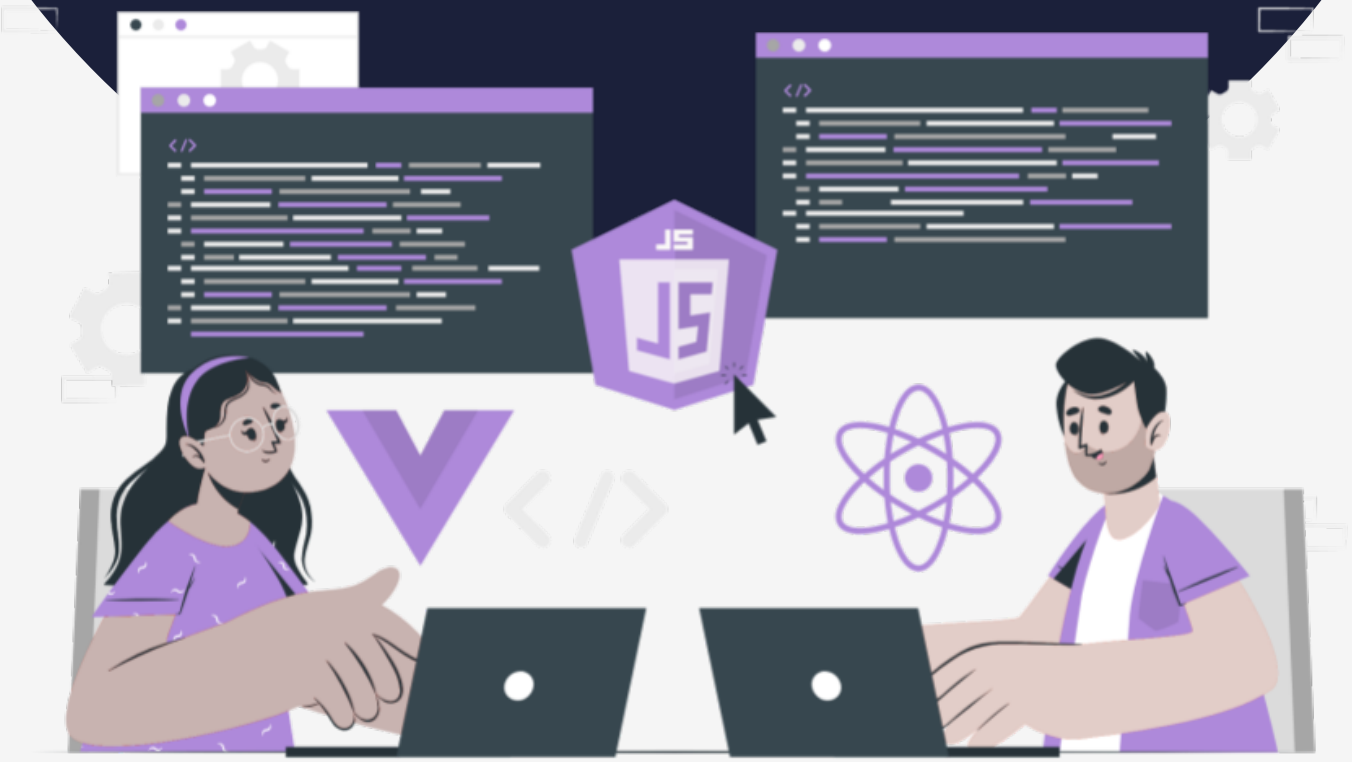
Lesson:

What and Why of CORS?



PW SKILLS

Topics

- What is Cross-Origin Resource Sharing (CORS)?
- The problem CORS trying to solve.
- How does CORS work?
- Why are you getting a CORS error?
- Few ways to resolve CORS errors.

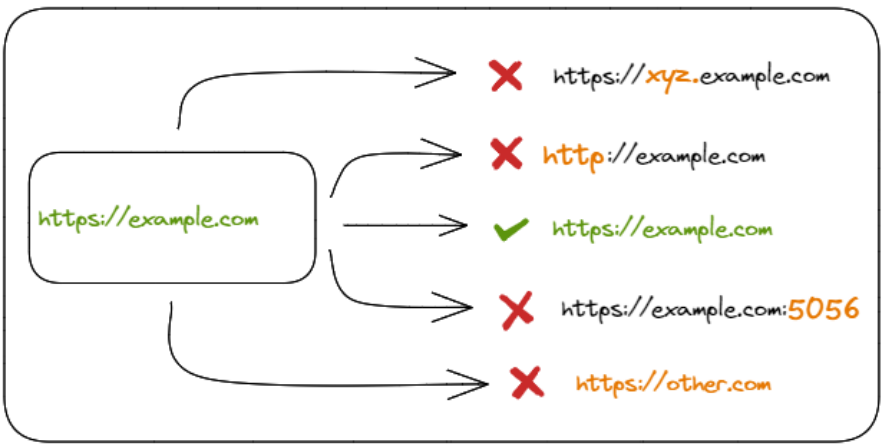
What is Cross-Origin resource sharing (CORS)?

"Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources."

CORS works by allowing servers to specify which domains are allowed to access their resources. When a web page running on one domain makes a request to a server on another domain.

The problem CORS is trying to solve.

Before the CORS mechanism was introduced, Browsers never allowed web applications to share resources between different origins by origin, suppose there is one origin **"example.com"**, so the browser did not allow this origin to share resources with another origin as you can see in the diagram below.



It **blocks** the request from those who have:
Same origin with the **subdomain** "https://xyz.example.com",
Same origin with a **different protocol** "http://example.com",
Same origin with a **different port** "http://example.com:5056", and a
Different origin "https://other.com"

This is because of the **Same Origin Policy**, web browsers only allow web pages to access resources from the same origin (i.e., domain, protocol, and port) as the page itself. If a script, cookie, or other resource tries to access content from a different origin, the web browser blocks the access. It is a fundamental concept in web security that helps to prevent malicious web applications from accessing the sensitive data of other web applications.

For example, suppose you have a web application running on **https://www.example.com.** In that case, scripts from other origins, such as **https://www.attacker.com** or **http://evil.com**, cannot access cookies or other sensitive data stored on **https://www.example.com.** This helps to prevent Cross-Site Scripting **(XSS)** attacks, Cross-Site Request Forgery **(CSRF)** attacks, and other types of web-based attacks.

Sharing the resource safely across the origin is the main issue before the CORS mechanism was introduced.

How does CORS work?

Let's first see what is **CORS preflight mechanism.**

preflight mechanism: It is a part of the Cross-Origin Resource Sharing **(CORS)** standard that is used to check if a cross-origin request is safe to send before actually sending it. This is done to prevent certain types of attacks, such as cross-site scripting **(XSS)** and cross-site request forgery **(CSRF)**, which can occur when a web page can make requests to a different domain that the user did not intend.

CORS preflight call: It is a special type of **HTTP OPTIONS** request that a browser sends to a server to check whether the server will allow a subsequent cross-origin request. It includes information about the origin, the request method, and headers.

Now let's see how CORS works

Let's say you have a website hosted on "**example.com**" and you want to make a request to an API located on "**api.example.org**".

- website on "example.com" makes a request to the API on "**api.example.org**"
- Before the actual request is sent, the browser sends a preflight request to the server hosting the API ("api.example.org"). This preflight request is an **HTTP OPTIONS** request that asks the server whether it will allow the actual request.
- The server hosting the API responds to the preflight request with a set of headers that indicate whether the request is allowed or not. These headers include "**Access-Control-Allow-Origin**", "**Access-Control-Allow-Methods**", and "**Access-Control-Allow-Headers**"
- If the server responds with headers that allow the request, the browser sends the actual request to the API and retrieves the requested data.

For example, the preflight request might look like this:

```
Unset
OPTIONS /api/data HTTP/1.1
Host: api.example.org
Origin: https://example.com
Access-Control-Request-Method: GET
Access-Control-Request-Headers: authorization
```

And the server might respond with headers like this:

```
Unset
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Headers: authorization
```

This indicates that the request from "**example.com**" is allowed, and the actual GET request to the API can proceed.

Why are you getting a CORS error?

One of the top reasons for CORS errors in development is that the server you are sending a request to might not include the expected **access-control-allow-origin** header in the responses it sends back to you. Or if it does, it doesn't include your frontend app's URL in the list of approved origins.

Few ways to resolve CORS errors:

Code example in node.js

1) Setting CORS headers in the API response.
You can set the CORS headers in the response of your API using the **cors npm package** in Node.js. This will allow the client-side JavaScript code to make cross-origin requests to your API. Here is an example:

```JavaScript
const express = require('express');
const cors = require('cors');
const app = express();

app.use(cors());
// your API endpoints here
```

2. Enabling core in server-side API:
If you have control over the server-side API, you can also enable CORS by setting the appropriate headers in your API response. Here is an example:

```JavaScript
app.get('/api', (req, res) => {

// any origin (i.e. domain or IP address) is allowed to
access the API.
  res.setHeader('Access-Control-Allow-Origin', '*');

//res.setHeader('Access-Control-Allow-Origin',
"https://example.com") allows only the mentioned origin to
access the API

  res.setHeader('Access-Control-Allow-Methods', 'GET, POST,
PUT, DELETE');

  res.setHeader('Access-Control-Allow-Headers',
'Content-Type');

  // your API logic here
});
```

Note: When using the **Access-Control-Allow-Origin** header with the value *, this allows any domain to access your API. For security reasons, it's recommended to set this header to the specific domain(s) that should be allowed to access your API.