AJAX Introduction

AJAX is a developer's dream, because you can:

* Update a web page without reloading the page
* Request data from a server - after the page has loaded
* Receive data from a server - after the page has loaded
* Send data to a server - in the background

**Making an Ajax call from JavaScript** means sending an asynchronous request to a server to fetch or send data without reloading the web page. This allows dynamic content updates, enhancing user experience by making the web application more interactive and responsive.

**Table of Content**

* [Using the XMLHttpRequest object](https://www.geeksforgeeks.org/how-to-make-ajax-call-from-javascript/#approach-1-using-the-xmlhttprequest-object)
* [Using jQuery](https://www.geeksforgeeks.org/how-to-make-ajax-call-from-javascript/#approach-2-using-jquery)
* [Using fetch() API](https://www.geeksforgeeks.org/how-to-make-ajax-call-from-javascript/#approach-3-using-fetch-api)

We’ll walk through each method with simple examples to help you understand how to use them in your projects.

## **Approach 1: Using the XMLHttpRequest object**

In this approach, we will use the [XMLHttpRequest object](https://www.geeksforgeeks.org/ajax-full-form/) to make an Ajax call. The**[XMLHttpRequest()](https://www.geeksforgeeks.org/how-to-make-put-request-using-xmlhttprequest-by-making-custom-http-library/)** method creates an XMLHttpRequest object which is used to make a request with the server.

**Syntax:**

let xhttp = new XMLHttpRequest();

Above syntax is used to create an XMLHttpRequest object. This object has many different methods used to interact with the server to send, receive or interrupt responses from the server. In the response, we get a string from the server that we print.

For a complete guide on AJAX and asynchronous JavaScript, our [**JavaScript Course**](https://gfgcdn.com/tu/T2i/) provides tutorials on making API calls and handling responses efficiently.

**Example:**In this example, we will use the [XMLHttpRequest object](https://www.geeksforgeeks.org/ajax-full-form/) to make an Ajax call.

* The XMLHttpRequest object is used to make an asynchronous GET request.
* It sends a request to a specified URL and waits for a response.
* On success, the JSON response is logged to the console.

JavaScript

**function** run() {

*// Creating Our XMLHttpRequest object*

**let** xhr = **new** XMLHttpRequest();

*// Making our connection*

**let** url = 'https://jsonplaceholder.typicode.com/todos/1';

xhr.open("GET", url, **true**);

*// function execute after request is successful*

xhr.onreadystatechange = **function** () {

**if** (**this**.readyState == 4 && **this**.status == 200) {

console.log(**this**.responseText);

}

}

*// Sending our request*

xhr.send();

}

run();

**Output:**

"{  
 "userId": 1,  
 "id": 1,  
 "title": "delectus aut autem",  
 "completed": false  
}"

## **Approach 2: Using jQuery**

In this approach, we will use [jQuery](https://www.geeksforgeeks.org/jquery-tutorial/" \t "_blank)to make an Ajax call. The[**ajax()**](https://www.geeksforgeeks.org/jquery-ajax-method/)method is used in jQuery to make ajax calls. It is used as a replacement for all approaches which are not working to make ajax calls.

**Syntax:**

$.ajax({arg1: value, arg2: value, ... });

**Parameter:**It takes a configuration file that configures the URL, type, and function to call when we get our response or if error, etc.

**Example:**In this example

* The ajaxCall() function makes an AJAX GET request to a sample URL using jQuery.
* On success, the response data is converted to a JSON string and logged to the console.
* If the request fails, an error message is logged to the console.

HTML

<!DOCTYPE HTML>

<**html**>

<**head**>

<**script** src=

"https://code.jquery.com/jquery-3.6.0.min.js">

</**script**>

</**head**>

<**body**>

<**script**>

**function** ajaxCall() {

$.ajax({

*// Our sample url to make request*

url:

'https://jsonplaceholder.typicode.com/todos/1',

*// Type of Request*

type: "GET",

*// Function to call when to*

*// request is ok*

success: **function** (data) {

**let** x = JSON.stringify(data);

console.log(x);

},

*// Error handling*

error: **function** (error) {

console.log(`Error **${**error**}**`);

}

});

}

ajaxCall();

</**script**>

</**body**>

</**html**>

**Output:**

{  
 "userId": 1,  
 "id": 1,  
 "title": "delectus aut autem",  
 "completed": false  
}

## **Approach 3: Using fetch() API**

In this approach, we will use [**fetch()**](https://www.geeksforgeeks.org/fetch-api/)API which is used to make XMLHttpRequest with the server. Because of its flexible structure, it is easy to use. This API makes a request to the server and gets the result as a [promise](https://www.geeksforgeeks.org/javascript-promises/) which is resolved to the string.

**Syntax:**

fetch(url, {config}).then().catch();

**Parameter:**It takes URL and config of the request as parameters.

We will configure the data required and make the request to the server. Since it is a resolved promise we use [**then()**](https://www.geeksforgeeks.org/why-we-use-then-method-in-javascript/)function and the **catch()** function to create output for the result. In response, we get the string that we print.

**Example:**In this example, we will use [**fetch()**](https://www.geeksforgeeks.org/fetch-api/)API to make XMLHttpRequest with the server.

* A GET request is made to the specified URL using the fetch() API.
* The JSON response is logged to the console, and the title field is specifically printed.
* Errors during the request are caught and logged to the console.

JavaScript

*// Url for the request*

**let** url = 'https://jsonplaceholder.typicode.com/todos/1';

*// Making our request*

fetch(url, { method: 'GET' })

.then(Result => Result.json())

.then(string => {

*// Printing our response*

console.log(string);

*// Printing our field of our response*

console.log(`Title of our response : **${**string.title**}**`);

})

.**catch**(errorMsg => { console.log(errorMsg); });

**Output:**

{ userId:1 ,id:1 ,title : "delectus aut autem" ,completed : false  
\_\_proto\_\_:Object }  
Title of our response : delectus aut autem