common application is to check the status property to make sure that the request was successful (200) and then to output the message body to a div on the HTML page.
- To run the demo, first start the Node.js server:
- Navigate to the directory AjaxBasics/Demos/.
- Type npm install.
- Type npm start to start the Node.js server.
- With the server started, you can then browse to http://localhost:8080/UsingXMLHttpRequest-Get.html to view the page.



```
function start() {
        var xmlhttp = new XMLHttpRequest();
        var contentDiv = document.getElementById("Content");

        xmlhttp.open("GET", "Demo?FirstName=Nat&LastName=Dunn", true);
        xmlhttp.onreadystatechange = function() {
                if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                        contentDiv.innerHTML = xmlhttp.responseText;
                }
        }
        xmlhttp.send(null);
}
```

This page simply "copies" the response text (xmlhttp.responseText) and "pastes" it into the "Content" div on the page

Ajax Using the POST Method

```
function start() {
        var xmlhttp = new XMLHttpRequest();
        var contentDiv = document.getElementById("Content");

        xmlhttp.open("POST", "Demo", true);
        xmlhttp.onreadystatechange = function() {
                if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                        contentDiv.innerHTML = xmlhttp.responseText;
                }
        }
        xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8");
        xmlhttp.send("FirstName=Nat&LastName=Dunn");</strong>
}
```

We specify "POST" as the method for our Ajax call in xmlhttp.open and, in xmlhttp.setRequestHeader specify the MIME type (application/x-www-form-urlencoded) and character encoding (charset=UTF-8) of the request we send to the /Demo response route.
We pass along two POST variables, FirstName and LastName, with values Nat and Dunn, respectively.

Get all response headers

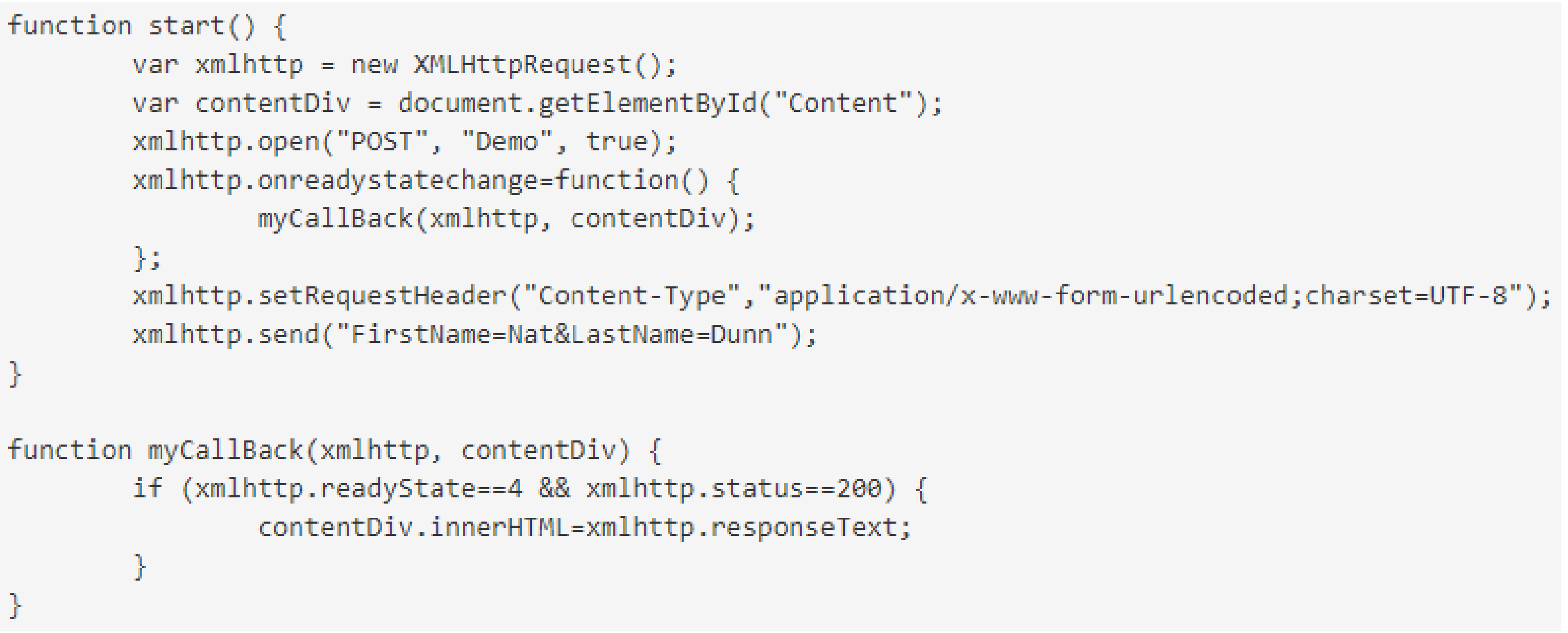
```
function start() {
        var xmlhttp = new XMLHttpRequest();
        var contentDiv = document.getElementById("Content");

        xmlhttp.open("POST", "Demo", true);
        xmlhttp.onreadystatechange = function() {
                if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                        contentDiv.innerHTML = xmlhttp.responseText;
                        var headers = xmlhttp.getAllResponseHeaders();
                        contentDiv.innerHTML += headers.replace(/\n/g, "<br>");
                }
        }
        xmlhttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded;charset=UTF-8");
        xmlhttp.send("FirstName=Nat&LastName=Dunn");
}
```

We use innerHTML to set the contents of #contentDiv, retrieving the response headers with xmlhttp.getAllResponseHeaders() and displaying with newlines converted to HTML <br /> tags.

The Callback Function

- Option 1



```
function start() {
        var xmlhttp = new XMLHttpRequest();
        var contentDiv = document.getElementById("Content");
        xmlhttp.open("POST", "Demo", true);
        xmlhttp.onreadystatechange=function() {
                myCallBack(xmlhttp, contentDiv);
        };
        xmlhttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded;charset=UTF-8");
        xmlhttp.send("FirstName=Nat&LastName=Dunn");
}

function myCallBack(xmlhttp, contentDiv) {
        if (xmlhttp.readyState==4 && xmlhttp.status==200) {
                contentDiv.innerHTML=xmlhttp.responseText;
        }
}
```

The Callback Function

- Option 2

```
function start() {
        var xmlhttp = new XMLHttpRequest();
        var contentDiv = document.getElementById("Content");
        xmlhttp.open("POST", "Demo", true);
        xmlhttp.onreadystatechange=myCallBack;
        xmlhttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded;charset=UTF-8");
        xmlhttp.send("FirstName=Nat&LastName=Dunn");

        function myCallBack() {
                if (xmlhttp.readyState==4 && xmlhttp.status==200) {
                        contentDiv.innerHTML=xmlhttp.responseText;
                }
        }
}
```

References

- https://www.w3schools.com/xml/ajax_intro.asp
- https://www.webucator.com/tutorial/learn-ajax/intro-ajax-the-nodejs-server.cfm

- Code Download Link: http://www.webucator.com/class-files/index.cfm?CourseID=JSC401

Geoffrey Okongo, Built 3 Plugins in JavaScript for iQuery

Answered May 9, 2017

Ajax is a technology that enables fetching/sending data from/to the server **without requiring a page reload**. *Angularjs* is a framework that implements Ajax technology to make building **Single Page Applications** a walk in the park.