What is CORS?

Cross-Origin Resource Sharing

CORS is a browser behavior

- allowing JS-based service calls
- to endpoints that are on a different domain/port
- than the currently loaded page

This is done for security reasons.

Before CORS: Wild West

Why CORS?

Consider life *before* CORS:

1st try: browser JS can do anything, anywhere

- Security problems, particularly with cookies
- What if I call a service on your bank site from my cat videos webpage?
    - it would have your bank cookies, but is my JS

Same Origin Policy

2nd try: Same Origin Policy (SOP)

- Pages can only load resources from same "origin"
    - origin = (protocol + domain + port)
- Except for images, JS, and CSS files
    - don't break the existing web

Same Origin Policy not enough

SOP Secure, but people WANTED Cross-Origin

- Including their own subdomains
    - **http://example.com**, **http://api.example.com**
- Workarounds included JSONP
    - Hides service call as a JS file to load and run
    - Which is NOT secure
        - remote service runs JS on your page
            - remote service may not be yours!

Adopting CORS

3rd Try: CORS (Cross-Origin Resource Sharing)

- response headers say what the service allows
    - methods, headers, allowed origins
- browser refuses to give data to JS if not allowed
- ENFORCED BY BROWSER
    - no browser, no CORS enforcement
    - full security requires server-side enforcement

CORS Preflight

Non-"simple" requests send a "preflight" request

- Not GET/POST is non-simple
- Sending custom headers is non-simple
- Sending auth headers (like cookies) is non-simple

Preflight:

- An OPTIONS (http method) request
    - checks response headers before real request
- Browser auto-sends and checks
    - bad check = no real request made

Triggering CORS

Simply load a page, then run some JS that makes a `fetch()` call to a different origin.

```
$ serve public/
```

In browser `Devtools > Console`:

```
fetch('http://example.com/api/');
```

What are the origins of:

- the loaded page?
- the request url in the fetch?

Misleading CORS message

Access to fetch at '**http://example.com/api/**' from origin '**http://127.0.0.1:9000**' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

I hate this message.

- `no-cors` is not what you want
    - you will NOT see the response ("opaque")
- Error is because response lacked CORS headers
- Fix is: server to add headers to response
- Can't turn security off just by asking
    - that would be bad security

What about CORB?

CORB is related browser-enforced security block

Blocks a resource if it appears to be the wrong kind. Example:

- Try to load a CSS file that doesn't exist
- Express returns a 404 with `text/html` content-type
- Browser refuses to show 404 content because `text/html` isn't CSS

Fix: Show appropriate content-type

- or make sure file exists

CORS workarounds

Don't try to "get around" CORS when it blocks you

- CORS is security
- any "workaround" will be fixed

Options:

- (best) Have the server side send CORS headers
- (okay) Have a backend proxy
    - write/find a service you CAN call
    - it makes the cross-origin request
    - it gives you the data

Easy CORS practice

- Set up a server running a service on one port
- Call that service from a page on a different port

Test different combinations:

- Simple calls vs non-simple calls
    - See OPTIONS preflight call in the Network tab
- Add `Access-control-allow-origin` header
    - See CORS error vs non-error

Common CORS issues

Issue 1: No `access-control-allow-origin` header

- Fix: Add header to allow origin `*` (or see Issue 2)

Issue 2: origin `*` is allowed, but still errors

- Why: Auth headers aren't allowed with origin `*`
- Fix: get origin from req, allow that origin in res

Issue 3: CORS set up, but get CORS error

- Why: Was response 200? CORS headers on errors?
- Fix 1: CORS error is distraction, fix actual error
- Fix 2: Add CORS headers on error responses

CORS takeaways

- CORS is enforced by the browser
- It exists for good security reasons
- "Fix/workaround" is to follow the protocol
- CORS error messages can be misleading
    - Make sure you know the problem
- Backend folks often don't know CORS
    - because browser-side only
    - service will work for them
        - using non-browser tests