Engineer

CORS

how a server lets pages on other origins read its responses

What it is

CORS, written out as Cross-Origin Resource Sharing, is the set of rules by which a server tells a browser that pages from other websites may read what it sends back. Browsers keep origins apart. An origin is a scheme, a host and a port taken together, so `https://app.example.com`, `https://api.example.com` and `http://app.example.com` are three different origins, and script running on one of them may not read a response from another unless that other one agrees. CORS is how it agrees: a small family of HTTP headers, nearly all of them starting `Access-Control-`, that the server adds to its responses and the browser checks before handing anything to the calling script.

The protocol lives in the WHATWG Fetch Standard, the living specification of how browsers fetch resources. Its reasoning is stated plainly there. A page should be able to use data from another origin, and to send requests richer than an HTML form can, but only when the other side opts in. Without the opt-in, a page from the open web could read servers that sit behind a company firewall, or read a user's private data on another site with that user's cookies attached.

The core exchange is short. When script asks for a resource on another origin, the browser adds an `Origin` header naming the page's own origin. The server answers with `Access-Control-Allow-Origin`, whose value is either that origin, written exactly, or `*` for anybody. If it matches, the script receives the response. If it does not, the script receives a network error and nothing of the body.

Some requests need permission before they are sent at all. A `GET`, `HEAD` or `POST` with only a few ordinary headers and a body of type `application/x-www-form-urlencoded`, `multipart/form-data` or `text/plain` goes straight out, because an HTML form could send the same thing from any site already. Anything beyond

that, such as a `PUT`, a `DELETE`, a JSON body or an `Authorization` header, is preceded by a preflight: an `OPTIONS` request that names the method in `Access-Control-Request-Method` and any extra headers in `Access-Control-Request-Headers`. The real request follows only if the answer has an ok status, such as 200 or 204, and allows both.

Use the concept in a decision

Put “What is CORS?” to work in a real decision. We can help relate the concept to the system you are building or buying. Accounts, passwords and permissions arranged so people hold exactly the access their job needs, and leavers lose it that day.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

Almost every web application with a separate API meets CORS on the first day. The frontend is served from one origin and the API from another, and the first `fetch` from one to the other fails in the browser console until the API sends the right headers. Teams that meet it this way tend to remember it as an obstacle and to configure it as one, by allowing whatever makes the error go away. That is where the risk lives.

The headers decide who may read your users’ data through their browsers. A user who is logged in to your application carries a session cookie. If your API allows any origin to make credentialed requests and read the answers, then any page that user visits can call your API as them and read the result. The Fetch Standard is blunt about sharing credentialed responses: “extreme care has to be taken to avoid the confused deputy problem”. The browser, holding the user’s credentials, is the deputy, and a careless CORS policy lets a stranger’s page direct it.

It is just as important to know what CORS does not do. It does not protect the server. A simple request reaches the server and runs whether or not the response is later shared, so a `POST` that changes data has changed it before the browser looks at a single CORS header. Outside a browser the `Origin` header is whatever the caller types, and the OWASP testing guide warns that it may be

spoofed. Who may call an endpoint and what they may do there is decided by authentication and authorisation on the server, every time. CORS only decides which web pages may read the answer inside a browser.

What it looks like in practice

Suppose a single-page application at `https://app.example.com` saves a note through an API at `https://api.example.com`, using the session cookie the API set at login:

```
await fetch("https://api.example.com/v1/notes/42", {
  method: "PUT",
  credentials: "include",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ text: "Ship on Friday" }),
});
```

A `PUT` with a JSON body is more than a form could send, so the browser asks first. The preflight carries no cookies; it only names what the real request will use:

```
OPTIONS /v1/notes/42 HTTP/1.1
Host: api.example.com
Origin: https://app.example.com
Accept: */*
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: content-type
```

The API checks the origin against its own list and answers:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://app.example.com
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: PUT
Access-Control-Allow-Headers: content-type
Access-Control-Max-Age: 600
Vary: Origin
```

The browser now sends the `PUT` with the cookie and an `Origin` header, and the API's response repeats `Access-Control-Allow-Origin` and `Access-Control-Allow-Credentials: true`. Without both, the note is saved on the server but the script never learns it. `Access-Control-Max-Age` lets the browser reuse the preflight answer for up to 600 seconds instead of asking before every call; when the header is missing, the standard keeps it for five seconds. The `Vary: Origin` header matters because the answer depends on who asked, and a cache that ignored that could hand one origin's answer to another.

On a platform holding professional and organisational data, the API layer runs rate limiting, CORS, request-size limits and input validation before anything reaches the database. The same page explains why CORS is one entry in that list and not the whole defence: the platform assumes the client is hostile, and the server checks each request's entitlements again whatever the browser has allowed.

Where it goes wrong

The server copies any `Origin` back into `Access-Control-Allow-Origin`. It is the quickest way to make the console error disappear, and with `Access-Control-Allow-Credentials: true` it hands every website on the internet the user's view of the API. Keep an explicit list of the origins that should have access, compare the incoming `Origin` against it exactly, and send the headers only on a match.

The list is checked with a prefix or a substring. A rule that accepts any origin starting with `https://app.example.com` also accepts `https://app.example.com.attacker.example`, and a substring match is worse. Compare whole serialised origins. Note that a serialised origin has no path and no trailing slash, so `https://app.example.com/` in the list will never match anything.

`*` **is expected to work with credentials.** When a request includes credentials, the browser refuses `*` in `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` and `Access-Control-Expose-Headers`, and `Access-Control-Allow-Credentials` must be exactly `true`, in lower case. A public, read-only API with no cookies can safely answer `*`; an API that relies on a session has to name the origin.

`null` **is on the allowed list.** Pages loaded from a `data:` URL get an opaque origin, and a request that has been redirected across origins also reports `Origin: null`, so many unrelated documents present the same value. Allowing `null`

allows all of them. Leave it off.

`Vary: Origin` **is missing.** If the API sends `Access-Control-Allow-Origin` only when a request carries a matching `Origin`, a cache can store an answer without the header and serve it to a later CORS request, or store one origin's answer and serve it to another. The Fetch Standard asks for `Vary: Origin` whenever the answer depends on the origin. If the value is `*` or one fixed origin, send it on every response and leave `Vary` out.

`Access-Control-Allow-Headers: *` **is trusted to cover everything.** The wildcard never covers `Authorization`, which has to be listed by name, and it covers nothing at all for credentialed requests. Name the headers the client really sends.

CORS is treated as access control. A server that refuses requests because the `Origin` looks wrong, and accepts them otherwise, has written a check that any script outside a browser walks past. Authenticate the caller, authorise the action on the server, and let CORS do its one job, which is deciding what a browser shows to a page.

Further reading

- Fetch Standard: the CORS protocol — the definition: the request and response headers, the preflight, the credentials table and the advice on `Vary`.
- RFC 6454: The Web Origin Concept — what an origin is, how scheme, host and port decide it, and how it is serialised into the `Origin` header.
- OWASP Web Security Testing Guide: Testing Cross Origin Resource Sharing — how testers look for a wildcard origin or an origin echoed back without checks.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is CORS?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/cors/