#8 / Cross-Origin Resource Sharing (CORS)

The browsers's same-origin policy blocks
reading a resource from a different origin.

브라우저의 same-origin 정책은
다른 출처를 가진 리소스를 읽는 것을 차단합니다.

This mechanism stops a malicious site from reading another site's data, but it also prevents legitimate uses.

이러한 메커니즘은 악의적인 사이트가 다른 사이트를 읽는 것을 막습니다,
하지만 이는 합법적인 이용도 막습니다.

What if you wanted to get weather data from another country?

만약 당신이 다른 나라의 날씨 데이터를 얻고 싶다면 어떡하시겠습니까?

In a modern web application, an application often wants to get resources from a different origin.

현대 웹 어플리케이션에서는 종종 한 어플리케이션이 다른 출처로부터 온
리소스를 얻고 싶어합니다.

For example, you want to retrieve JSON data from a different domain or load images from another site into a <canvas> element.

예를 들어, 당신은 다른 domain 에서 온 JSON 데이터를 얻거나
다른 사이트의 이미지를 <canvas> 요소에 로드하고 싶습니다.

In other words, there are `public resources` that should be available for anyone to read, but the same-origin policy blocks that.

요약하면, 누구나 읽을 수 있어야 하는 공공 리소스들이 있지만,
sample-origin 정책은 이를 차단합니다.

Developers have used work-arounds such as JSONP,
but `Cross-Origin Resource Sharing (CORS)` fixes this in a standard way.

개발자들은 JSONP 라는 차선책을 사용하기도 하지만,
`Cross-Origin Resource Sharing (CORS)` 가 이를 해결하는
기본적인 방법입니다.

Enabling `CORS` lets the server tell the browser
it's permitted to use an additional origin.

`CORS` 를 활성하함으로써 서버가 브라우저에게
추가적인 출처로 사용이 가능하다는 것을 알려줄 수 있습니다.

How does a resource request work on the web?

어떻게 리소스 요청이 웹에 작용하는가?

Figure: Illustrated client request and server response
그림: 클라이언트 요청과 서버의 응답을 표현

A browser and a server can exchange data over the network using the `Hypertext Transfer Protocol (HTTP)`.

브라우저와 서버는 `Hypertext Transfer Protocol (HTTP)` 를 사용해서
네트워크를 통해 데이터를 교환할 수 있습니다.

HTTP defines the communication rules between the requester and the responder, including what information is needed to get a resource.

HTTP 는 리소스를 얻기 위해서는 어떤 정보를 포함시켜야 하는지,
요청자와 응답자간에 통신 규칙을 정의합니다

The HTTP header is used to negotiate the type of message exchange between the client and the server and is used to determine access.

HTTP 헤더는 클라이언트와 서버가 교환하는 메세지 타입을 결정하고
접근을 허용하는데 사용됩니다.

Both the browser's request and the server's response message are divided into two parts: `header` and `body`:

브라우저의 요청과 서버의 응답 메세지는 크게 두 유형으로 나누어집니다:
`헤더` 와 `바디`

Header

헤더

Information about the message such as the type of message or the encoding of the message.

메시지의 타입과 인코딩 방식같은 메세지와 관련된 정보입니다.

A header can include a variety of information expressed as key-value pairs.

헤더는 키-값 쌍으로 표현된 다양한 정보가 포함될 수 있습니다.

The request header and response header contain different information.

요청 헤더와 응답 헤더는 다른 정보가 들어있습니다.

It's important to note that headers cannot contain comments.

헤더에는 comments(?) 가 포함되지 않는 다는 점을 주의하십시오.

Sample Request header

예시 요청 헤더

```
Accept: text/html Cookie: Version=1
```

Python

The above is equivalent to saying "I want to receive HTML in response. Here is a cookie I have."

위는 "나는 HTML 응답을 받길 원해. 이건 내가 가진 쿠키야." 라고
말하는 것과 같습니다.

Sample Response header

예시 응답 헤더

```
Content-Encoding: gzip Cache-Control: no-store
```
Python

The above is equivalent to saying "Data is encoded with gzip. Do not cache this please."

위는 "데이터는 gzip 으로 인코딩되었어. 이걸 캐시하지는 마" 라고
말하는 것과 같습니다.

---

body

바디

The message itself.

메세지 그 자체.

This could be plain text, an image binary, JSON, HTML, and so on.

이는 plain text 나 image binary, JSON, HTML 외
어떤 것도 될 수 있습니다.

How does CORS work?

어떻게 CORS 가 작동하는 것입니까?

Remember, the same-origin policy tells the browser to block cross-origin requests.

다시 기억을 되살려보면, same-origin 정책은 브라우저에게
cross-origin 요청을 차단하라고 말해줍니다.

When you want to get a public resource from a different origin, the resource-providing server needs to tell the browser "This origin where the request is coming from can access my resource."

당신이 다른 출처의 공공 데이터를 얻고싶을 때, 리소스를 제공하는 서버는
브라우저에게 "이 요청을 보낸 출처는 내 리소스에 접근하기를 원해" 라고
말해주는 것이 필요합니다.

The browser remembers that and allows cross-origin resource sharing.

브라우저는 이를 기억하고 cross-origin 리소스를 공유하는 것을
허용합니다.

Step 1: client (browser) request

스텝 1: 클라이언트 (브라우저) 요청

When the browser is making a cross-origin request,
the browser adds an `Origin` header with the current origin (scheme, host, and port).

브라우저가 cross-origin 요청을 만들 때,
브라우저는 현재 origin (scheme, host, port) 정보를
가진 `Origin` 을 헤더에 추가합니다.

Step 2: server response

스텝 2: 서버 응답

On the server side, when a server sees this header, and wants to allow access, it needs to add an `Access-Control-Allow-Origin` header to the response specifying the requesting origin (or `*` to allow any origin.)

서버가 이 헤더를 보고 접근을 허용하게 하려면,
요청한 origin 정보를 담은 `Access-Control-Allow-Origin` 을
응답 헤더에 추가합니다.(또는 `*` 로 어떤 origin 이든 허용할 수 있습니다.)

Step 3: browser receives response

스템 3: 브라우저가 응답을 받음

When the browser sees this response with an appropriate `Access-Control-Allow-Origin` header,
the browser allows the response data to be shared with the client site.

적절한 `Access-Control-Allow-Origin` 이 담긴 헤더를 브라우저가 보면,
브라우저는 클라이언트 사이트에서 응답 데이터가 공유되는 것을 허용합니다.

Share credentials with CORS

CORS 로 신원정보 공유하기

For privacy reasons, CORS is normally used for "anonymous requests"—ones where the request doesn't identify the requestor.

사생활과 관련되어있다는 이유로, CORS 는 보통 "익명의 요청자"—요청이 있을 때, 요청자를 확인하지 않는 것을 기본으로 하여 사용됩니다.

If you want to send cookies when using CORS (which could identify the sender), you need to add additional headers to the request and response.

만약 보낸자가 누구인지 알 수 있는 쿠키를 CORS 를 사용해서 보낸다면, 당신은 요청과 응답에 추가적인 헤더를 추가해야 합니다.

Request

요청

Add credentials: 'include' to the fetch options like below. This will include the cookie with the request.

신원정보 추가: 아래처럼 fetch options 에 'include' 를 넣습니다. 이는 요청에 쿠키를 포함시킬 것입니다.

```
fetch('https://example.com', { mode: 'cors', credentials: 'include' })
```

Python

Response

응답

`Access-Control-Allow-Origin` must be set to a specific origin (no wildcard using `*`) and must set `Access-Control-Allow-Credentials` to `true`.

구체적인 origin 이 명시된 `Access-Control-Allow-Origin` 이
설정되어야 하며 (여기서는 와일드카드 `*` 사용불가), `Access-Control-Allow-Credentials` 를
`ture` 로 설정합니다.

```
HTTP/1.1 200 OK Access-Control-Allow-Origin: https://example.com Access-Control-
Allow-Credentials: true
```

Python

Preflight requests for complex HTTP calls

복잡한 HTTP 요청을 위한 preflight 요청

If a web app needs a complex HTTP request,
the browser adds a preflight requests to the front of the request chain.

만약 웹 앱이 복잡한 HTTP 요청이 필요하다면,
브라우저는 요청 절차 앞에 preflight 요청을 추가합니다.

The CORS specification defines a `complex request` as

CORS 명세에서 `complex request` 란

- A request that uses methods other than GET, POST, or HEAD

  GET, POST, HEAD 가 아닌 다른 방법을 사용한 요청
- A request that includes headers other than `Accept`, `Accept-Language` or `Content-Language`

  `Accept`, `Accept-Language`, `Content-Language` 가 아닌
  다른 헤더를 포함한 요청
- A request that has a `Content-Type` header other than `application/x-www-form-urlencoded`, `multipart/form-data`, or `text/plain`

  `Content-Type` 에 `application/x-www-form-urlencoded`, `multipart/form-data`, `text/plain` 가 아닌 헤더를 가진 요청

Browsers create a preflight request if it is needed.

브라우저는 필요하다면 preflight 요청을 만듭니다.

It's an OPTIONS request like below and is sent before the actual request message.

이는 아래처럼 실제 요청 메세지를 보내기 전의 OPTIONS 요청을 가리킵니다.

```
OPTIONS /data HTTP/1.1 Origin: https://example.com Access-Control-Request-Method: DELETE
```

Python

On the server side, an application needs to respond to the preflight request with information about the methods the application accepts from this origin.

서버측면의 어플리케이션은 이 origin 으로부터 받을 수 있는 방법이 담긴 정보와 함께 preflight 요청에 응답할 필요가 있다

```
HTTP/1.1 200 OK Access-Control-Allow-Origin: https://example.com Access-Control-Allow-Methods: GET, DELETE, HEAD, OPTIONS
```

Python

The server response can also include an `Access-Control-Max-Age` header to specify the duration (in seconds) to cache preflight results so the client does not need to make a preflight request every time it sends a complex request.

서버의 응답은 또한 preflight 요청에 대한 결과를 언제까지 지니고 있을 지 명시하는 지속시간을 (초 단위) `Access-Control-Max-Age` 헤더를 포함시켜, 그럼으로써 클라이언트가 복잡한 요청을 보낼때마다 preflight 요청을
만들 필요가 없도록 할 수 있다.