Instructor Notes:

Add instructor notes here.

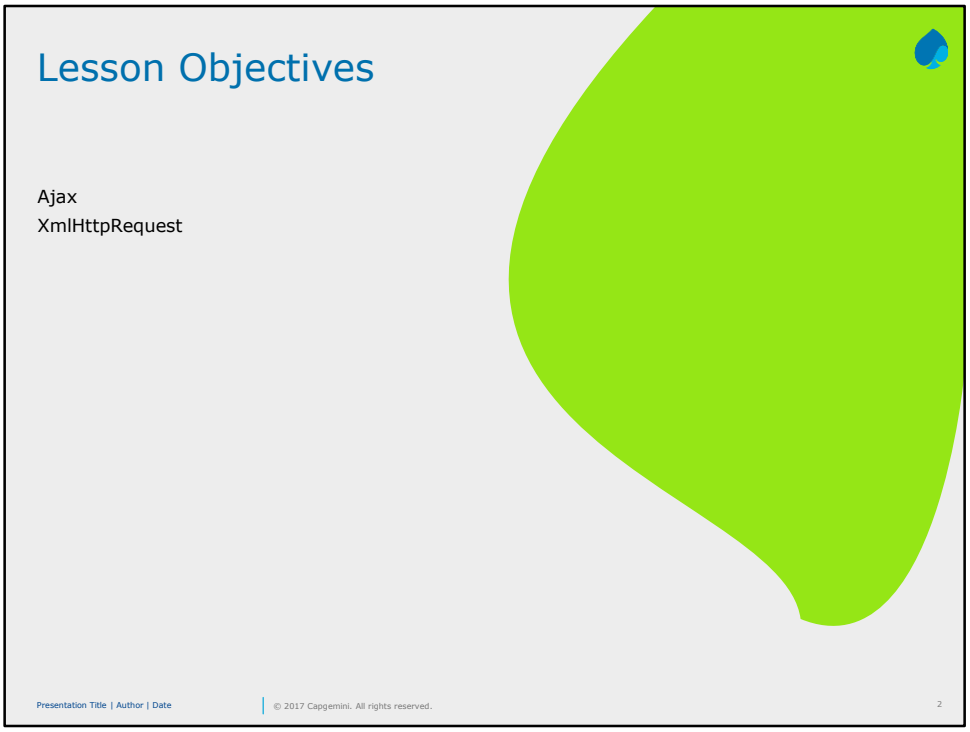
JavaScript ES6

Lesson 11-Ajax Developmement

Capgemini

Instructor Notes:

Add instructor notes here.



Following contents would be covered:

1.1 : What are Web services

1.1.1 Web service components and architecture

1.1.2 How do Web services work

1.2: HTTP and SOAP messages

1.3: Overview of JAX – WS and JAX – RS

Instructor Notes:

Specify the clear purpose of the Ajax. It is used to add Asynchronous behavior to web applications.

10.3. Working with JSON Object

AJAX

- "Asynchronous JavaScript And XML"
- AJAX is not a programming language, but a technique for making the user interfaces of web applications more responsive and interactive
- It provide a simple and standard means for a web page to communicate with the server without a complete page refresh.

Asynchronous JavaScript and XML (Ajax) is a technique for making the user interfaces of web applications more responsive and interactive.
If you've surfed the web at all lately, most likely you've seen Ajax in action without realizing it. Ajax can help increase the speed and usability of an application's web pages by updating only part of the page at a time, rather than requiring the entire page to be reloaded after a user-initiated change. Through the power of Ajax, the pages of your application can exchange small amounts of data with the server without going through a form submit. The Ajax technique accomplishes this by using the following technologies:

JavaScript that allows for interaction with the browser and responding to events
The DOM for accessing and manipulating the structure of the HTML of the page
XML, which represents the data passed between the server and client.
An XMLHttpRequest object for asynchronously exchanging the XML data between the client and the server.

Instructor Notes:

10.3. Working with AJAX

Why AJAX?

Intuitive and natural user interaction

- No clicking required. Call can be triggered on any event
- Mouse movement is a sufficient event trigger

"Partial screen update" replaces the "click, wait, and refresh" user interaction model

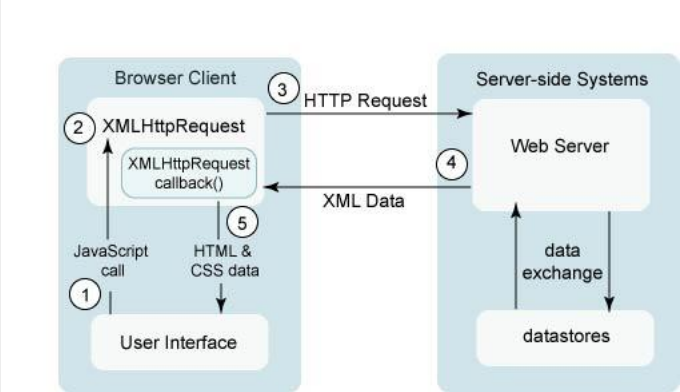
- Only user interface elements that contain new information are updated (fast response)

The rest of the user interface remains displayed as it is without interruption (no loss of operational context)

Instructor Notes:

10.3. Working with AJAX

Introduction to How Does Ajax works

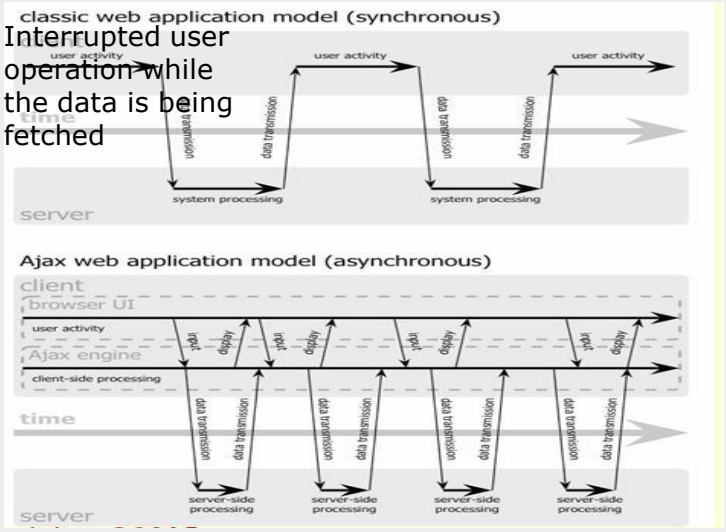


Instructor Notes:

.

10.3. Working with AJAX

Introduction to How Does Ajax works



Interrupted user operation while the data is being fetched

Uninterrupted user operation while data is being fetched

The user generates an event, such as by clicking a button. This results in a JavaScript call.
An XMLHttpRequest object is created and configured with a request parameter that includes the ID of the component that generated the event and any value that the user might have entered.
The XMLHttpRequest object makes an asynchronous request to the web server. An object (such as a servlet or listener) receives the request, processes it, and stores any data in the request to the data store. In the case of Ajax-aware JavaServer Faces components, the object that processes the request is a PhaseListener object. We'll cover that more later in the document.
The object that processed the request returns an XML document containing any updates that need to go to the client.
The XMLHttpRequest object receives the XML data, processes it, and updates the HTML DOM representing the page with the new data.

Instructor Notes:

.

10.3. Working with AJAX

Introduction to How Does Ajax works

JavaScript.

- Loosely typed scripting language
- Allows programmatic interaction with the browser's capabilities
- JavaScript function is called when an event in a page occurs

DOM:

- API for accessing and manipulating structured documents.
- Represents the structure of XML and HTML documents

Instructor Notes:

10.3. Working with AJAX

Introduction to How Does Ajax works

CSS

- Allows for a clear separation of the presentation from the content and may be changed programmatically by JavaScript

HTTP

- XMLHttpRequest

XML

- which represents the data passed between the server and client

Instructor Notes:

10.3. Working with AJAX

Introduction to Ajax

JavaScript object
- Created within a JavaScript function

XMLHttpRequest object for asynchronously exchanging the XML data between the client and the server

Communicates with a server via standard HTTP GET/POST

XMLHttpRequest object works in the background
- Does not interrupt user operation

Instructor Notes:

10.3. Working with AJAX

XML with Ajax works

open("method", "URL", syn/asyn)
- Assigns destination URL, method, mode

send(content)
- Sends request including string or DOM object data

abort()
- Terminates current request

The *open()* Method
You call the open(DOMString method, DOMString uri, boolean async, DOMString username, DOMString password) method to initialize an XMLHttpRequest object. The method parameter is required and specifies the HTTP method (GET, POST, PUT, DELETE, or HEAD) that you want to use to send the request. To send data to the server, use the POST method. To retrieve data from the server, use the GET method. The uri parameter specifies the server URI to which the XMLHttpRequest object sends the request. The uri resolves to an absolute URI using the window.document.baseURI property—in other words, you can use relative URIs which will be resolved in the same way that the browser resolves relative URIs. The async parameter specifies whether the request is asynchronous; the default value is true. To send a synchronous request, set the parameter to false. For servers that require authentication, you can supply the optional username and password parameters. After calling the open() method, the XMLHttpRequest objects sets its readyState property to 1 (Open) and resets the responseText, responseXML, status, and statusText properties to their initial values. It also resets the request headers. Note that the object resets these values if you call the open() method when readyState is 4.

The *send()* Method
After preparing a request by calling the open() method, you send the request to the server. You may call send() only when the readyState value is 1, otherwise the XMLHttpRequest object raises an exception. The request gets sent to the server using the parameters supplied to the open() method. The send() method returns immediately when the async parameter is true, letting other client script processing continue. The XMLHttpRequest object sets the readyState value to 2 (Sent) after the send() method has been called. When the server responds, before receiving the message body, if any, the XMLHttpRequest object sets readyState to 3 (Receiving). When the request has completed loading it sets readyState to 4 (Loaded). For a request of type HEAD, it sets the readyState value to 4 immediately after setting it to 3.

The send() method takes an optional parameter that may contain data of varying types. Typically, you use this to send data to the server using the POST method. You can explicitly invoke the send() method with null, which is the same as invoking it with no argument. For most other data types, set the Content-Type header using the setRequestHeader() method (explained below) before invoking the send() method. If the data parameter in the send(data) method is of type DOMString, encode the data as UTF-8. If data is of type Document, serialize the data using the encoding specified by data.xmlEncoding, if supported or UTF-8 otherwise.

Instructor Notes:

10.3. Working with AJAX

XML with Ajax works

getAllResponseHeaders()
- Returns headers (labels + values) as a string

getResponseHeader("header")
- Returns value of a given header

setRequestHeader("label","value")
- Sets Request Headers before sending

The *setRequestHeader()* Method
The setRequestHeader(DOMString header, DOMString value) method sets request headers. You may call this method after calling the open() method—when the readyState value is 1—otherwise you'll get an exception.

The *getResponseHeader()* Method
You use getResponseHeader(DOMString header, value) method used to retrieve response header values. Call getResponseHeader() only when the readyState value is 3 or 4 (in other words, after the response headers are available); otherwise, the method returns an empty string.

The *getAllResponseHeaders()* Method
The getAllResponseHeaders() method returns all the response headers as a single string with each header on a separate line. The method returns null if readyState value is not 3 or 4.

Instructor Notes:

10.3. Working with AJAX

XMLHTTPRequest

onreadystatechange

- Event handler that fires at each state change
- You implement your own function that handles this

readyState values – current status of request

- 0 = uninitialized
- 1 = loading
- 2 = loaded
- 3=interactive (some data has been returned)
- 4=complete

Status

- HTTP Status returned from server: 200 = OK

Instructor Notes:

10.3. Working with AJAX

XmlHTTPRequest

responseText
- String version of data returned from server

responseXML
- XML DOM document of data returned

statusText
- Status text returned from server

The *responseText* Property
The responseText property contains the text of the HTTP response received by the client. When the readyState value is 0, 1, or 2 responseText contains an empty string. When the readyState value is 3 (Receiving), the response contains the incomplete response received by the client. When readyState is 4 (Loaded) the responseText contains the complete response.

The *responseXML* Property
The responseXML property represents the XML response when the complete HTTP response has been received (when readyState is 4), when the Content-Type header specifies the MIME (media) type as text/xml, application/xml, or ends in +xml. If the Content-Type header does not contain one of these media types, the responseXML value is null. The responseXML value is also null whenever the readyState value contains any value other than 4.

The responseXML property value is an object of type Document interface, and represents the parsed document. If the document cannot be parsed (for example if the document is malformed or the character encoding of the document is not supported) the responseXML value is null.

The *status* Property
The status property represents the HTTP status code and is of type short. The status attribute is available only when the readyState value is 3 (Receiving) or 4 (Loaded). Attempting to access the status value when readyState is less than 3 raises an exception.

The *statusText* Property
The statusText attribute represents the HTTP status code text and is also available only when the readyState value is 3 or 4. Attempting to access the statusText property for other readyState values raises an exception.

Instructor Notes:

Introduction to Ajax

States of XMLHttpRequest

ReadyState Value	Description
0	Represents an "uninitialized" state in which an XMLHttpRequest object has been created, but not initialized.
1	Represents a "loading" state in which code has called the XMLHttpRequest open() method and the XMLHttpRequest is ready to send a request to the server.
2	Represents a "sent or loaded" state in which a request has been sent to the server with the send () method, but a response has not yet been received.
3	Represents a "receiving or interactive" state in which the HTTP response headers have been received, but message body has not yet been completely received.
4	Represents a "loaded or complete" state in which the response has been completely received.

The XMLHttpRequest object fires a readystatechange event whenever the readyState value changes.
The onreadystatechange property accepts an Event Listener value, specifying the method that the object will invoke whenever the readyState value changes.

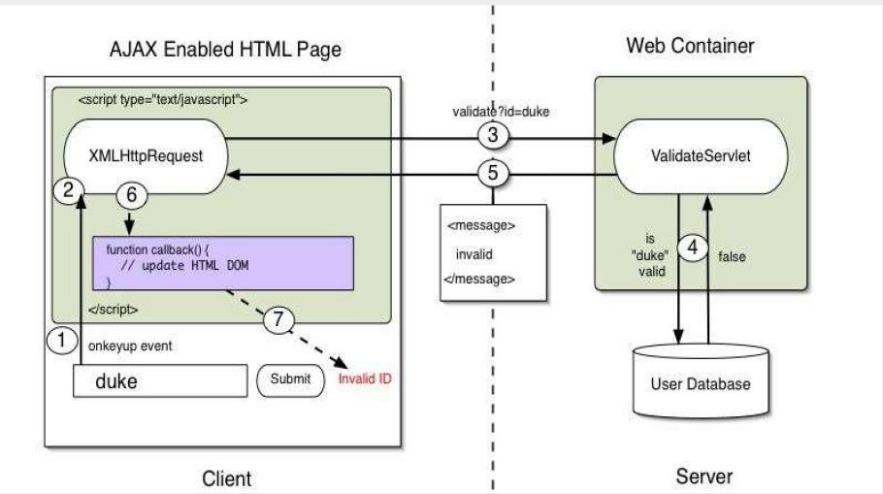
Instructor Notes:

Demo

Instructor Notes:

XMLHttpRequest Object

Ajax Interaction using XMLHttpRequest



Instructor Notes:

XMLHttpRequest Object

Steps Of Interaction

1. A client event occurs
2. An XMLHttpRequest object is created
3. The XMLHttpRequest object is configured
4. The XMLHttpRequest object makes an async. Request
5. The request is processed by the ValidateServlet.
6. The ValidateServlet returns an XML document containing the result
7. The XMLHttpRequest object calls the callback() function and processes the result
8. The HTML DOM is updated

Instructor Notes:

XMLHttpRequest Object

A Client event occurs

A JavaScript function is called as the result of an event

Example: validateUserId() JavaScript function is mapped to a onkeyup event on a link or form component

```
<input type="text"
size="20"
id="userid"
name="id"
onkeyup="validateUserId();">
```

Instructor Notes:

XMLHttpRequest Object

An XMLHttpRequest object is created and Configured

```
var req;
function initRequest(url) {
if (window.XMLHttpRequest) {
req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
isIE = true;
req = new ActiveXObject("Microsoft.XMLHTTP");
}
} function validateUserId() {
if (!target) target = document.getElementById("userid");
var url = "validate?id=" + escape(target.value);
initRequest(url);
req.onreadystatechange = processRequest;
req.open("GET", url, true);
req.send(null);
}
```

Instructor Notes:

An XMLHttpRequest object is configured with Callback function

```
var req;
function initRequest(url) {
if (window.XMLHttpRequest) {
req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
isIE = true;
req = new ActiveXObject("Microsoft.XMLHTTP");
}
} function validateUserId() {
if (!target) target = document.getElementById("userid");
var url = "validate?id=" + escape(target.value);
initRequest(url);
req.onreadystatechange = processRequest;
req.open("GET", url, true);
req.send(null);
}
```

Instructor Notes:

XMLHttpRequest Object

XMLHttpRequest object makes an async. request

```
var req;
function initRequest(url) {
if (window.XMLHttpRequest) {
req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
isIE = true;
req = new ActiveXObject("Microsoft.XMLHTTP");
}
} function validateUserId() {
if (!target) target = document.getElementById("userid");
var url = "validate?id=" + escape(target.value);
initRequest(url);
req.onreadystatechange = processRequest;
req.open("GET", url, true);
req.send(null);
```

URL is set to validate?id=greg

Instructor Notes:

The request is processed by the Validate Servlet at the Server

```
public class ValidationServlet extends HttpServlet {
        private ServletContext context;
        private HashMap accounts = new  HashMap();
        public void init(ServletConfig config) throws ServletException {
        this.context = config.getServletContext();
        accounts.put("greg","account data");
        accounts.put("duke","account data");
}
```

Instructor Notes:

The request is processed by the Validate Servlet at the Server (contd..)

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {
        String targetId = request.getParameter("id");
        if ((targetId != null) && !accounts.containsKey(targetId.trim())) {
        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("<valid>true</valid>");
} else {
        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("<valid>false</valid>");
} }
```

Instructor Notes:

The ValidateServlet returns an XML document containing the results

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {
        String targetId = request.getParameter("id");
        if ((targetId != null) && !users.containsKey(targetId.trim())) {
                response.setContentType("text/xml");
                response.setHeader("Cache-Control", "no-cache");
                response.getWriter().write("valid");
        } else {
        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("invalid");
} } }
```

Instructor Notes:

XML Http Request object calls callback() function and processes the result

The XMLHttpRequest object was configured to call the processRequest() function when there are changes to the readyState of the XMLHttpRequest object

```
-function processRequest() {

if (req.readyState == 4) {

if (req.status == 200) {

var message =

req.responseXML.getElementsByTagName("valid")[0].childNo

des[0].nodeValue;

setMessageUsingDOM(message);
...
```

Browser and DOM
Browsers maintain an object representation of the documents being displayed
In the form of Document Object Model (DOM)
JavaScript technology in an HTML page has access to the DOM
APIs are available that allow JavaScript technology to modify the DOM programmatically
Object Representation of the XML Document Retrieved
Following a successful request, JavaScript technology code may modify the DOM of the HTML page
The object representation of the XML document that was retrieved from the ValidateServlet is available to JavaScript technology code using the req.responseXML, where req is an XMLHttpRequest object

DOM APIs for Navigation

DOM APIs provide a means for JavaScript technology to navigate the content from that document and use that content to modify the DOM of the HTML page

```
<message> valid </message>

function parseMessage() {
var message =
req.responseXML.getElementsByTagName("message")[0];
setMessage(message.childNodes[0].nodeValue);
}
```

Instructor Notes:

XMLHttpRequest Object

The HTML DOM is updated

JavaScript technology can gain a reference to any element in the HTML DOM using a number of APIs

The recommended way to gain a reference to an element is to call

-document.getElementById("userIdMessage"), where "userIdMessage" is the ID attribute of an element appearing in the HTML document

JavaScript technology may now be used to modify the element's attributes; modify the element's style properties; or add, remove, or modify child elements

Instructor Notes:

XMLHttpRequest Object

The HTML DOM is updated (contd..)

```
<script type="text/javascript">
        function setMessage(message) {
          mdiv = document.getElementById("userIdMessage");
          if (message == "invalid") {
              mdiv.innerHTML = "<div style=\"color:red\">Invalid User Id</
div>";
          } else {
             mdiv.innerHTML = "<div style=\"color:green\">Valid User Id</
div>";
           }
       }
</script>
<body>
<div id="userIdMessage"></div>
</body>
```

Instructor Notes:

XMLHttpRequest Object

Change the Body content of an Element

```
<script type="text/javascript">
function setMessage(message) {
mdiv = document.getElementById("userIdMessage");
if (message == "invalid") {
mdiv.innerHTML = "<div style=\"color:red\">Invalid User Id</
div>";
} else {
mdiv.innerHTML = "<div style=\"color:green\">Valid User Id</
div>";
}
}
</script>
<body>
<div id="userIdMessage"></div>
</body>
```

Instructor Notes:

Ajax using JSON

```
{ "weatherdetails":
        {
                "weatherreport":
                        [
                                {"city": "Mumbai", "weather": "32" },
                                {"city": "Delhi", "weather": "28" },
                                {"city": "Chennai", "weather": "34" },
                                {"city": "Kolkata", "weather": "26" }
                        ]
        }
}
```

JSON data

```
xmlHttpRequest.open("GET","FetchWeather.jsp?city="+city+"&type="+type, true);

        xmlHttpRequest.onreadystatechange = StateChangeForJSON;

            xmlHttpRequest.send(null);
```

Configuring ajax Request and initializing

```
function StateChangeForJSON()
{

if(xmlHttpRequest.readyState == 4 && xmlHttpRequest.status == 200)
        {
                var json = eval('('+ xmlHttpRequest.responseText +')');
                for(var i=0; i < json.weatherdetails.weatherreport.length; i++)
                {.
                 .  //code for populating the HTML elements form the JSON data recived from server
                 .
```

Receiving JSON data using XMLHttpRequest object

Instructor Notes:

Add instructor notes here.

Demo

Demo1
Demo2
DemoOnAjax

Add the notes here.

Instructor Notes:

Add instructor notes here.

Lab

Lab 3

Add the notes here.

Instructor Notes:

Add instructor notes here.

Summary

In this lesson we have learned about –

- Creating the XMLHttpRequest Object
- Managing Ajax Requests