CORS, Preflight Request, OPTIONS Method

Monday, 28 August 2023 4:54 PM

→ CORS — Cross origin resource sharing.

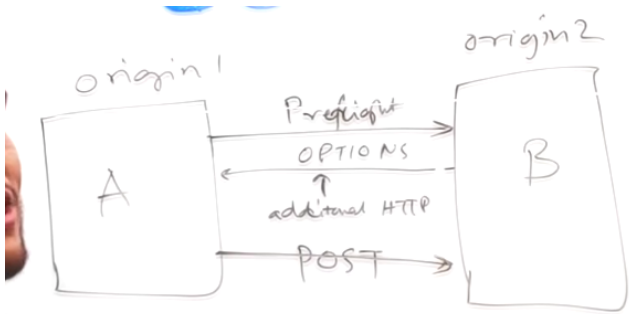
This is a mechanism which uses additional HTTP headers to tell the browser whether a specific webApp can share resource with another webApp. But, both webApps should have different origin. If they have same origin, they can share resources easily but if they have different origin, they must use CORS mechanism.

→ Before CORS, browsers never allowed webapps to share origins.
Eg: If yathaarthsuri.in wants to access data from google.com/api/getData then it wasn't allowed.
→ subdomains were also not allowed to access. Eg: api.yathaarthsuri.in
→ Different ports' access was not allowed. Eg yathaarthsuri.in was not allowed to access yathaarthsuri.in:5050
→ Different protocols were also not allowed. Eg: https://yathaarth.in could not access http://yathaarth.in

→ But how CORS has become a web standard so now we can share resources.

→ How does CORS or resource sharing between 2 websites work?
ans. Suppose if 2 webapps having different origins i.e domains want to share resources, a CORS preflight mechanism is followed.
So, a Preflight Options call is made before the actual API call.

So, if 'A' wants to post some data to 'B' then the browser makes a preflight call or an options call to 'B' i.e the server.
Then 'B' takes the responsibility of verifying whether this call is valid or not.
If call is valid then B will set some additional HTTP headers, then the browser will know that 'B' is safe & then the actual API call is made.



So, this is how resources are shared.

↳ Some majorly used additional headers are:

- Access-Control-Allow-Origin
  ↳ When we make a public API, so what we do is, server sets this above header as * which means that any domain outside of that domain can access this. We can also restrict access to specific domains, so then we give the value to above header as the domain name itself.
  eg: [Access-Control-Allow-Origin : *]
  eg: [Access-Control-Allow-Origin : https://yathaarth.suri.in]
  ↳ This will just allow this domain name to access the API or make call to B.

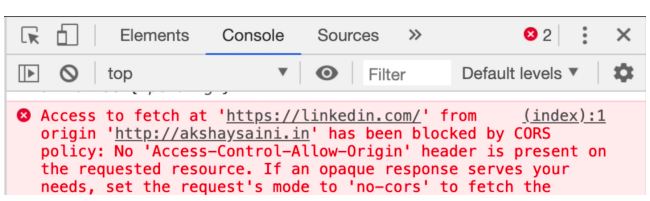
- Access-Control-Allow-Methods
  ↳ To restrict methods (eg: GET, PUT, POST, etc.)

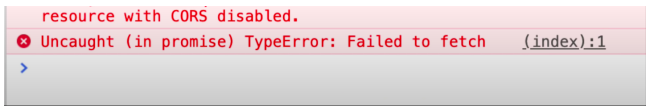
Note: This preflight call is not made for all API calls.
There are 2 types of access-control mechanisms:
1) Preflight requests 2) Simple requests
So, for some requests, browser already tags them as simple requests so the browser will not make a preflight call for them but directly calls it.
So what is a simple request?
A simple request is which meets all the following conditions.

A *simple request* is one that **meets all the following conditions**:

- One of the allowed methods:
  - `GET`
  - `HEAD`
  - `POST`
- Apart from the headers automatically set by the user agent (for example, `Connection`, `User-Agent`, or the other headers defined in the Fetch spec as a *forbidden header name*), the only headers which are allowed to be manually set are those which the Fetch spec defines as a CORS-safelisted request-header, which are:
  - `Accept`
  - `Accept-Language`
  - `Content-Language`
  - `Content-Type` (please note the additional requirements below)
  - `Range` (only with a simple range header value; e.g., `bytes=256-` or `bytes=127-255`)
- The only type/subtype combinations allowed for the media type specified in the `Content-Type` header are:
  - `application/x-www-form-urlencoded`
  - `multipart/form-data`
  - `text/plain`
- If the request is made using an `XMLHttpRequest` object, no event listeners are registered on the object returned by the `XMLHttpRequest.upload` property used in the request; that is, given an `XMLHttpRequest` instance `xhr`, no code has called `xhr.upload.addEventListener()` to add an event listener to monitor the upload.
- No `ReadableStream` object is used in the request.

→





This error comes when we make an error while following the CORS mechanism.
To resolve this, we can set the header from the server.