**Engineer**

# CORS

decidir qué orígenes pueden leer respuestas en el navegador

## Qué es

CORS, abreviatura de Cross-Origin Resource Sharing, es el conjunto de reglas con el que un servidor indica al navegador que las páginas de otros sitios pueden leer sus respuestas. Los navegadores mantienen separados los orígenes. Un origen es la combinación de esquema, host y puerto, de modo que `https://app.example.com`, `https://api.example.com` y `http://app.example.com` son tres orígenes distintos y un script ejecutado en uno no puede leer una respuesta de otro salvo que este lo autorice. CORS expresa esa autorización mediante una pequeña familia de cabeceras HTTP, casi todas con el prefijo `Access-Control-`, que el servidor añade a sus respuestas y que el navegador comprueba antes de entregar nada al script que hizo la llamada.

El protocolo forma parte de WHATWG Fetch Standard, la especificación viva de cómo obtienen recursos los navegadores. Expone claramente su razonamiento: una página debe poder usar datos de otro origen y enviar peticiones más complejas que las de un formulario HTML, pero solo cuando la otra parte lo autorice expresamente. Si no existiera ese requisito de autorización explícita, una página de la web abierta podría leer servidores situados tras el cortafuegos de una empresa o los datos privados de un usuario en otro sitio utilizando las cookies de ese usuario.

El intercambio básico es breve. Cuando un script solicita un recurso de otro origen, el navegador añade una cabecera `Origin` que identifica el origen de la propia página. El servidor responde con `Access-Control-Allow-Origin`, cuyo valor es ese origen escrito exactamente o `*` para cualquiera. Si coincide, el script recibe la respuesta. Si no, recibe un error de red y nada del cuerpo.

Algunas peticiones necesitan permiso antes incluso de enviarse. Un `GET`, `HEAD` o `POST` con únicamente unas pocas cabeceras habituales y un cuerpo de tipo `application/x-www-form-urlencoded`, `multipart/form-data` o `text/plain` sale directamente, porque un formulario HTML ya podría enviar lo mismo desde cualquier sitio. Todo lo que vaya más allá, como un `PUT`, `DELETE`, cuerpo JSON o cabecera `Authorization`, lleva antes una consulta previa, o preflight: una petición `OPTIONS` que indica el método en `Access-Control-Request-Method` y las cabeceras adicionales en `Access-Control-Request-Headers`. La petición real solo se envía si la respuesta tiene un estado de éxito, como 200 o 204, y permite tanto el método como las cabeceras.

Utilice el concepto en una decisión

Aplique «¿Qué es CORS?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. Cuentas, contraseñas y permisos organizados para que cada persona tenga exactamente el acceso que necesita y lo pierda el mismo día que deja la empresa.

**Explore nuestros servicios** Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. 1.000 reglas RBAC para servidores Linux — 1,000 reglas de acceso escritas para cerrar una brecha de seguridad

**1.000 reglas RBAC para servidores Linux** Póngase en contacto →

## Por qué importa

Casi toda aplicación web con una API separada se encuentra con CORS el primer día. El frontend se sirve desde un origen y la API desde otro, y el primer `fetch` entre ellos falla en la consola del navegador hasta que la API envía las cabeceras correctas. Los equipos que lo conocen así tienden a recordarlo como un obstáculo y a configurarlo como tal, permitiendo lo que haga desaparecer el error. Ahí reside el riesgo.

Las cabeceras deciden quién puede leer los datos de sus usuarios a través de sus navegadores. Un usuario que ha iniciado sesión en su aplicación lleva una cookie de sesión. Si la API permite que cualquier origen haga peticiones con credenciales y lea las respuestas, cualquier página que visite ese usuario puede llamar a la API en su nombre y leer el resultado. Fetch Standard es tajante sobre compartir respuestas a peticiones con credenciales: «hay que extremar las precauciones para evitar el problema del representante confundido». El navegador, que posee las credenciales del usuario, es el representante, y una política CORS descuidada permite que la página de un desconocido lo dirija.

Es igual de importante saber qué no hace CORS. No protege el servidor. Una petición simple llega al servidor y se ejecuta, se comparta o no después la respuesta, así que un `POST` que modifica datos ya lo ha hecho antes de que el navegador lea una sola cabecera CORS. Fuera de un navegador, la cabecera `Origin` contiene lo que escriba quien llama, y la guía de pruebas de OWASP advierte que puede falsificarse. Quién puede llamar a un endpoint y qué puede hacer en él lo deciden la autenticación y la autorización en el servidor, en cada ocasión. CORS solo decide qué páginas web pueden leer la respuesta dentro de un navegador.

## Cómo se aplica en la práctica

Suponga una aplicación de página única en `https://app.example.com` que guarda una nota mediante una API en `https://api.example.com`, utilizando la cookie de sesión que la API estableció al iniciar sesión:

```
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" }),
});
```

Un `PUT` con cuerpo JSON supera lo que podría enviar un formulario, así que el navegador pregunta antes. La consulta previa no lleva cookies; solo indica qué utilizará la petición real:

```
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
```

La API compara el origen con su propia lista y responde:

```
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
```

Ahora el navegador envía el `PUT` con la cookie y una cabecera `Origin`, y la respuesta de la API repite `Access-Control-Allow-Origin` y `Access-Control-Allow-Credentials: true`. Si falta cualquiera de las dos, la nota se guarda en el servidor, pero el script nunca lo sabe. `Access-Control-Max-Age` permite al navegador reutilizar la respuesta de la consulta previa durante un máximo de 600 segundos en lugar de preguntar en cada llamada; si falta la cabecera, el estándar la conserva cinco segundos. La cabecera `Vary: Origin` importa porque la respuesta depende del origen de la solicitud, y una caché que lo ignorara podría entregar a un origen la respuesta destinada a otro.

En una plataforma con datos profesionales y de organizaciones, la capa de API ejecuta limitación de frecuencia, CORS, límites del tamaño de petición y validación de entradas antes de que nada llegue a la base de datos. La misma página explica por qué CORS es un elemento de esa lista y no toda la defensa: la plataforma supone que el cliente es hostil y el servidor vuelve a comprobar los permisos de cada petición, independientemente de lo que haya permitido el navegador.

## Dónde surgen los problemas

**El servidor copia cualquier `Origin` en `Access-Control-Allow-Origin`.** Es la forma más rápida de hacer desaparecer el error de consola y, con `Access-Control-Allow-Credentials: true`, entrega a todos los sitios de Internet la vista de la API que tiene el usuario. Mantenga una lista explícita de los orígenes que deben tener acceso, compare exactamente el `Origin` entrante con ella y envíe las cabeceras solo cuando coincida.

**La lista se comprueba por prefijo o subcadena.** Una regla que acepta cualquier origen que empiece por `https://app.example.com` también acepta `https://app.example.com.attacker.example`, y buscar una subcadena es peor. Compare orígenes serializados completos. Un origen serializado no tiene ruta ni barra final, por lo que `https://app.example.com/` en la lista nunca coincidirá con nada.

**Se espera que `*` funcione con credenciales.** Cuando una petición incluye credenciales, el navegador rechaza `*` en `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` y `Access-Control-Expose-Headers`, y `Access-Control-Allow-Credentials` debe ser exactamente `true`, en minúsculas. Una API pública de solo lectura y sin cookies puede responder `*` de forma segura; una API que depende de una sesión debe nombrar el origen.

**Se incluye `null` en la lista permitida.** Las páginas cargadas desde una URL `data:` reciben un origen opaco, y una petición redirigida entre orígenes también indica `Origin: null`, por lo que muchos documentos sin relación presentan el mismo valor. Permitir `null` los permite a todos. Déjelo fuera.

**Falta `Vary: Origin`.** Si la API solo envía `Access-Control-Allow-Origin` cuando una petición lleva un `Origin` coincidente, una caché puede guardar una respuesta sin esa cabecera y servirla a una petición CORS posterior, o guardar la respuesta de un origen y servirla a otro. Fetch Standard pide `Vary: Origin` cuando la respuesta depende del origen. Si el valor es `*` o un origen fijo, envíelo en todas las respuestas y omita `Vary`.

**Se confía en que `Access-Control-Allow-Headers: *` cubra todo.** El comodín nunca cubre `Authorization`, que debe enumerarse por su nombre, y no cubre nada en las peticiones con credenciales. Indique las cabeceras que envía realmente el cliente.

**Se trata CORS como control de acceso.** Un servidor que rechaza peticiones porque `Origin` parece incorrecto y las acepta en caso contrario ha escrito una comprobación que cualquier script fuera del navegador puede eludir. Autentique a quien llama, autorice la acción en el servidor y deje a CORS su única tarea: decidir lo que muestra un navegador a una página.

## Lecturas adicionales

- Fetch Standard: el protocolo CORS — la definición: cabeceras de petición y respuesta, consulta previa, tabla de credenciales y recomendaciones sobre `Vary`.
- RFC 6454: el concepto de origen web — qué es un origen, cómo lo determinan esquema, host y puerto y cómo se serializa en la cabecera `Origin`.

- Guía de pruebas de seguridad web de OWASP: probar el intercambio de recursos entre orígenes — cómo se prueba si el servidor permite un origen comodín o devuelve un origen sin comprobarlo.

Glosario

Elija según sus restricciones reales

Describa la ruta de despliegue, los riesgos operativos y quiénes se encargan de las guardias. Podemos definir el alcance de un cambio con comprobaciones, una vía de recuperación y responsabilidades claras.

**Vea cuánto cuesta** Póngase en contacto →

## Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es CORS?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

**Explore nuestros servicios**

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

**Creado con esmero y cariño por D.**

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

**Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.**

**Impreso desde https://engineer.company/es/glossary/cors/**