

06.HTTP & AJAX

4 novembre 2025

Développement web il3

HTTP & AJAX

HE-Arc (DGR) 2025

HyperText Transfer Protocol

- Protocole application : invention www en 1990 (v0.9)
 - Connexion, GET, réponse, fermeture
- HTTP 1.0 (1996)
 - Entêtes de requête (Host, Referer, User-Agent, ...) et réponse (Content-Type, Set-Cookie, Location, ...)
- HTTP 1.1 (1997)
 - Nouveaux entêtes (Keep-alive, pipelining, cache, ...), Host obligatoire
- HTTP 2.0 ¹ (2015, basé sur SPDY)
 - Binaire, multiplexage connexions, compression entêtes, push, ...
 - Supporté par presque tous ² les navigateurs, une majorité de serveurs
- HTTP 3.0 ³ (2019, basé sur QUIC)
 - **UDP**, TLS 1.3 obligatoire, correction erreur, contrôle congestion, multiplexage (0 RTT)
 - TCP+TLS (http2) remplacés par UDP+QUIC

1. <https://docs.google.com/presentation/d/1eqae3OBCxwWswOsaWMAWRpqnmrVVrAfPQclfSqPkXrA/presentation#slide=id.p19>

2. <https://caniuse.com/#feat=http2>

3. <https://http3-explained.haxx.se/fr/>

Codes de réponse

- 1xx : Information
- 2xx : Succès
- 3xx : Redirection
- 4xx : Erreur Client
- 5xx : Erreur Serveur

Méthodes HTTP (verbes)

- GET : Demander une ressource
- POST : Création d'une ressource
- PUT : Remplacement total d'une ressource
- PATCH : Remplacement partiel d'une ressource
- DELETE : Suppression d'une ressource
- HEAD : Demande l'entête de la réponse, sans la ressource
- TRACE, OPTIONS, CONNECT

idempotentes sûres

Echanges HTTP

- Requête

```
GET / HTTP/1.1[CRLF]
Host: www.cff.ch[CRLF]
Connection: close[CRLF]
User-Agent: Opera/9.20 (Windows NT 6.0; U; en)[CRLF]
Accept-Encoding: gzip[CRLF]
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7[CRLF]
Cache-Control: no[CRLF]
Accept-Language: de,en;q=0.7,en-us;q=0.3[CRLF]
Referer: http://web-sniffer.net/[CRLF]
[CRLF]
```

- Réponse

```
HTTP Status Code: HTTP/1.1 302 Found
Date: Mon, 16 Nov 2009 08:01:35 GMT
Server: Apache
Location: http://www.sbb.ch/fr/
Content-Length: 205
```

Connection: close
Content-Type: text/html; charset=iso-8859-1

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head><title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="http://www.sbb.ch/fr/">here</a>.</p>
</body></html>
```

HTTP

- Requête POST : paramètres dans le corps

```
POST /login.jsp HTTP/1.1
Host: www.mysite.com
User-Agent: Mozilla/4.0
Content-Length: 27
Content-Type: application/x-www-form-urlencoded
```

userid=joe&password=guessme

- Outils HTTP
 - CLI : curl
 - Browser dev tools
- Exemples PATCH : mnot ⁴, SOA bits ⁵

AJAX : Historique

- Asynchronous Javascript And Xml
- Buzzword, Jesse James Garret ⁶, 2005
- Mise à jour sans rechargement intégral
- Utilisation de Remote Scripting ⁷ et de DOM
- Historique de techniques de remote scripting
 - (i)frames
 - Bibliothèques JS (ex : JSRS ⁸)

4. <https://www.mnot.net/blog/2012/09/05/patch>

5. <https://soabits.blogspot.ch/2013/01/http-put-patch-or-post-partial-updates.html>

6. <https://web.archive.org/web/20110102130434/http://www.adaptivepath.com/ideas/essays/archives/000385.php>

P

7. https://en.wikipedia.org/wiki/Remote_scripting

8. <https://www.ashleyit.com/rs/jsrs/test.htm>

- Utilisation des images/cookies (ex : GIF ⁹)
- Applets, Flash, ActiveX, ...
- XHR : XML HTTP Request (IE5, 1999 pour OWA)
- Fetch API
- Pas obligatoire d'avoir du JS, XML ni d'être asynchrone !

AJAX

- XHR est devenue la méthode standard jusqu'à 2018
 - Popularisée par Google (GMaps, GMail, ...)
 - Le w3c fait évoluer un draft ¹⁰ depuis 2006
- Principe
 1. Envoi de requête HTTP
 2. La réponse provoque l'exécution de la fonction de rappel
 3. Le DOM de la page est mis à jour
- Applications
 - GUI ressemblant à des app natives
 - MAJ dynamiques de formulaires, autocompletion
 - Validation avec interrogation du serveur
 - ...

L'objet *XMLHttpRequest*

- Initiative de Microsoft
 - Composant ActiveX de IE5
 - Adopté par Mozilla 1.0 et Safari 1.2
 - Standardisation W3C en cours
- Requête HTTP en JS
- Fonction de rappel (callback)
- Asynchrone : Non bloquant
- Non standard => différentes implémentations
- Supporté par la majorité des navigateurs
- Alternative souhaitable si JS désactivé

9. <https://web.archive.org/web/20100916110710/http://depressedpress.com/Content/Development/JavaScript/Articles/GIFAsPipe/Index.cfm>

10. <https://www.w3.org/TR/XMLHttpRequest/>

XHR en JS

```
var xhr;  
function createXMLHttpRequest()  
{  
    if (window.ActiveXObject)  
    {  
        xhr = new ActiveXObject("Microsoft.XMLHTTP");  
    }  
    else if (window.XMLHttpRequest)  
    {  
        xhr = new XMLHttpRequest();  
    }  
}
```

— Dans son contexte ¹¹

XHR en jQuery avec *load()*

```
<!DOCTYPE html>  
<html>  
<head>  
<script src="jquery.js"></script>  
<script>  
$(document).ready(function(){  
    $("button").click(function(){  
        $("#div1").load("demo_test.txt");  
    });  
});  
</script>  
</head>  
  
<body>  
    <div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>  
    <button>Get External Content</button>  
</body>  
</html>
```

— Tester ¹²

11. <https://www.xul.fr/xml-ajax.html#ajax-exemple>

12. https://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_ajax_load

XHR : propriétés et méthodes

- readyState, status, onreadystatechange
- responseText, responseXML
- open (Verbe, URI, async) :
 - Verbe HTTP : "GET", "POST" ou "PUT"
 - URI : destinataire de la requête
 - async (bool) : true = asynchrone, false = bloquant
- send (null | string) : peut être bloquante
- setRequestHeader(header, value)
- getResponseHeader(string)
- abort()

Envoi de données

- GET
 - Obtenir des données
 - Longueur URL limitée par le navigateur (2'048 pour IE)
 - Utilise le cache (navigateur, proxy)
 - manipulables par l'utilisateur (bookmarks, partage, ...)
- POST
 - Faire quelque chose
 - Données sensibles
 - Longueur limitée par le serveur (assez large)
 - Utilisation de la méthode send() de XHR
 - Requête Ajax en 2 temps (entête, puis données)

Envoi de données

- Cache
 - Client : Construire des URL uniques¹³
 - Serveur : Envoi d'entêtes¹⁴ interdisant le cache

```
MyXhr.open("GET", "fichier.xml", true);
MyXhr.setRequestHeader("Cache-Control", "no-store, no-cache, must-revalidate,
    post-check=0, pre-check=0");
MyXhr.setRequestHeader("Pragma", "no-cache");
MyXhr.setRequestHeader("Expires", "Wed, 09 Aug 2000 08:21:57 GMT");
```

13. <https://stackoverflow.com/questions/367786/prevent-browser-caching-of-jquery-ajax-call-result>

14. <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching>

Préférer GET, sauf

Détails ¹⁵

Réponse en texte

- Si la requête aboutit :
 - `readystate == 4`
 - `status == 200`
- La réponse est dans l'attribut `responseText`
- ou dans `responseXML`
 - Utilisation du DOM (`getElementsByTagName()`, ...)

Réponse en XML

```
<?xml version="1.0" ?>
<liste>
  <personne>
    <nom>Berger</nom>
    <prenom>Laurent</prenom>
  </personne>
  <personne>
    <nom>Borgo</nom>
    <prenom>Sébastien</prenom>
  </personne>
  <personne>
    <nom>Bux</nom>
    <prenom>Rémy</prenom>
  </personne>
</liste>
```

- Dans `responseXML`

Réponse en JSON ¹⁶

- Standard ¹⁷ depuis octobre 2013 (Douglas Crockford ¹⁸)

15. <https://blog.teamtreehouse.com/the-definitive-guide-to-get-vs-post>

16. <https://www.json.org/>

17. <https://ecma-international.org/publications-and-standards/standards/ecma-404/>

18. <https://www.crockford.com/>

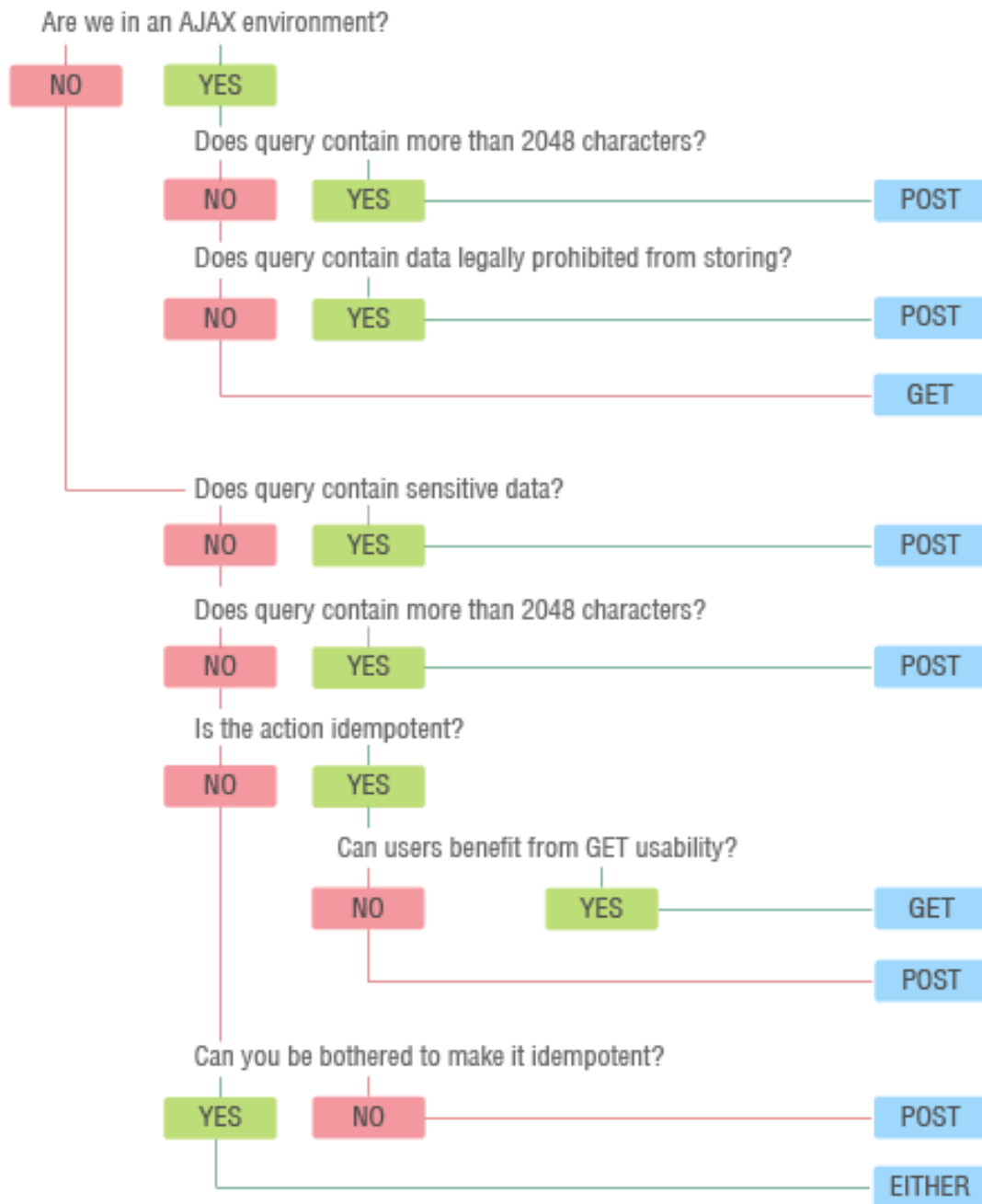


FIGURE 1 – “GETorPOST”

- Tableau d'objets js :
 - pour chacun, ses attributs sont des paires clé:valeur

```
{ "nom": "Berger", "prénom": "Laurent" }
```

```
[ "zéro", 1, 2, 3 ]
```

```
[
  { "nom": "Berger", "prénom": "Laurent" },
  { "nom": "Borgo", "prénom": "Sébastien" },
  { "nom": "Bux", "prénom": "Rémy" }
]
```

- Utilisation de : `var users = eval('(' + myXHR.responseText + ')');` pour créer le tableau d'objets correspondant

« eval is Evil » ¹⁹

- `eval()` : évalue et exécute la chaîne en paramètre
- Risque : instructions au lieu d'un tableau d'objets
- Solution : le parser ²⁰ JSON

```
var users = JSON.parse(myXHR.responseText);
var myString = JSON.stringify(users);
```

- Avec jQuery :

```
var obj = jQuery.parseJSON( '{ "nom": "Berger" } ' );
alert(obj.nom);
```

Fetch API

- Le successeur d'XHR est fetch ²¹ : Exemple ²²
- Fetch a un *polyfill* pour les navigateurs ne le supportant pas
- L'API Fetch est native et utilise les promesses ²³ plutôt que les callbacks

```
<div style="flex: 50%;">
```

19. <https://javascriptweblog.wordpress.com/2010/04/19/how-evil-is-eval/>

20. https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/JSON/parse

21. <https://fetch.spec.whatwg.org/>

22. https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

23. <https://www.promisejs.org/>

```

fetch("https://api.site.com/data")
  .then(function(response) {
    return response.json()
  })
  .then(function(json) {
    console.log(json);
  })
  .catch(function(error) {
    console.error("erreur", error)
  })

```

</div>

<div style="flex: 50%;">

```

async function getData() {
  try {
    const response = await fetch("https://api.site.com/data");
    const data = await response.json();
    console.log("Données reçues :", data);
  } catch (erreur) {
    console.error("Erreur :", erreur);
  }
}

```

</div>

Autres alternatives AJAX

- Utiliser une bibliothèque comme Axios ²⁴
- htmx ²⁵ : html repensé pour l'utilisation post-2020

Same Origin Policy (SOP ²⁶)

- Restriction pour les documents et scripts provenant d'une *origine*
- *origine* = protocole + hôte + port

24. <https://axios-http.com/docs/intro>

25. <https://htmx.org/docs/#ajax>

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

- Concerne les requêtes envoyées par des **scripts** (fetch, XHR)
- **But** : éviter l'accès au DOM par d'autres onglets
 - lecture cookies de sessions, requêtes auth, réponses API, ...
- Protège les sessions (interdit à 1 "onglet" d'accéder aux autres)
- Exceptions pour les éléments *inertes* : images, fonts, CSS, JS, medias (audio/video)
- CORS permet de contrôler l'accès depuis d'autres origines

Cross-Origin Resource Sharing (CORS ²⁷)

- **But** : Contourner la SOP quand c'est justifié
- Le serveur *décide*, le navigateur *contrôle*
- Utilisation des entêtes de réponses
- Avant de demander la ressource, le client demande l'autorisation :
 - Requête *preflight* OPTION
 - Réponse avec les autorisations dans les entêtes :
 - Access-Control-Allow-Origin : protocole + hôte + port
 - Access-Control-Allow-Methods : GET, POST, PUT, DELETE
 - Access-Control-Allow-Headers : entêtes des requêtes client
 - Access-Control-Allow-Credentials : cookies ou tokens

Traitement d'erreurs

- Utiliser les entêtes HTTP ²⁸
 - Champ Status
 - Code d'erreur
 - Corps structuré en JSON
- En PHP

```
header("Status: Message d'erreur explicite", true, 400);
```

- Afficher le message au client :

```
myXHR.getResponseHeader("Status");
```

27. <https://developer.mozilla.org/fr/docs/Web/HTTP/Guides/CORS>

28. <https://www.bennadel.com/blog/1860-using-appropriate-status-codes-with-each-api-response.htm>

Penser à l'utilisateur !

- Requêtes XHR non enregistrées dans l'historique :
 - Bouton précédent non opérationnel (sauf GET et URL uniques)
 - Pas de bookmark
 - solution via History API ²⁹
- Utilisabilité : signaler à l'utilisateur ce qui est en cours :
 - GIF AJAX loading ³⁰
 - Rectangle Loading en haut à droite (Google)
 - Yellow Fade Technique ³¹ (37signals) : partie modifiée
- Code client :
 - Pas de maîtrise performance
 - Mauvais code == Appli lente
- En cas de doute, faire tester des utilisateurs

Bonnes pratiques d'utilisabilité

- Trafic minimal
- Pas de surprise, Respect des conventions (UI Patterns ³²)
- Pas de distraction
- A11y (Contrast Checker ³³, Checklist ³⁴, ARIA ³⁵, Resources ³⁶)
- Ne pas switcher AJAX/non-AJAX
- Se mettre à la place de l'utilisateur

Sources

29. <https://html.spec.whatwg.org/multipage/nav-history-apis.html>

30. <https://loading.io/>

31. <https://codepen.io/Mestika/pen/KVGwKb>

32. <https://ui-patterns.com/patterns>

33. <https://color.a11y.com/>

34. <https://www.a11yproject.com/checklist/>

35. <https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA>

36. <https://www.a11yproject.com/resources/>