Cross-Origin Resource Sharing (CORS) is a protocol that enables scripts running on a browser client to interact with resources from a different origin. This is useful because, thanks to the [same-origin policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy) followed by XMLHttpRequest and fetch, JavaScript can only make calls to URLs that live on the same origin as the location where the script is running. For example, if a JavaScript app wishes to make an AJAX call to an API running on a different domain, it would be blocked from doing so thanks to the same-origin policy.

Most of the time, a script running in the user's browser would only ever need to access resources on the same origin (think about API calls to the same backend that served the JavaScript code in the first place). So the fact that JavaScript can't normally access resources on other origins is a good thing for security.

In this context, "other origins" means the URL being accessed differs from the location that the JavaScript is running from, by having:

* a different scheme (HTTP or HTTPS)
* a different domain
* a different port

However, there are legitimate scenarios where cross-origin access is desirable or even necessary. For example, if you're running a React SPA that makes calls to an API backend runningon a different domain. [Web fonts also rely on CORS work](https://www.w3.org/TR/css-fonts-3/#font-fetching-requirements).

**Identifying a CORS Response**

When a server has been configured correctly to allow cross-origin resource sharing, some special headers will be included. Their presence can be used to determine that a request supports CORS. Web browsers can use these headers to determine whether or not an XMLHttpRequest call should continue or fail.

There are [a few headers that can be set](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#The_HTTP_response_headers), but the primary one that determines who can access a resource is [Access-Control-Allow-Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Access-Control-Allow-Origin). This header specifies which origins can access the resource. For example, to allow access from any origin, you can set this header as follows:

Access-Control-Allow-Origin: \*

Access-Control-Allow-Origin: <https://example.com>

## Understanding CORS Request Types

There are two types of CORS request: "simple" requests, and "preflight" requests, and it's the browser that determines which is used. As the developer, you don't normally need to care about this when you are constructing requests to be sent to a server. However, you may see the different types of requests appear in your network log and, since it may have a performance impact on your application, it may benefit you to know why and when these requests are sent.

Let's have a look at what that means in more detail in the next couple of sections.

### **Simple requests (GET, POST, and HEAD)**

The browser deems the request to be a ["simple" request](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#Simple_requests) when the request itself meets a certain set of requirements:

* One of these methods is used: GET, POST, or HEAD
* A [CORS safe-listed header](https://fetch.spec.whatwg.org/#cors-safelisted-request-header) is used
* When using the Content-Type header, only the following values are allowed: application/x-www-form-urlencoded, multipart/form-data, or text/plain
* No event listeners are registered on any XMLHttpRequestUpload object
* No [ReadableStream](https://developer.mozilla.org/en-US/docs/Web/API/ReadableStream" \t "_blank) object is used in the request

The request is allowed to continue as normal if it meets these criteria, and the Access-Control-Allow-Origin header is checked when the response is returned.

### **Preflight requests (OPTIONS)**

If a request does not meet the criteria for a **simple** request, the browser will instead make an automatic [preflight request](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS" \l "Preflighted_requests" \t "_blank) using the OPTIONS method. This call is used to determine the exact CORS capabilities of the server, which is in turn used to determine whether or not the intended CORS protocol is understood. If the result of the OPTIONS call dictates that the request cannot be made, the actual request to the server will not be executed.

The preflight request sets the mode as OPTIONS and sets a couple of headers to describe the actual request that is to follow:

* Access-Control-Request-Method: The intended method of the request (e.g., GET or POST)
* Access-Control-Request-Headers: An indication of the custom headers that will be sent with the request
* Origin: The usual origin header that contains the script's current origin

# **Which CORS headers do you need to send an Authorization header?**

Sep 12, 2018

In cross origin requests, the authorization header can be sent in two ways: either by the browser or specified along with the request. This article explains which CORS headers you need for each.

## Authorization header

The Authorization HTTP header provides authentication information on a request. There are several types of authentication that use this header, and some are supported by browsers, such as [basic authentication](https://en.wikipedia.org/wiki/Basic_access_authentication). When an unauthenticated request is received by the server, it will respond with a HTTP 401 Unauthorized response with a WWW-Authenticate header. This will trigger the browser to ask the user for credentials. The browser will then perform the same request, but include an Authorization header with the entered credentials.

In contrast, some applications use the Authorization header without any intervening from the browser. A JavaScript app may obtain a token from the server and send that with each request to authenticate the request. This if called bearer authentication and the Authorization header is often used to send the token.

## Cross origin access with credentials

If you want to send an Authorization header along with a request to another site, that site has to notify the browser that that is permitted. After all, sites can’t just access each other’s pages. It would be insecure if this site could perform an AJAX request to your bank’s site, using the cookies from your browser. However, there are some use cases for cross-site access. In that case, the CORS HTTP response headers can grant access to another site. These are response headers, so the application that handles the request has to give its OK that the response is used by another application.

## XHR requests with Authorization header

When performing a cross-origin request which includes authorization header, the server needs to respond with approval of the use of credentials. How this is done differs depending on whether the Authorization header is set by the browser or from your application.

### **By the browser**

Browsers support HTTP basic authentication as described above, where the browser asks for a username and password and sends it with every subsequent request. To use this, you need to enable credentials on your request. This will send cookies, client-side certificates, and basic authentication information in the Authorization header along with the request. To do this, you need three things:

* On the client, specify that you want to include credentials. Set [Request.credentials](https://developer.mozilla.org/en-US/docs/Web/API/Request/credentials) to include.
* On the server, respond with Access-Control-Allow-Credentials: true. This lets the client know that authenticated requests are permitted.
* On the server, respond with Access-Control-Allow-Origin header, containing the origin that performs the request. You must specify a URL, a wildcard won’t work with authenticated requests.