




CORS

☰ 태그	Web
⚙ 상태	시작 전

아니..이놈 뭐야

✖ Access to fetch at '<https://example.com/api/data>' from origin '<http://localhost:5173>' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: 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. 

프론트엔드에서 개발을 하다가 서버에 비동기 요청을 보낼 때 때론 위 와 같은 에러가 발생하는 것을 자주 목격하게 된다.

맞다. 유명하디 유명한 CORS 에러... 분명 나는 올바르게 요청을 했는데, 이런 에러가 발생해서 골치가 아플 때가 많이 있었을 것이다.

같은 URI 요청을 Postman 등 프로그램으로 똑같이 보냈을 때는 잘 동작하게 되는 모습을 보게된다. 분명 같은 요청을 보냈는데, 왜 브라우저에서는 다른 결과가 일어나는지 CORS가 무엇인지 알아보자.

자바스크립트는 무궁무진하다.

웹 브라우저에서 주로 많이 사용하는 자바스크립트는 무궁무진한 능력을 가지고 있다! 자바스크립트를 통해 브라우저에서 제공하는 WebAPI 는 전부 외우기 힘들 정도로 정말 많이 있고, 이것을 통해 DOM 조작은 물론이고 Cookie를 다루는 일, AJAX 비동기요청 등 여러가지 작업을 할 수 있다.

특히 Fetch API 를 통해 외부에서 데이터를 받아오는 요청의 경우에는 외부와 네트워크가 연결되는 과정에서 언제든지 개인의 정보를 탈취할 수 있는 큰 취약점이 발생할 수있다.

이때 발생할 수 있는 공격 중 한가지가 **CSRF (Cross-Site Request Forgery)** 이다.



CSRF (Cross-Site Request Forgery)

“신뢰할 수 있는 사용자를 사칭해 웹 사이트에 원하지 않는 명령을 보내는 공격”

예시


- 만일, A라는 사이트의 사용자 개인 비밀번호 변경을 하는 주소 패턴이 ' `http://example.com/user.do?cmd=user_passwd_change&user=admin&newPwd=1234` ' 라고 한다면 이러한 링크를 사용자의 메일로 보내는데, 만약 사용자가 메일을 읽게 되면 해당 사용자의 패스워드가 1234로 초기화된다.
이를 관리자에게 보내서 일반 계정을 관리자 계정으로 바꾸도록 하거나, 관리자 계정 패스워드를 바꾸는 데 이용한다면 해당 사이트의 모든 정보가 해킹당하는 데는 오랜 시간이 걸리지 않는다.

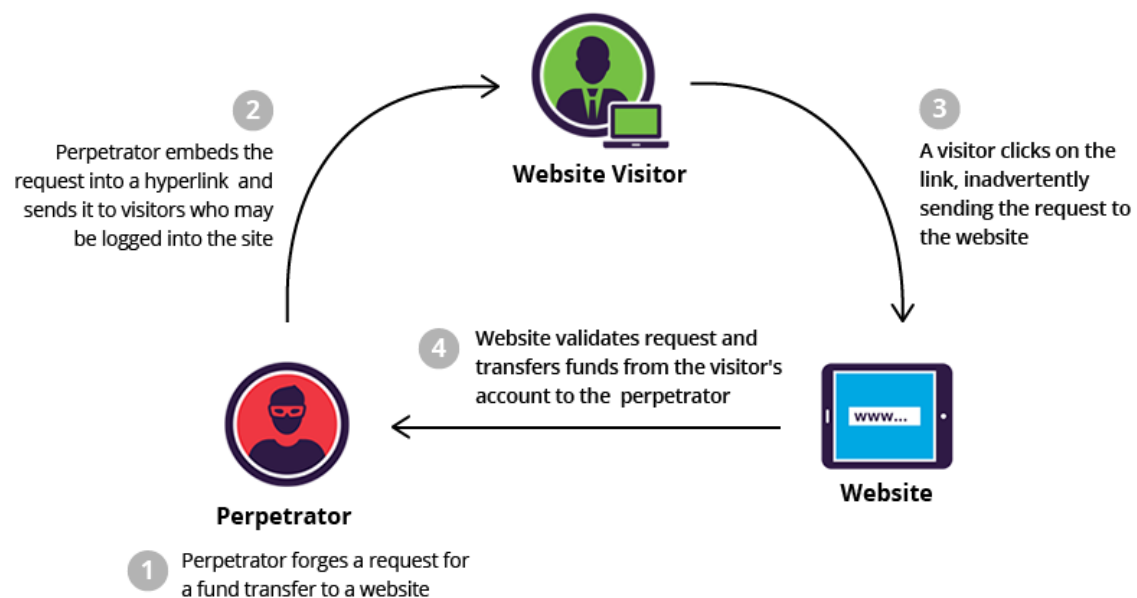
- 스팸 메일 등으로 접근하여, 공격자의 사이트에 접근하게 하여 사용자의 브라우저에 개인정보가 담긴 **Cookie**에 접근하고, 탈취하는 자바스크립트 파일을 실행하도록 한다.
이러한 공격은 **의도치 않은 사이트 접속으로 인해 사용자의 브라우저에서 외부의 서버에 요청을 보내는 코드**가 담긴 자바스크립트 파일이 실행되었기 때문에 일어날 것입니다.

교차 사이트 요청 위조 (CSRF) - MDN Web Docs 용어 사전: 웹 용어 정의 | MDN

교차 사이트 요청 위조(Cross-Site Request Forgery, CSRF)는 신뢰할 수 있는 사용자를 사칭해 웹 사이트에 원하지 않는 명령을 보내는 공격입니다.

 <https://developer.mozilla.org/ko/docs/Glossary/CSRF>

 mdn web docs



XSS(Cross-Site Scripting)

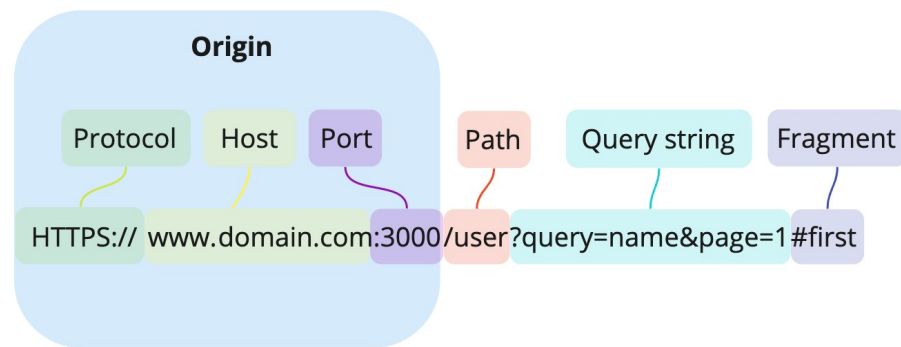
•

이러한 공격은 서로의 신뢰여부를 알 수 없는 다른 Origin 끼리의 요청이 가능하기 때문에 일어나게 된다.

그래서 브라우저는 같은 출처(Origin)에서만 통신할 수 있도록 하는 **동일 출처 정책(SOP)**를 기본적으로 도입하게 된다.

▼ (참고) Origin의 정의

“Protocol + Host + Port”



출처 - MDN Web Docs 용어 사전: 웹 용어 정의 | MDN

웹 콘텐츠의 출처(origin)는 접근할 때 사용하는 URL의 스킴(프로토콜), 호스트(도메인), 포트로 정의됩니다. 두 객체의 스킴, 호스트, 포트가 모두 일치하는 경우 같은 출처를 가졌다고 말합니다.

<https://developer.mozilla.org/ko/docs/Glossary/Origin>

mdn web docs

SOP (Same Origin-Policy) : 동일 출처 정책

"같은 Origin 에서의 접근만을 허용한다!!"

"다른 Origin 에서의 접근을 모두 차단한다!"

동일 출처 정책 - 웹 보안 | MDN

동일 출처 정책은 어떤 출처에서 불러온 문서나 스크립트가 다른 출처에서 가져온 리소스와 상호 작용할 수 있는 방법을 제한하는 중요한 보안 메커니즘입니다.

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

mdn web docs

웹 생태계의 변화

인터넷의 발달로 웹 생태계가 많이 다양해지고 커지게 되면서, 인터넷 상에 있는 여러 서비스들끼리 다양한 데이터를 주고 받을 필요가 생기게 되었다.

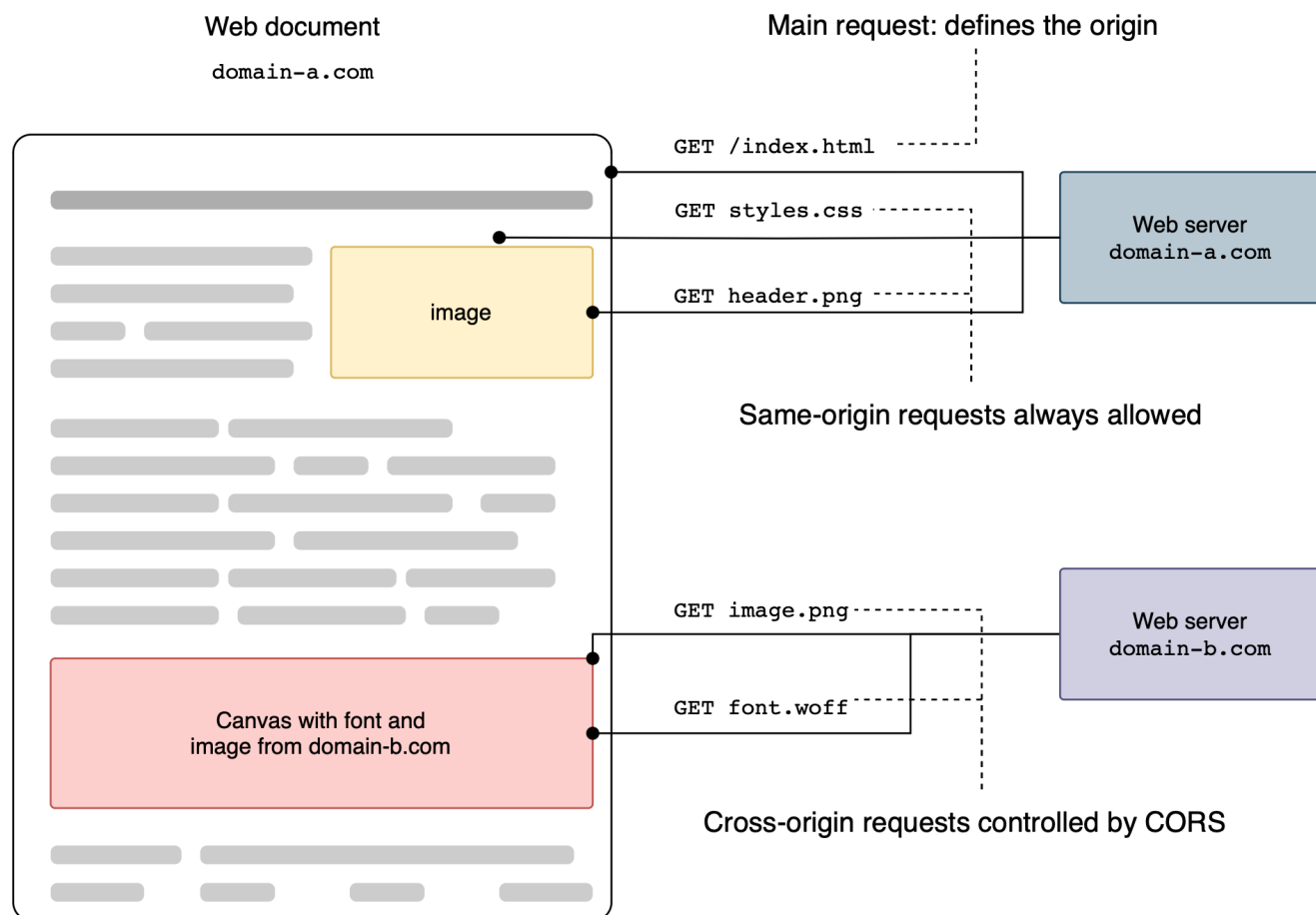
외부 사이트에서 데이터를 주고 받는 일이 당연해지고 있는 상황에서 브라우저의 SOP 정책으로 인해 이러한 상황을 제한하게 되면서 데이터의 교환에 있어서 자유롭지 못하다는 문제가 발생한다.

그래서 브라우저에서는 이러한 문제를 해결하기 위해 "서로 합의된 서로 다른 Origin 간에 합법적으로 통신을 허용해주기 위해" 새롭게 등장한 개념이 바로 **CORS(Cross-Origin Resource Sharing)** 이다.

브라우저 : "CORS 규칙을 지키면, '다른 출처'에서도 데이터를 불러올 수 있게 해줄게!"

👉 CORS(Cross-Origin Resource Sharing) : 교차 출처 리소스 공유

"서로 다른 Origin 간에 데이터를 공유하는 것을 허용하도록 서버가 허가해주는 HTTP 헤더 기반 메커니즘"



이쯤되면 **CORS**는 우리의 적(?)이 아니라 동료라는 것을 알게 되었을 것이다. (사격중지!!)

이 정책이 아니었다면 어쩌면 우리는 서로가 서로를 믿지 못하는 세상에서 살고 있을 지도 모른다..?

브라우저에서 CORS를 적용하고 있는 요청은 아래와 같이 **MDN 문서**에서 확인해볼 수 있다.

어떤 요청이 CORS를 사용합니까?

이 [교차 출처 공유 표준](#) 은 다음과 같은 경우에 교차 출처 HTTP 요청을 가능하게 합니다.

- 위에서 언급한 `fetch()` 또는 `XMLHttpRequest` 의 호출.
- 웹 폰트(CSS 내 `@font-face` 에서 교차 도메인 폰트 사용 시), [서버가 교차 출처로만 로드될 수 있고 허가된 웹사이트에서만 사용할 수 있는 TrueType 폰트를 배포할 수 있게 합니다.](#)
- [WebGL 텍스처](#).
- [drawImage\(\)](#) (영어)를 사용해 캔버스에 그린 이미지/비디오 프레임.
- [이미지로부터 추출하는 CSS Shapes.](#) (영어)

CORS를 사용하는 브라우저 요소들

교차 출처 리소스 공유 (CORS) - HTTP | MDN

교차 출처 리소스 공유(Cross-Origin Resource Sharing, CORS)는 브라우저가 자신의 출처가 아닌 다른 어떤 출처(도메인, 스킴 혹은 포트)로부터 자원을 로딩하는 것을 허용하도록 서버가 허가 해주는 HTTP 헤더 기반 메커니즘입니다. 또한 CORS 는 교차 출처 리소스를 호스팅하는 서버가 실제 요청을 허가할 것인지 확인하기 위해 브라우저가 보내는 "사전

<https://developer.mozilla.org/ko/docs/Web/HTTP/CORS>

mdn web docs

앞서 정의한 내용 대로 CORS는 **HTTP header에 기반한 메커니즘**이다.

서로 다른 Origin에 요청을 보냈을 때 서버가 웹 브라우저에서 해당 정보를 읽는 것이 허용된 출처를 설명할 수 있도록 HTTP Header 를 추가하도록 하여 동작한다.

지금부터 브라우저의 CORS 기본 동작을 살펴보고, CORS를 통해 외부 Origin 서버가 접근하는 시나리오를 한번 알아보자.

🔍 브라우저의 CORS 기본 동작 과정

1 클라이언트에서 HTTP 요청의 헤더에 Origin을 담아 전달

⇒ HTTP 프로토콜을 이용하여 서버에 요청을 보낼 때, **Request Header**에 **Origin** 이라는 필드에 요청하는 Origin을 함께 담아 보낸다.

▼ Request Headers	<input type="checkbox"/>
	Raw
Accept:	application/json, text/plain, */*
Accept-Encoding:	gzip, deflate, br, zstd
Accept-Language:	ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Connection:	keep-alive
Cookie:	JSESSIONID=37EA90AC4C921F98D22231F37423EDDE
Host:	localhost:8080
Origin:	http://localhost:5173
Referer:	http://localhost:5173/
Sec-Ch-Ua:	"Google Chrome";v="131", "Chromium";v="131", "Not_A Brand";v="24"
Sec-Ch-Ua-Mobile:	?0
Sec-Ch-Ua-Platform:	"macOS"
Sec-Fetch-Dest:	empty
Sec-Fetch-Mode:	cors
Sec-Fetch-Site:	same-site
User-Agent:	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36

2 서버는 응답헤더에 Access-Control-Allow-Origin 을 담아 클라이언트로 전달

⇒ **Access-Control-Allow-Origin** 의 값으로 **"리소스에 접근하는 것이 허용된 Origin"** 을 보낸다.

▼ Response Headers	<input type="checkbox"/>
	Raw
Access-Control-Allow-Credentials:	true
Access-Control-Allow-Origin:	http://localhost:5173
Cache-Control:	no-cache, no-store, max-age=0, must-revalidate
Connection:	keep-alive
Content-Type:	application/json
Date:	Tue, 14 Jan 2025 15:35:08 GMT
Expires:	0
Keep-Alive:	timeout=60
Pragma:	no-cache
Transfer-Encoding:	chunked
Vary:	Origin
Vary:	Access-Control-Request-Method
Vary:	Access-Control-Request-Headers
X-Content-Type-Options:	nosniff
X-Frame-Options:	DENY
X-Xss-Protection:	0

3 클라이언트에서 Origin 과 서버가 보내준 Access-Control-Allow-Origin 을 비교

⇒ 응답을 받은 브라우저는 자신이 보냈던 Origin과 서버에서 받은 **Access-Control-Allow-Origin** 을 비교

- 같다면 (유효 하다면)
 - → 문제 없이 리소스를 가져옴
- 다르다면 (유효하지 않다면)
 - → 받아온 응답을 사용하지 않고, 에러를 발생시킨다. (CORS 에러)

🔍 CORS 접근 제어 시나리오

1. 단순 요청 (Simple Request)
2. 사전 요청 (Preflighted requests)
3. 자격 증명을 포함한 요청 (Credentialed Request)

1 사전 요청 (Preflighted requests) :

“본 요청을 보내기 전에 안전한 요청인지 (서버와 잘 통신이 되는지) 미리 확인 하는 것”





실제 요청을 보내는 것이 안전한지 판단하기 위해 브라우저가 먼저 **OPTIONS** 메서드를 사용해 다른 출처의 리소스에 HTTP 요청을 보냄

- 브라우저가 미리 먼저 예비로 요청을 보내는 것을 **Preflight** 라고 부른다.
- 이 사전 요청에서는 **HTTP Method**를 **OPTION** 이라는 요청으로 보냄

OPTIONS - HTTP | MDN

HTTP OPTIONS 메서드는 주어진 URL 또는 서버에 대해 허용된 통신 옵션을 요청합니다. 클라이언트는 이 방법으로 URL을 지정하거나 별표(*)를 지정하여 전체 서버를 참조할 수 있습니다.

 <https://developer.mozilla.org/ko/docs/Web/HTTP/Methods/OPTIONS>

 mdn web docs

⇒ 서버의 데이터에 영향을 줄 수 있는 요청들일 때 안전하게 확인하기 위해 **Preflighted requests** 를 보냄

Preflighted requests 순서

1. 브라우저는 **OPTION** 메서드와 몇가지 요청 사항을 다른 Origin에 HTTP 요청을 보낸다.
2. 웹 서버는 **request header** 에 담겨온 정보를 확인하고, 자신이 허락하는 내용 (**Access-control-Allow ...**) 를 **response header** 에 담아서 브라우저에게 넘겨준다
3. 브라우저는 웹 서버가 보낸 **response header** 내용을 확인하고, 요청이 가능하다면 실제 요청을 보낸다.

▼ Preflighted requests 시 요청 Header에 보내는 내용

```
Origin : { 브라우저의 Origin }
Access-Control-Request-Method : { 실제 요청시 보내는 method }
Access-Control-Request-Headers : { 실제 요청의 추가 header }

// ...언어, 인코딩 방식, 운영체제, 브라우저 정보 등
```

▼ Preflighted requests 보낸 후 서버의 응답 Header 에 들어 있는 내용

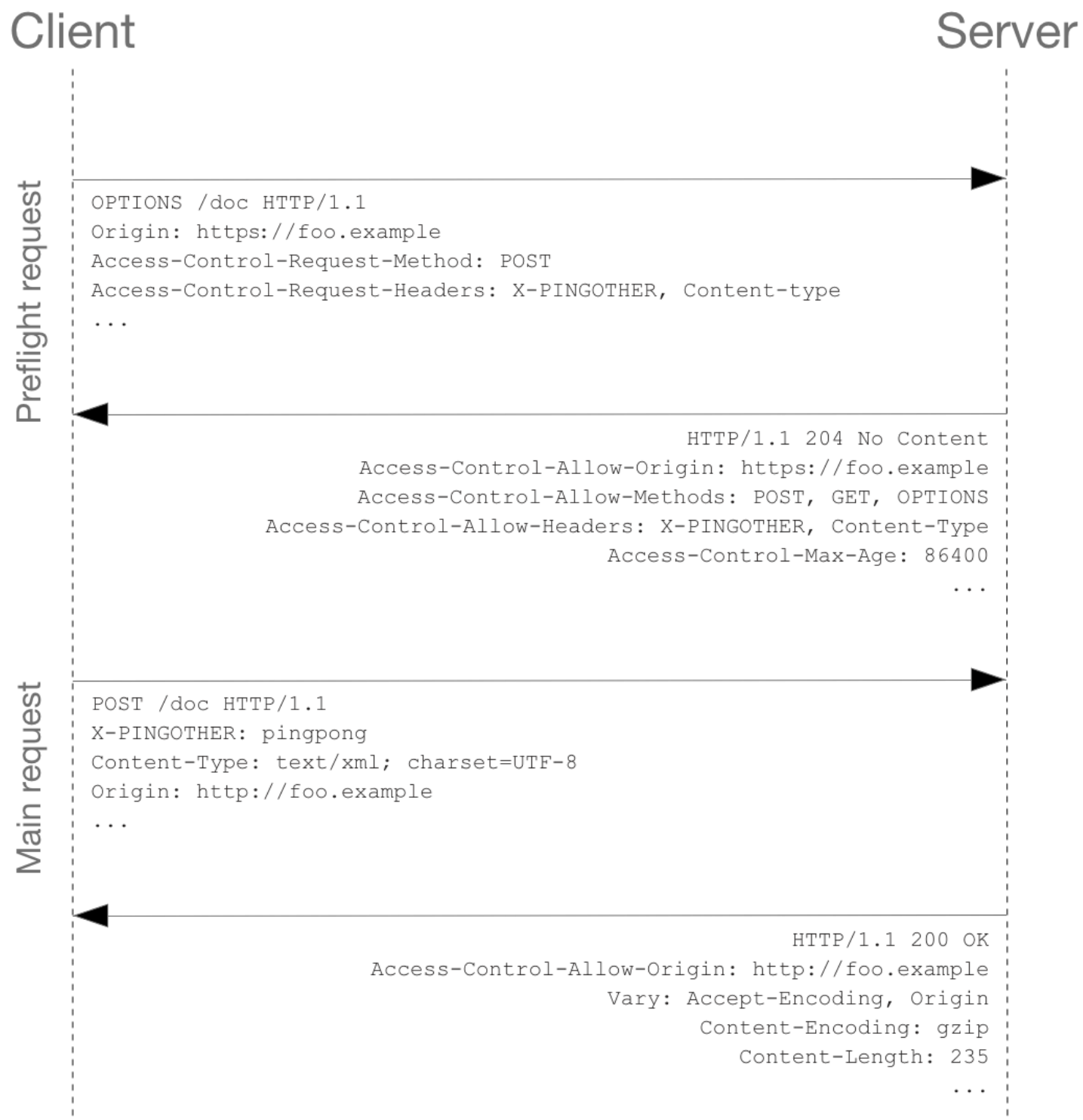
```
Access-Control-Allow-Origin : { 서버에서 허가하는 Origin }

Access-Control-Allow-Method : { 서버에서 허가하는 Method }

Access-Control-Allow-Headers : { 서버에서 허가하는 Header }

Access-Control-Allow-Max-Ag : { Preflight 응답 캐시 기간 }
```

위와 같은 내용으로 서로 다른 Origin 간에 요청을 주고 받은 후 내용확인 후 요구 사항이 맞다면, 본 요청을 할 수 있게 됨

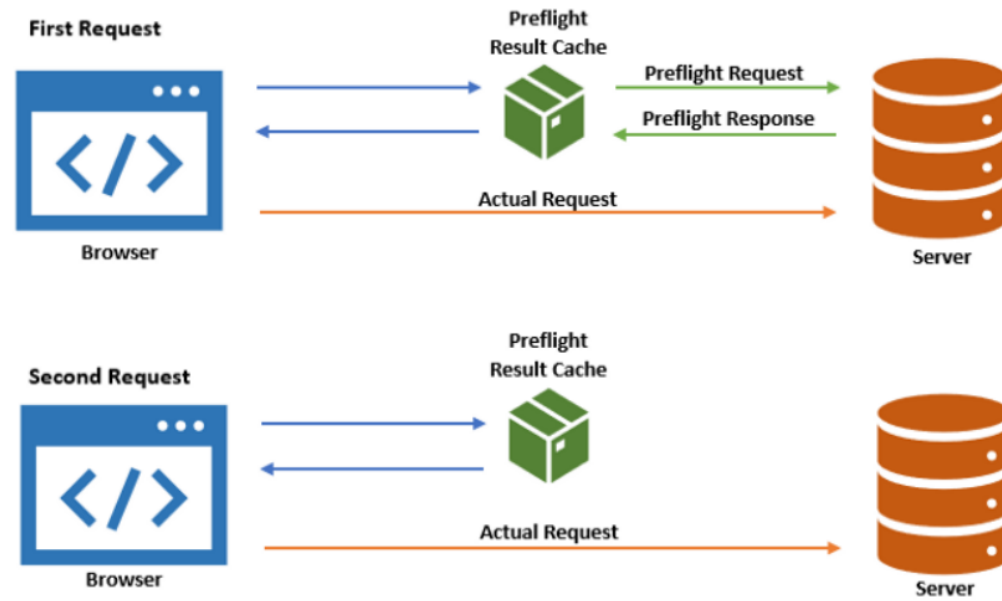


Preflighted requests 단점

- 실제 요청에 걸리는 시간이 늘어나게 되어 **어플리케이션의 성능에 영향**을 미친다.
- API 호출 수가 많아지면, 서버 운용 비용이 늘어날 수 있다

⇒ 브라우저 캐시를 이용해서 Preflighted 요청을 캐싱 시켜 최적화 시킨다.

Access-Control-Max-age : 600 // 서버에서 설정해줌



2 단순 요청 (Simple Request) :

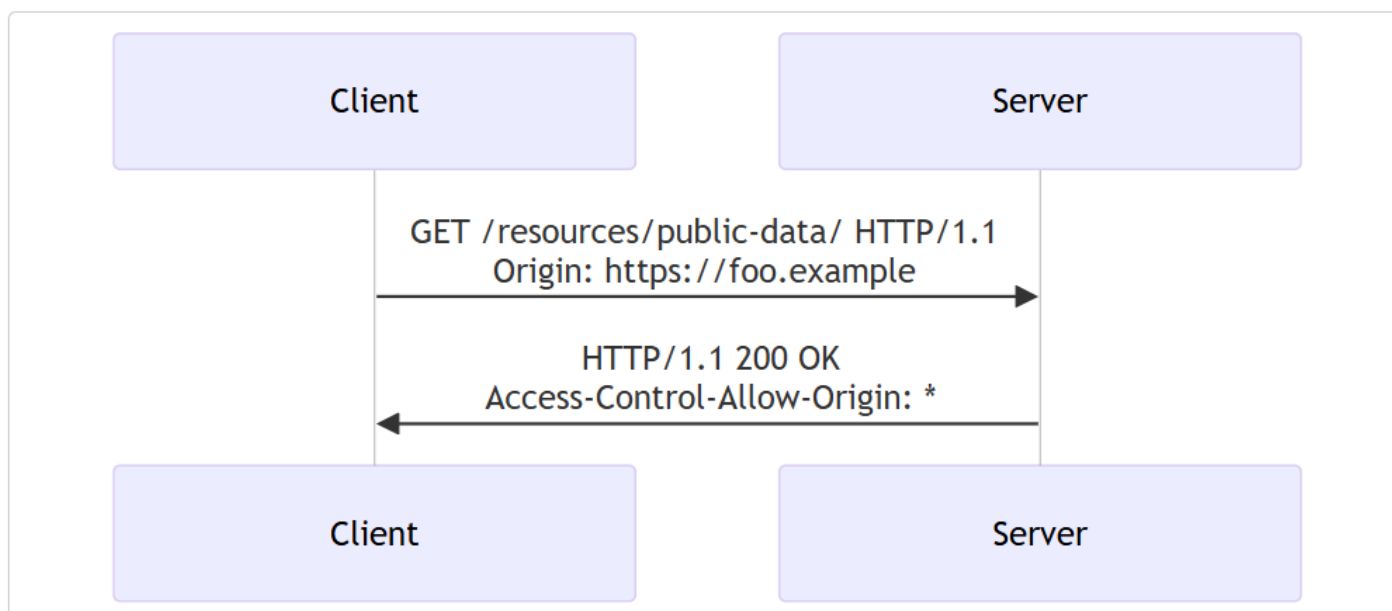
“어떠한 조건을 만족하면 **CORS 정책 검사를 진행하지 않는 요청**”

Preflighted requests” 을 생략하고 바로 서버에 직행으로 보냄

- 클라이언트와 서버 간의 간단한 교환을 수행하며, 권한 처리를 위해 CORS 헤더를 사용

! Simple Request 요청을 처리하는 조건

- GET, HEAD, POST 요청 중 한 가지 이어야 함
- Request Header에는 다음 속성만 허용
 - Accept
 - Accept-Language
 - Content-Language
 - Content-Type, DPR, Downlink, Save-Data, Viewport-Width, Width
 - 만약 Content-Type 를 사용하는 경우에는 아래 내용만 허용
 - application/x-www-form-urlencoded
 - multipart/form-data
 - text/plain



“Simple Request” 동작 순서

1. 브라우저에서 위 조건에 해당하는 내용을 요청 Header에 담아서 보낸다.
2. 서버는 Access-Control-Allow-Origin 에 모든 출처에서 접근을 허용하는

3 자격 증명을 포함한 요청 (Credentialed Request)

“자격 인증 정보를 함께 담아 요청할 때 보내는 요청”

- 브라우저와 서버에서 각각 설정해주어 요청을 보내야 한다.

! 자격 인증 정보

“Session ID 등 이 저장되어 있는 Cookie 혹은 Authorization 헤더에 설정하는 토큰 값”

동작 메커니즘

1. 브라우저에서 **인증 정보 설정을 지정한** HTTP 요청을 서버에 보낸다. (Cookie 포함)

⇒ HTTP Method에 따라서 simple request, preflight request를 보낸다.

✓ 브라우저에서 설정하기

“인증 정보를 보내도록 설정”

`credentials` 옵션을 설정한다.

만약, 어떤 설정도 해주지 않으면, 쿠키 등의 인증 정보는 자동으로 서버에 전송되지 않는다.

! **credentials 옵션**

옵션 값	설명
same-origin (기본값)	같은 출처 간 요청에만 인증 정보를 담도록 함
include	모든 요청에 인증 정보를 담을 수 ○
omit	모든 요청에 인증 정보를 담지 ✕

예시

```
// fetch
fetch("http://localhost:8080/v1/api/user" , {
  method: "POST",
  credential : "include",
  body : {
    ...
  }
})
```

```

})

// axios
axios.post('' , body , withCredentials: true )

```

✅ 서버에서 설정하기

“인증된 요청에 대한 헤더 설정하기”

일반적인 CORS 요청과는 다르게 응답 헤더를 설정해주어야 한다.

- **Access-Control-Allow-Credentials** : `true` 로 설정해주어야 함
- **Access-Control-Allow-Origin** : * 는 사용할 수 ❌
- **Access-Control-Allow-Methods** : * 는 사용할 수 ❌
- **Access-Control-Allow-Headers** : * 는 사용할 수 ❌

⇒ 모든 Origin 에 대해서 허용을 해주지 않고, **분명한 Origin으로 설정**되어있어야 한다!!!

(인증 정보는 민감한 중요 정보이기 때문에)

⇒ **Access-Control-Allow-Credentials** 헤더가 true 로 설정 되어 있지 않으면 브라우저에서 CORS에 의해 응답이 거부된다.



CORS 3가지 시나리오 작동 체험 사이트

CORS Tutorial

Web site created using create-react-app

<https://chuckchoiboi.github.io/cors-tutorial/>

🌟 CORS 를 허용할 수 있는 방법 정리

1 서버에서 Access-Control-Allow-Origin 헤더 세팅하기

“가장 일반적이고, 이상적인 해결 방법”

- 서버의 문법에 맞게 HTTP 헤더를 설정해준다!!

2 Proxy 서버를 이용한다.

- 브라우저와 서버 중간의 **모든 Origin**을 허용한 **Proxy 서버**를 구축하여 Proxy 서버를 통해 API 요청을 주고 받는다.


⇒ CORS는 브라우저에서 세워 놓은 정책에 의해 발생하는 오류이기 때문에


3 Chrome 확장 프로그램 이용

- 로컬 (localhost) 환경에서 API 테스트 시, CORS 문제를 해결 할 때 사용

Allow CORS: Access-Control-Allow-Origin - Chrome Web Store

Easily add (Access-Control-Allow-Origin: *) rule to the response header.


 <https://chromewebstore.google.com/detail/allow-cors-access-control/lhobafahddgcelffkeicbaginigejlf?pli=1>



reference


[WEB] CORS 정책

CORS 정책이란 ?


 <https://velog.io/@pcjo1202/CS-CORS-%EC%A0%95%EC%B1%85>

WEB

CORS 정책

 악명 높은 CORS 개념 & 해결법 - 정리 끝판왕 🙌

악명 높은 CORS 에러 메시지 웹 개발을 하다보면 반드시 마주치는 멍멍 같은 에러가 바로 CORS 이다. 웹 개발의 신입 신고식이라고 할 정도로, CORS는 누구나 한 번 정도는 겪게 된다고 해도 과언이 아니다. 프론트엔드 개발자 입장에선 요청 코드를 이상하게 적은것도 아니고, 백엔드 개발자 입장에선 서버 코드나 세팅이 이상한것도 아니다. 모든게 멀쩡한데

 <https://inpa.tistory.com/entry/WEB-%F0%9F%93%9A-CORS-%F0%9F%92%AF-%EC%A0%95%EB%A6%AC-%ED%95%B4%EA%B2%B0-%EB%B0%A9%EB%B2%95-%F0%9F%91%8F>

CORS

Access to XMLHttpRequest has been blocked