CORS: It’s not scary

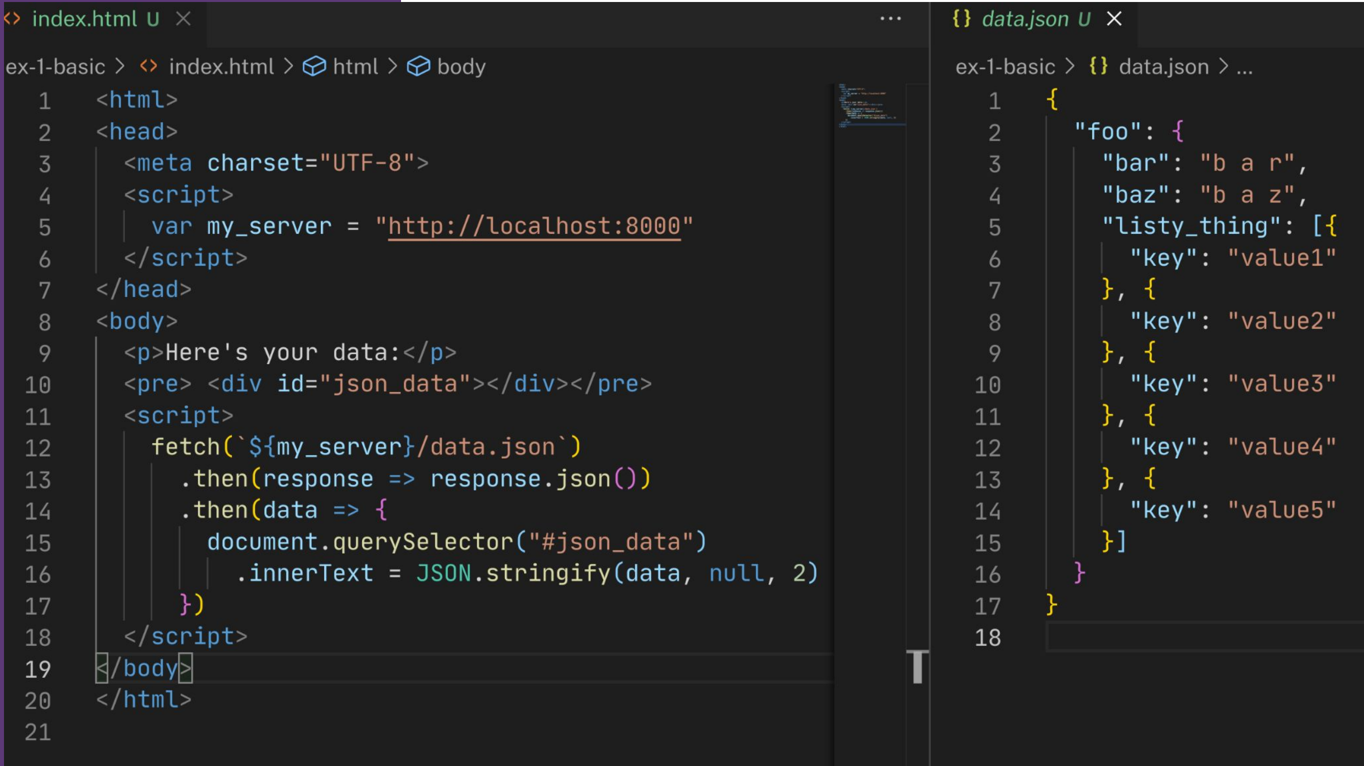
A lightning talk by
Charles Bushong

2021-11-08

We’ve all been there

“I wanna load some data on my site”

It starts innocently enough


index.html U
ex-1-basic > index.html > html > body
data.json U
ex-1-basic > {} data.json > ...

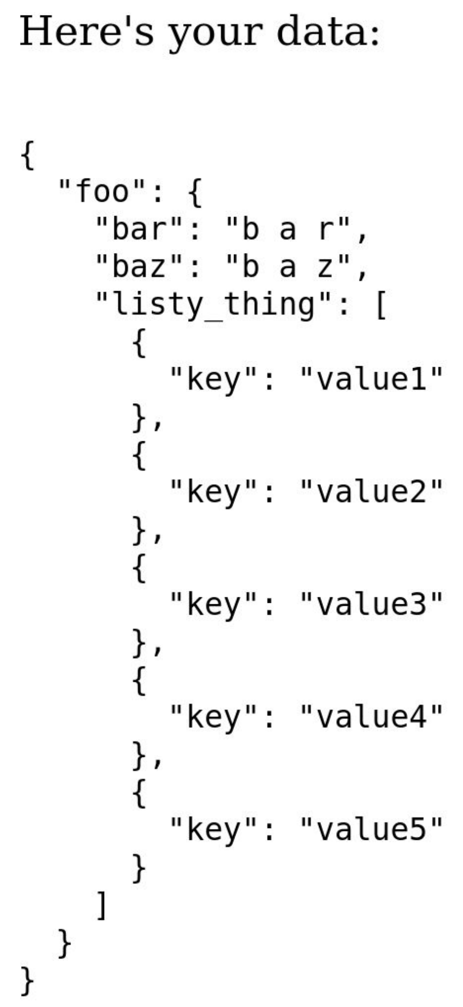
```html
<html>
<head>
  <meta charset="UTF-8">
  <script>
    var my_server = "http://localhost:8000"
  </script>
</head>
<body>
  <p>Here's your data:</p>
  <pre> <div id="json_data"></div></pre>
  <script>
    fetch(`${my_server}/data.json`)
      .then(response => response.json())
      .then(data => {
        document.querySelector("#json_data")
          .innerText = JSON.stringify(data, null, 2)
      })
  </script>
</body>
</html>

```

```json
{
  "foo": {
    "bar": "b a r",
    "baz": "b a z",
    "listy_thing": [{
      "key": "value1"
    }, {
      "key": "value2"
    }, {
      "key": "value3"
    }, {
      "key": "value4"
    }, {
      "key": "value5"
    }]
  }
}

```

It might even work!



Here's your data:

```
{
  "foo": {
    "bar": "b a r",
    "baz": "b a z",
    "listy_thing": [
      {
        "key": "value1"
      },
      {
        "key": "value2"
      },
      {
        "key": "value3"
      },
      {
        "key": "value4"
      },
      {
        "key": "value5"
      }
    ]
  }
}
```

But then something changes

Now you have two URLs

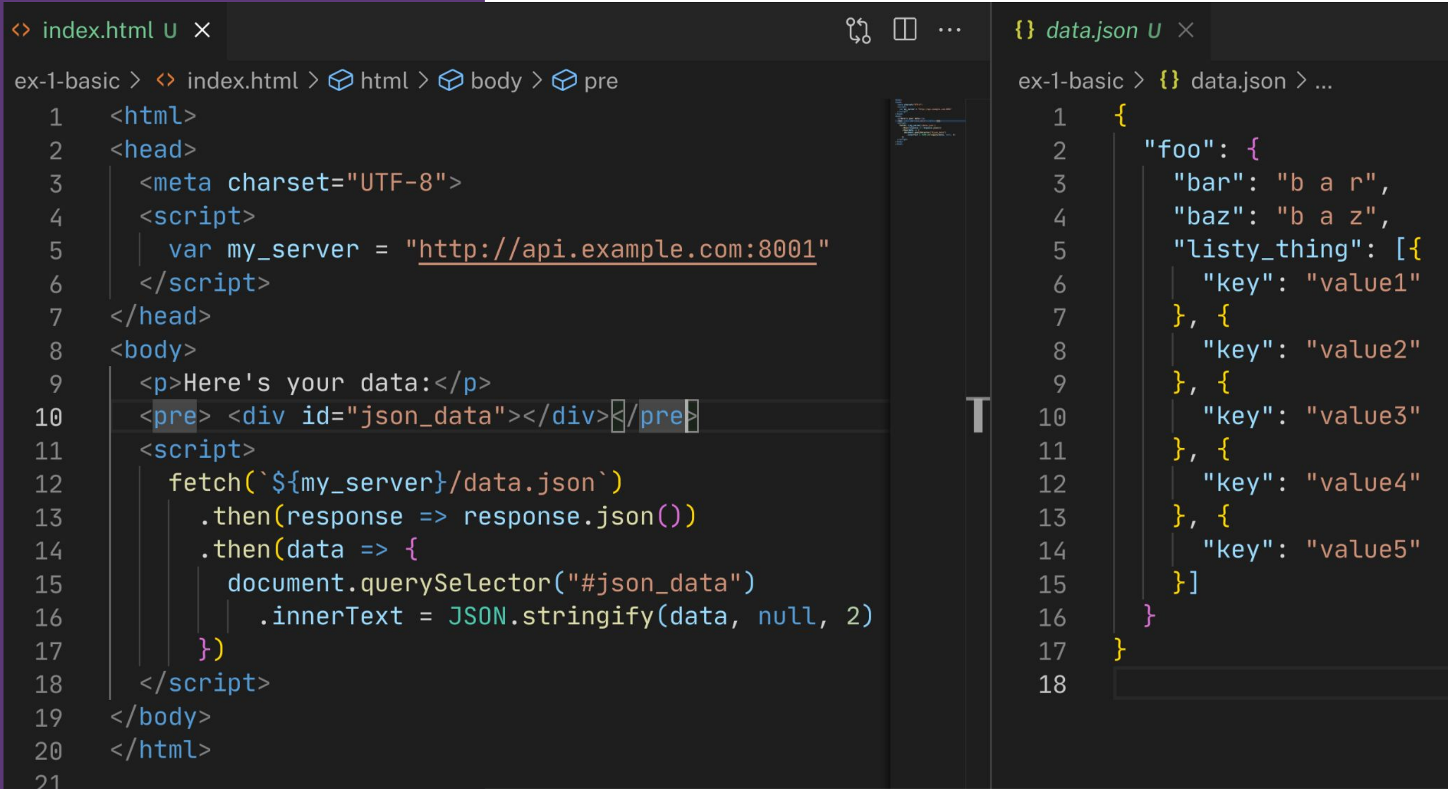
A static front-end
www.example.com

- /index.html

A separated API backend
api.example.com

- /data.json

Makes sense to me



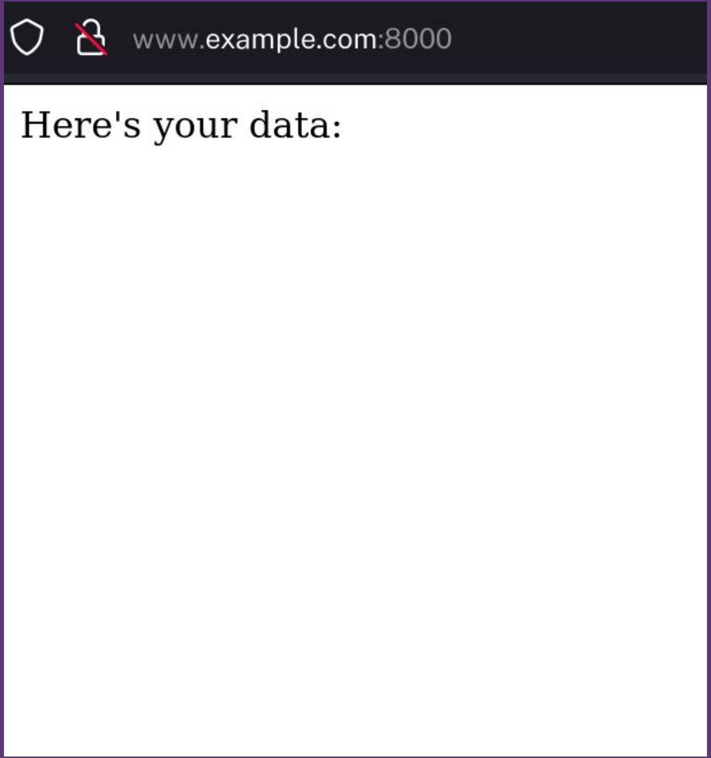
ex-1-basic > index.html > html > body > pre

ex-1-basic > data.json > ...

```html
<html>
<head>
  <meta charset="UTF-8">
  <script>
    var my_server = "http://api.example.com:8001"
  </script>
</head>
<body>
  <p>Here's your data:</p>
  <pre> <div id="json_data"></div></pre>
  <script>
    fetch(`${my_server}/data.json`)
      .then(response => response.json())
      .then(data => {
        document.querySelector("#json_data")
          .innerText = JSON.stringify(data, null, 2)
      })
  </script>
</body>
</html>
```

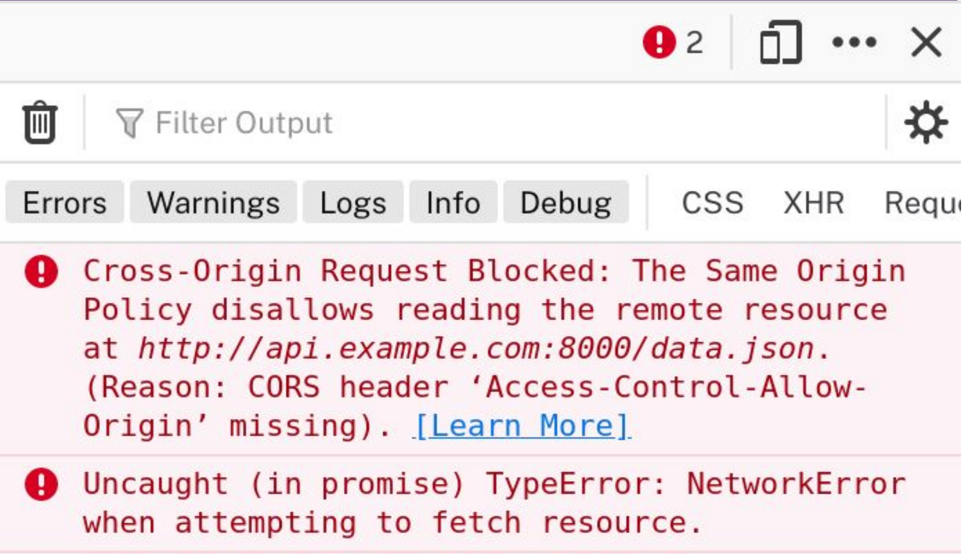
```json
{
  "foo": {
    "bar": "b a r",
    "baz": "b a z",
    "listy_thing": [{
      "key": "value1"
    }, {
      "key": "value2"
    }, {
      "key": "value3"
    }, {
      "key": "value4"
    }, {
      "key": "value5"
    }]
  }
}
```

NOPE!


www.example.com:8000

Here's your data:

Opening dev tools…

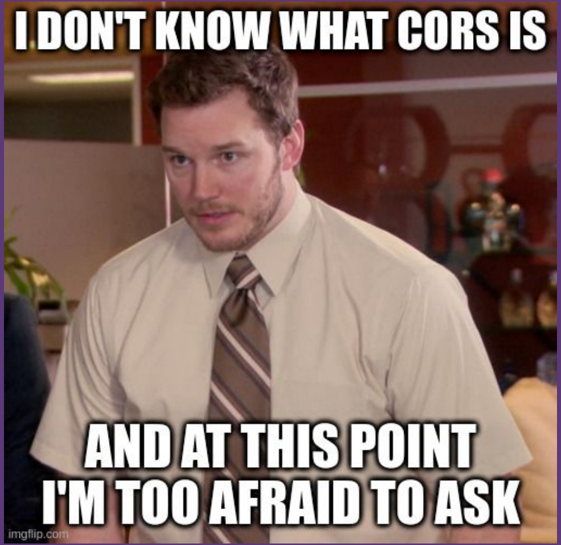


```
Cross-Origin Request Blocked: The Same Origin
Policy disallows reading the remote resource
at http://api.example.com:8000/data.json.
(Reason: CORS header ‘Access-Control-Allow-
Origin’ missing). [Learn More]

Uncaught (in promise) TypeError: NetworkError
when attempting to fetch resource.
```

"CROSS-ORIGIN REQUEST BLOCKED: THE SAME ORIGIN POLICY DISALLOWS READING THE REMOTE RESOURCE"
WHAT THE FUCK DOES THAT MEAN?
imgflip.com

Cross-Origin Resource Sharing


I DON'T KNOW WHAT CORS IS
AND AT THIS POINT I'M TOO AFRAID TO ASK
imgflip.com

What’s CORS?

~~What~~Why's CORS?

- Login to "**mybank.com**/mymoney"
- Open a new tab to "**hackerwebsite.com**/index.html"
- Hacked page requests:
  - **mybank.com**/send-money-to-hacker.asp

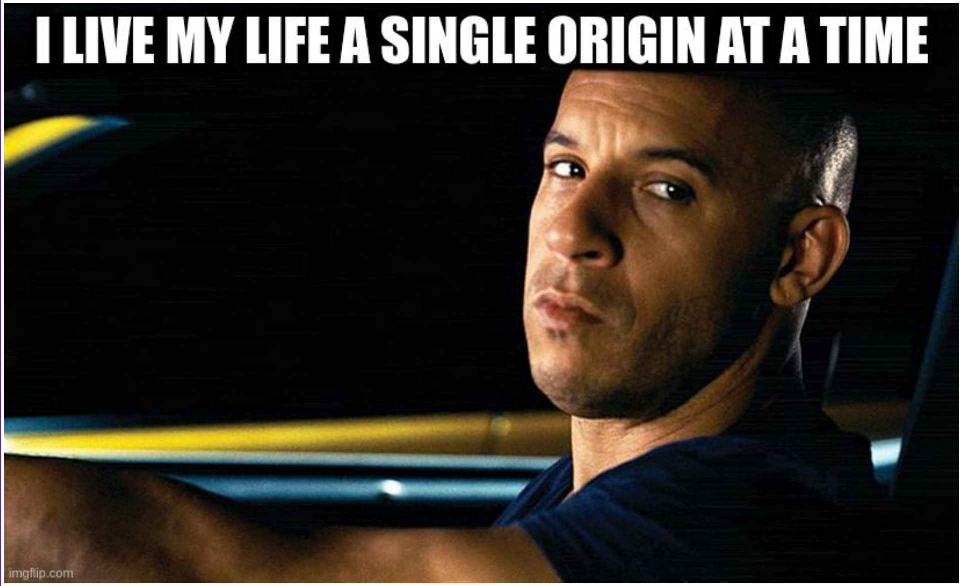
- These sites have a different origin
- So it violates the "*Same-origin Policy*"

Same-Origin Policy

I DON'T KNOW WHAT SAME-ORIGIN POLICY IS
AND AT THIS POINT I'M TOO AFRAID TO ASK
imgflip.com

Same-origin Policy

- A **client side** verification
- Protects unsuspecting users
- Client application [browser, request library]:
    - “I do solemnly swear to honor the same origin policy”



https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Same-origin Policy

Protects against...

- A webpage loading a resource from another origin that the resource owner didn't want

Same-origin Policy

Does NOT protect against...

- A hacked bank server
- Dodgy WIFI
- An unlocked laptop
- Malicious code executed by:
  - A remote connected hacker
  - A user copy pasting it into browser console
  - An npm module

Same-origin Policy?

- https://www.example.com/foo/bar.html
  - https://api.example.com/foo.bar ❌
  - http://www.example.com/foo.bar ❌
  - https://www.example.com:8443/foo.bar ❌

  - https://www.example.com:443/baz/bar ✅

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Back to CORS

CORS allows controlled exceptions to the Same-Origin Policy

How?

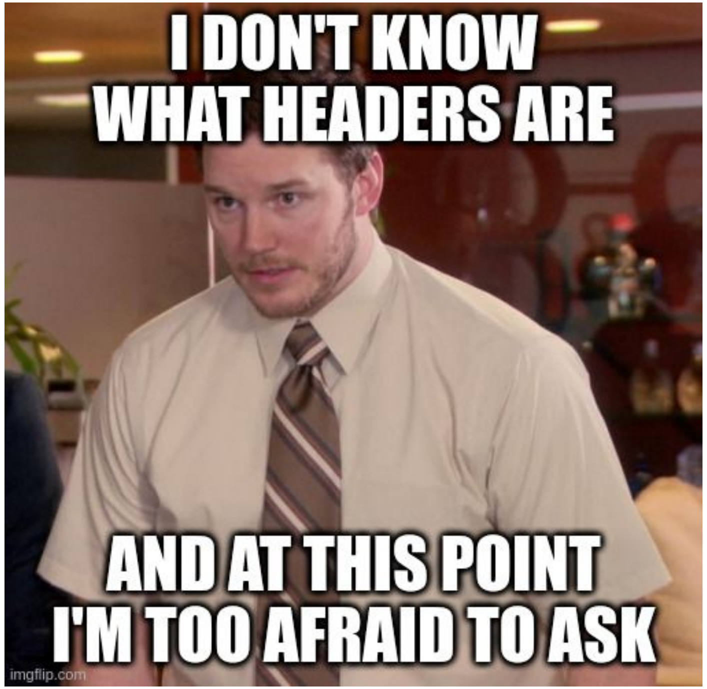
In two parts:

- Preflight request headers
- Response headers

Headers

(just in case)

- Headers are simply **key-value pairs** attached to a HTTP request.
- Key-value pairs are simply two strings, a name and some contents


I DON'T KNOW WHAT HEADERS ARE
AND AT THIS POINT I'M TOO AFRAID TO ASK
imgflip.com

Response Headers: Fail

GET **api**.example.com/foo.js -- “Hey , can you send me /foo.js?”
Origin: www.example.com -- “FYI, my user is on www.example.com”

200 OK -- “Sure thing, ”
Access-Control-Allow-Origin: **api**.example.com -- “But I’d appreciate it if you would
only use this if your user were
on ***api***.example.com”

Cross-Origin Request Blocked -- “Sorry, user, but said I can’t give this to you.
Don’t worry, i’ll just throw the data out.”

Response Headers: Success

GET **api**.example.com/foo.js -- “Hey , can you send me /foo.js?”
Origin: www.example.com -- “FYI, my user is on www.example.com”

200 OK -- “Sure thing, ”
Access-Control-Allow-Origin: **www**.example.com -- “But I’d appreciate it if you would only use this if your user were on ***www***.example.com”

“Thanks! Here ya go, User”

With response headers, the server runs code and returns regardless of pass/fail

That feels… ripe for abuse

Preflight Requests

A request before a request

- Goal to prevent bad requests
- Browser cooperates with Server
- Server responds to “OPTIONS” request, includes ACAO header
- Browser requests OPTIONS, validates
  - If valid, browser requests GET
  - If not valid, GET is never called

Preflight: Fail

OPTIONS **api**.example.com/foo.js -- “Hey , what’s the deal with /foo.js?”
Origin: www.example.com -- “FYI, my user is on www.example.com”

200 OK -- “Here’s the info, ”
Access-Control-Allow-Origin: **api**.example.com -- “I’d appreciate it if you would only make your request if your user were on ***api***.example.com”

Cross-Origin Request Blocked -- “Sorry, user, but said I can’t make that request.”

Preflight: Success

OPTIONS **api**.example.com/foo.js -- “Hey , what’s the deal with /foo.js?”
Origin: www.example.com -- “FYI, my user is on www.example.com”

200 OK -- “Here’s the info, ”
Access-Control-Allow-Origin: **api**.example.com -- “I’d appreciate it if you would
only make your request if your user
were on ***api***.example.com”

Origin == Access-Control-Allow Origin -- “Great! Now i’ll make a Response Header CORS
request and I already know the answer”

Pitfalls

Where things go wrong

- Access-Control-Allow-Origin (ACAO) headers **can** be a wildcard "*"
  - Defeats the purpose of CORS, SOP
- ACAO headers **cannot** *have* a wildcard
  - "*.example.com" is not allowed
- ACAO headers **cannot** be a list
  - Only one domain
- Wait, what if I need more than one domain?
  - You need to do some dumb hacks to work around the shortsighted spec
    - Seriously?
      - Yeah

Dumb Hacks

Working around the CORS single domain issue

At a high level:

- Server listens to OPTIONS
- On request, look at “Origin: <domain>”
- If “<domain>” matches list of good domains
  - Return with header “ACAO: <domain>”
- If not
  - Return with header “ACAO: <self-domain>”
- Also, do that same “header” bit for every other request that comes in

Or find a library that handles the heavy lifting for you

References

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

https://www.serverless.com/blog/cors-api-gateway-survival-guide

https://www.imgflip.com

fearless.tech

@fearlessbmore