Make an HTTP Request

In order to make an HTTP request to the server with JavaScript, you need an instance of an object with the necessary functionality. This is where XMLHttpRequest comes in.

- Create XMLHttpRequest Object

```
const xhr = new XMLHttpRequest ();
```

- Initialize created request

```
xhr.open(“GET”, “data.txt”, true);
```

- Send The Request

```
xhr.send();
```

Make an HTTP Request

In order to make an HTTP request to the server with JavaScript, you need an instance of an object with the necessary functionality. This is where XMLHttpRequest comes in.

- Create XMLHttpRequest Object

```
const xhr = new XMLHttpRequest ();
```

- Initialize created request

```
xhr.open("POST", "signupform.php", true);
```

- If method type POST then you may have to set Request Header before sending request

```
xhr.setRequestHeader('Context-Type', 'application/JSON')
```

- Send The Request

```
xhr.send(body);
```

Handling Server Response

After making a request, you will receive a response back.

At this stage, you need to tell the XMLHttpRequest object which JavaScript function will handle the response, by setting the onreadystatechange property of the object and naming it after the function to call when the request changes state.

```
xhr.onreadystatechange = nameOfTheFunction;


function nameOfTheFunction() {

}
```

Using Anonymous Function

```
xhr.onreadystatechange = function(){
   // Process the server response here.
};
```

Using Arrow Function

```
xhr.onreadystatechange = () => {
   // Process the server response here.
};
```

Handling Server Response

```
xhr.onreadystatechange = showdata;
function showdata() {
}
```

Using Anonymous Function

```
xhr.onreadystatechange = function(){
   // Process the server response here.
};
```

Using Arrow Function

```
xhr.onreadystatechange = () => {
   // Process the server response here.
};
```

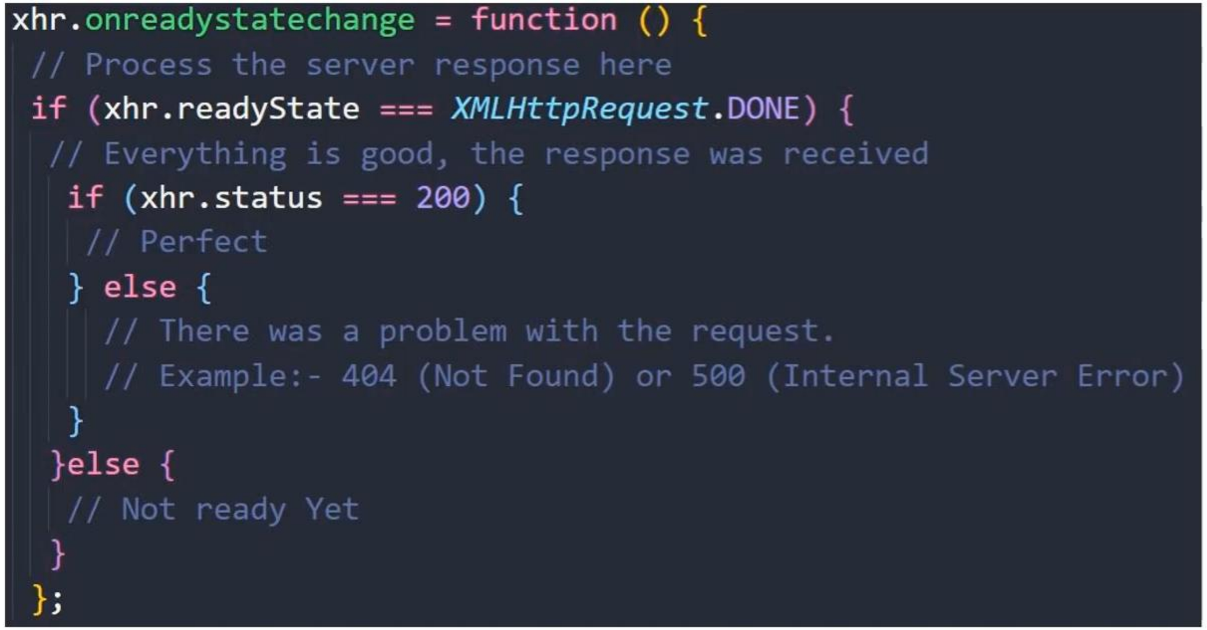
Response Handling Function

First, the function needs to check the request's state. If the state has the value of XMLHttpRequest.DONE (corresponding to 4), that means that the full server response was received and it's OK for you to continue processing it.

```
xhr.onreadystatechange = function(){
        // Process the server response here.
        if (xhr.readyState === XMLHttpRequest.DONE) {
                // Everything is good, the response was received.
        } else {
                // Not ready yet.
        }
};
```

Response Handling Function

Next, check the HTTP response status codes of the HTTP response. we differentiate between a successful and unsuccessful AJAX call by checking for a 200 OK response code.



```
xhr.onreadystatechange = function () {
 // Process the server response here
 if (xhr.readyState === XMLHttpRequest.DONE) {
  // Everything is good, the response was received
   if (xhr.status === 200) {
    // Perfect
   } else {
     // There was a problem with the request.
     // Example:- 404 (Not Found) or 500 (Internal Server Error)
   }
  }else {
   // Not ready Yet
  }
 };
```

Response Handling Function

After checking the state of the request and the HTTP status code of the response, you can do whatever you want with the data the server sent. You have two options to access that data:

- xhr.responseText – It returns the server response as a string of text
- xhr.responseXML – It returns the response as an XMLDocument object you can traverse with JavaScript DOM functions

Ajax Request Response

```
const xhr = new XMLHttpRequest();
 xhr.open("GET", "data.txt", true);
 xhr.onreadystatechange = function () {
  if (xhr.readyState === XMLHttpRequest.DONE) {
   if (xhr.status === 200) {
    console.log(xhr);
    console.log(xhr.responseText);
   } else {
    console.log("Problem with Request");
   }
  }
 };
 xhr.send();
```

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'data.txt', true);
xhr.onload = function(){
   if (xhr.status === 200) {
         console.log(xhr)
         console.log(xhr.responseText);
    } else {
          console.log('Problem with Request');
    }
};
xhr.send();
```

Ajax Request Response

```
const xhr = new XMLHttpRequest();
 xhr.open("GET", "data.txt", true);
 xhr.onreadystatechange = () => {
   if (xhr.readyState === XMLHttpRequest.DONE) {
     if (xhr.status === 200) {
       console.log(xhr);
       console.log(xhr.responseText);
     } else {
       console.log("Problem with Request");
     }
   }
 };
 xhr.send();
```

XMLHttpRequestEventTarget

XMLHttpRequestEventTarget is the interface that describes the event handlers you can implement in an object that will handle events for an XMLHttpRequest.

- onload
- onprogress
- onerror
- onloadstart
- onabort
- ontimeout
- onloadend

onload

The XMLHttpRequestEventTarget.onload is the function called when an XMLHttpRequest transaction completes successfully.

Syntax:-

XMLHttpRequest.onload = callback;

Example:-

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'data.txt', true);
xhr.onload = function(){
  if (xhr.status === 200) {
          console.log(xhr.responseText);
   }
};
xhr.send();
```

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'data.txt', true);
xhr.onload = () => {
  if (xhr.status === 200) {
          console.log(xhr.responseText);
   }
};
xhr.send();
```

onprogress

The XMLHttpRequestEventTarget.onprogress is the function called periodically with information when an XMLHttpRequest before success completely.

Syntax:- XMLHttpRequest.onprogress = callback;

Example:-

```
xhr.onprogress = function(e){
        e.loaded;        // The Amount of data currently transferred.
        e.total;         // The Total Amount of Data to be transferred.
};


xhr.onprogress = (e) => {
        e.loaded;        // The Amount of data currently transferred.
        e.total;         // The Total Amount of Data to be transferred.
};
```

onerror

The XMLHttpRequestEventTarget.onerror is the function called when an XMLHttpRequest transaction fails due to an error.

Syntax:- XMLHttpRequest.onerror = callback;

Example:-

```
xhr.onerror = function(){
            console.log('Error has occured')
};
```

onloadstart

The XMLHttpRequestEventTarget.onloadstart is the function called when an XMLHttpRequest transaction starts transferring data.

Syntax:- XMLHttpRequest.onloadstart = callback;

Example:-

```
xhr.onloadstart = function(){
        console.log('The transaction started…')
};
```

onloadend

The XMLHttpRequestEventTarget.onloadend is the function called when an XMLHttpRequest transaction ends transferring data.

Syntax:- XMLHttpRequest.onloadend= callback;

Example:-

```
xhr.onloadend = function(){
        console.log('The transaction end !!')
};
```

onabort

The XMLHttpRequestEventTarget.onabort is the function called when an XMLHttpRequest transaction is aborted, such as when the XMLHttpRequest.abort() function is called.

Syntax:- XMLHttpRequest.onabort= callback;

Example:-

```
xhr.onabort = function(){
        console.log('Terminated !!')
};
```

ontimeout

The function that is called if the event times out and the timeout event is received by this object; this only happens if a timeout has been previously established by setting the value of the XMLHttpRequest object's timeout attribute.

Syntax:- XMLHttpRequest.ontimeout= callback;

Example:-

```
xhr.ontimeout = function(){
        console.log('Time out !!')
};
```