W07 - AJAX

AJAX allows websites to communicate asynchronously with servers, dynamically updating pages w/out reloading

- Clients and Servers - internet has two parts clients and servers
  - setting up a local development server may be appropriate to facilitate Ajax or server side development.
- History
  - It all started with static content.
  - 1999 Microsoft implemented XMLHTTP ActiveX control allowing asynchronous data.
  - 2004/5 Google maps and gmail web applications used asynchronous loading techniques.
  - 2005 Jesse James Garrett coined the term "Ajax"
    Asynchronous JavaScript and XML:
    ASYNCHRONOUS - program doesn't have to wait for requested data before carrying on with program.
    JAVASCRIPT - Always considered a front end language, AJAX enabled JavaScript to send and receive requests from a server.
    XML - At the beginning XML documents were used to return data. JSON is the most common form today
- The Fetch API - XMLHttpRequest object became the standard. It is now superceded by the Fetch API.
  - uses promises to avoid callback hell.

Basic Usage - global fetch() method - one mandatory argument: url of the resource.

Basic example:

```
fetch('https://example.com/data')
  .then(// code that handles response)
  .catch(// code that handles error)
```

W07 - AJAX

Response Interface - allow effective processing of the response.

- ok property - checks to see if response is successful, based on HTTP status code
- status property - used to access status code.
  - 200 if successful
  - 201 if resource created
  - 204 if request successful but no content returned.
- ok returns true if status property between 200 - 299

If block to check success of request:

```
const url = 'https://example.com/data';

fetch(url)
.then ( (response) => {
   if (response.ok) {
      return response;
   }
throw Error (response.statusText);
})
.then (response => // do something)
.catch (error => console.log('error occured'))
```

other properties of Response object:

- headers - A Headers object containing any headers associated with response.
- url - string containing url of responses
- redirect - boolean value response result of a redirect
- type - "basic", "cors", "error", or "opaque"
  - basic - response from same domain
  - cors - received from valid cross-origin request from different domain.
  - opaque - response recieved from no-cors from different domain.
  - error - network error

redirect() - used to redirect to another URL

#W07 - AJAX

Redirect example - no support currently in any browser.

```
fetch (url)
  .then ( response => response.redirect (newUrl));
  .then ( // do something else)
  .catch (error => console.log ('ERROR:', error))
```

Text Responses - text() method takes a stream of text returns a promise resolving to a USVString object treated as a string in JavaScript.

```
fetch (url)
 .then ( response => response.text() ) ; // transform text stream into JS string)
 .then ( text => console.log(text))
 . catch ( error => console.log ('ERROR', error))
```

File Responses - blob() method used to read file of raw data.

```
fetch (url)
 .then (response => response.blob() ); // transforms data into a blob object
 .then ( blob => console.log ( blob.type)) // logs type of file received
 . catch (error => console.log ('ERROR', error))
```

JSON Responses - json() method used to transform a stream of JSON data into a promise that resolves into a JS object.

```
fetch (url)
.then ( response => response.json() );
.then (data => console.log ( Object. entries (data))) // view key/valu
. catch (error => console.log ('ERROR:', error))
```

Creating Response Objects - can create using a constructor function

```
const response = new Response ('Hello!', {
    ok: true,
    status: 200,
    statusText: 'OK',
    type: 'cors',
    url: '/api'
});
```

useful when creating an API or dummy response while testing

W07 - AJAX

Request Interface - using a request object as an argument allows options to be set about the request.
- created using Request() constructor

Properties:
- url - request resource
- method - string specifying HTTP method used for request. defaults to GET.
- headers - a Headers object providing details of request's headers
- mode - allows specification of CORS or not. enabled by default
- cache - specify how to use browser's cache
- credentials - specify if cookies should be allowed
- redirect - specify what to do if response returns a redirect.

GET - request to retrieve resources
POST - requests
PUT - requests to upsert, insert or update
PATCH - requests to make partial update
DELETE - requests to delete a resource

constructor function used to create a new Request object.

```
const request = new Request('https://example.com/data', {
    method: 'GET',
    mode: 'cors',
    redirect: 'follow',
    cache: 'no-cache'
});
```

↑ Required. rest made up from any of the properties listed.

↑ used as a parameter of fetch()

```
fetch(request)
.then(// do something)
.catch(//handle error)
```

url and object can be entered directly

Headers Interface - headers are used to pass additional information

```
const headers = new Headers(); // constructor can include optional arguments containing 'initial values'
```

- has() - boolean to check if contains header provided as an argument

```
headers.has('Content-type'); // returns true if has it
```

W07 - AJAX

- set() - used to set a value of existing header or create a new header

```
headers.set('Content-type', 'application/json');
```

- append() - adds a new header

```
headers.append('Accept-Encoding', 'gzip, deflate');
```

- delete() - Removes specified header

```
headers.delete('Accept-Encoding');
```

- keys(), values(), entries() - Iterators

```
for (const entry of headers.entries()) {
  console.log(entry);
}
```

Putting It All Together - use Headers, Request, Response objects to together a typical example

```
const url = 'https://example.com/data';
const headers = new Headers({'content-type': 'text/plain', ...});
const request = (url, {headers: headers})

fetch(request)
then(function(response) {
  if (response.ok) {
   return response;
  }
throw Error(response.statusText);
})
.then(response => //do something with response)
.catch(error => console.log('ERROR'))
```

W07 - AJAX

- Receiving Information

  SEE ajax.html, ajaxExample.js in week7

  rewrote a little of the example code, adding notes to explain the processes

- Sending Information. we can also use AJAX to send information.

  SEE todoAjax.html and todoAjaxExample.js

  most forms have an action attribute that specifies the URL to use if the form is sent without using AJAX and a method attribute that will specify the HTTP verb to use.

  More generalize request using the form.attributes:

```
const request = new Request (form.action,
 {
   method: form.method,
   headers: headers,
   body: data
 }
)
```

FORM DATA - Fetch API includes FormData interface, making it easier to submit information in forms using AJAX.

FormData object is created using a constructor function

```
const data = new FormData();
```

If a form is passed to this constructor function as an argument, the form data instance is serialized automatically.

This reduces the amount of code necessary when submitting forms.

FormData is particularly useful for file uploads.