CIS 193 – Web Development Using JavaScript

Lecturer: Joseph Su

Reading Assignment

- AJAX
  - https://www.w3schools.com/xml/ajax_intro.asp
  - https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data
- XMLHttpRequest
  - https://en.wikipedia.org/wiki/XMLHttpRequest
  - https://www.w3.org/TR/XMLHttpRequest/
- HTTP Response Status Code:
  - https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
- Fetch API
  - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
  - https://developers.google.com/web/updates/2015/03/introduction-to-fetch
- Promise API
  - https://javascript.info/promise-api
  - https://bitsofco.de/javascript-promises-101/

Overview

- What is AJAX?
  - Technology Stack
  - Properties
- XMLHttpRequest
  - Overview
  - Requests
  - readyState / HTTP Status Code
  - Examples
- JSON
  - Structures
  - JSON vs XML
- JSON vs JavaScript
- Fetch API
  - Request
  - Response
- Promise
- Chaining Promises
- POST Request

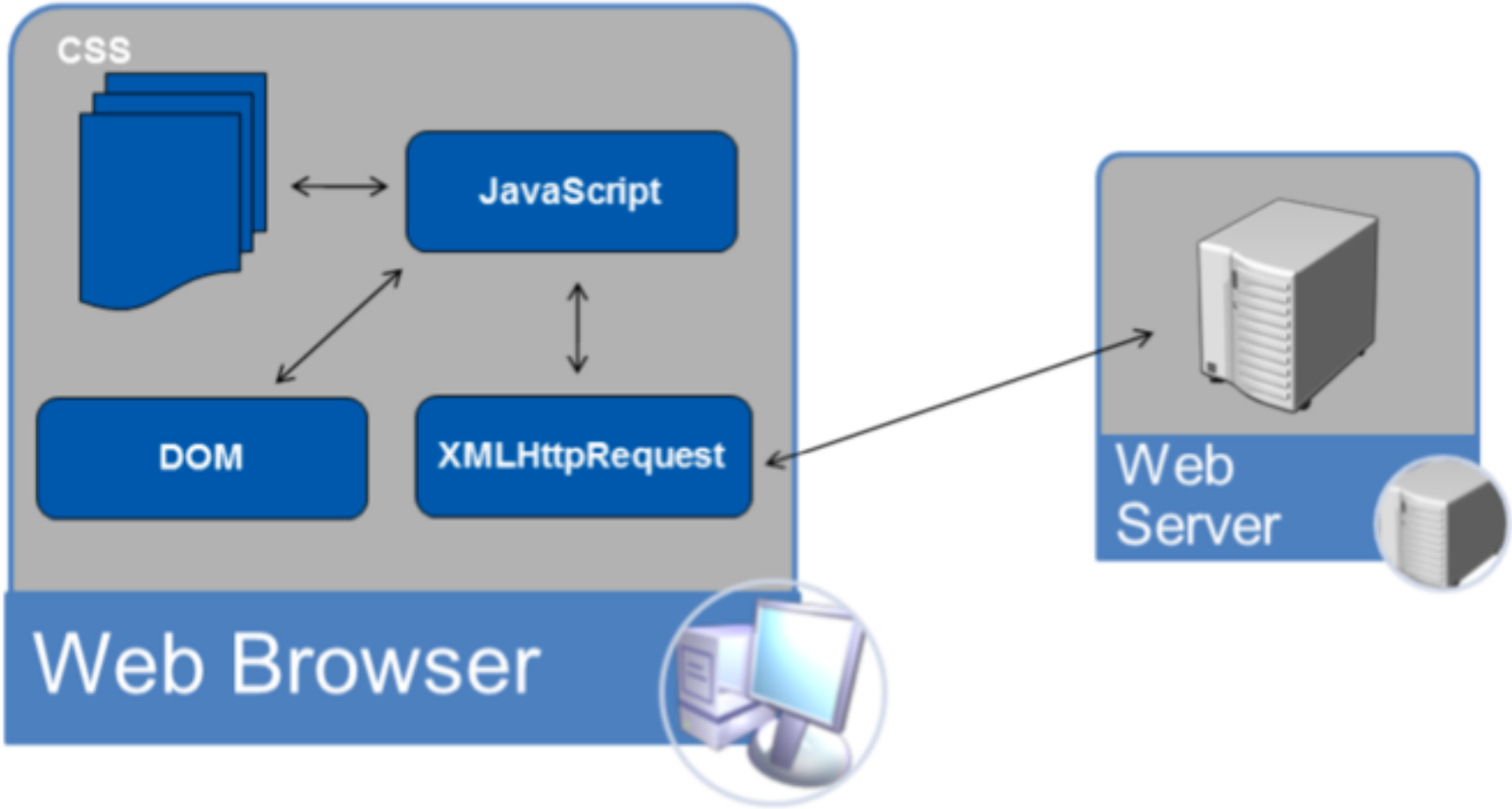
AJAX: What is It

- Technology for creating rich internet applications
  - For creating highly responsive apps and services
  - Rich content and interactions
- A group of technologies with built-in XMLHttpRequest object.
- A client-focused model
  - JavaScript, CSS, HTML, DHTML,
- A user-focused model
  - UX
  - User-first development model
- Async model
  - Communicate with server via asynchronous communications
  - User activity not interrupted

AJAX: Properties

- The browser hosts an application, not HTML content.
  - Data is sent to the browser.
  - JavaScript manages client-side UX with user.
- The server delivers data, not HTML content.
  - Requests from client to server
  - Response from server to client to fill data into HTML
- User interaction can be continuous and fluid.
  - Event driven apps
  - Near instantaneous response to user
- Client – server communication requires coding disciplines.
  - Fidelity in API contracts between client and server
  - Communication synchronicity

CSS, DOM, JavaScript, XMLHttpRequest



XMLHttpRequest Overview

- API
- It is a JavaScript Object with methods for communicating data between a browser and server.
- Provided by the browser's JavaScript Just-In-Time environment.
- HTTP/HTTPS as transport protocol.
- Support standard CRUDs:
  - GET
  - POST
  - HEAD
  - PUT
  - DELETE
  - OPTIONS

XMLHttpRequest: Overview

- Handle request process
  - W3C specification (http://www.w3.org/TR/XMLHttpRequest)

- Benefits:
  - Easy to use
  - Standard CRUD support
  - Can be used synchronously or asynchronously
  - Request headers can be added
  - Response headers can be added
  - Support in all modern browsers

XMLHttpRequest: Requests

- **`open` method**
  - Sets up the XMLHttpRequest object for communications

```
request.open(sendMethod, sendUrl[, boleanAsync, stringUser, stringPwd]);
```

- **`send` method**
  - Initiates the request

```
request.send([varData]);
```

- **`abort` method**
  - Cancels a request currently in process
- **`setRequestHeader` method**
  - Adds custom HTTP headers to the request
    - Used mainly to set *Content-Type*

```
request.setRequestHeader(sName, sValue);
```

XMLHttpRequest: Requests

- **`readystatechange` event**
  - Fires for each stage in the request cycle
  - Handle to be able to retrieve the content of the response
- **`readyState` property – progress indicator (0 to 4)**
  - Most important is 4 (Loaded); you can access the data
- **`responseXXX` property – retrieves the response**
  - `responseText` – as a string
  - `responseBody` – as an array of unsigned bytes
- **`status` property, `statusText` property**
  - Return the HTTP response code or friendly text respectively

HTTP Response Status Code:

https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

XMLHttpRequest: readyState

readyState value	Description
0	Unsent
1	Opened
2	Headers Received
3	Loading
4	Loaded (Done) – data is fully loaded

XMLHttpRequest: Examples

Using XMLHttpRequest

- **Create a new `XMLHttpRequest` object**
- **Set the request details using the `open` method**
- **Hook-up the `readystatechange` event to a callback function**
  - Easiest way is to use an anonymous function
- **`send` the request**

```
  function loadAjaxTxt() {
            var xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function () {
                if (xmlhttp.readyState == 4 && xmlhttp.status == 200)
{
                    document.getElementById('content').innerHTML =

        xmlhttp.responseText;}
                }
            //set up communication
            xmlhttp.open("GET", "ajax_info.txt", true);
            //intiates the request
            xmlhttp.send();
        }
```

XMLHttpRequest: Important Info

XMLHttpRequest support

- **Supported in all modern browsers**
  - As a native JavaScript object `XMLHttpRequest`
- **IE 5.x/6 use through an ActiveXObject**
- **Use *object detection* to get the right object**

A cross browser complaint request

```
var xmlHttp = null;
function ensureXMLHttp() {
    if (window.XMLHttpRequest) {
        xmlHttp = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        try {
             xmlHttp = new
ActiveXObject("Microsoft.XMLHTTP");
            }
        catch(e) {}
    }
}
```

JavaScript Objection Notation (JSON)

- Lightweight data interchange format
  - Vs XML
- Simple format
  - Human readable and writeable
  - Easy to parse, convert, and generate
- JSON is TEXT format
  - Agnostic of programming language
  - Similar syntactical convention in C#, C++, C, and JavaScript
    - Associative array
    - Easily convertible into a JavaScript object

JSON: Structures

- Universal data structures compliant with modern programming languages.
- Collection of keys – values as JavaScript objects
  - Associative array
- Ordered list of values as JavaScript array
  - Indexed array
- JSON object
  - Unordered set of name-value pairs
  - Begin with {
  - End with }
  - Each name followed by a : (colon)
  - Name/value pairs separated by a , (comma)

JSON: Example

- Can be validated using https://jsonlint.com/

```
{
    "glossary": {
        "title": "example glossary",
        "GlossDiv": {
            "title": "S",
            "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                    "SortAs": "SGML",
                    "GlossTerm": "Standard Generalized Markup Language",
                    "Acronym": "SGML",
                    "Abbrev": "ISO 8879:1986",
                    "GlossDef": {
                        "para": "A meta-markup language.",
                        "GlossSeeAlso": ["GML", "XML"]
                    },
                    "GlossSee": "markup"
                }
            }
        }
    }
}
```

XML: Example

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
      <glossary><title>example glossary</title>
       <GlossDiv><title>S</title>
            <GlossList>
                  <GlossEntry ID="SGML" SortAs="SGML">
                        <GlossTerm>Standard Generalized Markup Language</GlossTerm>
                        <Acronym>SGML</Acronym>
                        <Abbrev>ISO 8879:1986</Abbrev>
                        <GlossDef>
                              <para>A meta-markup language.</para>
                         <GlossSeeAlso OtherTerm="GML">
                         <GlossSeeAlso OtherTerm="XML">
                        </GlossDef>
                        <GlossSee OtherTerm="markup">
                  </GlossEntry>
            </GlossList>
       </GlossDiv>
       </glossary>
```

JSON vs JavaScript

JSON is a subset of the object literal notation of JavaScript

- **Can be used in the JavaScript language with no problems**

```
var myJSONObject = { "searchResults": [
    { "productName" : "Aniseed Syrup", "unitPrice" : 10 },
    { "productName" : "Alice Mutton", "unitPrice" : 39 }
  ]
};
```

- **Object created by this example**
  - Single member `searchResults`
  - Contains two objects each containing `productName` and `unitPrice` members
  - Can use dot or subscript operators

```
alert(myJSONObject.searchResults[0].productName); // alerts
"Aniseed Syrup"
```

Fetch API

- Provides interface for fetching resources locally or across network.
- Similar to XMLHttpRequest but more powerful and flexible

fetch() allows you to make network requests similar to XMLHttpRequest (XHR)

- **Uses promises**
- **Simpler and cleaner**

```
// url (required), options (optional)
fetch('https://qa.com/some/url', {
      method: 'get'
}).then(function(response) {

}).catch(function(err) {
      // Error :(
});
```

Fetch Request: Example

Parse response as JSON

```
fetch('./api/some.json')
  .then(
    function(response) {
      if (response.status !== 200) {
        console.log('Looks like there was a problem. Status
Code: ' +
          response.status);
        return;
      }

      // Examine the text in the response
      response.json().then(function(data) {
        console.log(data);
      });
    }
  )
  .catch(function(err) {
    console.log('Fetch Error :-S', err);
  });
```

Fetch Response: Metadata

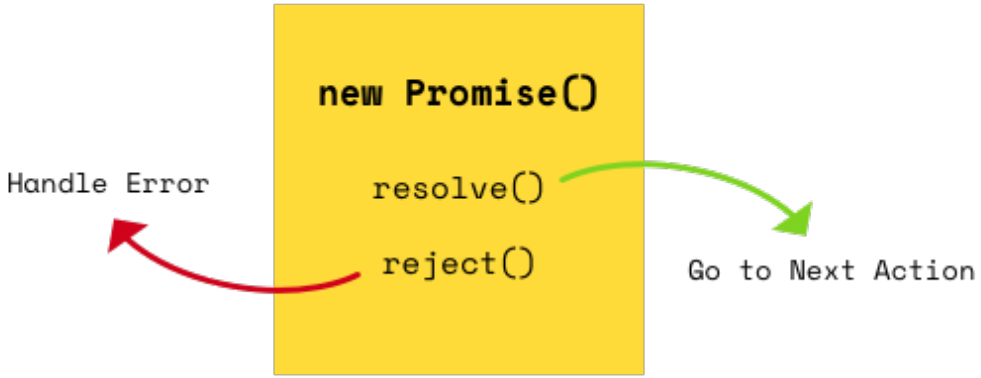
Other metadata we may want to access, like headers, are illustrated below.

```
fetch('users.json').then(function(response) {
    console.log(response.headers.get('Content-Type'));
    console.log(response.headers.get('Date'));

    console.log(response.status);
    console.log(response.statusText);
    console.log(response.type);
    console.log(response.url);
});
```

Promise

- Standard Built-in JavaScript API
- A JavaScript object that may produce a single value some time in the future: either a resolved value, or a reason for why it is not resolved (e.g., network error).
- Represents the result of an operation that has not been completed, but will, at some later time to be determined.
- Uses Stream to ensure multiple asynchronous operations with callbacks can be managed much easily than JavaScript event callback functions.



Chaining Promises

- **Great features of promises**
  - Chain them together
- **For fetch**
  - This allows you to share logic across fetch requests

```
fetch('users.json')
  .then(status)
  .then(json)
  .then(function(data) {
    console.log('Request succeeded with JSON response',
data);
  }).catch(function(error) {
    console.log('Request failed', error);
  });
```

Chaining Promises

- From the previous example:
  - Both status and json are JavaScript functions
  - Can be defined separately to check status and parse JSON

```
function status(response) {
    if (response.status >= 200 && response.status < 300) {
        return Promise.resolve(response);
    } else {
        return Promise.reject(new Error(response.statusText));
    }
}

function json(response) {
    return response.json();
}
```

POST Request

To call an API with a POST Method. Set method and body parameters in the fetch() option.

```
fetch(url, {
    method: 'post',
    headers: {
      "Content-type": "application/x-www-form-
urlencoded; charset=UTF-8"
    },
    body: 'foo=bar&lorem=ipsum'
  })
  .then(json)
  .then(function (data) {
    console.log('Request succeeded with JSON response',
data);
  })
  .catch(function (error) {
    console.log('Request failed', error);
  });
```