LIMITATIONS OF CLASSIC WEB APPLICATION

1. Performance

A web app is directly linked to a web browser. Due to this, the app size tends to get increased. The impact of this can be seen in the performance of a web application. A large web app performs considerably slower than a native desktop app.

2. Security

Web apps lack the feature of quality control system. As a result, both safety and security is reduced to a higher level. However, by the methods such as SSL enforcement, the users can prevent data breach to some extent.

3. Availability

Web apps are rare to find since they are not available in any App Store or Play Store. Thus, it is difficult to make awareness among audiences that such apps are available. Only the audience knowing their apps will be visiting through the respective website.

4. Web Issues

The web application is entirely dependent on the website. Which means that, if the website happens to undergo failure, then most likely the application will be failing too? Hence, for the best performance of a web application it is necessary having a quality website.

5. Internet Dependence

An internet connection is compulsory when running a web application. Still there are many parts of the world where internet is not accessible. Without a reliable internet connection you cannot either browse the web or run the web application.

Advantages of AJAX

Ajax has come a long way since the time it was introduced. It has become so popular because of the benefits it has over the traditional page wise style of website creation. Some of its major benefits are:

- Ajax helps in improving user interactivity with the application interface.
- Ajax can buffer data before the user actually needs it. This helps in increasing the overall speed of the web application and reduces wait time for the user.
- Helps in reducing bandwidth requirements for an application as only the data that is needed is requested and transferred – refreshing the entire page is not necessary.
- Ajax helps in executing queries that take a long time to run. Instead of waiting for the results after clicking the submit button, Ajax can make the data request in the background, while the user can still continually interact with the page.
- Dynamic data filtering works well with Ajax. An example of the same is the Cascading dropdown control of the ASP.NET Ajax control toolkit.
- Ajax is really good for form submissions. Feedback can be given to a user as the form is filled. There is no need to wait for the form to be submitted. For example, Hints can be given to a user while he is filling a form.
- Programmers can use a number of different languages or formats to develop Ajax pages for their specific goal(s). For example, raw data (usually obtained in XML) from a server-side database is separated from the format or structure of the webpage (which is usually structured in XHTML). Dynamic handling of DOM can be enabled. CSS use allows for the separating of style elements on the page, like fonts and picture placement.
- Ajax also separates the functionality of web pages by combining different elements in different ways. For example, JavaScript on the client-side browser is combined with XMLHttp to enable communication between client and server browsers. Then any server-side program or scripting language allows the programmer to quickly respond to client requests in a language and format they are familiar with

Applications of Ajax on the Web

The popularity of Ajax has become enormous since the time it was introduced. This is clear by the number of Ajax-based web applications that have crawled on to the World Wide Web.

- Google is using the Ajax approach to develop many of its applications including Gmail, Orkut, Google Maps, Google Suggest, Google Groups.
- Many features of *Flickr* depend on Ajax.
- Amazon's search engine *A9.com* applies similar techniques.
- Facebook has been using Ajax since a long time.
- Online documentation and presentation tools such as Google Docs, 280sildes.com are a great application of Ajax on the web.

The web applications described in this section demonstrate that Ajax is practical for real-world applications, in addition to being technically sound. Ajax applications can be any size - from the very simple, single-function Google Suggest; to the very complex and sophisticated Google Maps.

AJAX PRINCIPLES

Introduction

Ajax a new web application model which defines a new user experience to the end user. The following principles makes web application as a good Ajax application.

- **Minimal traffic** Ajax applications should send and receive a very little information to and from the web server. Ajax can minimize the traffic between server and client by not sending any unnecessary information and images.
- **Interaction Models** Traditional web model uses click and wait for interaction where AJAX uses interface paradigms such as drag-and-drop or double-clicking. Be consistent when using the AJAX models.
- **Conventions** Use established conventions for user interaction via page so that there will be a minimal learning curve.
- **No Distractions** Avoid unnecessary page elements and distractions such as animations and blinking page sections.
- **Accessibility** Donâ€™t develop sites for your self consider your primary users of your site.
- **Avoid entire page downloads** All server communication after initial page download should be managed by the AJAX engine.

The common thread in all the above principles is usability. Ajax is primarily for enhancing the web experience for your users.

Technologies behind Ajax

- **DOM** Dynamic updating of a loaded page
- **XML** Data exchange format
- **XSLT** Transforms XML into XHTML
- **XMLHttp** Primary communication
- **JavaScript** scripting language used to program an Ajax engine.
-

AJAX Example

HTML Page

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
  <h2>Let AJAX change this text</h2>
  <button type="button" onclick="loadDoc()">Change Content</button>
</div>

</body>
</html>
```

The HTML page contains a section and a <button>.

The section is used to display information from a server.

The <button> calls a function (if it is clicked).

The function requests data from a web server and displays it:

Function loadDoc()

```
function loadDoc()

{
  var xhttp= new XMLHttpRequest();
 xhttp.onreadystatechange = function()

{
    if (this.readyState == 4 && this.status == 200)

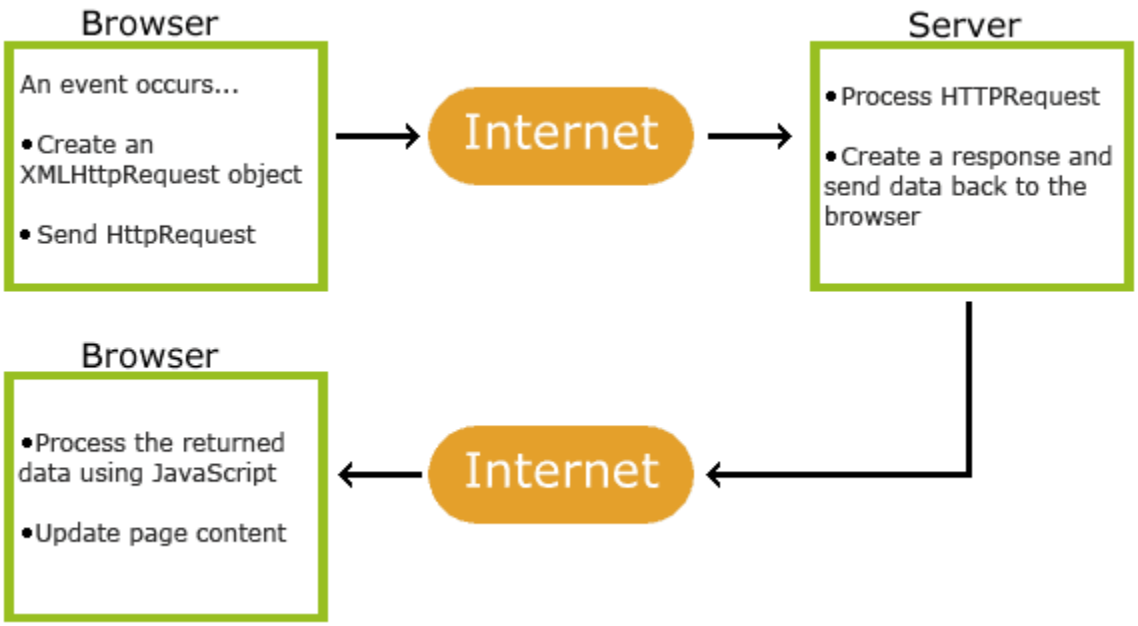
{
    document.getElementById("demo").innerHTML = this.responseText;
    }
 };
 xhttp.open("GET", "ajax_info.txt", true);
 xhttp.send();
}
```

The "ajax_info.txt" file used in the example above, is a simple text file and looks like this:

```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
```

AJAX is a technique for accessing web servers from a web page.
AJAX stands for Asynchronous JavaScript And XML.

How AJAX Works



AJAX - The XMLHttpRequest Object

The keystone of AJAX is the XMLHttpRequest object.

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

All modern browsers (Chrome, Firefox, Edge (and IE7+), Safari, Opera) have a built-in XMLHttpRequest object.

Syntax for creating an XMLHttpRequest object:

variable = new XMLHttpRequest();

Example

```
var xhttp = new XMLHttpRequest();
```

The "ajax_info.txt" file used in the example above, is a simple text file and looks like this:

```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
<p>AJAX is a technique for accessing web servers from a web page.</p>
<p>AJAX stands for Asynchronous JavaScript And XML.</p>
```

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.

The examples on W3Schools all open XML files located on the W3Schools domain.

If you want to use the example above on one of your own web pages, the XML files you load must be located on your own 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
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

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
status	Returns the status-number of a request 200: "OK" 403:"Forbidden" 404:"Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Asynchronous (AJAX Web-Application Model)

An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform other operations also. In such case, JavaScript Engine of the browser is not blocked.

As you can see in the above image, full page is not refreshed at request time and user gets response from the AJAX Engine. Let's try to understand asynchronous communication by the image given below. AJAX Components AJAX is not a technology but group of inter-related technologies.

AJAX Technologies includes: HTML/XHTML and CSSϖ DOMϖ XML or JSON(JavaScript Object Notation)

XMLHttpRequest Object JavaScript HTML/XHTML and CSS

- These technologies are used for displaying content and style. It is mainly used for presentation. DOM
- It is used for dynamic display and interaction with data. XML or JSON(Javascript Object Notation)
- For carrying data to and from server. JSON is like XML but short and faster than XML. XMLHttpRequest Object
- For asynchronous communication between client and server. JavaScrip
- t• It is used to bring above technologies together. Independently, it is used mainly for clientside validation

Creating Ajax applications using JavaScript and HTML

Example 1☹using Javascript)

```
<script>
    function run() {

        // Creating Our XMLHttpRequest object
        var xhr = new XMLHttpRequest();

        // Making our connection
        var url = 'https://jsonplaceholder.typicode.com/todos/1';
        xhr.open("GET", url, true);

        // function execute after request is successful
        xhr.onreadystatechange = function () {
            if (this.readyState == 4 && this.status == 200) {
                console.log(this.responseText);
```

```
                }
            }
            // Sending our request
            xhr.send();
        }
        run();
</script>
```

Output:

```
"{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}"
```

Example 2:(using HTML)

```
<!DOCTYPE HTML>
<html>

<head>
        <script src=
"https://code.jquery.com/jquery-3.6.0.min.js">
        </script>
</head>

<body>

        <script>

            function ajaxCall() {
                $.ajax({

                    // Our sample url to make request
                    url:
        'https://jsonplaceholder.typicode.com/todos/1',

                    // Type of Request
                    type: "GET",
```

```
                        // Function to call when to
                        // request is ok
                        success: function (data) {
                                var x = JSON.stringify(data);
                                console.log(x);
                        },

                        // Error handling
                        error: function (error) {
                                console.log(`Error ${error}`);
                        }
                });
        }
        ajaxCall();
    </script>
</body>
</html>
```

AJAX XMLHttpRequest Object

AJAX XMLHttpRequest Object - create an XMLHttpRequest object, open a URL, and send the request.

What is Ajax XMLHttpRequest object?

XMLHttpRequest object is an API for fetching any text base format data, including XML without user/visual interruptions.
All most all browser platform support XMLHttpRequest object to make HTTP requests.

u sing Ajax XMLHttpRequest object you can **make many things easier**. So many new things **can't possible** using **HEAD request**.

This object allows you to **making HTTP requests** and **receive responses** from the **server in the background**, without requiring the user to **submit the page** to the server (without **round trip process**).

Using **DOM** to **manipulate received data** from the **server** and make responsive contents are **added into live page** without **user/visual interruptions**.

Using this object you can make very **user interactive** web application.

Following are sequence of **step** for working with **XMLHttpRequest object**:

- **Define instance** of this XMLHttpRequest.

- **Create a asynchronous call** to a server page, also defining a **callback function** that will **automatically execute** when the server response is received.
- Callback function **getting server response**.
- **DOM manipulate** received data and **added into live page**.

Define instance of this XMLHttpRequest object

Create new instance of the XMLHttpRequest object, using *new* keyword,

```
var xhr = new XMLHttpRequest();
```

Microsoft Internet Explorer 6 or earlier version, you may not able to deal
with XMLHttpRequest object. Use following<="" p="" style="box-sizing: border-box;">

```
var xhr = new ActiveXObject("Msxml2.XMLHTTP");
var xhr = new ActiveXObject("Microsoft.XMLHTTP");
```

So now our example code is implemented using try catch block,

```
<script type="text/javascript">
   if (window.XMLHttpRequest || window.ActiveXObject) {
      if (window.ActiveXObject) {
         try {
            xhr = new ActiveXObject("Msxml2.XMLHTTP");
         } catch(exception) {
            xhr = new ActiveXObject("Microsoft.XMLHTTP");
         }
      } else {
         xhr = new XMLHttpRequest();
      }
   } else {
      alert("Your browser does not support XMLHTTP Request...!");
   }
</script>
```

Initialize open() method

Now we can initialize open() method along with required parameter and optional parameter.

Syntax

```
open( "method",  "URL" [ , asynchronous_flag [ , "username" [ , "password" ] ] ] )
```

Parameter Description

Parameter	Required?	Description
method	required	Specifies the HTTP method. Valid value GET, POST, HEAD, PUT, DELETE, OPTIONS
URL	required	Specifies URL that may be either relative parameter or absolute full URL.
asynchronous_flag	optional	Specifies whether the request should be handled asynchronously or not. Valid value TRUE, FALSE Default value FALSE TRUE means without waiting for a response, next code processing to execution queue on after the send() method. FALSE means wait for a response after the next code processing.
username	optional	Specifies username of authorize user otherwise set to null.
password	optional	Specifies password of authorize user otherwise set to null.

```
<script type="text/javascript">
   …
   …
   xhr.open("GET", "demo_textfile.txt", true);    // Make sure file is in same server
</script>
```

Request send using send() method

Finally send() method execute to send it along with above specified parameters.

Syntax

```
xhr.send();
```

Example

```
<script type="text/javascript">
   …
   …
   xhr.send(null);
</script>
```

readystatechange Event To Set Callback Function

Before you calling send method, Using onreadystatechange event to set callback (handler) function to be executed when the status of the request changes,

Syntax

```
xhr.onreadystatechange = function() {
    ......
};
```

Response text received from server that data assign into textarea tag using DOM.

Example

```
<script type="text/javascript">
    ...
    ...
    xhr.onreadystatechange = function(){
        document.getElementById("textArea").value = xhr.responseText;
    };
</script>
```

Simple Example Code

```
<html>
<body>
    <script language="javascript" type="text/javascript">
    function send_with_ajax(){
        if (window.XMLHttpRequest || window.ActiveXObject) {
            if (window.ActiveXObject) {
                try {
                    xhr = new ActiveXObject("Msxml2.XMLHTTP");
                } catch(exception) {
                    xhr = new ActiveXObject("Microsoft.XMLHTTP");
                }
            } else {
                xhr = new XMLHttpRequest();
```

```
        }
    } else {
        alert("Your browser does not support XMLHTTP Request...!");
    }
xhr.open("GET", "demo_textfile.txt", true);    // Make sure file is in same server
    xhr.send(null);
     xhr.onreadystatechange = function(){
        document.getElementById("textArea").value = xhr.responseText;
    };
  }
  </script>
<button onClick="send_with_ajax()">Get Contain</button><br />
  <textarea id="textArea" cols="40" rows="5"></textArea>
</body>
</html>
```

Data Download in AJAX

Using HTML

```
<button type="button" id="GetFile">Get File!</button>
```

Using Javascript

```
$('#GetFile').on('click', function () {
  $.ajax({
    url: 'https://s3-us-west-2.amazonaws.com/s.cdpn.io/172905/test.pdf',
    method: 'GET',
    xhrFields: {
      responseType: 'blob'
    },
    success: function (data) {
      var a = document.createElement('a');
      var url = window.URL.createObjectURL(data);
      a.href = url;
      a.download = 'myfile.pdf';
      document.body.append(a);
      a.click();
```

```
            a.remove();
            window.URL.revokeObjectURL(url);
        }
    });
});
```

Displaying the fetched data

Example Explained - The showCustomer() Function

When a user selects a customer in the dropdown list above, a function called showCustomer() is executed. The function is triggered by the onchange event:

showCustomer

```
function showCustomer(str) {
  if (str == "") {
    document.getElementById("txtHint").innerHTML = "";
    return;
  }
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("txtHint").innerHTML = this.responseText;
  }
  xhttp.open("GET", "getcustomer.php?q="+str);
  xhttp.send();
}
```

The showCustomer() function does the following:

- Check if a customer is selected
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The AJAX Server Page

- The page on the server called by the JavaScript above is a PHP file called "getcustomer.php".
- The source code in "getcustomer.php" runs a query against a database, and returns the result in an HTML table:
- ```
  <?php
  $mysqli = new mysqli("servername", "username", "password", "dbname");
  if($mysqli->connect_error) {
    exit('Could not connect');
  ```
  ```

```
}

$sql = "SELECT customerid, companyname, contactname, address, city, postalcode,
country
FROM customers WHERE customerid = ?";

$stmt = $mysqli->prepare($sql);
$stmt->bind_param("s", $_GET['q']);
$stmt->execute();
$stmt->store_result();
$stmt->bind_result($cid, $cname, $name, $adr, $city, $pcode, $country);
$stmt->fetch();
$stmt->close();

echo "<table>";
echo "<tr>";
echo "<th>CustomerID</th>";
echo "<td>" . $cid . "</td>";
echo "<th>CompanyName</th>";
echo "<td>" . $cname . "</td>";
echo "<th>ContactName</th>";
echo "<td>" . $name . "</td>";
echo "<th>Address</th>";
echo "<td>" . $adr . "</td>";
echo "<th>City</th>";
echo "<td>" . $city . "</td>";
echo "<th>PostalCode</th>";
echo "<td>" . $pcode . "</td>";
echo "<th>Country</th>";
echo "<td>" . $country . "</td>";
echo "</tr>";
echo "</table>";
?>
```

Sending data to server using GET and POST methods

The jQuery get() and post() methods are used to request data from the server with an HTTP GET or POST request.

HTTP Request: GET vs. POST

Two commonly used methods for a request-response between a client and server are: GET and POST.

- **GET** - Requests data from a specified resource
- **POST** - Submits data to be processed to a specified resource

GET is basically used for just getting (retrieving) some data from the server. **Note:** The GET method may return cached data.

POST can also be used to get some data from the server. However, the POST method NEVER caches data, and is often used to send data along with the request.

To learn more about GET and POST, and the differences between the two methods, please read our HTTP Methods GET vs POST chapter.

jQuery $.get() Method

The $.get() method requests data from the server with an HTTP GET request.

Syntax:

$.get(*URL,callback*);

The required URL parameter specifies the URL you wish to request.

The optional callback parameter is the name of a function to be executed if the request succeeds.

The following example uses the $.get() method to retrieve data from a file on the server:

Example

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

he first parameter of `$.get()` is the URL we wish to request ("demo_test.asp").

The second parameter is a callback function. The first callback parameter holds the content of the page requested, and the second callback parameter holds the status of the request.

jQuery $.post() Method

The $.post() method requests data from the server using an HTTP POST request.

Syntax:

$.post(*URL,data,callback*);

The required URL parameter specifies the URL you wish to request.

The optional data parameter specifies some data to send along with the request.

The optional callback parameter is the name of a function to be executed if the request succeeds.

The following example uses the $.post() method to send some data along with the request:

Example

```
$("button").click(function(){
  $.post("demo_test_post.asp",
  {
    name: "Donald Duck",
    city: "Duckburg"
  },
  function(data, status){
    alert("Data: " + data + "\nStatus: " + status);
  });
});
```

The first parameter of $.post() is the URL we wish to request ("demo_test_post.asp").

Then we pass in some data to send along with the request (name and city).

The ASP script in "demo_test_post.asp" reads the parameters, processes them, and returns a result.

The third parameter is a callback function. The first callback parameter holds the content of the page requested, and the second callback parameter holds the status of the request.