
# AJAX, c'est quoi ?

*Les sprays nettoyants ou autres détergents ne seront pas évoqués ici…*

**AJAX** ou Asynchronous JavaScript And XML est un terme qui s'est démocratisé grâce à cet article de Jesse James Garrett en 2005.

C'est une autre approche, une autre architecture particulièrement adaptée aux applications web.

## 1. Un peu d'histoire

- 1996 – JavaScript est ajouté au navigateur Netscape
- 1998 – Le Document Object Model (DOM)
- 1998 – Microsoft créer XMLHTTP, un composant ActiveX pour Internet Explorer 5.
- 2002 ~ 2005 – Ajouté à la norme ECMAScript relative au langage JavaScript, XMLHttpRequest est mit en oeuvre sur la plupart des navigateurs (Mozilla, Safari, Opéra …).
- 2005 – Article de Jesse James Garrett qui démocratise le terme et l'approche AJAX
- 2006 – XMLHttpRequest (xhr) devient une recommandation du W3C
- 2012 – XMLHttpRequest Level 2 élaborée par le W3C
- 2014 – whatwg continue le travail autour de xhr

## 2. L'approche classique

La plupart des actions effectuées par un utilisateur déclenche une requête HTTP vers un serveur web. Ce serveur effectue nombre d'opérations, decomposer la requête, appeler les services nécessaires, retrouver des données et suite à ce processus, retourner une page HTML à l'utilisateur.

Ce fonctionnement est adapté pour une consultation « classique » de contenus mais pas forcément pour des applications web.

## 3. Une autre approche

Dans l'article de Jesse James Garrett, AJAX y est décrit non pas comme une technologie mais bien comme une association de plusieurs technologies, une nouvelle façon de penser l'architecture d'une application web :

- Présentation | HTML / CSS
- Interaction | DOM
- échange et manipulation de données | XML
- Récupération asynchrone de données | XMLHttpRequest
- Liaisons et assemblage | JavaScript

En comparant une application web classique et en AJAX il est plus simple de saisir les subtilités.

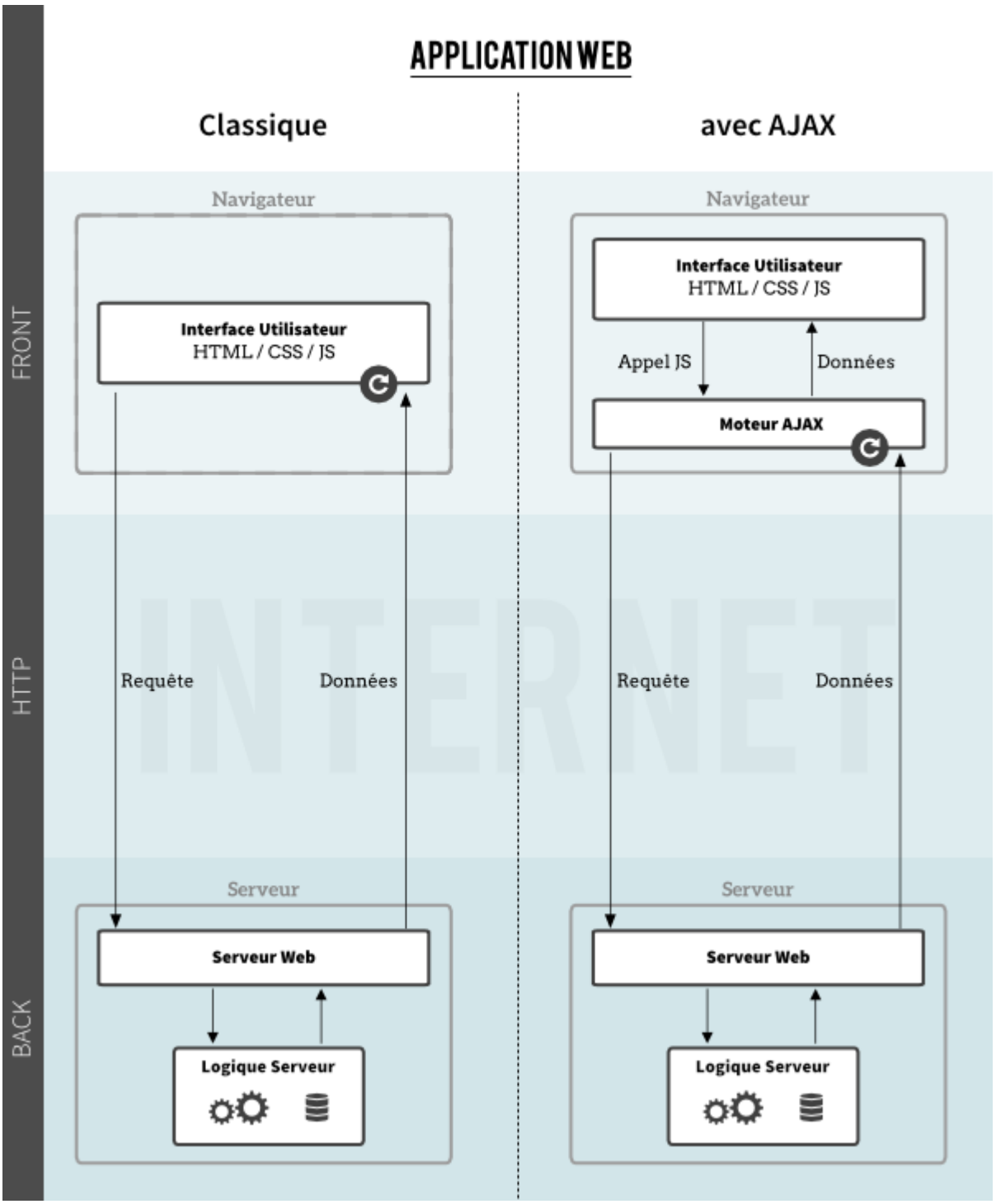


Schéma inspiré de celui proposé par Jesse James Garrett

# Synchrone contre asynchrone

La différence principale entre synchrone et asynchrone est que chaque action d'un utilisateur qui devrait normalement générer une requête HTTP (synchrone) prendra à la place la forme d'un appel JavaScript au moteur AJAX (asynchrone). Chaque réponse à une action utilisateur ne nécessite donc pas d'attendre une réponse du serveur.

Le moteur AJAX prendra en charge les actions utilisateurs et il se chargera de récupérer les données nécessaire, si il en a besoin. Le moteur fait donc des requêtes asynchrones sans obliger l'utilisateur à subir un changement d'état complet de l'application à chacune de ses actions.

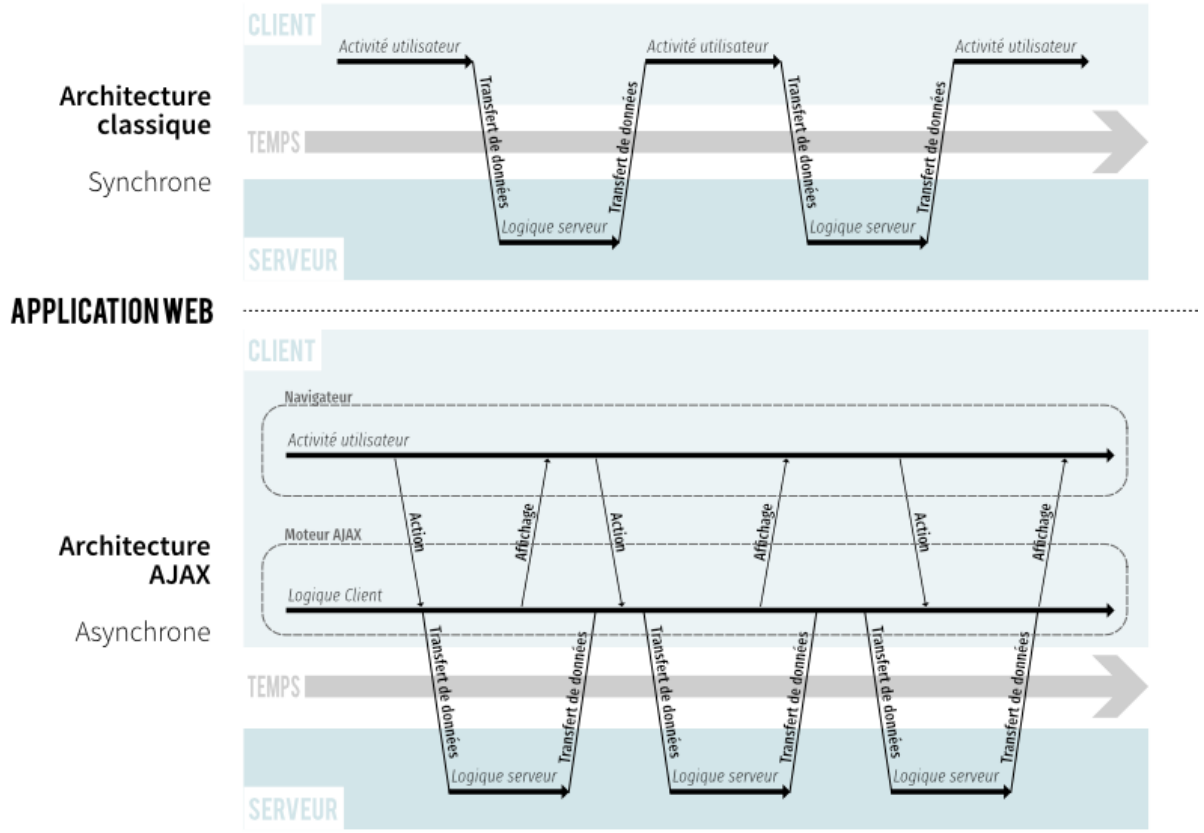


Schéma inspiré de celui proposé par Jesse James Garrett

## Exemple : Le lazy loading

Smashing Magazine (synchrone) VS Twitter (asynchrone)


# Ajax avec jQuery

jQuery.ajax()

## Étape 1 : Création d'une requête

### Juste l'objet XHR

```
var xhr = $.ajax();
```

### Une requête complète

```
var xhr = $.ajax('demo.php');
```

Il est possible de passer plusieurs options

```
var xhr = $.ajax('demo.php', { method: 'POST' });
var dataToPost = {id: 10};
var xhr = $.ajax('demo.php', { method: 'POST', data: dataToPost });
```

## Étape 2 : Agir en fonction du retour serveur

```
// En cas de succes
xhr.done(function(data){
  console.log(data); // Les données retournées sont accessibles via
'data'
});

// En cas d'échec
xhr.fail(function(){});

// Dans tous les cas
xhr.always(function(){});
```

# Quelques astuces et outils

## Rendre un formulaire exploitable en AJAX très rapidement avec jQuery

**Sous forme d'un chaîne de caractères** avec .serialize()

```
$('form#contact').serialize();
```

**Sous forme de tableau** avec .serializeArray()

```
$('form#contact').serializeArray();
```

## Bloquer les requêtes autres que via AJAX en PHP

La super globale $_SERVER contient beaucoup d'informations très interessante sur le serveur et les headers d'une requête

Une fonction bien utile s'appuyant sur $_SERVER pour définir si une requête est effectuée en AJAX

```
function isAjax() {
  return isset($_SERVER['HTTP_X_REQUESTED_WITH']) &&
strtolower($_SERVER['HTTP_X_REQUESTED_WITH']) === 'xmlhttprequest';
}
```

## Envoyer un header d'erreur en PHP

```
header($_SERVER['SERVER_PROTOCOL'] . ' 500 Internal Server Error',
true, 500);
```

## Nettoyer une chaîne de caractères en PHP

Une petite fonction intéressante pour nettoyer efficacement une chaîne avec trim, strip_tags et htmlspecialchars

```
function sanitize($str){
  return htmlspecialchars(strip_tags(trim($str)));
}
```

Une autre option intéressante, mais avec quelques subtilités à prendre en compte

```
filter_var($str, FILTER_SANITIZE_STRING);
```

# JS Comprendre XMLHttpRequest

The Hard Way

XMLHttpRequest est un objet sur lequel repose l'architecture AJAX.

## Étape 1 : Création de l'objet XHR

```
var xhr = new XMLHttpRequest();
```

L'objet possède une longue liste de propriétés et méthodes (consultable sur MDN ou Devdocs.io)

À noter que la syntaxe au-dessus est vrai pour la plupart des navigateurs, toutefois pour les navigateurs plus anciens il faudra passer par une syntaxe plus complexe.

```
function createXhrObject() {
  if (window.XMLHttpRequest)
    return new XMLHttpRequest();

  if (window.ActiveXObject) {

    var names = [
      "Msxml2.XMLHTTP.6.0",
      "Msxml2.XMLHTTP.3.0",
      "Msxml2.XMLHTTP",
      "Microsoft.XMLHTTP"
    ];

    for(var index in names) {
      try{
        return new ActiveXObject(names[index]);
      }
      catch(err){}
    }

  }

  window.alert("XMLHttpRequest n'est pas pris en charge.");

  return null; // non supporté
}

var xhr = createXhrObject(); //Objet XMLHttpRequest ou null
```

## Étape 2 : Écouter les changements de statut de l'objetXHR

```
xhr.onreadystatechange = function(evt) {
  console.log(this.readyState);
  if (this.readyState == 4 ) {
    console.log(this.responseText);
  }
};
```

## Étape 3 : Initialiser une requête de l'objet XHR vers le serveur

open('methode', 'destination', asynchrone ou non)

```
xhr.open('GET','demo.php', true);
```

## Étape 4 : Effectuer la requête

```
xhr.send();
```

#  Travailler avec Fetch :

Explication :

Fetch est une maniere plus récente de travailler avec des requêtes asynchrones en JavaScript. C'est une alternative a XMLHttpRequest qui est plus logique et standardisée avec les avancée de JS.

## Comment on s'en sert :

```
fetch(MyURL, myOptions).then((response) => {
  console.log(response)
})
```

*myURL* = C'est l'url/endpoint de la ressource que l'on souhaite récupérer

*myOptions* = C'est dans cet objet qu'on met les options de notre requête, par exemple :  GET ou POST ou la gestion du cache

*then* = contient le retour de notre requête asynchrone, que l'on va pouvoir utiliser dans la fonction qui à comme paramètre ledit retour

*response* = la réponse à notre requête

En pratique, tout appel Ajax / requête HTTP / requête XHR aura le même code de base. Le code utilisant fetch est un peu plus léger et lisible.

**Exemple de requête en méthode GET :**

```
/**
 * Options de la future requête HTTP
 */
let fetchOptions = {
  // --- Toujours défini :

  // La méthode HTTP (GET, POST, etc.)
  method: 'GET',

  // --- Bonus (exemples) :

  // On utilisera souvent Cross Origin Resource Sharing qui apporte
  // une sécurité pour les requêtes HTTP effectuée avec XHR entre 2
  // domaines différents.
  mode: 'cors',
  // Veut-on que la réponse puisse être mise en cache par le
navigateur ?
  // Non durant le développement, oui en production.
  cache: 'no-cache'

  // Si on veut envoyer des données avec la requête => décommenter
et remplacer data par le tableau de données
  // , body : JSON.stringify(data)
};


/**
 * La fonction native fetch() permet de lancer une requête HTTP
depuis JS.
 *
 * Dans DevTools > onglet "Network", on pourra voir cette requête
avec le
 * filtre "XHR".
 *
 * Arguments de fetch() :
 * 1. l'URL à laquelle on veut accéder
 * 2. les options de cette requête HTTP
 */
request = fetch('http://www.domaine.tld/endpoint', fetchOptions);
```

```
// La requête vient d'être envoyée !

// On n'a pas encore la réponse, mais on se prépare déjà à la
recevoir.
// Une fois la réponse reçue, la fonction de callback précisée en
argument
// sera automatiquement appelée.
request.then(
  // Cette fonction de callback est définie directement "à la volée"
=> fonction anonyme.
  // Elle recevra en argument la réponse brute provenant du serveur.
  function(response) {
    // On sait que la réponse est au format JSON (JavaScript Object
Notation),
    // donc on transforme la réponse : conversion texte => objet JS
    return response.json();
  }
)

// On peut enchaîner les fonctions de traitement de la réponse.
.then(
  // Celle-ci étant chaînée à la précédente, elle recevra en
argument la réponse
  // précédemment convertie en objet JS.
  function(jsonResponse) {
    // Désormais, on a accès aux données facilement et on peut
travailler avec :
    console.log(jsonResponse);

    // TODO, utiliser ces données pour modifier la page, afficher
les données, etc.
  }
);
```

On changera essentiellement, selon les besoins :

- la méthode HTTP
- l'URL de la requête HTTP
- le traitement à faire une fois les données reçues au format JSON

**Exemple de requête en méthode POST :**

```
// On stocke les données à transférer
const data = {
  firstname: "Serge",
  lastname: "Karamazov",
  gimmick: "Aucun lien"
};

// On prépare les entêtes HTTP (headers) de la requête
// afin de spécifier que les données sont en JSON
const httpHeaders = new Headers();
httpHeaders.append("Content-Type", "application/json");

// On consomme l'API pour ajouter en DB
const fetchOptions = {
    method: 'POST',
    mode: 'cors',
    cache: 'no-cache',
    // On ajoute les headers dans les options
    headers: httpHeaders,
    // On ajoute les données, encodées en JSON, dans le corps de la
requête
    body: JSON.stringify(data)
};

// Exécuter la requête HTTP avec FETCH
fetch('http://lacitedelapeur.io/api/characters', fetchOptions)
.then(
    function(response) {
        // console.log(response);
        // Si HTTP status code à 201 => OK
        if (response.status == 201) {
            alert('ajout effectué');

            // TODO selon ce qu'on veut faire une fois la réponse
récupérée
        }
        else {
            alert('L\'ajout a échoué');
        }
    }
)
```