Ajax

© Philippe ROOSe
IUT de Bayonne / UPPA

Constat // Applications Riches pour Internet (RIA)

- C'est reproduire ou du moins s'approcher de l'expérience utilisateur des applications Desktop, dans une appli web
- Le programme répond rapidement et intuitivement
- Feedback quasi instantané
  - Une cellule dans un tableur change de couleur quand on passe la souris dessus,
  - Un password est validé à chaque touche tapée,
- Les choses se passent naturellement
  - Pas besoin de cliquer sur un bouton pour déclencher un événement

Appli Web < 2.0

- "Clique, attend que la page se ré-affiche" user interaction
  - Chaque communication avec le serveur implique un nouveau rendu de la page HTML
- Modèle de communication "requête/réponse" synchrone
  - L'utilisateur doit attendre que la réponse revienne, il ne peut rien faire dans la même page en attendant.
- Ce modèle est "Page-driven": le Workflow de l'application est par page
  - La logique de la navigation par page est déterminée par l'application côté serveur.

Problème

- Interruption des opérations de l'utilisateur
  - Il ne peut rien faire tant que la réponse n'est pas revenue et qu'une nouvelle page est affichée,
- Perte de contexte lorsque la nouvelle page revient
  - Plus rien sur l'écran, ré-affichage,
  - Perte de la position dans la page, il faut re-scroller,
- Pas de "petit feedback", bulles, messages, aides interactives
- Contraintes de HTML
  - Pas de widgets riches (calendrier) etc.

Technos possibles pour du RIA

- Applet, Java Web Start, Java FX
- Macromedia Flash/Air
- HTML5 (tags html+css+javascript)
- **Ajax/javascript**
- Silverlight (Windows only)
- GWT (gmail etc)
- Ruby on rails, Play, Grails
- Ajax4.Net

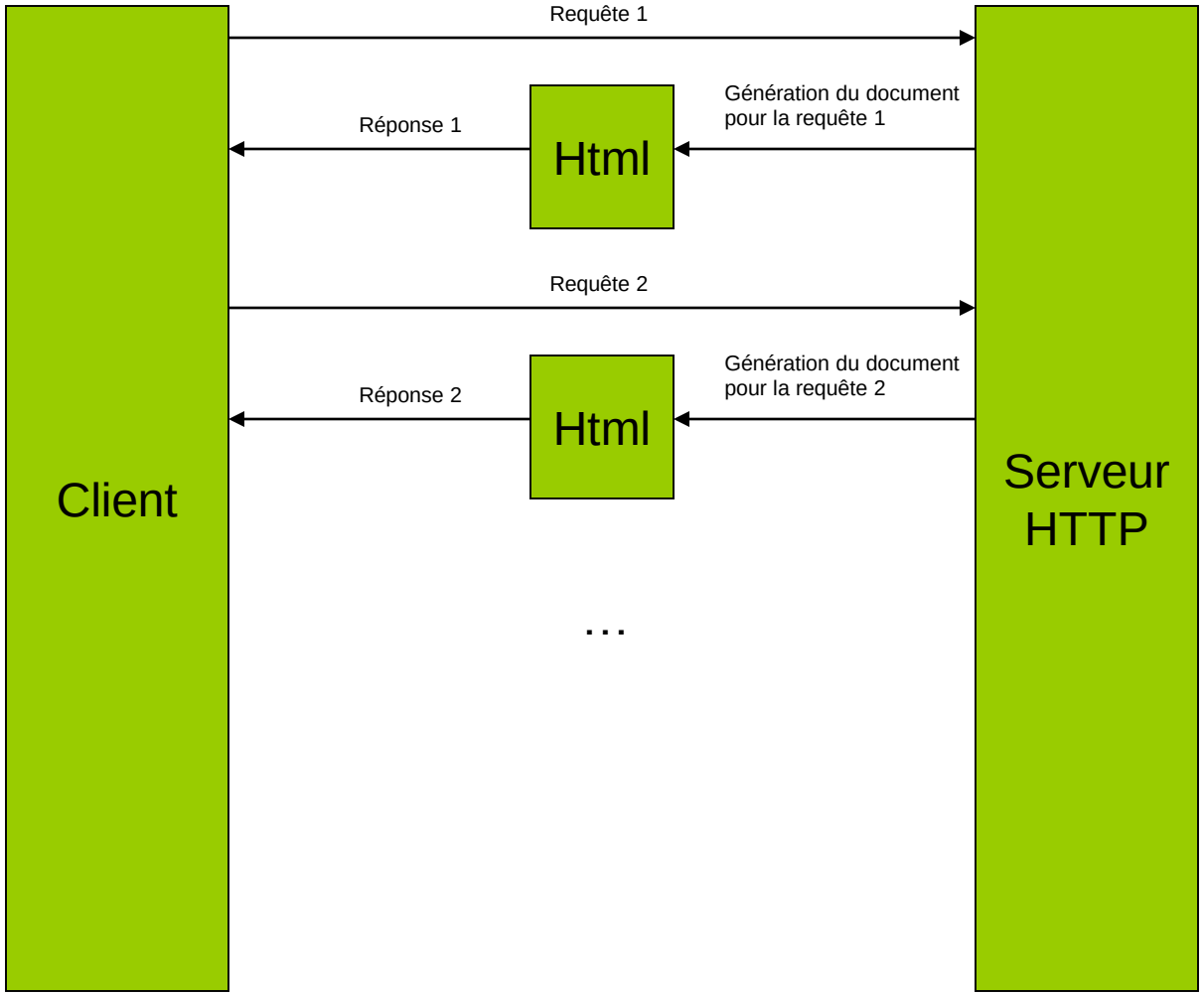
Contexte

- 1/ Terminaux des ordinateurs centraux

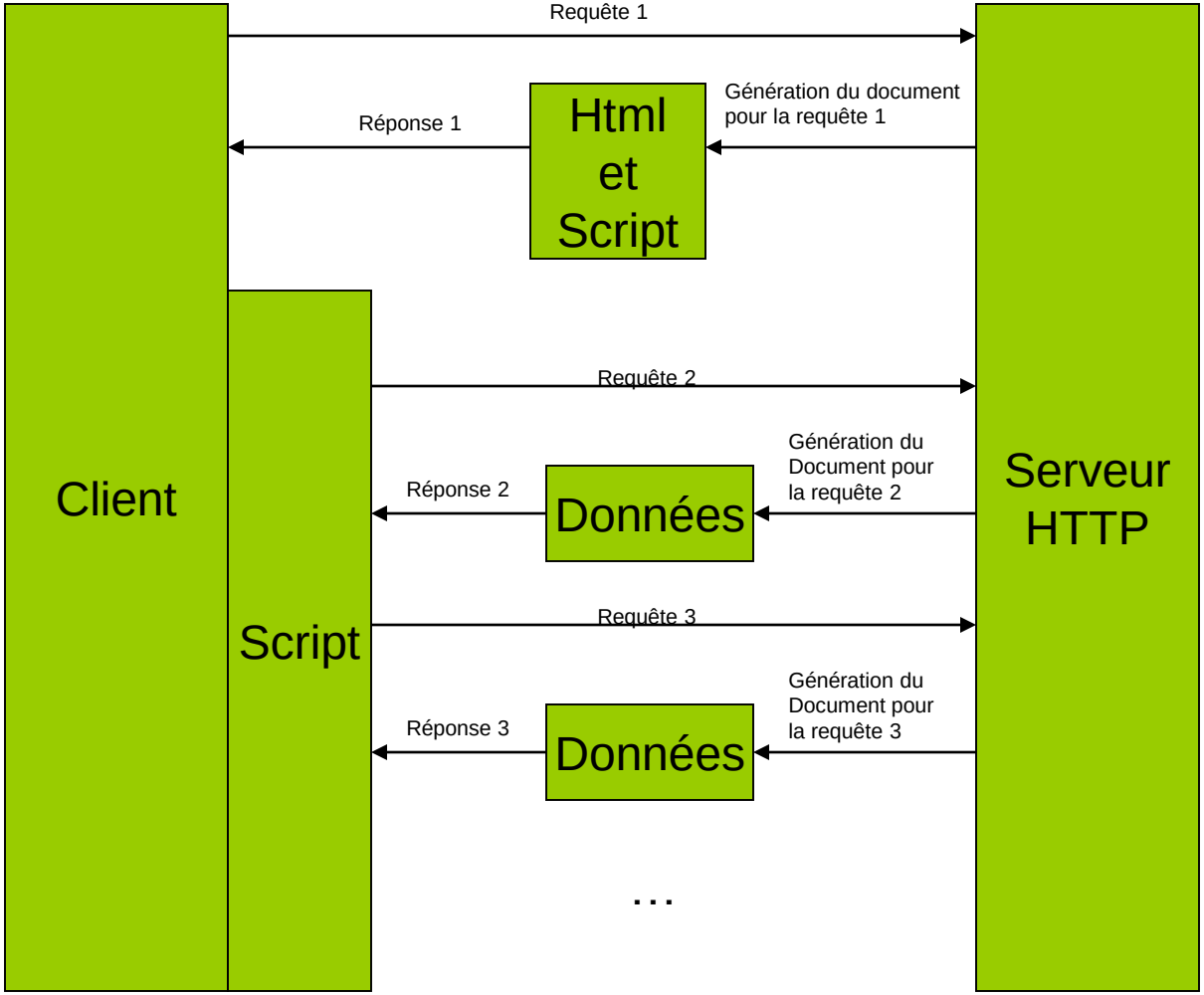
- 2/ Évolution vers les architecture client/serveur (2-tiers)
  - Avènement des IHM graphiques, souris, évènements, etc.
- 3/ Arrivée du Web : Applications accessibles partout, au détriment de l'ergonomie/convivialité
- 4/ Web 2.0 / Ajax => Augmentation de l'interaction
  - ***Asynchronous Javascript*** *and* ***XML***.
  - **Important** : Ajax n'est surtout pas un langage ni une nouvelle technologie, mais plutôt une nouvelle façon d'utiliser celles qui existaient déjà.

- Le terme AJAX a été mentionné pour la première fois par Jesse James Garrett d'Adaptive Path, en février 2005, dans un article intitulé « AJAX : A New Approach To Web Applications ».

Web < 2.0 (avant)


Requête 1
Génération du document pour la requête 1
Réponse 1
Html
Requête 2
Génération du document pour la requête 2
Réponse 2
Html
...
Client
Serveur HTTP

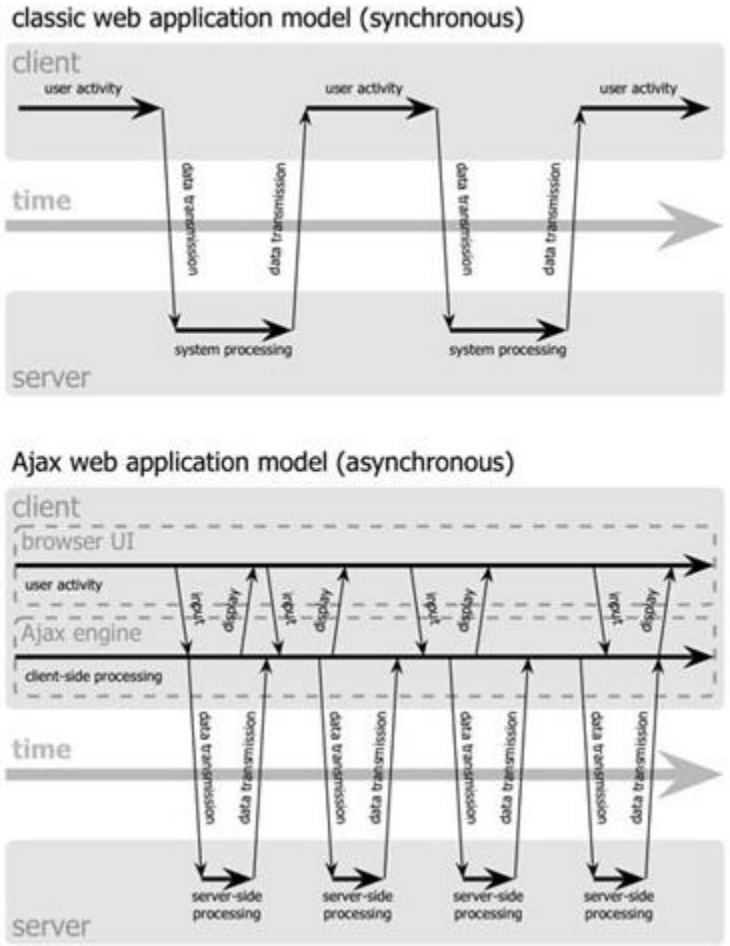
Web ≥ 2.0 (Après)


Requête 1
Réponse 1
Html
et
Script
Génération du document
pour la requête 1
Client
Script
Requête 2
Réponse 2
Données
Génération du
Document pour
la requête 2
Serveur
HTTP
Requête 3
Réponse 3
Données
Génération du
Document pour
la requête 3
...

1 – Qu'est-ce que Ajax ?

- À chaque fois que l'utilisateur interagi avec la page, le navigateur doit envoyer une requête au serveur et attendre sa réponse avant de rafraîchir la page.
  - délai parfois néfaste, pénibles pour l'utilisateur dans le cas des applications de bureau.
  - Impossibilité de gérer certains évènements, tel le mouvement de la souris.

- Objectif :
  - Exemple 1 : accélérer le processus en téléchargeant d'avance les données susceptibles d'être consulté par la suite alors que l'utilisateur celles à l'écran ?



1 – Qu'est-ce que Ajax ?

- Ajax permet de faire une requête au serveur sans recharger la page.
  - Permet de ne rafraîchir qu'une partie de la page.
- Repose sur des technologies éprouvées
  - Le langage Javascript,
  - L'objet XMLHttpRequest, format XML.
- Les techniques Ajax sont, en soi, indépendantes de la plateforme utilisée.

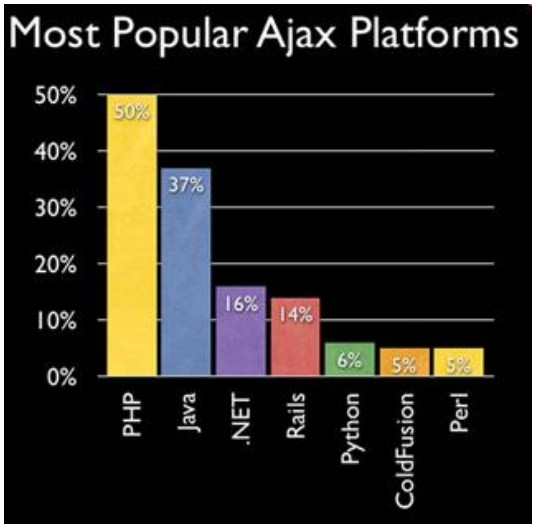
2) Comment fonctionne Ajax ?

- Le serveur HTTP envoie au client une page Web incluant un script.
- Le script utilise un objet XMLHttpRequest pour communiquer avec le serveur sans télécharger de nouveau la page.

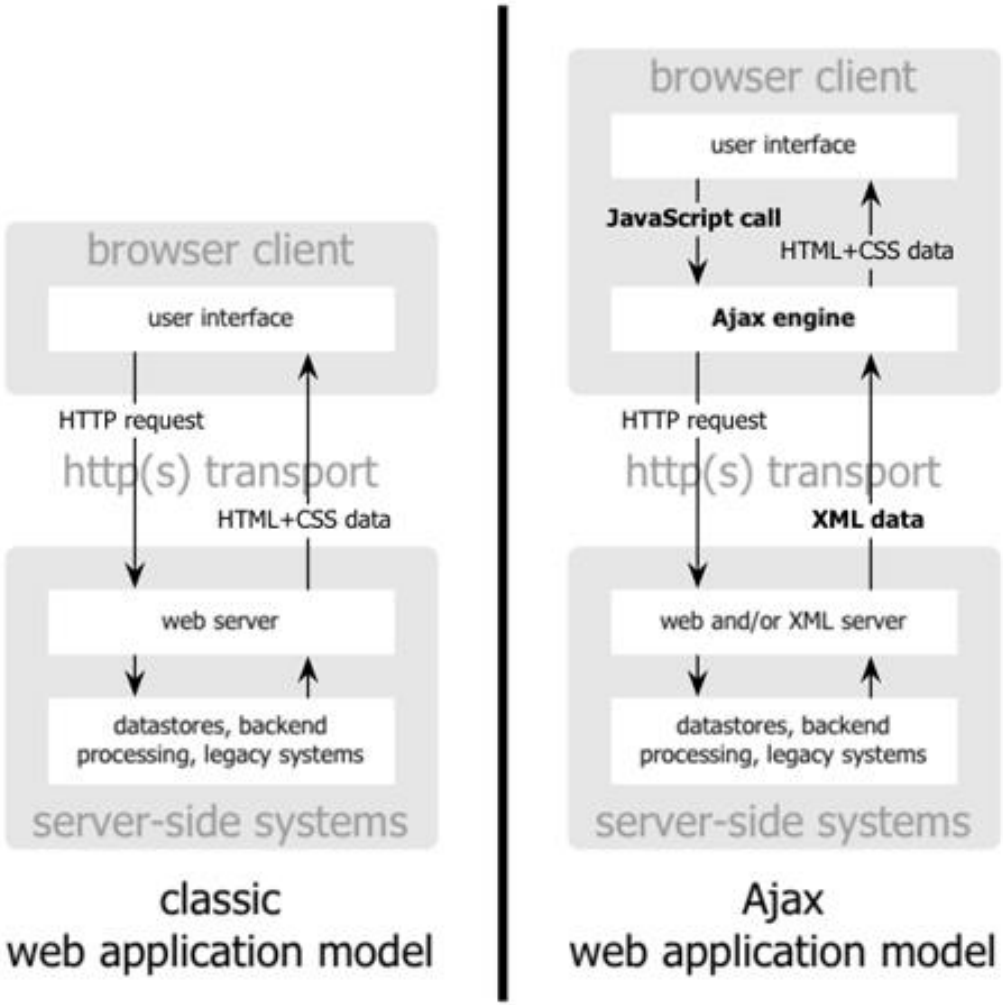
2) Comment fonctionne Ajax ?

- **Objet XMLHttpRequest**
  - Provient de Microsoft. Standard *de facto*: Implémenté par la plupart des principaux navigateurs Web.
  - Interface de programmation (API) semblable entre les navigateurs, mais ceux-ci ont chacun leurs particularités.

- Solution possible : utiliser des frameworks permettant de simplifier le paramétrage des requêtes, la spécification de leurs arguments et l'interprétation de la réponse.



Architecture de dialogue



Echange de données

- Le modèle de programmation serveur ne change pas beaucoup des applis classiques
  - Il reçoit des GETs/POSTs HTTP classiques,
- Le “content type” des réponses peut être :
  - text/xml
  - text/plain
  - text/json
  - text/javascript

Ajax en 7 étapes

1. Un événement client est émis,
2. Un objet XMLHttpRequest est créé,
3. L'objet XMLHttpRequest est configué,
4. L'objet XMLHttpRequest déclenche une requête asynchrone
5. La servlet ValidateServlet renvoie un document XML contenant le résultat,
6. L'objet XMLHttpRequest appelle la fonction callback() function et traite le résultat,
7. Le DOM HTML de la page est mis à jour.

Applications possibles

- On peut scroller la carte à la souris
  - Au lieu de cliquer sur un bouton ou quoi que ce soit…
  - Ceci déclenche une action sur le serveur.
- En coulisse : AJAX est utilisé
  - Des requêtes sont envoyées en tâche de fond pour demander de nouvelles données,
  - Les données arrivent de manière asynchrone et seule une partie de la page est rafraichie.
- Les autres parties de la page ne bougent pas
  - Pas de perte du contexte opérationnel.

Usages classiques

- Vérification en temps réel des données d'un formulaire, par appel du serveur
  - Identificateurs, numéros de séries, codes postaux…
  - Plus besoin de logique de validation compliquée impliquant de la navigation entre pages
- Auto-complétion
  - Emails, villes, etc… peuvent être autocomplétées ou suggérées en temps réel au fur et à mesure de la saise
- Maitrise des opérations dans le détail et de la GUI
  - Des actions de l'utilisateur peuvent appeler des informations plus détaillées, requêtées sur le serveur et affichées quasi instantanément.

Etape 1: L'événement javascript est émis

- Une fonction javascript est appelée lors de l'émission de l'événement,
- Exemple: la fonction validateUserId() est un "écouteur" de l'événement onkeyup sur le champ input don't l'attribut id vaut "userid".

```
<input type="text"
        size="20"
          id="userid"
        name="id"
      onkeyup="validateUserId();">
```

Etape 2: Un objet XMLHttpRequest est créé

```
var req;
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}
function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest;
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

Etape 3: L'objet XMLHttpRequest est configuré par une function de callback

```
var req;
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest; // callback
    function
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

Etape 4: L'objet XMLHttpRequest envoie une requête asynchrone

```
function initRequest() {
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function validateUserId() {
    initRequest();
    req.onreadystatechange = processRequest;
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

- L'URL relatif de la servlet appelée vaut ici : validate?id=greg

Etape 5: La servlet ValidateServlet renvoie un document XML de réponse

```
public  void doGet(HttpServletRequest request,
  HttpServletResponse  response)
      throws IOException, ServletException {

  String targetId = request.getParameter("id");

  if ((targetId != null) && !accounts.containsKey(targetId.trim())) {
        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("<valid>true</valid>");
    } else {
        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("<valid>false</valid>");
    }
  }
```

Etape 6: L'objet XMLHttpRequest appelle une fonction de callback pour traiter la réponse

- L'objet XMLHttpRequest a été configuré pour appeler la fonction processRequest() lorsque la valeur de son attribut readyState change de valeur :

```
function processRequest() {
    if (req.readyState == 4) {
        if (req.status == 200) {
            var message = ...;

...
```

Etape 7: Le DOM HTML est mis à jour

- Voici comment on procède :
  - document.getElementById("userIdMessage"), où "userIdMessage" est l'attribut ID d'un élément du document HTML

Methodes de XMLHttpRequest

- open(“HTTP method”, “URL”, syn/asyn)
  - GET ou POST ?, URL à appeler, mode
- send(content) : Envoi de la abort() : abandonne la requête en cours
- getAllResponseHeaders()
  - Renvoie les headers (labels + valeurs) sous la forme d’une string
- getResponseHeader(“header”)
  - Renvoie la valeur d’un attribut du header de la réponse
- setRequestHeader(“label”,”value”)
  - Initialise un attribut du header de requête

Propriétés de XMLHttpRequest

- onreadystatechange
  - Prend comme valeur un écouteur du changement de l'état de la requête
- readyState – Etat courant de la requête
  - 0 = uninitialized, 1 = loading, 2 = loaded, 3 = interactive (quelques données ont été retournées), 4 = complete
- status
  - Statut HTTP retourné : 200 = OK
- responseText
  - Versions String de la réponse,
- responseXML
  - Version XML de la réponse,
- statusText
  - Texte du statut retourné par le serveur.

innerHTML

- Utiliser innerHTML est plus facile : on peut modifier ou récupérer des sous-arbres HTML dans le DOM directement
- userMessageElement.innerHTML = messageText;

Conclusion

- **AJAX** n'est **pas une technologie**
- **AJAX** est **indépendant de tout environnement**
- **AJAX** peut être implémenté sur **toute plateforme Web** (*PHP, JSP, .NET, etc.*)

- Attention : Veiller à ce que ça ne surcharge pas trop les échanges avec le serveur => gain anéanti !