

# AJAX

IF3110 – Web-based Application Development  
School of Electrical Engineering and Informatics  
Institut Teknologi Bandung

# What is AJAX

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML

AJAX is not a programming language

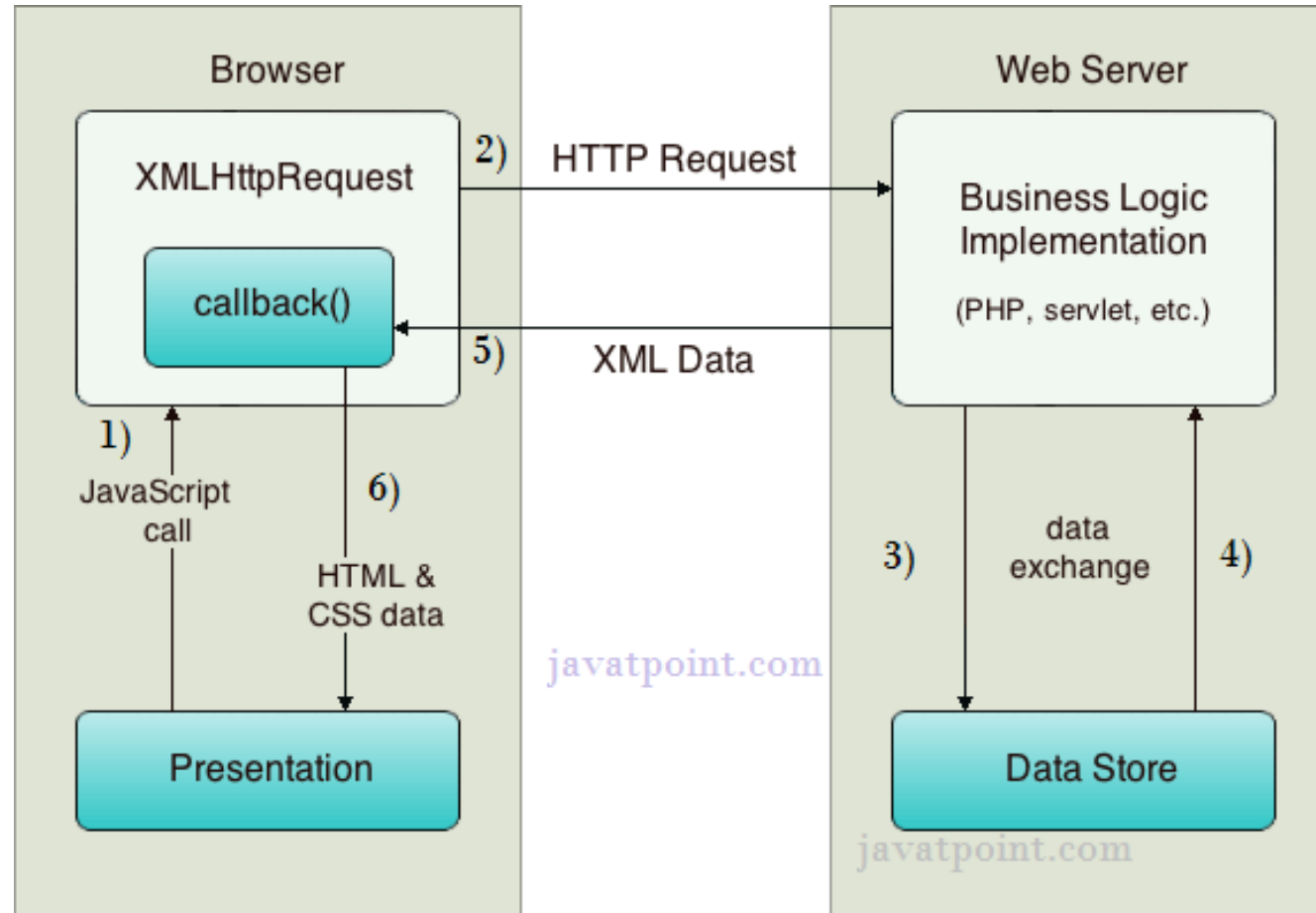
AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

# AJAX is a developer's dream

- 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 Interaction



# AJAX Example

```
<!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>
```

Let AJAX change this text

Change Content

# AJAX 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();  
}
```

ajax\_info.txt

```
<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>
```

# 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
- Modern browser can use **Fetch API** in a simpler way

# Access Across Domains

- For security reasons, modern browsers do not allow access across domains (same-origin policy)
- This means that both the web page and the XML file it tries to load, must be located on the same origin (same domain & scheme & port)
- Exception to this rule can be specified by CORS mechanism of the different origin:
  - Access-Control-Allow-Origin: <https://example.com>



# XMLHttpRequest Object Methods

Method	Description
<code>new XMLHttpRequest()</code>	Creates a new XMLHttpRequest object
<code>abort()</code>	Cancels the current request
<code>getAllResponseHeaders()</code>	Returns header information
<code>getResponseHeader()</code>	Returns specific header information
<code>open(<i>method</i>, <i>url</i>, <i>async</i>, <i>user</i>, <i>psw</i>)</code>	Specifies the request <i>method</i> : the request type GET or POST <i>url</i> : the file location <i>async</i> : true (asynchronous) or false (synchronous) <i>user</i> : optional user name <i>psw</i> : optional password
<code>send()</code>	Sends the request to the server Used for GET requests
<code>send(<i>string</i>)</code>	Sends the request to the server. Used for POST requests
<code>setRequestHeader()</code>	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
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")

# 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**.

# Sending information with GET/POST

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);  
xhttp.send();
```

```
xhttp.open("POST", "demo_post2.asp", true);  
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
xhttp.send("fname=Henry&lname=Ford");
```

# 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 **XMLHttpRequest** object:

```
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();
```

# Synchronous Request

- To execute a synchronous request, change the third parameter in the **open()** method to **false**.
- Sometimes `async = false` are used for quick testing.
- Since the code will wait for server completion, there is no need for an **onreadystatechange** function

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

- Synchronous **XMLHttpRequest** (`async = false`) is **not recommended** because the JavaScript will stop executing until the server response is ready

# The responseXML Property

- The **XMLHttpRequest** object has an in-built XML parser.
- The **responseXML** property returns the server response as an XML DOM object.

```
xmlDoc = xmlhttp.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;
xmlhttp.open("GET", "cd_catalog.xml", true);
xmlhttp.send();
```

# The responseXML Property ...

```
function loadDoc() {  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function() {  
        if (this.readyState == 4 && this.status == 200) {  
            myFunction(this);  
        }  
    };  
    xhttp.open("GET", "c  
    xhttp.send();  
}
```

```
function myFunction(xml) {  
    var i;  
    var xmlDoc = xml.responseXML;  
    var table="<tr><th>Title</th><th>Artist</th></tr>";  
    var x = xmlDoc.getElementsByTagName("CD");  
    for (i = 0; i < x.length; i++) {  
        table += "<tr><td>" +  
            x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +  
            "</td><td>" +  
            x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +  
            "</td></tr>";  
    }  
    document.getElementById("demo").innerHTML = table;  
}
```



# The getAllResponseHeaders() Method

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

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

# The `getResponseHeader()` Method

The **`getResponseHeader()`** method returns specific header information from the server response

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        document.getElementById("demo").innerHTML =
            this.getResponseHeader("Last-Modified");
    }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

# The Fetch API

```
async function getData() {  
  const url = "https://example.org/products.json";  
  try {  
    const response = await fetch(url);  
    if (!response.ok) {  
      throw new Error(`Response status: ${response.status}`);  
    }  
  
    const json = await response.json();  
    console.log(json);  
  } catch (error) {  
    console.error(error.message);  
  }  
}
```

The Fetch API provides an interface for fetching resources (including across the network). It is a more powerful and flexible replacement for XMLHttpRequest.

The `fetch()` method takes one mandatory argument, the path to the resource you want to fetch. It returns a Promise that resolves to the Response to that request

# WEB Platform APIs

The XMLHttpRequest and the Fetch API are part of the Web platform API defined by the standards bodies WHATWG and W3C.

<https://fetch.spec.whatwg.org/>

<https://developer.mozilla.org/en-US/docs/Web/API>

They are not part of ECMAScript standard.

# AJAX use case

- Autocomplete
- Voting and Rating
- Dynamically updating content
- Form Submission & Validation
- Chat Rooms And Instant Messaging
- Slicker UIs
- Widgets and adds
- JIT Help system
- Tooltips
- Accordion
- Tabbed pages

# AJAX Pattern

- **Multi Stage Download**

Large pages don't load all at once, but gradually

- **Periodic Refresh** (polling mechanism)

AJAX periodically checks if new data is available on the server

- **Predictive Fetch**

AJAX app guesses what the user is going to do next, and retrieves the appropriate data

- **Fallback Pattern**

If an error occurs on the server, meaning a status of something other than 200 is returned, AJAX need to decide what to do

- **Submission Throttling**

AJAX sends large data in several stages.