6 minutes

Les CORS

Les CORS

Le CORS (Cross-origin resource sharing) est un mécanisme de sécurité des navigateurs.

Il permet de définir si un utilisateur peut accéder à des ressources d'un serveur situé sur une autre origine que le site courant.

Cela permet d'éviter que du JavaScript d'un site Web puisse accéder aux données d'un autre site Web. Sinon des scripts malveillants pourraient très simplement récupérer beaucoup d'informations sur les utilisateurs d'une page.

Ce mécanisme est valable pour les Web APIs API XMLHttpRequest et Fetch.

Une **origine est constitué d'un domaine, d'un protocole et d'un port**.

Le mécanisme CORS s'applique donc dès lors que la requête est envoyée à un domaine différent, ou sur un port différent, ou en utilisant un protocole différent.

Cela signifie par exemple que du JavaScript récupéré en HTTPS ne pourra pas effectuer une requête HTTP sur le même domaine par défaut.

Par défaut, du JavaScript ne peut donc effectuer des requêtes que sur la même origine depuis laquelle il a été chargé (c'est ce qu'on appelle la Same-origin policy).

Seul le serveur qui contrôle la ressource peut décider de sa politique CORS, il s'agit donc d'une problématique uniquement serveur.

Mais il est très important de comprendre les mécanismes car tout développeur Web rencontrera un jour ou l'autre cette erreur.

L'origine est définie par le navigateur comme entête de la requête HTTP. Elle ne peut pas être modifiée manuellement.

Par exemple, si nous effectuons une requête avec du JavaScript provenant de https://dyma.fr, nous aurons :

```
origin: https://dyma.fr
```

Faisons un essai :

https://codesandbox.io/embed/js-c15-l4-1-46jzt

Nous avons un échec car la requête part de l'origine du `sandbox` qui n'est pas la même que `dyma.fr`. Or nous n'autorisons que nos domaines à accéder à nos ressources.

Les requêtes simples

Les requêtes simples sont des requêtes qui utilisent les méthodes `GET`, `POST` ou `HEAD` ET qui utilisent uniquement un ou plusieurs entêtes de la liste suivantes :

`Accept`, `Accept-Language`, `Content-Language`, `Content-Type`, `Last-Event-ID`, `DPR`, `Save-Data`, `Viewport-Width`, `Width`.

Dernière condition, si l'entête `Content-Type` est utilisé, les seules valeurs possibles sont : `application/x-www-form-urlencoded`, `multipart/form-data` et `text/plain`.

Si la requête respecte ces conditions elle est considérée comme simple.

Dans ce cas, le navigateur envoie directement la requête en spécifiant l'entête `Origin`.

Le serveur peut ensuite décider selon l'`Origin` de répondre ou non.

Si il accepte la requête `CORS` la réponse aura un entête `access-control-allow-origin` :

```
access-control-allow-origin: https://dyma.fr
```

La valeur peut être aussi `*` qui signifie que le serveur accepte toutes les origines pour les requêtes `CORS`.

Les autres requêtes

Toutes les autres requêtes nécessitent l'envoi d'une requête `CORS` préliminaire par le navigateur appelée `prelight request` avec la méthode `OPTIONS`.

Vous n'avez aucun contrôle sur cette requête : c'est le navigateur qui la fait.

Cette requête préliminaire aura trois entêtes importants :

`origin` qui va permettre au serveur de déterminer si il accepte l'accès.

`Access-Control-Request-Headers` qui va contenir tous les `headers` passés en option de la requête et permet au serveur de les contrôler.

`Access-Control-Request-Method` qui va contenir la méthode de la requête demandée.

Exemple :

```
...
access-control-request-headers: x-test
access-control-request-method: DELETE
origin: https://dyma.fr
...
```

Le serveur doit ensuite envoyer une réponse qui contiendra des entêtes que le navigateur analysera pour décider si il effectuera la requête ou renverra une erreur :

`Access-Control-Allow-Header` qui va permettre au navigateur de comparer.

`Access-Control-Allow-Headers` qui doit contenir tous les `headers` demandés.

`Access-Control-Allow-Method` qui doit contenir la méthode de la requête demandée.

`Access-Control-Max-Age` qui permet de spécifier un nombre de secondes pour la mise en cache par le navigateur de la politique `CORS` du serveur spécifié dans les entêtes précédents. Ainsi, le navigateur n'aura pas à faire de requêtes préliminaires si elles respectent les règles spécifiées par le serveur.

Exemple :

```
...
access-control-allow-headers: x-test
access-control-allow-methods: GET,HEAD,PUT,PATCH,POST,DELETE
access-control-allow-origin: https://dyma.fr
...
```

Si tous les entêtes correspondent alors le navigateur effectuera la requête, sinon il ne la fera pas.

Les entêtes (`headers`) accessibles en `JavaScript`

Lorsque vous ouvrez l'onglet `Network` sur votre navigateur vous verrez beaucoup d'entêtes sur la réponse :

```
access-control-allow-credentials: true
access-control-allow-origin: https://zex33.csb.app
age: 134
cache-control: max-age=14400
cf-cache-status: HIT
```

```
cf-ray: 550743a68cb8dbe3-LHR
content-encoding: br
content-type: application/json; charset=utf-8
date: Sun, 05 Jan 2020 17:45:08 GMT
etag: W/"160d-1eMSsxeJRfnVLRBmYJSbCiJZ1qQ"
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beac
expires: -1
pragma: no-cache
server: cloudflare
set-cookie: __cfduid=d9209030a6f832267668235323528259715782463 08; expires=Tue, 04-Feb
status: 304
vary: Origin, Accept-Encoding
via: 1.1 vegur
x-content-type-options: nosniff
x-powered-by: Express
```

Mais par défaut, pour des raisons de sécurité, le `JavaScript` ne peut pas accéder à tous ces entêtes lors de requêtes `CORs`.

Il ne peut accéder qu'aux entêtes `Cache-Control`, `Content-Language`, `Content-Type`, `Content-Type`, `Expires`, `Last-Modified` et `Pragmal`.

Pour exposer d'autres entêtes aux `JavaScript`, le serveur doit explicitement le spécifier en ajoutant un entête spécifique pour le navigateur.

Par exemple, pour rendre l'entête `Content-Length` accessible au `JavaScript` il faut que le serveur ajoute l'entête suivant :

```
Access-Control-Expose-Headers: Content-Length
```

Exemple

Nous allons prendre un exemple :

```
const reponse = await fetch("https://jsonplaceholder.typicode.com/users", {
  method: "delete",
  headers: { "x-test": 2 }
});
```

Ici, nous avons une requête non simple pour deux raisons : nous utilisons la méthode `DELETE` et nous avons un entête autre que ceux dans la liste que nous avons vue.

Le navigateur effectue donc une requête préliminaire, les entêtes de la requête `OPTIONS` qui nous intéresse sont les suivants :

```
…
access-control-request-headers: x-test
access-control-request-method: DELETE
origin: https://zex33.csb.app
…
```

Nous retrouvons la méthode et les entêtes de la requête demandée par le `JavaScript` et l'origine.

Le serveur va analyser ces entêtes et dans notre cas il a des règles `CORS` très ouvertes, il répond donc :

```
…
access-control-allow-headers: x-test
access-control-allow-methods: GET,HEAD,PUT,PATCH,POST,DELETE
access-control-allow-origin: https://zex33.csb.app
…
```

Ce qui signifie qu'il accepte l'entête demandé et toutes les méthodes `HTTP`, et qu'il accepte notre origine.

Le navigateur contrôle ces entêtes et peut donc effectuer la requête `DELETE`.

Vous pouvez tester en regardant la console :
https://codesandbox.io/embed/js-c15-l4-2-zex33

Les `credentials`

Par défaut, là encore pour des raisons de sécurité, les requêtes `CORS` n'envoient pas les informations relatives à l'authentification.

Elles n'envoient donc pas les cookies relatifs au domaine d'origine et pas non plus les cookies relatifs au domaine cible, elles n'envoient pas non plus d'entête `HTTP-Authorization` (contenant par exemple un `token JWT`).

Un `cookie` est une liste de paires clé/valeur stockée par votre navigateur.

Un serveur demande à votre navigateur de stocker le `cookie` et celui-ci s'exécute en incluant les règles spécifiées par le serveur.

Par défaut, le `cookie` est accessible par le `JavaScript` de l'origine et il est envoyé avec toutes les requêtes à la même origine (au serveur qui a demandé la création du `cookie` au navigateur).

Les `cookies` peuvent servir pour l'authentification (sessions), le tracking (analytics), la sauvegarde d'un panier, de préférences utilisateurs etc.

Pour inclure l'envoi des `cookies`, il faut utiliser l'option `credentials` :

```
fetch('https://dyma.fr, {
  credentials: "include"
});
```

Ici, la requête enverra tous les `cookies` créés avec l'origine `https://dyma.fr`.

Le serveur décide ensuite s'il accepte ou non de recevoir les `credentials`.

Si oui, il faut utiliser l'entête `Access-Control-Allow-Credential` :

```
…
Access-Control-Allow-Credentials: true
```

Si cet entête est reçu en réponse à une requête préliminaire, alors le navigateur enverra les informations d'authentification (contenues par exemple dans les `cookies`), sinon il n'enverra pas la requête.

Si la requête est simple, et que cet entête n'est pas renvoyée, (et qu'elle a donc été envoyée avec les informations d'authentification), la réponse sera ignorée par le navigateur et le `JavaScript` n'y aura pas accès.