Week 2 Intermediate JavaScript – Files, Patterns and Flags

# **Day 3: Fetch: Cross Origin Requests**

## Cross-Origin Requests and CORS

Origin Concept

An origin in web development refers to a combination of domain, port, and protocol

**https://example.com:3000**

Cross-Origin Requests

These are requests sent from one domain to another. They include requests to different domains, subdomains, or those using different protocols or ports.

CORS (Cross-Origin Resource Sharing):

It's a security policy implemented by web browsers that controls how resources from one domain can interact with resources from a different domain.

**try {**

**await fetch('http://example.com');**

**} catch(err) {**

**alert(err);** // Failed to fetch due to restrictions

**}**

**Importance of CORS:** CORS prevents malicious scripts on one site from tampering with or stealing data from another site. It's a fundamental security measure on the internet.

**Browser Enforcement:** Modern web browsers enforce CORS policies, allowing developers to control which domains can access their resources.

**Server-Side Configuration:** Additionally, servers can be configured to include appropriate CORS headers, specifying which domains are permitted to access their resources.

## Using Forms

In the early days of web development, it was feasible to make GET/POST requests to other sites, even without dedicated networking methods.

However, accessing the content of an <iframe> from a different site was restricted, making it impossible to read the response.

Although forms could send data to any destination, they couldn't receive the response. Some workarounds existed, but they're considered outdated and not commonly used today.

// form target

**<iframe name=”iframe”></iframe>**

// form could be dynamically generated/submitted

**<form target=”iframe” method=”POST” action=”http://another.com/...”>**

**</form>**

## Simple Requests

Types of Cross-Domain Requests:

* Simple Requests.
* All others (considered "non-simple")

Conditions for a Simple Request:

* Simple methods allowed: GET, POST, HEAD.
* Only specific custom headers are allowed.

Custom headers allowed:

Accept,

Accept-Language,

Content-Language,

Content-Type with the value application/x-www-form-urlencoded, multipart/form-data or text/plain.

Simple Request vs Non-Simple Request:

**Simple requests** can be made with a <form> or a <script>, without special methods.

**Non-simple requests** involve non-standard headers or methods like DELETE and require special handling.

Preflight Request:

Sent by the browser when making a non-simple request.

Asks the server if it agrees to accept such cross-origin requests.

Purpose of Restrictions:

Ensure new cross-origin capabilities are only accessible with explicit permission from the server.

## CORS for simple requests

Making simple requests

When a cross-origin request is made, the browser adds an **Origin header** to it.

This header contains the origin **(domain/protocol/port)** without a path.

The server checks this Origin and can choose to accept it.

**If accepted**, it responds with an Access-Control-Allow-Origin header, specifying the allowed origin. If not, it results in an error.

The browser ensures the correct Origin is sent and checks for the appropriate Access-Control-Allow-Origin in the response. If everything aligns, JavaScript gets access, otherwise, it's blocked with an error.

For instance, if we request

https://anywhere.com/request from https://javascript.info/page, the headers will be like:

**GET /request**

**Host: anywhere.com**

**Origin: https://javascript.info**

This Origin contains exactly the origin (domain/protocol/port), without a path.

A diagram of a computer program

Description automatically generated

Here’s an example of a permissive server response:

**200 OK**

**Content-Type:text/html; charset=UTF-8**

**Access-Control-Allow-Origin: https://javascript.info**

## Response Headers

By default, JavaScript from one domain can only access certain response headers (known as "simple response headers").

These include headers like

* Cache-Control
* Content-Language
* Content-Type
* Expires
* Last-Modified
* Pragma

Notably, the **Content-Length header** is not included in this list.

If you want JavaScript to be able to access other response headers, the server must explicitly allow them using the Access-Control-Expose-Headers header.

**200 OK**

**Content-Type:text/html; charset=UTF-8**

**Content-Length: 12345**

**API-Key: 2c9de507f2c54aa1**

**Access-Control-Allow-Origin: https://javascript.info**

**Access-Control-Expose-Headers: Content-Length,API-Key**

With this Access-Control-Expose-Headers header, the script is **permitted to access the Content-Length** and API-Key headers of the response.

## Non-Simple Requests

Non-simple requests

Allow any HTTP method, like PATCH, DELETE, etc.

Older web services might see non-standard methods as a sign that it's not a browser.

To avoid misunderstandings, browsers send a "preflight" request before making a non-simple request.

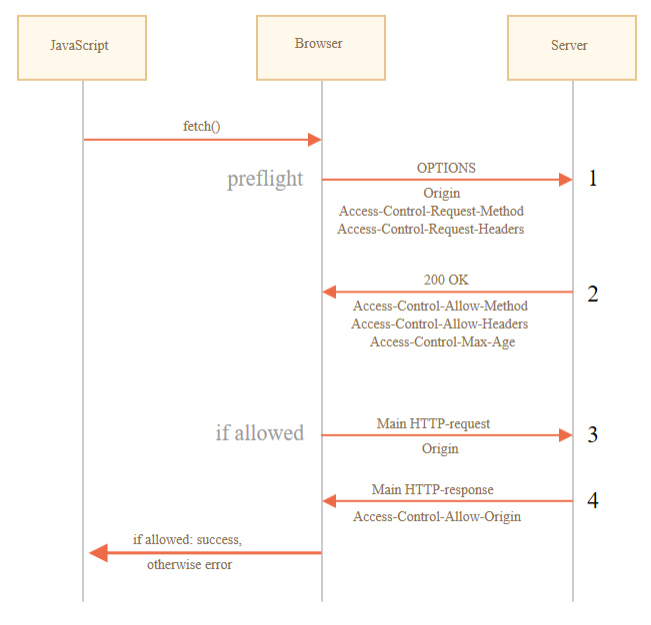
The preflight request

uses the **OPTIONS** method and doesn't contain a body.

It includes headers like **Access-Control-Request-Method (for the requested method)** **and Access-Control-Request-Headers (for non-simple headers).**

If the server approves, it should respond with status 200, and include the headers Access-Control-Allow-Methods (with the allowed method) and Access-Control-Allow-Headers (with the allowed headers).

Optionally, Access-Control-Max-Age header can specify a number of seconds to cache the permissions, saving preflight requests for subsequent requests with the same permissions



**Let response = await fetch(‘https://site.com/services.json’, {**

**method: ‘PATCH’,**

**headers: {**

**‘Content-Type’: ‘application/json’**

**‘API-Key’: ‘secret’**

**}**

**});**

**Understanding Cross-Domain PATCH Request:**

Step 1 (Preflight Request):

Browser sends an invisible preflight request automatically.

This request has method OPTIONS with the same path as the main request.

It includes special headers like Origin, Access-Control-Request-Method, and Access-Control-Request-Headers.

Step 2 (Preflight Response):

Server responds with status 200 and headers specifying allowed methods and headers.

In this example, it's PATCH method and headers Content-Type, API-Key.

Note: If the server expects other methods or headers in the future, it can allow them by adding to the list.

Step 3 (Actual Request):

Browser makes the real PATCH request after a successful preflight.

It includes headers like Origin, Content-Type, API-Key.

Step 4 (Actual Response):

Server includes Access-Control-Allow-Origin in the response header.

This allows JavaScript to read the full response.

Important Notes:

Preflight request is automatic and invisible to JavaScript.

JavaScript only gets the response to the main request or an error if there's no server permission.

## Credentials

Cross-Origin Requests and Credentials:

Cross-origin requests **do not** include credentials (like cookies) by default.

This is a security measure to prevent unauthorized access.

To allow credentials, the server must explicitly permit it with an additional header.

Sending Credentials:

**fetch('http://another.com', {**

**credentials: "include"**

**});**

**Server Response for Credentials:** Access-Control-Allow-Credentials: true

Simple Requests vs. Non-Simple Requests:

Simple requests include methods like GET, POST, or HEAD, and specific headers.

Non-simple requests include other methods and headers, requiring a preliminary "preflight" request.

Handling Simple Requests:

The browser sends the Origin header the request.

The server should set Access-Control-Allow-Origin to the origin.

**Access-Control-Allow-Origin: <origin>**

**Access-Control-Allow-Credentials: true**

Accessing Non-Simple Response Headers:

If JavaScript needs to access specific response headers, the server should list them in Access-Control-Expose-Headers.

Why Do We Need Origin?

Origin is more reliable than Referer, especially in cross-origin requests.

Referer can be absent in some cases, but Origin is guaranteed by the browser.

Solutions:

Origin is used as a reliable source of the request's origin.

It is necessary for security, especially in cases where Referer might not be sent.

## Fetch API

List of all fetch options



**Referrer**: Controls the HTTP Referer header.

* To send no referer, use an empty string or "no-referrer".
* To set another URL within the current origin, provide the URL.

**fetch('https://example.com/page', {**

**referrer: ''**

**});**

**fetch('https://example.com/page', {**

**referrer: 'https://example.com/otherpage'**

**});**

**ReferrerPolicy:** Sets general rules for Referer.

* Options include "no-referrer-when-downgrade", "no-referrer", "origin", and others.

Mode:

* Determines the behavior for cross-origin requests:
  + **"cors"**: Allows cross-origin requests (default).
  + **"same-origin"**: Forbids cross-origin requests.
  + **"no-cors"**: Only allows simple cross-origin requests.

Credentials:

* Specifies whether fetch should send cookies and HTTP-Authorization headers with the request.
  + **"same-origin"**: Default, don’t send for cross-origin requests.
  + **"include"**: Always send, requires **Access-Control-Allow-Credentials** from the cross-origin server.
  + **"omit"**: Never send, even for same-origin requests.

Cache:

* Fine-tunes HTTP caching behavior.
  + **"default"**: Uses standard HTTP-cache rules and headers.
  + **"no-store"**: Totally ignores HTTP-cache.
  + **"reload"**: Ignores HTTP-cache but populates it with the response.
  + **"no-cache"**: Creates a conditional request if there is a cached response.
  + **"force-cache"**: Uses a response from HTTP-cache, even if it’s stale.
  + **"only-if-cached"**: Uses a response from HTTP-cache, even if it’s stale.

Redirect:

* Controls how fetch handles HTTP-redirects:
  + **"follow"**: Follow HTTP-redirects (default).
  + **"error"**: Throws an error in case of HTTP-redirect.
  + **"manual"**: Doesn’t follow HTTP-redirect, but provides information for manual redirection.

Integrity:

* Checks if the response matches a known-ahead checksum.
  + Supports hash-functions like SHA-256, SHA-384, and SHA-512.

keepalive:

* Indicates that the request may outlive the page.
  + Essential for requests to succeed even after the page is unloaded.

**window.onunload = function() {**

**fetch('/analytics', {**

**method: 'POST',**

**body: "statistics",**

**keepalive: true**

**});**

**};**

Note: Each option serves specific purposes, allowing fine-grained control over the behavior of the fetch request.