AJAX

M1 MIAGE - Applications Web

Adil El Ghali

Plan

- Introduction
- Qu'est ce qu'AJAX, pourquoi faire?
- Principes de fonctionnement d'AJAX
- Les *frameworks* AJAX

Introduction

- Utilisateurs sont habitués à des applications ayant des interfaces *riches*
    - Réponses rapides au niveau de l'interface
    - Pas besoin de cliquer sur des boutons, pour avoir des changements au niveau de l'interface
- Avec les applications web conventionnelles c'est plutôt:

  cliquer, attendre la réponse du serveur, rafraîchir

Introduction

- L'utilisateur ne peut rien faire en attendant la réponse du serveur
- Il perd des informations contextuelles en rafraîchissant une page
- N'a pas de retour instantané à ces actions

Il faut des applications web ayant une interface riche

Introduction

- Beaucoup de technologies pour faire des RIA
  - Applets, Flash, Java Web Start,
  - DHTML (+iFrame)
  - AJAX

DHTML

- DHTML = javascript + DOM + CSS
  - Pas de communication asynchrone
  - besoin de rafraîchir les pages :-(
- iFrame :
  - un élément du DOM: déplaçable, modifiable, sans rafraîchir la page
  - comportement asynchrone

DHTML

- DHTML = javascript + DOM + CSS
  - Pas de communication asynchrone
  - besoin de rafraîchir les pages :-(
- iFrame :
  - un élément du DOM: déplaçable, modifiable, sans rafraîchir la page
  - comportement asynchrone

Hack !!

AJAX

- DHTML + communication asynchrone via XMLHttpRequest

+ technologie la plus aboutie pour les RIA
+ beaucoup de toolkits et de frameworks
+ pas besoin de plug-ins
- incompatibilité des navigateurs
- Javascript est difficile à maintenir

Exemples d'applications

- Google maps
  - http://maps.google.com/
- Google suggest
  - http://www.google.com/webhp?complete=1&hl=en
- Gmail
  - https://mail.google.com/mail/

Google maps

- L'utilisateur peut bouger dans la carte en utilisant la souris
- La carte est téléchargée du serveur par des appels AJAX de manière asynchrone
- Le reste de la page ne change pas

Cas d'usage d'AJAX

- Validation de formulaire par le serveur,
- Auto-complétion
- Enrichissement de la page avec des détails personnalisés (sans tout inclure dans la page)

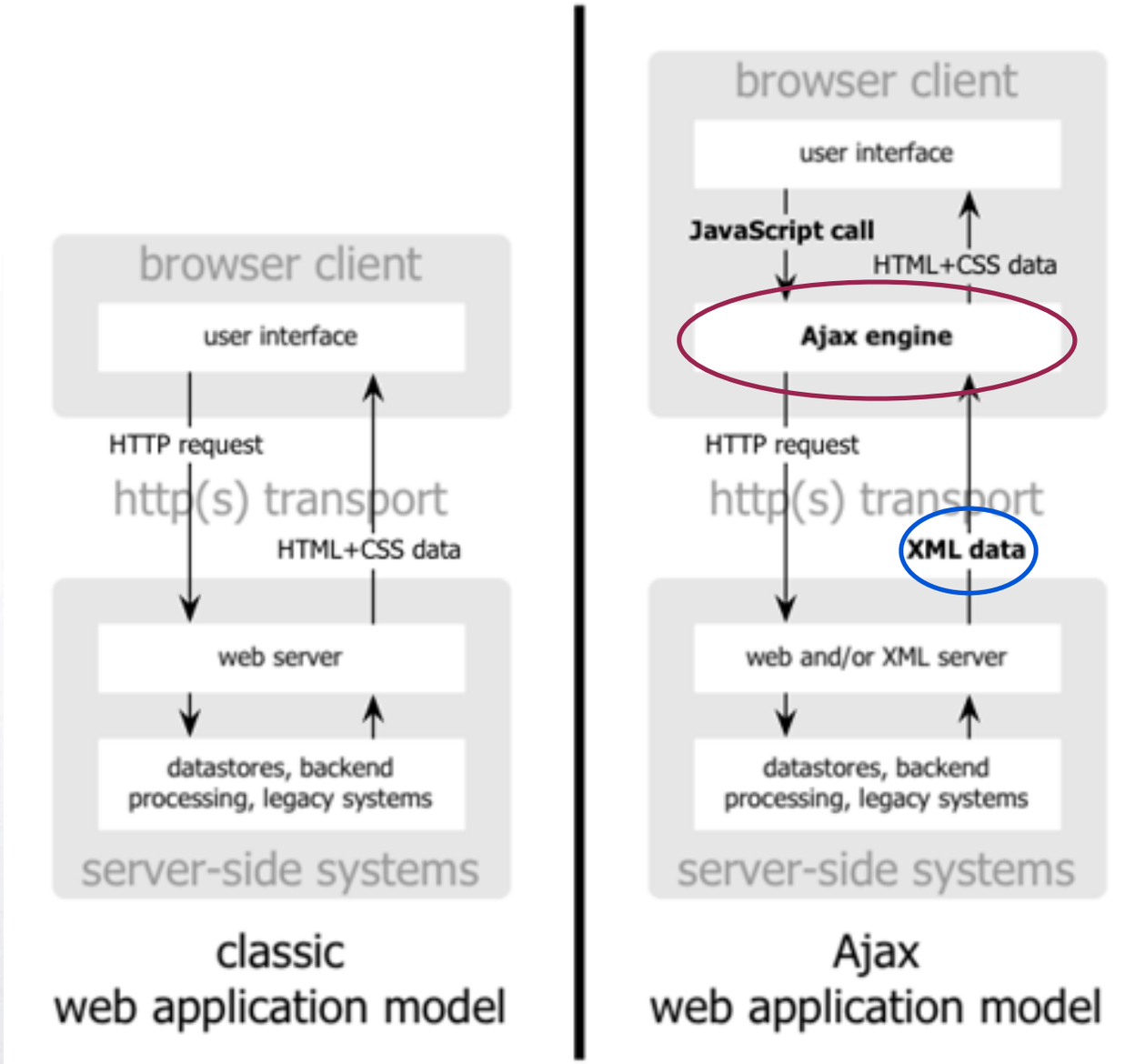
Cas d'usage d'AJAX

- Les *widgets* complexes dans les pages (arbres, menus dynamiques, barre de progression)
- Rafraîchissement automatiques des pages
- Simuler les notifications du serveur

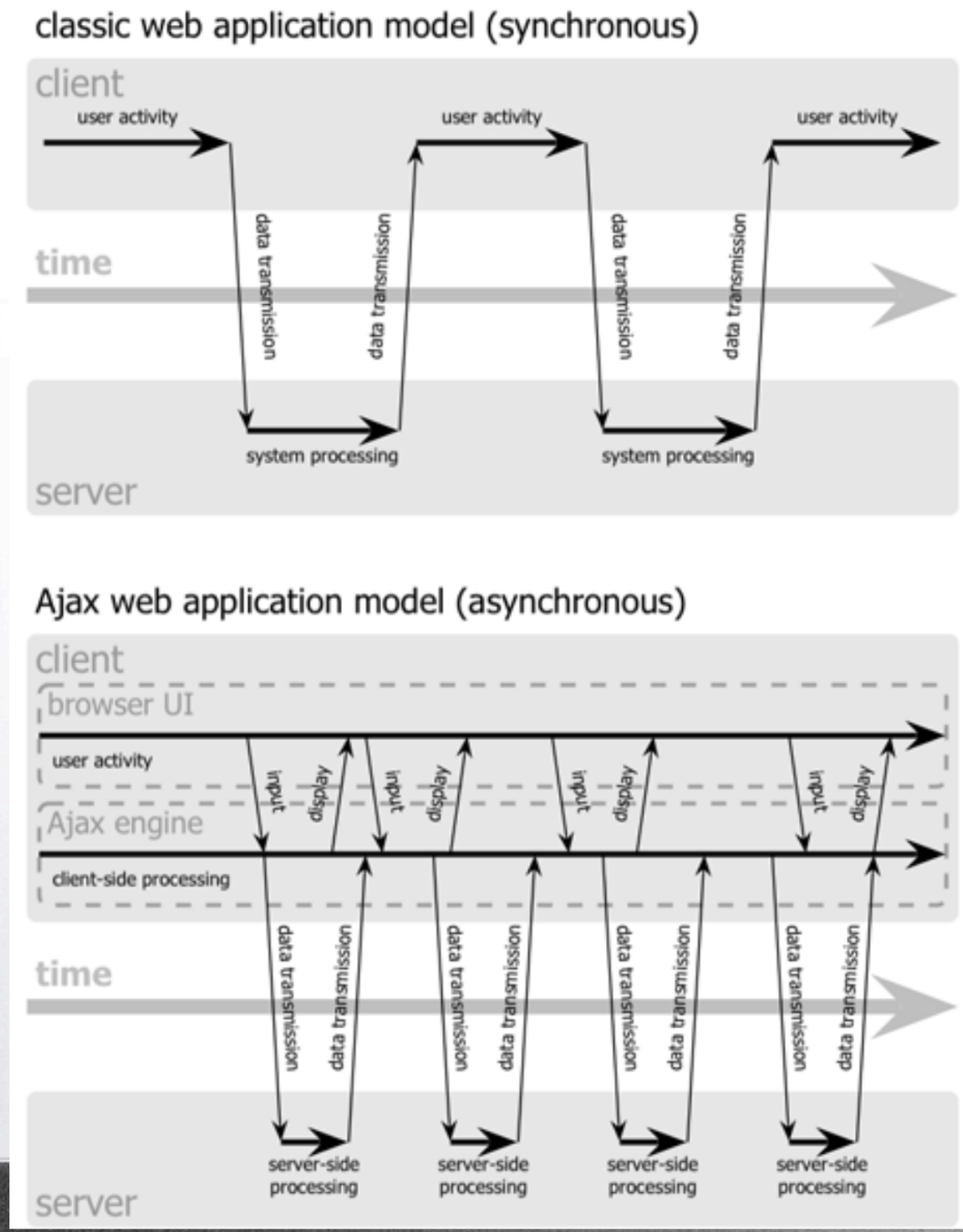
Pourquoi utiliser AJAX?

- Interactions naturelles et intuitives
- Mise à jour partielle des pages
- Interface pilotée par les données (vs structure de la page)
- Communication asynchrones remplace le modèle (requête / réponse)

modèle classique vs AJAX



modèle classique vs AJAX



Les actions de l'utilisateur sont interrompus pendant la récupérations des données

L'utilisateur peut continuer à travailler pendant que l'application va chercher des données sur le serveur

Les technologies

- AJAX est l'acronyme pour : "*Asynchronous JavaScript and XML*", basé sur:
  - Javascript
  - DOM (API d'accès à la structure des documents XML)
  - CSS
  - XMLHttpRequest: Objet JS qui s'occupe de dialoguer avec le serveur de manière asynchrone.

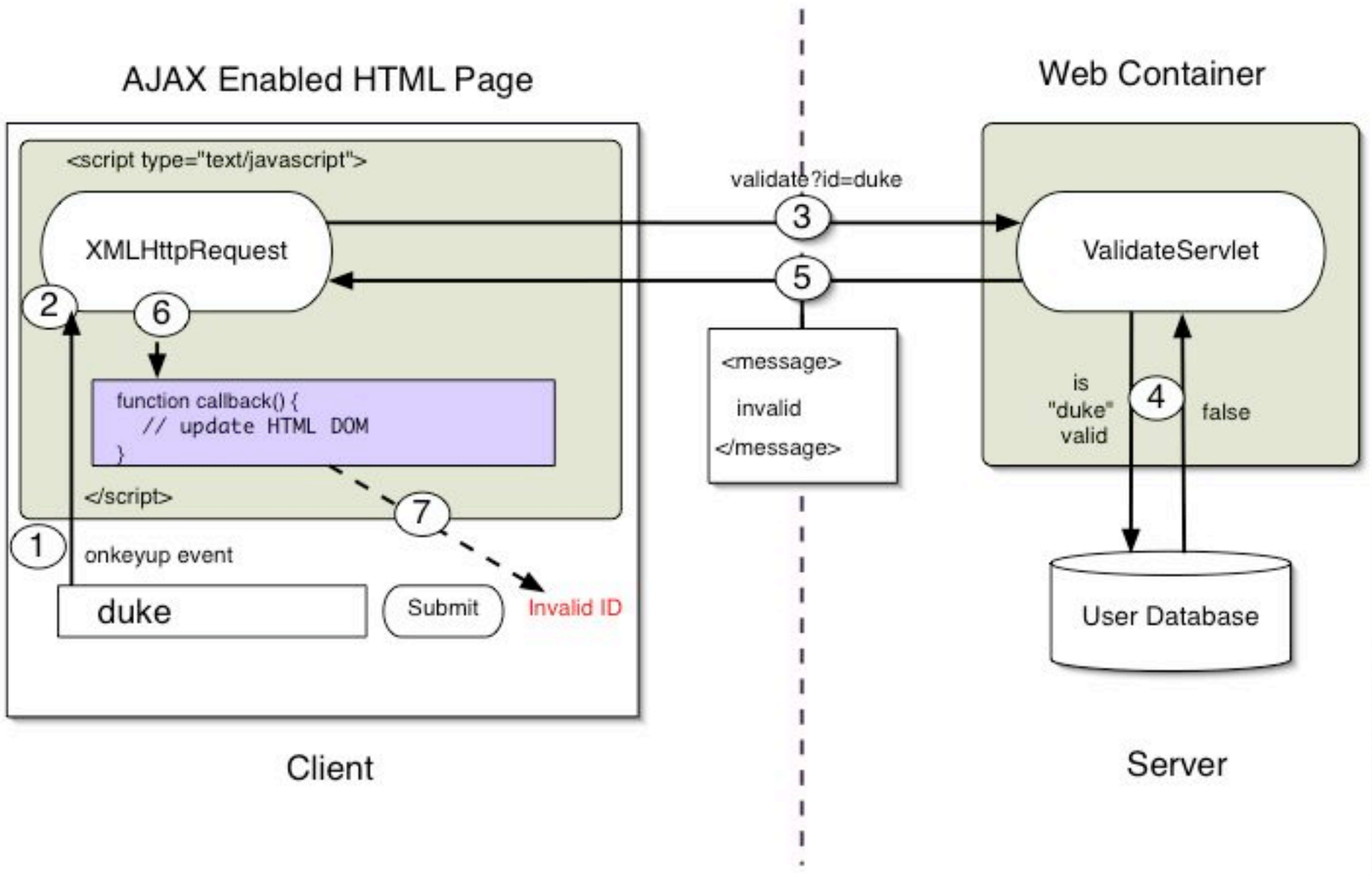
XMLHttpRequest

- Objet Javascript
- Implanté par les navigateurs:
  - Mozilla, Safari, Opera, IE
- Communication avec le serveur via HTTP GET/POST
- Fonctionne en arrière plan

Coté serveur

- Rien ne change!!
  - répondre à des requêtes GET/POST
  - Servlet, JSP, ...
- Quelques “petites” contraintes
  - Des requêtes plus fréquentes
  - Réponses typées: text/xml, text/plain, text/json, text/javascript

Fonctionnement



Les opérations

1. Evénement client
2. Création d'un objet XMLHttpRequest
3. Configuration de l'objet XMLHttpRequest
4. Requête asynchrone par l'objet
5. Le serveur renvoie un résultat XML
6. L'objet XMLHttpRequest appelle la fonction callback() et traite les résultats
7. Le DOM de la page est mis à jour

Opérations (1)

- Une fonction JS est appelé à la suite d'un événement client
  - ValidateUserid() est associé au remplissage du champ userid

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

Opérations (2)

```
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);
}
```

Création de l'objet XMLHttpRequest

Opérations (3)

```
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; // fonction de retour
    if (!target) target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);
    req.open("GET", url, true);
    req.send(null);
}
```

Configuration de l'objet avec une fonction de retour

Opérations (4)

```
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);
}
```

Exécution de l'appel asynchrone, à l'URL validate?id=*userid*

Opérations (5)

```
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>");
    }
}
```

Le serveur renvoie des données XML en réponse à la requête

Opérations (6)

- L'objet XMLHttpRequest a été configuré pour appeler la fonction de retour processRequest() quand son état devient readyState:

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


...
```

Opérations (7)

- Il ne reste plus qu'à modifier la page avec les informations reçues, en utilisant l'API DOM.

- Par exemple, on peut modifier l'élément userIdMessage, pour y accéder on utilise l'appel:

```
document.getElementById("userIdMessage")
```

```
<script type="text/javascript">
function setMessageUsingDOM(message) {
    var userMessageElement =
    document.getElementById("userIdMessage");
    var messageText;
    if (message == "false") {
        userMessageElement.style.color = "red";
        messageText = "Invalid User Id";
    } else {
        userMessageElement.style.color = "green";
        messageText = "Valid User Id";
    }
    var messageBody = document.createTextNode(messageText);
    // if the messageBody element has been created simple replace it
    otherwise
    // append the new element
    if (userMessageElement.childNodes[0]) {
        userMessageElement.replaceChild(messageBody,

    userMessageElement.childNodes[0]);
    } else {
        userMessageElement.appendChild(messageBody);
    }
}
</script>
<body>
    <div id="userIdMessage"></div>
</body>
```

XMLHttpRequest: méthodes

- open(“HTTP method”, “URL”, syn/asyn)
  - défini la méthode HTTP, l’URL destination et le mode
- send(content)
  - envoie la requête avec un contenu (chaîne de caractères, objet DOM)
- abort()
  - termine la requête
- getAllResponseHeaders()
  - renvoie les headers comme une chaîne de caractères
- getResponseHeader(“header”)
  - renvoie la valeur d’un header
- setRequestHeader(“label”,”value”)
  - modifie la valeur d’un header avant envoie au serveur

XMLHttpRequest: propriétes

- onreadystatechange
  - associé à un événément JS qui est levé à chaque changement d'état
- readyState – le statut de la requête
  - 0 = non initialisé
  - 1 = en cours de chargement
  - 2 = chargé
  - 3 = mode interactive (certaines données ont été retournées)
  - 4 = completé
- status
  - le statut HTTP renvoyé par le serveur: 200 = OK

XMLHttpRequest: propriétes

- responseText
  - Les données envoyées par le serveur comme *string*
- responseXML
  - Les données renvoyées par le serveur comme *XML*
- statusText
  - Le statut renvoyé par le serveur comme *string*

Sécurité AJAX

- Le code JS est visible, donc potentiellement utilisable pour profiter des trous de sécurité du serveur (XSS)
- Le code est téléchargé puis exécuté sur le client, c'est potentiellement dangereux pour le client
- Le code est contraint par le modèle Sandbox (accès limité pour le code)

Outils de développement

- L'extension FireBug pour FireFox
  - Espionner le trafic XMLHttpRequest
  - Débugger le JavaScript pas à pas et inspecter le source HTML (style, événements, layout et DOM)
  - Barre d'état pour montrer les messages d'erreurs
  - Une console pour les erreurs JS et CSS
  - Possibilité de logguer les messages d'erreurs

Mais ...

- Quelques problèmes:
  - Pas de standardisation
  - Pas de support dans les vieux navigateurs
  - Fortement dépendant de JS
  - Tous le code est visible

Mais ...

- Quelques problèmes:
  - Pas de standardisation
  - Pas de support dans les vieux navigateurs
  - Fortement dépendant de JS
    (donc des incompatibilités)
  - Tous le code est visible

Mais ...

- Quelques problèmes:
  - Pas de standardisation
  - Pas de support dans les vieux navigateurs
  - Fortement dépendant de JS
    (donc des incompatibilités)
  - Tous le code est visible
    (mais ce n'est pas un vrai problème, si vous en écrivez du bon)


Mais ...

- Ecrire des applications est complexe
  - Il faut une bonne coordination entre la partie serveur et la partie AJAX
- Ca peut être difficile à débugger
- Il faut utiliser des *frameworks*

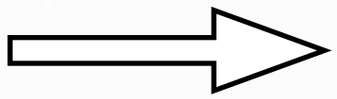
... de plus en plus

- Des libraires JSF comportant de l'AJAX
- Standardisation de XMLHttpRequest
- Meilleur support dans les navigateurs
- de plus en plus de documentations et d'outils sur le web
- des *frameworks* de plus en plus puissants:

Les *frameworks*

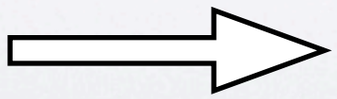
- "*Lourds*"

- DOJO
- GWT



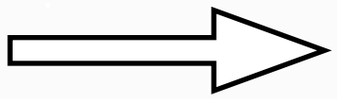
- "Légers"

- jQuery
- ExtJS

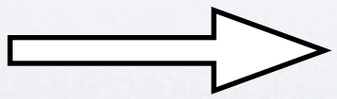


Les *frameworks*

- "*Lourds*"
  - DOJO
  - GWT



- "Légers"
  - jQuery
  - ExtJS



Prochains cours

Les *frameworks*

- "*Lourds*"
  - DOJO
  - GWT

⟹ Cours Web 2.0

- "Légers"
  - jQuery
  - ExtJS

⟹ Prochains cours

Exercice

- Ecrire un programme AJAX qui vérifie qu'un champs *user* contient le nom d'un utilisateur valide
- Ecrire un programme AJAX d'auto-complétion sur les noms de documents présents sur le serveur pour un champs de recherche.