



# CORS

Qu'est ce que c'est ?



Intro « Bertrand »

On peut appeler une api depuis n'importe quel url.

Ex depuis google.com + api StarWars

```
fetch('https://swapi.dev/api/people').then(response=>response.json())
```

On récupère une liste des personnages

Parfait, je vais pouvoir récupérer toutes les données de datasquare, alors!

```
fetch('https://api.datasquare.analytics.safran/blog/blogposts').then(response=>response.json())
```

Tiens, ça marche

Il y a datahub

```
fetch('https://datahub-api.analytics.safran').then(response=>response.json())
```

Ca ne marche pas

Regardons l'erreur dans le navigateur: "blocked by CORS policy"

Que se passe t'il ?

Un problème de CORS, c'est un problème de partage de ressources.

Ce problème est lié à un contexte précis. A savoir ici entre deux sources / url / api / bdd différentes via le navigateur.



# QUE SE PASSE T'IL ?

Une histoire de partage de ressources

A quoi ça sert un cors ?

C'est un mécanisme qui permet à un site web avec une url foo.com de faire des requêtes de datas sur d'autres url /serveur (bar.com, andrea.com, sharon.com...).

S'il n'y a pas CORS chez bar.com, andrea.com, sharon.com... pour autoriser l'accès à foo.com, il ne peut pas réussir de requête vers ces serveurs.



A QUOI ÇA SERT LES CORS ?

# A QUOI ÇA SERT LES CORS ?



**foo.com**



**bar.com**

Le CORS est ici !

Le site foo.com ne peut faire que des requêtes vers son propre serveur, un serveur de la même origine.

Les cors chez bar.com, andrea.com, sharon.com... permettent ainsi à foo.com de faire des requêtes réussies sur Url / serveur d'origine croisées comme bar.com, andrea.com, sharon.com...





**GET**  
*Origin:* foo.com





# CLIENT

foo.com

# SERVER



foo.com

**GET**

*Origin:* foo.com



**CLIENT**

foo.com

**SERVER** 

foo.com

**200  
OK**



**CLIENT**

foo.com

**200  
OK**

**SERVER** 

foo.com





**GET**

*Origin:* foo.com





**cross-origin  
request**



**GET**  
*Origin:* foo.com



**200**

*Access-Control-Allow-Origin:* foo.com





**200**

*Access-Control-Allow-Origin:* foo.com

On comprend son nom: « Cross-origin resource sharing » (CORS) ou « partage des ressources entre origines croisées / multiples ».

# CORS

Cross-origin resource sharing

Partage de ressource entre origines croisées / multiples

A noter

On parle d'origine, c'est quoi l'origine? C'est la combinaison protocole, domaine et port

Si l'un de ces 3 éléments change, l'origine n'est plus la même.



<https://www.foo.com:8000>

protocole

domaine

port



Le code est bien implémenté du côté serveur.

Pourtant c'est dans le navigateur, du côté client que ça se passe.

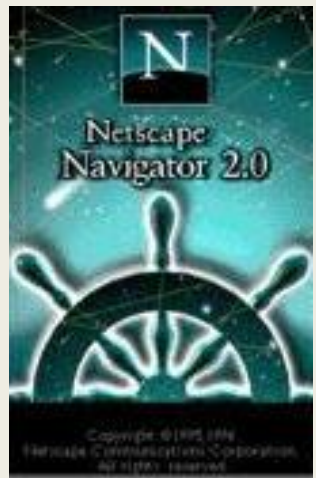
Depuis 1995, Netscape 2, les navigateurs appliquent une règle de sécurité: "same-origin policy" (politique de même origine).

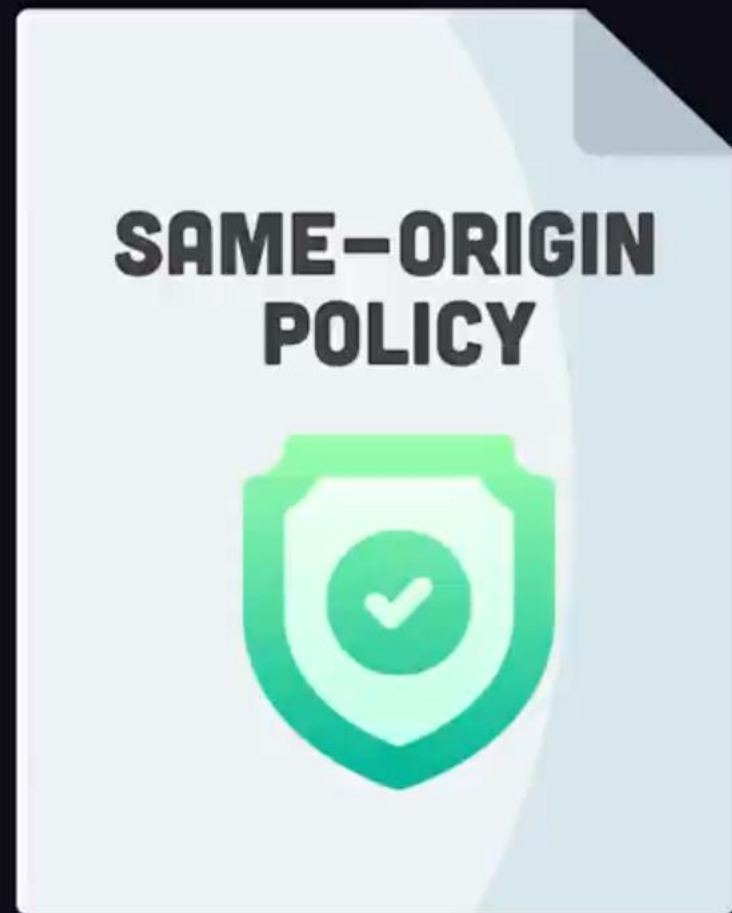
Les navigateurs, dans leur système de sécurité, autorisent donc un site web à demander librement des images et des datas depuis sa propre url. Mais bloque tout ce qui vient d'url externe sauf si certaines conditions sont réunies. A savoir la présence de cors chez bar.com.

Exemple avec starwars de tout à l'heure.

Les requêtes XMLHttpRequest (XHR) et l'API Fetch respectent la règle "same-origin policy" d'origine unique. Cela signifie qu'une application web qui utilise ces API peut donc uniquement émettre des requêtes vers la même origine que celle à partir de laquelle l'application a été chargée.

Si elle veut émettre des requêtes vers d'autres origines, elles devront avoir un CORS qui l'y autorise.





Plus généralement, ça sert à se protéger des failles de sécurité, en particulier: Cross site request forgery / XSRF dit c-surf (falsification des requêtes inter sites)



# **SAME-ORIGIN POLICY**



# XSRF

Cross site request forgery  
(falsification des requêtes inter sites)

On note quelques exceptions de ressources qui peuvent être embarqués malgré leur origine incompatible avec la same-origin policy.

Par exemple les cross origin *Embarqués* comme le iframe, on le rencontre souvent pour embed un contenu

Voir MDN:

[https://developer.mozilla.org/fr/docs/Web/Security/Same-origin\\_policy#acc%C3%A8s\\_r%C3%A9seau\\_cross-origin](https://developer.mozilla.org/fr/docs/Web/Security/Same-origin_policy#acc%C3%A8s_r%C3%A9seau_cross-origin)


## Ressources qui peuvent être embarquées malgré leur origine incompatible avec la *same-origin policy*

- JavaScript avec `<script src="..."></script>`. Les messages d'erreur de syntaxe ne sont disponibles que pour les scripts ayant la même origine.
- CSS avec `<link rel="stylesheet" href="...">`. Étant donnée la [souplesse des règles de syntaxe](#) du CSS, les CSS d'origine différentes nécessitent une entête `Content-Type` correcte. Les restrictions varient selon les navigateurs : [IE](#), [Firefox](#), [Chrome](#), [Safari](#) et [Opera](#).
- Images avec `<img>`. Les formats d'image supportés, comprenant PNG, JPEG, GIF, BMP, SVG...
- Fichiers média avec `<video>` et `<audio>`.
- Objets avec `<object>` (en-US), `<embed>` et `<applet>`.
- Fontes de polices avec `@font-face`. Certains navigateurs autorisent les fontes cross-origin, d'autres appliquent la same-origin policy.
- N'importe quoi avec `<frame>` (en-US) et `<iframe>`. Un site peut utiliser l'entête [X-Frame-Options](#) (en-US) pour interdire cela depuis une page n'ayant pas la même origine.

**SAME-ORIGIN  
POLICY**



# CORS



" J'accepte une requête *cross origin* dans ce ou ces cas particulier(s) "



COMMENT CA MARCHE ?

Le mécanisme consiste à ajouter dans des en-têtes HTTP un ou des user-agent. Ainsi l'**en-tête de la réponse HTTP** décrit exactement quels serveurs peuvent charger les données et les rendre disponibles à l'utilisateur.

Le user agent est (pour le moment pour nous) Acces-Control-Allow-Origin = d'où est ce que tu accepte que la requête arrive? Le serveur dit au client, ce sont à ces url que nous autorisons l'accès.

user-agent = information échangée sous forme de chaîne de caractères, entre le navigateur utilisé par l'internaute et le serveur du site web visité



**200**

*Access-Control-Allow-Origin:* foo.com



**CLIENT**

foo.com

**SERVER** 

bar.com

**200**

*Access-Control-Allow-Origin:* foo.com



**Access-Control-Allow-Origin**



**MATCH**

**Origin**



▼ Response Headers (278 B)

HTTP/1.1 200 OK  
X-Powered-By: Express  
Access-Control-Allow-Origin: https://www.foo.com  
Vary: Origin  
Content-Type: application/json; charset=utf-8  
Content-Length: 23  
ETag: W/"17-5bdkTC043EEL9K1bdRBU8k2HqAU"  
Date: Mon, 29 Mar 2021 12:47:12 GMT  
Connection: keep-alive

▼ Request Headers (341 B)

GET / HTTP/1.1  
Host: localhost:3000  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64  
Accept: \*/\*  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Referer: http://localhost: https://www.foo.com  
Origin: http://localhost: https://www.foo.com

**ACCES-CONTROL-  
ALLOW-ORIGIN**

**« Nous autorisons  
l'accès à ces url ! »**

A noter

Avec le caractère wildcard '\*', il est possible d'ouvrir à toutes les url.

Comme sur l'api starwars

Dans quelles conditions laisser cela?

Si c'est seulement du Get ou head, la bdd ne peut pas être modifiée. Mais toutes vos données sont publiques

Comme pour l'api starwars

# Access-Control-Allow-Origin



# Origin



## ▼ Response Headers (278 B)

HTTP/1.1 200 OK  
X-Powered-By: Express  
Access-Control-Allow-Origin: \*  
Vary: Origin  
Content-Type: application/json; charset=utf-8  
Content-Length: 23  
ETag: W/"17-5bdkTC043EEL9K1bdRBU8k2HqAU"  
Date: Mon, 29 Mar 2021 12:47:12 GMT  
Connection: keep-alive

## ▼ Request Headers (341 B)

GET / HTTP/1.1  
Host: localhost:3000  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64  
Accept: \*/\*  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Referer: http://localhost: https://www.foo.com  
Origin: http://localhost: https://www.foo.com



ET DANS LA PRATIQUE?

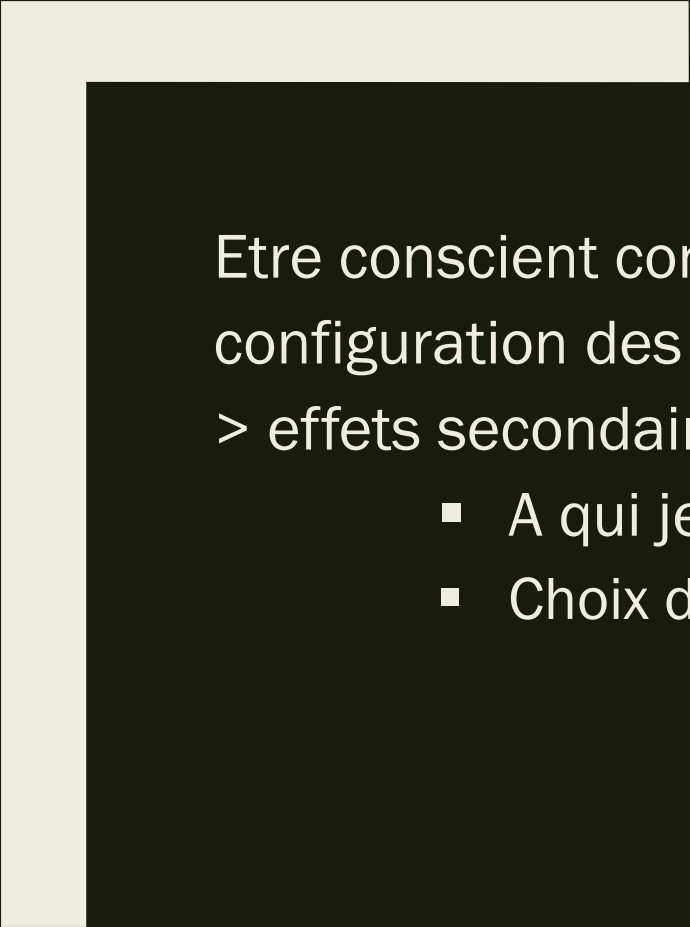
Et dans la pratique?

**Etre conscient contexte navigateur / api.** Si ouvre ailleurs ou dans un autre langage, il n'y a plus la sécurité !

**Etre conscient de la configuration des CORS.** Être vigilant sur la sécurité.


Que dois-je prendre en compte ?

- **Qui a un besoin légitime d'accès ?** Peut-être n'y a-t-il qu'un site Web spécifique ou un petit ensemble de sites Web qui ont besoin d'un accès. Si c'est le cas, dressez-en la liste. White list, black list
- **Tous les endpoints doivent-ils être exposés ?** Examinez vos différents endpoints et vous verrez peut-être qu'ils ont des utilisations spécifiques différentes qui nécessitent **des configurations CORS différentes**
- **Quels sont les effets secondaires pour les endpoints fournissant des données ?** Certains endpoints, généralement les endpoints **comme GET**, fourniront des données mais ne modifieront rien dans l'état de votre serveur (ne modifieront pas l'état d'authentification ou les données stockées). Pour ces endpoints, **vérifiez si les données que vous fournissez sont confidentielles** et si les CORS autorisés ont une raison légitime de les demander.
- **Et les effets sur les endpoints qui modifient les données ?** D'autres endpoints, généralement les endpoints de POST/DELETE/PUT, fournissent des données mais **modifient également l'état de votre serveur**. Dans ce cas, le risque est plus élevé car le CORS peut être utilisé pour **déclencher des actions involontaires**, telles que la modification des données de l'utilisateur, la modification des informations de connexion, l'envoi d'e-mails, etc.
- **Qu'en est-il des services tiers ?** Si vous devez fournir une API à des tiers, il est bon que vous implémentiez l'autorisation d'une manière qui requiert le consentement de l'utilisateur. Par exemple, OAuth vous permet d'autoriser des sites Web et des applications tiers et requiert le consentement explicite de l'utilisateur.



Etre conscient contexte navigateur / serveur et de la configuration des CORS

> effets secondaires sur la sécurité

- A qui je donne l'accès?
  - Choix des endpoints
- 



VOUS AVEZ BIEN SUIVI ?



# VOUS AVEZ BIEN SUIVI ?

C'est quoi un CORS alors?

1. Ca fait mal au pied
2. Un mécanisme coté serveur pour accepter ou pas une requête *cross origin* (origine croisée)
3. Un formidable groupe de pop rock irlandais des années 2000

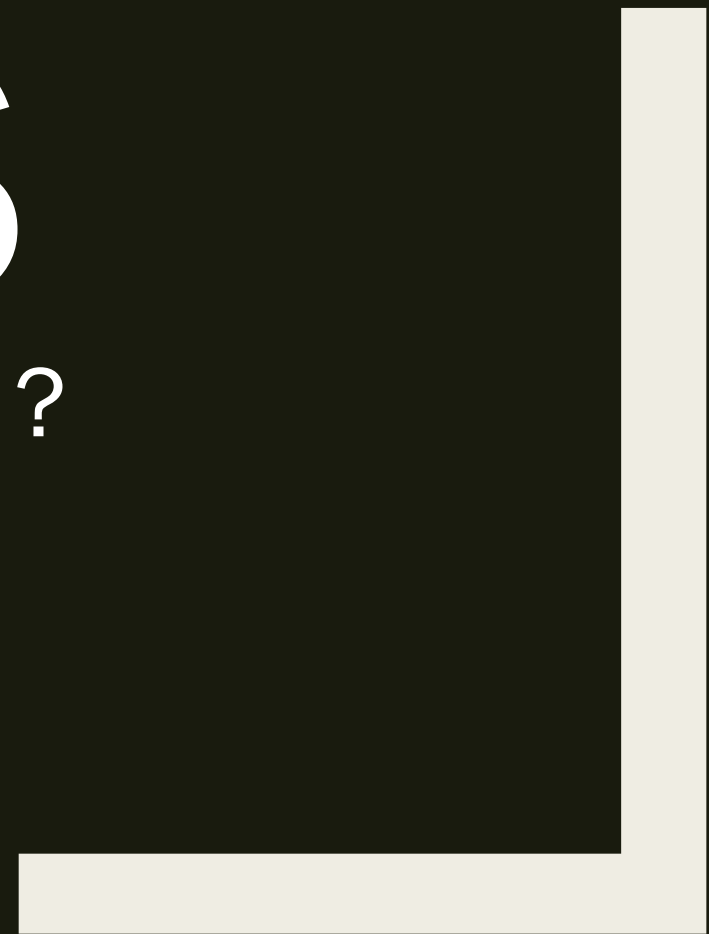
A decorative graphic consisting of a thick white L-shaped line. The vertical part of the 'L' is on the left, and the horizontal part extends to the right, positioned above the title.

# VOUS AVEZ BIEN SUIVI ?

1. Un mécanisme pour contourner la *same origin policy* du navigateur
2. Pour que le serveur distant accepte ou pas une requête *cross origin* (origine croisée)
3. Uniquement dans un contexte navigateur / url / javascript. Donc attention!

# CORS

Qu'est ce que c'est ?





# LES PRINCIPAUX MECANISMES

## requête simple

En principe, une Cross-Origin Request est une **requête HTTP**. Certaines méthodes ne posent en principe aucun problème. Comme nous l'avons vu GET et HEAD ne peuvent pas modifier les données et ne sont donc généralement pas perçus comme un risque lié à la sécurité.

Origin: <origin>:

Ceci indique l'origine de la demande du client. Lorsque vous travaillez avec un frontend et un backend, comme indiqué précédemment, ce sera l'hôte de votre application frontend.

Access-Control-Allow-Origin: <origin>: Ceci est utilisé pour spécifier l'origine autorisée à accéder à la ressource sur le serveur.

# REQUÊTE SIMPLE


Access-Control-Allow-*Origin* : quelle origine est autorisée ?

## ▼ Response Headers (278 B)



```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: http://localhost:5000
Vary: Origin
Content-Type: application/json; charset=utf-8
Content-Length: 23
ETag: W/"17-5bdkTC043EEL9K1bdRBU8k2HqAU"
Date: Mon, 29 Mar 2021 12:47:12 GMT
Connection: keep-alive
```

## ▼ Request Headers (341 B)



```
GET / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Origin: http://localhost:5000
```

## requête nécessitant une requête préliminaire

But: Sécurité quand on utilise des requêtes comme PATCH, PUT ou DELETE, peuvent être nuisibles

Principe: Le client envoie d'abord une **preflight requests (requête préliminaire)** dans laquelle il précise seulement la **méthode** HTTP qui sera ensuite adressée au serveur. Comme une **sonde** pour déterminer si le serveur prend en charge la requête principale sur le point d'être effectuée.

Lorsque la confirmation positive est obtenue, la requête principale est alors envoyée.

Puis il demande si la requête est considérée comme sûre. Pour cela, il faut utiliser l'en-tête OPTIONS. Ce n'est qu'après une réponse positive que la demande effective peut être effectuée.

La **méthode HTTP OPTIONS** est utilisée pour décrire les options de communication pour la ressource ciblée.

Entre les lignes 1 à 12 qui précèdent, on voit la requête préliminaire avec la méthode [OPTIONS](#). Le navigateur détermine qu'il est nécessaire d'envoyer cela à cause des paramètres de la requête fournie par le code JavaScript. De cette façon le serveur peut répondre si la requête principale est acceptable et avec quels paramètres. OPTIONS est une méthode HTTP/1.1 qui est utilisée afin de déterminer de plus amples informations à propos du serveur. La méthode OPTIONS est une méthode « sûre » (*safe*) et ne change aucune ressource. On notera, qu'avec la requête OPTIONS, deux autres en-têtes sont envoyés (cf. lignes 10 et 11) :

# REQUÊTE NECESSITANT UNE *REQUÊTE PRÉLIMINAIRE*

Access-Control-Allow-*Methods* : quelles méthodes de requête HTTP sont autorisées ?

## Preflight request

```
OPTIONS /resources/post-here/ HTTP/1.1    Request header
Host: truc.autre
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Origin: http://toto.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type
```

```
HTTP/1.1 200 OK    Response header
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2.0.61 (Unix)
Access-Control-Allow-Origin: http://toto.example
Access-Control-Allow-Methods: POST, GET
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 0
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

## Main request

```
POST /resources/post-here/ HTTP/1.1
Host: truc.autre
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Referer: http://toto.example/exemples/preflightInvocation.html
Content-Length: 55
Origin: http://toto.example
Pragma: no-cache
Cache-Control: no-cache
```

```
<?xml version="1.0"?><personne><nom>Toto</nom></personne>
```

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:40 GMT
Server: Apache/2.0.61 (Unix)
Access-Control-Allow-Origin: http://toto.example
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 235
Keep-Alive: timeout=2, max=99
Connection: Keep-Alive
Content-Type: text/plain
```



## requêtes avec informations d'authentification

But: Ajouter une couche de sécurité

Principe: Les serveurs peuvent également indiquer aux clients s'il est nécessaire de fournir des informations d'authentification (que ce soit des cookies ou des données d'authentification HTTP) avec les requêtes.

Par défaut les cookies sont bloqués. Dans la requête, la commande fetch, il faut ajouter `credential include`. Mais surtout, du côté de l'autre origine, il faut accepter dans le cors (`credential à true`).

Lorsque la confirmation positive est obtenue, la requête principale est alors envoyée.

# REQUÊTES AVEC INFORMATIONS D'AUTHENTIFICATION

Access-Control-Allow-**Credentials** : ajoute une couche de sécurité

```
GET /resources/access-control-with-credentials/ HTTP/1.1
Host: truc.autre
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: http://toto.example/exemples/credential.html
Origin: http://toto.example
Cookie: pageAccess=2

HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:34:52 GMT
Server: Apache/2.0.61 (Unix) PHP/4.4.7 mod_ssl/2.0.61 OpenSSL/0.9.7e mod_fastcgi/2
X-Powered-By: PHP/5.2.6
Access-Control-Allow-Origin: http://toto.example
Access-Control-Allow-Credentials: true
Cache-Control: no-cache
Pragma: no-cache
Set-Cookie: pageAccess=3; expires=Wed, 31-Dec-2008 01:34:53 GMT
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 106
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

Request header

Response header

# DIFFÉRENTS CORS HEADER

---

**Access-Control-Allow-Origin** : quelle origine est autorisée ?

---

**Access-Control-Allow-Credentials** : les requêtes sont-elles autorisées même si le Credentials Mode est réglé en include ?

---

**Access-Control-Allow-Headers** : quels en-têtes peuvent être utilisés ?

---

**Access-Control-Allow-Methods** : quelles méthodes de requête HTTP sont autorisées ?

---

**Access-Control-Expose-Headers** : quels en-têtes peuvent être affichés ?

---

**Access-Control-Max-Age** : quel est le délai de validité de la requête préliminaire ?

---

**Access-Control-Request-Headers** : quel en-tête HTTP est spécifié dans la requête préliminaire ?

---

**Access-Control-Request-Method** : quelle méthode HTTP est spécifiée dans la requête préliminaire ?    Origin : quelle est la source de la requête ?