NAME: SAMUEL OWUSU

STUDENT ID: 050919007

COURSE TITLE : WEB API

COURSE CODE: GTU304CEM

ASSIGNMENT

1. LIST ALL THE PROPERTIES OF THE XMLHTTPREQUEST CLASS.

The XMLHttpRequest Class Properties

Property	Description
onreadystatechange/td>	Defines a callback function that the browser triggers when the HTTP connection changes state
readyState	Contains the connection status of the HTTP connection
responseText	Contains the response sent by the web server in text format
responseXML	Contains the response sent by the web server in XML format
status	Contains the numeric HTTP response code from the web server
statusText	Contains the text HTTP response string from the web server

2. RESEARCH ON ALL REQUEST STATUS CODES WITH THEIR MEANINGS.

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

i. Informational responses (100–199)
ii. Successful responses (200–299)
iii. Redirection messages (300–399)
iv. Client error responses (400–499)
v. Server error responses (500–599)

The below status codes are defined by section 10 of RFC 2616. You can find an updated specification in RFC 7231.

Note: If you receive a response that is not in this list, it is a non-standard response, possibly custom to the server's software.

i. INFORMATION RESPONSES

100 Continue

This interim response indicates that the client should continue the request or ignore the response if the request is already finished.

101 Switching Protocols

This code is sent in response to an Upgrade request header from the client and indicates the protocol the server is switching to.

102 Processing (WebDAV)

This code indicates that the server has received and is processing the request, but no response is available yet.

103 Early Hints

This status code is primarily intended to be used with the Link header, letting the user agent start preloading resources while the server prepares a response.

ii. SUCCESSFUL RESPONSES

200 OK

The request succeeded. The result meaning of "success" depends on the HTTP method:

- GET: The resource has been fetched and transmitted in the message body.
- HEAD: The representation headers are included in the response without any message body.
- PUT or POST: The resource describing the result of the action is transmitted in the message body.
- TRACE: The message body contains the request message as received by the server.

201 Created

The request succeeded, and a new resource was created as a result. This is typically the response sent after POST requests, or some PUT requests.

202 Accepted

The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

203 Non-Authoritative Information

This response code means the returned metadata is not exactly the same as is available from the origin server, but is collected from a local or a third-party copy. This is mostly used for mirrors or backups of another resource. Except for that specific case, the 200 OK response is preferred to this status.

204 No Content

There is no content to send for this request, but the headers may be useful. The user agent may update its cached headers for this resource with the new ones.

205 Reset Content

Tells the user agent to reset the document which sent this request.

206 Partial Content

This response code is used when the Range header is sent from the client to request only part of a resource.

207 Multi-Status (WebDAV)

Conveys information about multiple resources, for situations where multiple status codes might be appropriate.

208 Already Reported (WebDAV)

Used inside a <dav:propstat> response element to avoid repeatedly enumerating the internal members of multiple bindings to the same collection.

226 IM Used (HTTP Delta encoding)

The server has fulfilled a GET request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.

iii. REDIRECTION MESSAGES

300 Multiple Choice

The request has more than one possible response. The user agent or user should choose one of them. (There is no standardized way of choosing one of the responses, but HTML links to the possibilities are recommended so the user can pick.)

301 Moved Permanently

The URL of the requested resource has been changed permanently. The new URL is given in the response.

302 Found

This response code means that the URI of requested resource has been changed *temporarily*. Further changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.

303 See Other

The server sent this response to direct the client to get the requested resource at another URI with a GET request.

304 Not Modified

This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same cached version of the response.

305 Use Proxy

Defined in a previous version of the HTTP specification to indicate that a requested response must be accessed by a proxy. It has been deprecated due to security concerns regarding in-band configuration of a proxy.

306 unused

This response code is no longer used; it is just reserved. It was used in a previous version of the HTTP/1.1 specification.

307 Temporary Redirect

The server sends this response to direct the client to get the requested resource at another URI with same method that was used in the prior request. This has the same semantics as the 302 Found HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a POST was used in the first request, a POST must be used in the second request.

308 Permanent Redirect

This means that the resource is now permanently located at another URI, specified by the Location: HTTP Response header. This has the same semantics as the 301 Moved Permanently HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a POST was used in the first request, a POST must be used in the second request.

iv. CLIENT ERROR RESPONSES

400 Bad Request

The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

401 Unauthorized

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

402 Payment Required

This response code is reserved for future use. The initial aim for creating this code was using it for digital payment systems, however this status code is used very rarely and no standard convention exists.

403 Forbidden

The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401 Unauthorized, the client's identity is known to the server.

404 Not Found

The server can not find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 Forbidden to hide the existence of a resource from an unauthorized client. This response code is probably the most well known due to its frequent occurrence on the web.

405 Method Not Allowed

The request method is known by the server but is not supported by the target resource. For example, an API may not allow calling DELETE to remove a resource.

406 Not Acceptable

This response is sent when the web server, after performing server-driven content negotiation, doesn't find any content that conforms to the criteria given by the user agent.

407 Proxy Authentication Required

This is similar to 401 Unauthorized but authentication is needed to be done by a proxy.

408 Request Timeout

This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection. This response is used much more since some browsers, like Chrome, Firefox 27+, or IE9, use HTTP pre-connection mechanisms to speed up surfing. Also note that some servers merely shut down the connection without sending this message.

409 Conflict

This response is sent when a request conflicts with the current state of the server.

410 Gone

This response is sent when the requested content has been permanently deleted from server, with no forwarding address. Clients are expected to remove their caches and links to the resource. The HTTP specification intends this status code to be used for "limited-time, promotional services". APIs should not feel compelled to indicate resources that have been deleted with this status code.

411 Length Required

Server rejected the request because the Content-Length header field is not defined and the server requires it.

412 Precondition Failed

The client has indicated preconditions in its headers which the server does not meet.

413 Payload Too Large

Request entity is larger than limits defined by server. The server might close the connection or return an Retry-After header field.

414 URI Too Long

The URI requested by the client is longer than the server is willing to interpret.

415 Unsupported Media Type

The media format of the requested data is not supported by the server, so the server is rejecting the request.

416 Range Not Satisfiable

The range specified by the Range header field in the request cannot be fulfilled. It's possible that the range is outside the size of the target URI's data.

417 Expectation Failed

This response code means the expectation indicated by the Expect request header field cannot be met by the server.

418 I'm a teapot

The server refuses the attempt to brew coffee with a teapot.

421 Misdirected Request

The request was directed at a server that is not able to produce a response. This can be sent by a server that is not configured to produce responses for the combination of scheme and authority that are included in the request URI.

422 Unprocessable Entity (WebDAV)

The request was well-formed but was unable to be followed due to semantic errors.

423 Locked (WebDAV)

The resource that is being accessed is locked.

424 Failed Dependency (WebDAV)

The request failed due to failure of a previous request.

425 Too Early

Indicates that the server is unwilling to risk processing a request that might be replayed.

426 Upgrade Required

The server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol. The server sends an Upgrade header in a 426 response to indicate the required protocol(s).

428 Precondition Required

The origin server requires the request to be conditional. This response is intended to prevent the 'lost update' problem, where a client GETs a resource's state, modifies it and PUTs it back to the server, when meanwhile a third party has modified the state on the server, leading to a conflict.

429 Too Many Requests

The user has sent too many requests in a given amount of time ("rate limiting").

431 Request Header Fields Too Large

The server is unwilling to process the request because its header fields are too large. The request may be resubmitted after reducing the size of the request header fields.

451 Unavailable For Legal Reasons

The user agent requested a resource that cannot legally be provided, such as a web page censored by a government.

v. SERVER ERROR RESPONSES

500 Internal Server Error

The server has encountered a situation it does not know how to handle.

501 Not Implemented

The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.

502 Bad Gateway

This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.

503 Service Unavailable

The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded. Note that together with this response, a user-friendly page explaining the problem should be sent. This response should be used for temporary conditions and the Retry-After HTTP header should, if possible, contain the estimated time before the recovery of the service. The webmaster must also take care about the caching-related headers that are sent along with this response, as these temporary condition responses should usually not be cached.

504 Gateway Timeout

This error response is given when the server is acting as a gateway and cannot get a response in time.

505 HTTP Version Not Supported

The HTTP version used in the request is not supported by the server.

506 Variant Also Negotiates

The server has an internal configuration error: the chosen variant resource is configured to engage in transparent content negotiation itself, and is therefore not a proper end point in the negotiation process.

507 Insufficient Storage (WebDAV)

The method could not be performed on the resource because the server is unable to store the representation needed to successfully complete the request.

508 Loop Detected (WebDAV)

The server detected an infinite loop while processing the request.

510 Not Extended

Further extensions to the request are required for the server to fulfill it.

511 Network Authentication Required

Indicates that the client needs to authenticate to gain network access.

3. LIST ALL THE CONTENT TYPES OF A HTTP AJAX REQUEST

a. XHR

XMLHttpRequest is an object such as (a native component in most other browsers, an ActiveX object in Microsoft Internet Explorer) that permits a web page to make a request to a server and get a response back without reloading the entire page. The user continues on the same page that is page did not reload, and more importantly, they will not actually see or notice the processing occur, they will not see a new page loading, not by default at least.

Using the XMLHttpRequest object makes it possible for a developer to change a page earlier loaded in the browser with data from the server without having to request the full page from the server again.

Performing GET request using XHR

```
const Http = new XMLHttpRequest();

const url='http://yourdomain.com/';

Http.open("GET", url);

Http.send();

Http.onreadystatechange=(e)=>{

console.log(Http.responseText)

}
```

Performing Post request using XHR

```
var xhr = new XMLHttpRequest();

xhr.open("POST", '/submit', true);

xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

xhr.onreadystatechange = function() {

   if (this.readyState === XMLHttpRequest.DONE && this.status === 200) {

      // Request finished. Do processing here.

   }
}

xhr.send("name=Ketan&id=1");
```

b. Fetch API

Fetch API is the new choice to XMLHttpRequest for getting resources from the server. Unlike XMLHttpRequest, it has a more powerful feature set and a more meaningful name. Fetch is also flexible and easy to use because of its syntax and structure. However, the thing that sets it apart from other AJAX HTTP libraries is that it is supported by all modern web browsers. Fetch follows a request-response approach where Fetch makes a request and returns a promise that resolves to the Response object.

Pros of using Fetch API

It’s flexible and easy to use

Callback hell is avoided by Promises

Supported by all modern browsers

Follows request-response approach

Cons of using Fetch API

Doesn’t send cookie by default

CORS is disabled by default

Performing GET Request in Fetch API

```
fetch('https://www.yourdomain.com', {

      method: 'get'

   })

   .then(response => response.json())

   .then(jsonData => console.log(jsonData))

   .catch(err => {

         //error block

      }
```

Performing POST Request in Fetch API

```
var url = 'https://www.yourdomain.com/updateProfile';

var data = {username: 'courseya'};
```

```
fetch(url, {
  method: 'POST', // or 'PUT'
  body: JSON.stringify(data), // data can be `string` or {object}!
  headers:{
    'Content-Type': 'application/json'
  }
}).then(res => res.json())
.then(response => console.log('Success:', JSON.stringify(response)))
.catch(error => console.error('Error:', error));
```

c. jQuery

jQuery is a client-side programming language you can be used to create cool and amazing web applications. It's free, yet powerful, comparatively easy to set up and learn, and it has multiple extensions and plugins available to do anything you could imagine or think off. You can get started quickly, and you won't outgrow it later when you get really good at it.

Pros of using Jquery

The greatest advantage of jQuery is its simplicity.

It is also incredibly flexible because jQuery allows users to add plug-ins.

It is also a very fast solution to your problems. While there may be "better" solutions, jQuery and its development teamwork to make sure you can implement jQuery quickly and effectively, which saves money.

Open-source software means quick growth and the freedom of developers to provide the best service possible without corporate red tape.

Cons of using Jquery

It is also, frequent updates mean community members are also unlikely to provide solutions.

There are also multiple varieties of jQuery available currently and some are less compatible than others.

Sometimes jQuery is slower compared to CSS in some cases. At that time its simplicity becomes a curse, as it is not meant for client-side interactions.

Jquery Interview Questions

Performing GET Request in Jquery

```
$.ajax({
    url: '/users',
    type: "GET",
    dataType: "json",
    success: function (data) {
        console.log(data);
    },
    error: function (error) {
        console.log(`Error ${error}`);
    }
});
```

Performing POST Request in Jquery

```
$.ajax({
    url: '/users',
    type: "POST",
    data: {
        name: "Ipseeta",
        id: 1
    },
    dataType: "json",
    success: function (data) {
        console.log(data);
    },
    error: function (error) {
        console.log(`Error ${error}`);
    }
});
```

d. Axios

Axios is one among many available promise-based HTTP client that works both in the browser and in a node.js environment. It basically provides a single API for dealing with XMLHttpRequest s and node's HTTP interface. Apart from that, it binds the requests using a polyfill for ES6 new's promise syntax.

Advantages of using Axios

Out-of-the-box Promise support

Client-side support for protecting against XSRF

Can capture requests or responses before they are carried out.

Automatic transforms for JSON data

Supports Promise API

Can set or cancel a request

Can set a response timeout

It works on both Nodejs and Browser

Performing GET Request in Axios

```
axios.get('/get-user', {
    params: {
      ID: 1
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  })
  .then(function () {
    // always executed
  });
```

Performing POST Request in Axios

```
axios.post('/user', {
    name: 'Sanjeev',
    id: 1
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

e. Request

The Request library is one of the simplest ways to make HTTP calls. The structure and syntax are very similar to that of how requests are handled in Node.js. Currently, the project has 18K stars on GitHub and deserves a mention for being one of the popular HTTP libraries available.

Syntax

```
var request = require('request');
request('http://www.yourdomain.com', function (error, response, body) {
  console.log('error:', error);
  console.log('statusCode:', response && response.statusCode);
  console.log('body:', body);
});
```

f. SuperAgent

SuperAgent is a lightweight and progressive AJAX library that's focused more on readability and flexibility. SuperAgent also boasts of a gentle learning curve unlike other libraries out there. SuperAgent has a request object that accepts methods such as GET, POST, PUT, DELETE, and HEAD.

Pros of SuperAgent

It has a plugin-based environment and ecosystem where plugins could be built and developed for extra or additional functionality.

Easily Configurable.

Nice interface for making HTTP requests.

Multiple functions chaining to send requests.

Has to support for upload and download progress.

Has support for chunked transfer encoding.

Old-style callbacks are supported

Numerous plugins available for many common features

Performing Get Request

```
request
   .get('/user')
   .query({ id: 1 })
   .then(res => {
   });
```

Performing Post Request

```
request.post('/user')
    .set('Content-Type', 'application/json')
    .send('{"name":"Ipseeta","id":1}')
    .then(callback)
    .catch(errorCallback)
```