

IMY 220 ● Lecture 12

# AJAX

Acronym for Asynchronous JavaScript and XML

Coined in 2005

AJAX is not limited to the technologies mentioned in the acronym.

The term AJAX simply means almost any technique or technology that lets a user's browser **interact with a server without disturbing the existing page.**

# AJAX

The alternative to AJAX involves the user clicking a link or submitting a form, which sends a request to the server.

The current browser page is removed and the server responds with a fresh page of HTML, which is loaded and displayed to the user.

And while the page is loading, the user is forced to wait...

# AJAX

AJAX lets us fire requests from the browser to the server without page reload

So we can update a part of the page while the user continues working.

This helps us mimic the feel of a desktop application, and gives the user a more responsive and natural experience.

# AJAX

As AJAX runs on the browser, we need a way to interact dynamically with the server.

Each web browser tends to supply slightly different methods for achieving this.

# AJAX

The use of the `XMLHttpRequest` object allows us to communicate with server-side scripts

It can send as well as receive information in a variety of formats, including **JSON**, **XML**, **HTML**, and even text files.

It is **asynchronous** which means it can communicate with the server without having to refresh the page.

You can update **portions of a page** based upon user events.

[https://developer.mozilla.org/en-US/docs/AJAX/Getting\\_Started](https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started)

# AJAX

The two main features of AJAX:

1. Make requests to the server without reloading the page
2. Receive and work with data from the server

# AJAX

## Steps:

### 1. Make an HTTP request

1. Create XMLHttpRequest object
2. Tell the HTTP request object which JavaScript function will **handle processing** the response
3. Making the request with **open()** and **send()**

### 2. Handle the server response

1. Check for the **state** of the request
2. Check the **response code**
3. Do something with the data

The next example will demonstrate the steps.



# A simple example

The user clicks the link "Make a request" in the browser;

The event handler calls the `makeRequest()` function with a parameter – the name `test.html` of an HTML file in the same directory;

The request is made and then, using `onreadystatechange`, the execution is passed to `alertContents()`

`alertContents()` checks if the response was received and it's an OK and then `alerts` the contents of the `test.html` file.

# A simple example

Files:

- index.html
- test.html
- script.js

# A simple example

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title> Welcome </title>
  </head>
  <body>
    <a href="#" id="ajaxButton">
      Make a request
    </a>
    <script type="text/javascript" src="script.js"></script>
  </body>
</html>
```

# A simple example

test.html

I'm a test.

# A simple example

script.js

```
document.getElementById("ajaxButton").onclick = makeRequest;  
  
// We still have to define the makeRequest function which  
// calls the AJAX functions and gets the data
```

# A simple example

## 1: Make an HTTP request

```
let httpRequest;

function makeRequest() {
    httpRequest = new XMLHttpRequest();

    if (!httpRequest) {
        alert('Cannot create an XMLHttpRequest instance');
        return false;
    }
    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('GET', 'test.html');
    httpRequest.send();
}
```

# A simple example

1.1: Create `XMLHttpRequest` object (in script.js)

In order to make an HTTP request to the server using JavaScript, **you need an instance of a class that provides this functionality.**


## `XMLHttpRequest`

This class was originally introduced in Internet Explorer as an `ActiveX` object called `XMLHTTP`. Then, Mozilla, Safari and other browsers followed, implementing an `XMLHttpRequest` class that supports the methods and properties of Microsoft's original `ActiveX` object. Meanwhile Microsoft has implemented `XMLHttpRequest` as well.

# A simple example

## 1.1: Create XMLHttpRequest object (in script.js)

```
let httpRequest;

function makeRequest() {
    httpRequest = new XMLHttpRequest(); 
    if (!httpRequest) {
        alert('Cannot create an XMLHttpRequest instance');
        return false;
    }
    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('GET', 'test.html');
    httpRequest.send();
}
```



# A simple example

1.2: Tell the HTTP request object which JavaScript function will **handle processing** the response

This is done by setting the `onreadystatechange` property of the object to the name of the JavaScript function that should be called when the state of the request changes

```
httpRequest.onreadystatechange = nameOfTheFunction;
```

Note that there are no parentheses after the function name and no parameters passed, because you're simply assigning a reference to the function, rather than actually calling it

# A simple example

1.2: Tell the HTTP request object which JavaScript function will **handle processing** the response

```
let httpRequest;

function makeRequest(url) {
    httpRequest = new XMLHttpRequest();

    if (!httpRequest) {
        alert('Cannot create an XMLHttpRequest instance');
        return false;
    }
    httpRequest.onreadystatechange = alertContents; ←
    httpRequest.open('GET', url);
    httpRequest.send();
}
```

# A simple example

## 1.3: Making the request with `open()` and `send()`

call the `open()` and `send()` methods of the HTTP request class

# A simple example

## open()

The first parameter of the call to `open()` is the **HTTP request method** – GET, POST, HEAD or any other method that is supported by your server.

Keep the method capitalized as per the HTTP standard; otherwise some browsers (like Firefox) might not process the request.

The second parameter is the **URL** of the page you're requesting. As a security feature, you cannot call pages on 3rd-party domains. Be sure to use the exact domain name on all of your pages or you will get a "permission denied" error when you call `open()`.

The optional third parameter **sets whether the request is asynchronous**. If TRUE (the default), the execution of the JavaScript function will continue while the response of the server has not yet arrived. This is the A in AJAX.

# A simple example

## 1.3: Making the request with `open()` and `send()`

```
let httpRequest;

function makeRequest(url) {
    httpRequest = new XMLHttpRequest();

    if (!httpRequest) {
        alert('Cannot create an XMLHttpRequest instance');
        return false;
    }

    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('GET', url);
    httpRequest.send();
}
```

# A simple example

## send()

The parameter to the `send()` method can be any data you want to send to the server if POST-ing the request. Form data should be sent in a format that the server can parse easily, e.g., a query string, JSON, SOAP, etc.


# A simple example

## 1.3: Making the request with `open()` and `send()`

```
let httpRequest;

function makeRequest(url) {
    httpRequest = new XMLHttpRequest();

    if (!httpRequest) {
        alert('Cannot create an XMLHttpRequest instance');
        return false;
    }

    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('GET', url);
    httpRequest.send(); 
}
```

# A simple example

## 2: Handle the server response

```
function alertContents() {  
    if (httpRequest.readyState === XMLHttpRequest.DONE) {  
        if (httpRequest.status === 200) {  
            alert(httpRequest.responseText);  
        } else {  
            alert('There was a problem with the request.');        }  
    }  
}
```



# A simple example

## 2.1: Check for the **state of the request**.

If the state has the value of `XMLHttpRequest.DONE` (evaluating to 4), that means that the full server response has been received and it's OK for you to continue processing it.

# A simple example

## 2.1: Check for the **state of the request**.

```
function alertContents() {  
    if (httpRequest.readyState === XMLHttpRequest.DONE) {  
        if (httpRequest.status === 200) {  
            alert(httpRequest.responseText);  
        } else {  
            alert('There was a problem with the request.');        }  
    }  
}
```

# A simple example

2.2: Check the **response code** of the HTTP server response.

All the possible codes are listed on the W3C site.

In the example, we differentiate between a **successful** or **unsuccessful** AJAX call by checking for a **200** OK response code.

# A simple example

Some error response codes include:

- 400 – Bad request (The server could not understand the request due to invalid syntax.)
- 403 – Forbidden (The client does not have access rights to the content)
- 404 – Not found (The server can not find requested resource)
- 502 – Bad gateway (The server, while working as a gateway to get a response needed to handle the request, got an invalid response.)

You can find a full list of HTTP response codes here:

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

# A simple example

2.2: Check the **response code** of the HTTP server response.

```
function alertContents() {  
    if (httpRequest.readyState === XMLHttpRequest.DONE) {  
        if (httpRequest.status === 200) {  
            alert(httpRequest.responseText);  
        } else {  
            alert('There was a problem with the request.');        }  
    }  
}
```

# A simple example

2.3: Do something with the data.

In the example, we just alert the data.

# A simple example

## 2.3: Do something with the data.

```
function alertContents() {  
    if (httpRequest.readyState === XMLHttpRequest.DONE) {  
        if (httpRequest.status === 200) {  
            alert(httpRequest.responseText);  
        } else {  
            alert('There was a problem with the request.');        }  
    }  
}
```

# A simple example

2.3: Do something with the data.

Note that you can't use this code in the same way you use synchronous code

For example, you can't set a variable asynchronously and use it in the next line, because the variable might not have been set



# A simple example

```
let text = "Example";

function alertContents() {
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
        if (httpRequest.status === 200) {
            text = httpRequest.responseText;
        } else {
            alert('There was a problem with the request.');
        }
    }
}

// Server call is made and alertContents is called
console.log(text);
// Output: ???
```

# A simple example

```
let text = "Example";

function alertContents() {
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
        if (httpRequest.status === 200) {
            text = httpRequest.responseText;
        } else {
            alert('There was a problem with the request.');
        }
    }
}

// Server call is made and alertContents is called
console.log(text);
// Output: Example
```

# A simple example

2.3: Do something with the data.

You always have to use callback functions (or promises) when dealing with asynchronous functions

# A simple example

## 2.3: Do something with the data.

```
function alertContents() {  
    if (httpRequest.readyState === XMLHttpRequest.DONE) {  
        if (httpRequest.status === 200) {  
            console.log(httpRequest.responseText);  
            // Output: I'm a test  
  
        } else {  
            alert('There was a problem with the request.');        }  
    }  
}
```

# A simple example with Promises

## Recap

*“A Promise is an object representing the eventual completion or failure of an asynchronous operation”*

*“Essentially, a Promise is a returned object to which you attach callbacks, instead of passing callbacks into a function”*

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using\\_promises](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises)

# A simple example with Promises

As of ES6 we can use Promises instead of relying on (potentially deeply nested) callback functions

This is done by wrapping the functionality of the AJAX call inside a Promise...

...and then adding functionality to the success or failure of the Promise

# A simple example with Promises

```
let httpRequest;

function makeRequest(url) {
    httpRequest = new XMLHttpRequest();

    if (!httpRequest) {
        alert('Cannot create an XMLHttpRequest instance');
        return false;
    }

    httpRequest.onreadystatechange = alertContents;
    httpRequest.open('GET', url);
    httpRequest.send();
}
```

We want to encapsulate this AJAX request as a promise...

...and then do something with it

# A simple example with Promises

To do that, we redefine `makeRequest` as a Promise object

```
let httpRequest;  
  
let makeRequest = new Promise((resolve, reject) => {  
    // AJAX code goes here  
})
```



# A simple example with Promises

Instead of using callbacks to handle the success or failure of the AJAX call, we use the resolve and reject functions

```
let httpRequest;

let makeRequest = new Promise((resolve, reject) => {
  // AJAX code goes here
  // If the AJAX call was successful, do something with the data
  resolve(httpRequest.responseText);

  // If the AJAX call failed, return a failure message
  reject('There was a problem with the request.');
```

```
});
```

```
let makeRequest = new Promise((resolve, reject) => {
  httpRequest = new XMLHttpRequest();

  if(!httpRequest){
    reject('Cannot create an XMLHttpRequest instance');
  }

  httpRequest.onreadystatechange = () => {
    if(httpRequest.readyState === XMLHttpRequest.DONE){
      if(httpRequest.status === 200){
        resolve(httpRequest.responseText);
      } else {
        reject('There was a problem with the request.');
```

# A simple example with Promises

Then we still need to assign it to our `onclick` handler

We pass the button's event handler an anonymous function which calls the Promise object's `.then` and logs the data

```
document.getElementById("ajaxButton").onclick = () => {  
    makeRequest  
    .then(data => {  
        console.log(data) ;  
    }) ;  
}
```

# A simple example with Async/Await

We can also use async/await to call the Promise

```
document.getElementById("ajaxButton").onclick = () => {  
    let message = await makeRequest;  
    console.log(message);  
}
```

Output: ???

# A simple example with Async/Await

We can also use async/await to call the Promise

```
document.getElementById("ajaxButton").onclick = () => {  
    let message = await makeRequest;  
    console.log(message);  
}
```

Output: error, because you can only use await within `async` functions

We can fix this by defining our anonymous event-handler function as `async`

# A simple example with Async/Await

We can also use async/await to call the Promise

```
document.getElementById("ajaxButton").onclick = async () => {  
  let message = await makeRequest;  
  console.log(message);  
}
```

Now the contents of test.html should be logged to the console

Remember that, unlike with Promises, the execution of the script will actually halt at **await makeRequest** until the Promise resolves or rejects

# Extra video

Asynchrony: Under the Hood

(ClickUP -> Lecture Slides -> Additional Resources)

Good overview of different asynchronous behaviours in JS and how they operate (highly recommended viewing)

# References

[https://developer.mozilla.org/en-US/docs/AJAX/Getting\\_Started](https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started)

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise)

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async\\_function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function)