M1IF03
Conception d'applications Web

CONCEPTION ET DÉVELOPPEMENT D'APPLICATIONS WEB CÔTÉ CLIENT

LIONEL MÉDINI
NOVEMBRE 2020

Plan du cours

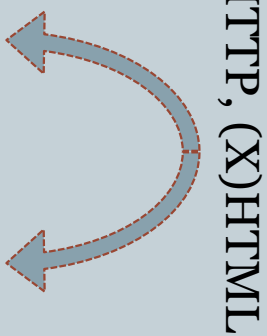
- Asynchronous Javascript And XML (AJAX)
  - Mécanismes de requêtes asynchrones
  - Composants d'une application
  - Quelques patterns de conception en AJAX
  - Considérations de sécurité
- Fetch API
- Outils de conception et de développement
  - jQuery
  - jQuery UI
- Single-Page Application
- Conclusion

Introduction

- Objectif : concevoir des applications Web « riches »
  - Web-based
    - Paradigme client-serveur, HTTP
      - ➔ Programmation côté serveur et côté client
  - Expérience utilisateur proche des applications natives
    - Interface utilisateur fluide, ergonomique, dynamique
      - ➔ Traitement de l'interface côté client (JavaScript, CSS , DOM)
      - ➔ Échanges client-serveur asynchrones (AJAX)
  - Logique métier complexe
    - Outils « évolués » de modélisation, conception, développement
      - ➔ IDE, POO, UML, design patterns, méthodes agiles, XP…
  - ➔ Où placer la logique métier ? La couche données ?

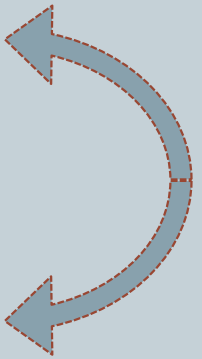
Asynchronous Javascript And XML (AJAX)

- Composants d'une application Web « classique »
  - Côté serveur
    - Contrôleur général de l'application (index.jsp)
    - Ressources statiques
      - Modèle de document, bibliothèques de scripts, feuilles de style
    - Traitements dynamiques des données (couche métier)
    - Composition dynamique de l'interface (couche vue)
  - Côté client
    - Gestion des événements utilisateur
    - Composition dynamique de l'interface (couche vue)



Asynchronous Javascript And XML (AJAX)

- Composants d'une application Web AJAX
  - Côté serveur
    - Contrôleur général de l'application (index.php)
    - Ressources statiques
      - Modèle de document, bibliothèques de scripts, feuilles de style
    - Traitements dynamiques des données (couche métier)
  - Côté client
    - Contrôleurs délégués relatifs à un type de vue
    - Gestion des événements utilisateur
    - Traitement des données reçues (couche métier)
    - Composition dynamique de l'interface (couche vue)



Asynchronous Javascript And XML (AJAX)

- Généralités sur AJAX
  - Applications web avec interface utilisateur
  - Déporter un maximum de code sur le client
    - Réduction des ressources consommées côté serveur
    - Réduction de la bande passante réseau
  - Applications Web AJAX les plus connues
    - Google (Mail, Map, Earth…)
    - Suggestions automatiques
    - Traitement de texte
    - …
  - Exemple
    - http://www.standards-schmandards.com/exhibits/ajax/

Asynchronous Javascript And XML (AJAX)

- Fonctionnement
  - Requête asynchrone au serveur dans une fonction JavaScript (déclenchée par un événement quelconque)
  - Transfert asynchrone de données en XML
  - Traitement dynamique côté client
    - Affichage (inclusion au document HTML, transformation XSLT…)
    - Logique applicative (fonctions JavaScript dédiées)
- Spécificité de la technologie AJAX
  - Requête asynchrone sur un document XML *via* un
    - Objet XMLHttpRequest (Mozilla)
    - Contrôle ActiveX XMLHTTP (IE)

Asynchronous Javascript And XML (AJAX)

- Fonctionnement
  - Étapes d'une communication AJAX côté client
    - Envoi de la requête
      - Créer un objet requête
      - Spécifier les éléments de la requête
        - URL, méthode , headers HTTP, paramètres
      - Lui associer un gestionnaire d'événement
      - L'envoyer
    - Réception de la réponse
      - À chaque changement d'état de la requête, tester si l'état est « ready »
      - Traiter les données reçues
        - Ajout à l'interface, transformation XSL…

Asynchronous Javascript And XML (AJAX)

- Fonctionnement
  - Étapes d'une communication AJAX côté serveur
    - Que doit faire un serveur Web à la réception d'une requête asynchrone AJAX ?

Asynchronous Javascript And XML (AJAX)

- Exemple de code : création d'un objet requête

```
var req = null;

function getRequest()
{
   if (window.XMLHttpRequest)
   {
      req = new XMLHttpRequest();
   }
   else if (typeof ActiveXObject != "undefined")
   {
      req=new ActiveXObject("Microsoft.XMLHTTP");
   }
   return req;
}
```

Safari / Mozilla

Internet Explorer

Asynchronous Javascript And XML (AJAX)

- Exemple de code : chargement asynchrone

```
function GetDataUsingAJAX (HttpMethod, url, params, elt)
{
   if(req != null)
   {
      // méthode avec paramètres
      req.onreadystatechange = function() {stateChange(elt)};
      // méthode sans paramètre
      // req.onreadystatechange = stateChange;

      req.open(HttpMethod, url, true);
      req.setRequestHeader("Accept", "application/xml");
      req.send(params);
    }
}
```

Association d'une fonction de callback aux changements d'état de la réponse

Asynchronous Javascript And XML (AJAX)

- Exemple de code : gestion de l'état

```
function stateChange (elt)
{
   if(req.readyState == 4) {
      if (req.responseXML != null) {
         var docXML= req.responseXML;
      } else {
         var docXML= req.responseText;
         docXML=parseFromString(docXML);
      }
      var docXMLresult = traiteXML(docXML);
      var str = (new XMLSerializer()).serializeToString(docXMLresult);
      document.getElementById(elt).innerHTML += str;
   }
}
```

READY_STATE_COMPLETE

Asynchronous Javascript And XML (AJAX)

- Exemple de code : transformation XSLT

```
//Après chargement asynchrone des documents XML et XSLT
function transform XSLT (XMLDoc, XSLDoc, id)
{
   if(XMLDoc == null || XSLDoc == null) {return;}
   try {
      if (window.ActiveXObject)
      {
         var target = document.getElementById(id);
         target.innerHTML = xml.transformNode(xsl);
      }
```

Internet Explorer

Asynchronous Javascript And XML (AJAX)

- Exemple de code : transformation XSLT

```
    } else if (window.XSLTProcessor) {
        var fragment;
        var xsltProcessor = new XSLTProcessor();
        xsltProcessor.importStylesheet(xsl);
        fragment = xsltProcessor.transformToFragment(xml, document);
        var target = document.getElementById(id);

        target.appendChild(fragment);
    }
  } catch (e) {
    return e;
  }
}
```

Safari / Mozilla

Asynchronous Javascript And XML (AJAX)

- Implémentation de la logique applicative
  - Programmation d'un ensemble de fonctions JavaScript
    - Réécriture de fonctionnalités existantes
    - Mélange de la logique métier et des fonctionnalités techniques
    - Pas forcément à l'épreuve des changements technologiques
    - Réutilisabilité moyenne
    - Code parfois un peu « fouillis »
  - ➔Utiliser / s'approprier des outils existants
    - Langages / IDE spécifiques (ou plugins de votre IDE préféré)
    - Lirairies / frameworks open source

Asynchronous Javascript And XML (AJAX)

- Implémentation de la logique applicative
  - Standardisation de la communication avec les langages de programmation côté serveur : JSON
    - Spécification liée à ECMAScript – RFC 4627
    - Implémentée par tous les navigateurs
    - Permet de sérialiser des types de données (alternative à XML)
    - Définit des types de données de façon simple
    - Indépendant du langage de programmation utilisé

      ➔ Permet les échanges de données entre serveur et client
    - Syntaxe : des inclusions
      - d'objets sous forme d'une liste de membres
        { nommembre1 : valmembre1, nommembre2: valmembre2, … }
      - de tableaux sous forme d'une liste de valeurs
        [valeur1, valeur2, valeur3, …]

Asynchronous Javascript And XML (AJAX)

- Implémentation de la logic
  - Standardisation de la commun
    programmation côté serveur :
    - Exemple de fichier
      au format JSON :

```
{ "menu": "Fichier", "commandes":
[ { "title": "Nouveau",
"action":"CreateDoc" }, {
"title": "Ouvrir", "action":
"OpenDoc" }, { "title": "Fermer",
"action": "CloseDoc" } ] }
```

    - Equivalence en XML :

```
<?xml version="1.0" ?>
<root>
  <menu>Fichier</menu>
  <commands>
     <item>
         <title>Nouveau</value>
         <action>CreateDoc</action>
     </item>
     <item>
         <title>Ouvrir</value>
         <action>OpenDoc</action>
     </item>
     <item>
         <title>Fermer</value>
         <action>CloseDoc</action>
     </item>
  </commands>
</root>²
```

  - Source :
    http://www.xul.fr/ajax-format-json.html

Asynchronous Javascript And XML (AJAX)

- Implémentation de la logique applicative
  - Standardisation de la communication avec les langages de programmation côté serveur : JSON
    - Utilisation côté client :

```
req.open("GET", "fichier.json", true); // requête
…
 var doc = JSON.parse(req.responseText); // récupération
…
var nomMenu = document.getElementById('jsmenu'); // recherche
nomMenu.value = doc.menu.value;            // assignation
…
doc.commands[0].title // lire la valeur "title" dans le tableau
doc.commands[0].action // lire la valeur "action" dans le tableau
```

    - Utilisation côté serveur : librairies *ad hoc*

Asynchronous Javascript And XML (AJAX)

- Quelques règles de conception en AJAX
  - Utiliser des design patterns
    - Adaptateur
      - Le plus utilisé
      - Testez la fonctionnalité à utiliser, pas le navigateur…
    - MVC
      - De préférence type 2 (avec contrôleurs délégués)
      - Isoler les parties du modèle
      - Répartir les traitements de chaque partie entre serveur et client
      - Indiquer à la vue comment restituer les objets du modèle
    - Observateur
      - Permet de définir un modèle événementiel
      - Si celui de JavaScript est insuffisant
      - Il en existe plusieurs dans des librairies open source (W3C)
    - …

Asynchronous Javascript And XML (AJAX)

- AJAX a aussi ses inconvénients
  - Toute une application dans la même page
    - Bouton « Back » inutilisable
    - Définition de bookmarks sur une vue particulière impossible
  - Génération dynamique des contenus
    - Indexation par des moteurs de recherche impossible
  - Téléchargement du code applicatif sur le client
    - Temps de latence importants au lancement de l'application
  - Nécessite d'avoir activé JavaScript
    - Prévoir une solution de repli « acceptable » lorsqu'il est désactivé
  - Complexité des développements
    - Appropriation et utilisation des différentes technos parfois coûteuse

Source : http://dico.developpez.com/html/1710-Internet-Ajax-Javascript-Asynchrone-et-XML.php

Asynchronous Javascript And XML (AJAX)

- Sécurité
  - Déporter de la logique applicative sur le client présente des risques
  - Remarque
    - L'envoi d'une requête asynchrone XHR à un autre serveur que celui ayant délivré le script est impossible (en principe)
  - Types d'attaques
    - Usurpation de session/d'identité :
      - on ne peut jamais être sûr que le client est celui qu'il prétend être
      - la partie applicative tournant sur le client est-elle réellement celle envoyée par le serveur ?

      ➔ Double validation (mots de passe)

Asynchronous Javascript And XML (AJAX)

- Sécurité
  - Types d'attaques
    - Cross-site scripting (XSS)
      http://cwe.mitre.org/top25/index.html#CWE-79
      https://www.owasp.org/index.php/XSS
      - violation de la *same-origin policy*
      - exécution de scripts malicieux dans le contexte d'un site « trusté »
      - exemple: injection de scripts dans les commentaires des forums
      - ➔ Revenir au HTML de base pour les données sensibles
      - ➔ Vérifier le contenu saisi par les utilisateurs
    - Cross-site request forgery (CSRF)
      http://cwe.mitre.org/top25/index.html#CWE-352
      https://www.owasp.org/index.php/CSRF
      - utiliser l'authentification d'un utilisateur pour réaliser des actions à son insu
      - souvent permise par l'authentification par cookies
      - ➔ Utiliser des champs hidden ou l'en-tête HTTP Referer

Fetch API

- Principe
  - Fournir des primitives de plus haut niveau que `XmlHttpRequest`
  - Accepter les réponses streamées (« chunked »)
  - Récupérer du texte ou du JSON (pas de XML)
  - ➔ Encapsuler les requêtes asynchrones dans des promesses
- Exemple (simple)

```
fetch('./monUrl/quiRenvoie/du.json')
.then(res => res.json())
.then(json => console.log(json));
```

Fetch API

- La méthode `fetch()`
  - Seul paramètre obligatoire : URL
  - Le reste est sous forme d'options dans un objet JSON

```
fetch(url, {
  method: "POST", // *GET, POST, PUT, DELETE, etc.
  mode: "cors", // no-cors, cors, *same-origin
  cache: "no-cache", // *default, no-cache, reload, force-cache, only-if-cached
  credentials: "same-origin", // include, *same-origin, omit
  headers: {
    "Content-Type": "application/json; charset=utf-8",
  },
  redirect: "follow", // manual, *follow, error
  referrer: "no-referrer", // no-referrer, *client
  body: JSON.stringify(data), // body data type must match "Content-Type" header
})
```

  - Remarque : il faut explicitement autoriser `fetch()` à envoyer des credentials (cookies…)

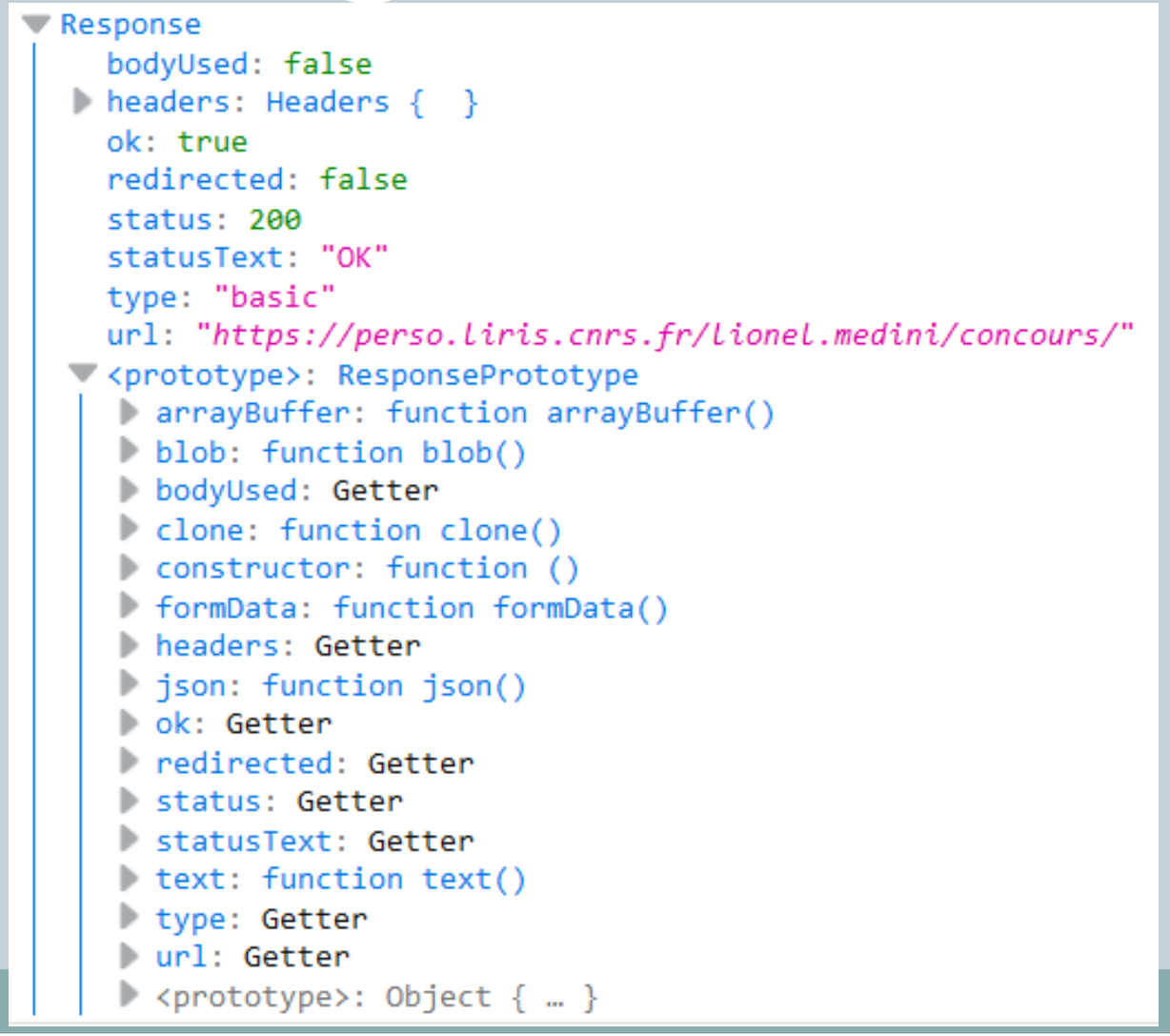
Fetch API

- La méthode `fetch()`
  - Renvoie une réponse
    - Qui peut être wrappée dans différents formats (en fonction du Content-Type)
    - Pas d'erreur (sauf erreur réseau)
      - ➔ Il faut tester le code de retour HTTP
  - La valeur de retour est une promesse
    - Pour permettre l'émission en plusieurs fois (objet `Observable`)

```
>> fetch('https://perso.liris.cnrs.fr/lionel.medini/concours/')
   .then(res => console.log(res.json));
<- ▶ Promise { <state>: "pending" }
   ▶ function json()
```

Fetch API

- Exemple de réponse



Fetch API

- Gestion des erreurs
  - `fetch()` lève une erreur en cas de :
    - Problème réseau
    - Réponse non conforme au résultat attendu (parsing JSON)
  - ...mais pas en cas d'erreur HTTP (404, 500...)

    ➔ C'est à vous de tester si la réponse correspond à vos attentes

```
fetch(https://maRessourceCORS/', {mode: 'cors'})
  .then(response => {
    if(response.ok) { return response.text(); }
    throw new Error('Erreur HTTP : ' + response.status);
  })
  .then(text => { console.log('Corps de la réponse :', text); })
  .catch(error => {
    console.log('Erreur dans la réception de la requête : ', error);
  });
```

Fetch API

- Références
  - Spec
    - https://fetch.spec.whatwg.org/
  - Tutos
    - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
    - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
    - https://developers.google.com/web/updates/2015/03/introduction-to-fetch
    - https://www.sitepoint.com/introduction-to-the-fetch-api/

Outils d'aide à la programmation côté client

- Bibliothèques
  - Ensembles de fonctions JavaScript réalisant des traitements spécifiques
  - Peuvent être réutilisées dans des applications
- Frameworks AJAX
  - Programmation dans un autre langage
  - Génération du code JavaScript
  - Mécanismes de communication standard entre client et serveur
- Référence :
  - http://softwareas.com/ajax-patterns/

Outils d'aide à la programmation côté client

- jQuery
  - Présentation
    - Bibliothèque de fonctions d'aide à la génération d'applications Web
      - Navigation dans un document et sélection d'éléments (X)HTML
      - Gestion d'événements
      - AJAX
      - Animations…
    - Utilisation très répandue
    - Existence de plugins développés par la communauté
    - Remarque : 2 versions
      - Compressée (production) / Lisible (développement)
  - Site Web
    http://jquery.com/
  - Documentation
    http://docs.jquery.com/

Outils d'aide à la programmation côté client

- jQuery
  - Quelques détails
    - L'objet jQuery
      - Équivalent : $
      - Fonction membre de l'objet window
      - Plusieurs utilisations
        - jQuery(selector [, context])
          - Renvoie tous les éléments DOM
            - Correspondant au sélecteur selector
            - À partir de l'élément DOM donné en context
        - jQuery(html [, ownerDocument])
          - Renvoie objet jQuery correspondant à un ou plusieurs élément(s) DOM
            - Rajouté(s) au document ownerDocument
            - Correspondant à la chaîne de caractères html
        - jQuery(callback)
          - Appelle une fonction de callback quand le DOM est chargé
          - Équivalent : jQuery(document).ready()

Outils d'aide à la programmation côté client

- jQuery
  - Quelques détails
    - Sélecteurs
      - Tous les sélecteurs CSS (versions 1 à 3)
      - Des attributs et fonctions spécifiques
        - :checked, :empty, :even, :header…
        - :eq(), :lt(), :not(), :nth-child()…
      - Des notations particulières
        - Sélecteurs multiples : ("selector1", "selector2 ", "selector3 ")
        - Next adjacent selector : ("previous + next ")
        - Next sibling selector : ("previous ~ sibling ")
      - Référence : http://api.jquery.com/category/selectors/

Outils d'aide à la programmation côté client

- jQuery
  - Quelques détails
    - Objets jQuery
      - Toutes les méthodes jQuery retournent un ou plusieurs (tableau) **objets jQuery**
      - Chaque objet jQuery possède l'ensemble des méthodes définies par l'API jQuery

      ➔ On peut donc chaîner les méthodes entre elles :

      ```
      $('h1#titre').html($('title').html()).before('Voici le titre :').click(mafonction);
      ```

      - Le chaînage s'appliquera pour chacun des objets retournés par chaque fonction de la chaîne
      - Exemples : http://www.siteduzero.com/tutoriel-3-160891-jquery-ecrivez-moins-pour-faire-plus.html?all=1

Outils d'aide à la programmation côté client

- jQuery
  - Quelques détails
    - Gestion des événements
      - Fourniture de fonctions pour l'ajout d'EventHandlers d'événements standards…
        - click(), dblclick(), load()
      - …Ou définis par la bibliothèque
        - ready()
      - Permet d'attacher une callback à un événement quelconque
        - bind(), unbind()
      - Référence : http://api.jquery.com/category/events/
      - Remarque : l'objet Event est lui aussi surchargé par un objet jQuery spécifique http://api.jquery.com/category/events/event-object/

Outils d'aide à la programmation côté client

- jQuery
  - Quelques détails
    - Requêtes asynchrones
      - AJAX

```
$.ajax({
  url: "test.html",
  context: document.body,
  success: function(){
    $(this).addClass("done");
  }
});
```

      - JSON
      - Générale

Outils d'aide à la programmation côté client

- jQuery
  - Quelques détails
    - Requêtes asynchrones
      - AJAX
      - JSON
        - jQuery.getJSON(url [, data] [, success(data, textStatus, jqXHR)])
        - Équivalent à :
        - ```
          $.ajax({
            url: "test.html",
            datatype: 'json',
            context: document.body,
            success: success
          });
          ```
      - Générale
          ```

Outils d'aide à la programmation côté client

- jQuery
  - Quelques détails
    - Requêtes asynchrones
      - AJAX
      - JSON
      - Générale
        - jQuery.get(url [, data] [, success(data, textStatus, jqXHR)] [, dataType])
        - Équivalent à :
        - ```
          $.ajax({
            url: "test.html",
            datatype: datatype,
            context: document.body,
            success: success
          });
          ```
      - Référence : http://api.jquery.com/category/ajax/
          ```

Outils d'aide à la programmation côté client

- jQuery UI
  - Extension de jQuery
    - Bibliothèque d'éléments d'interface (thèmes, widgets, primitives d'interaction)
    - Permet de rajouter facilement des interactions complexes
    - Permet de rendre une application Web plus dynamique
    - Exemple
      - Drag'n drop : http://jqueryui.com/demos/draggable/
    - Utilisation
      1. Identifier les éléments dont on a besoin
      2. Construire et télécharger sa bibliothèque personnalisée
      3. L'utiliser dans son application
    - Site Web
      http://jqueryui.com/

Single-Page Application (SPA)

- Principes
  - Toute l'application côté client est dans une unique page Web
    - Une seule page à charger (→ moins de « lag » entre les vues)
    - Contient toutes les vues de l'application
      - Toutes les vues sont « hidden » sauf une
    - Mécanisme de routage fondé sur le hash de l'URL pour sélectionner la vue à montrer
  - Permet d'implémenter un pattern MV* complet côté client
    - Model : scripts exécutés localement + échanges asynchrones de données avec le serveur
    - View : ensemble de <section> dans la page HTML + mécanisme de templating
    - Controller / Presenter / View-Model : routage + gestion de événements + …

Single-Page Application (SPA)

- Routage
  - Principe
    - Intercepter le changement de hash dans l'URL
    - Récupérer les – éventuels – paramètres
    - Déclencher un callback
  - Outils
    - événement `hashchange`
    - hash `window.location.hash`

Single-Page Application (SPA)

- Routage
  - Exemple de code

```
<style>
  .active { display: block; }
  .inactive { display: none; }
</style>
<script>
function show(hash) {
  $('.active')
    .removeClass('active')
    .addClass('inactive');
  $(hash)
    .removeClass('inactive')
    .addClass('active');
}
</script>
```

```
<h1>Ma première SPA</h1>
<section id='index' class='active'>
  <p>Ceci est la vue d'accueil.</p>
  <ul>
    <li><a href='#vue1'>Vue 1</a></li>
    <li><a href='#vue2'>Vue 2</a></li>
  </ul>
</section>
<section id='vue1' class='inactive'>
  <p>Ceci est la vue 1.</p>
  <p><a href='#index'>Vue accueil</a></p>
</section>
<section id='vue2' class='inactive'>
  <p>Ceci est la vue 2.</p>
  <p><a href='#index'>Vue accueil</a></p>
</section>
<script>
  window.addEventListener(
    'hashchange',
    () => { show(window.location.hash); }
  );
</script>
```

Conclusion

- De plus en plus d'applications Web « riches »
  - Charge répartie entre client et serveur
  - Outils de conception et de développement matures
  - Bonne ergonomie grâce aux technologies CSS, JavaScript
  - Standardisation
    - Indépendance vis-à-vis de l'OS
    - Ne correspond pas aux stratégies des vendeurs d'OS ou de logiciels
  - Disponibles sur Internet
    - Indépendance vis-à-vis de la machine utilisée

Conclusion

- Quelques règles pour développer une application Web riche
  - Outils de développement
    - Utilisez les ressources à votre disposition
      - Choisissez une bibliothèque aussi standard que possible
      - Il existe aussi des feuilles de style CSS open source
        Exemple : http://www.oswd.org/
    - Échafaudez (« scaffhold ») vos projets
  - Compatibilité avec les navigateurs
    - Vérifiez la compatibilité avec les navigateurs visés
    - Testez la fonctionnalité à utiliser, pas le navigateur…
    - Utilisez des façades aussi souvent que possible

Perspectives

- Quelles applications Web pour demain ?
  - Deux types d'applications Web
    - RDA : Rich Desktop Application
      - S'exécute dans un complément installé sur le poste de travail
        - Microsoft SilverLight
        - Adobe AIR

      ➔ Moins de restrictions de sécurité
    - RIA : Rich Internet Application
      - S'exécute dans un navigateur

      ➔ Doit être compatible avec une majorité de navigateurs

      ➔ Doit être aussi performante qu'une RDA (chargement, exécution)
  - Ne pas perdre de vue l'arrivée de l'informatique ubiquitaire
    - PDA, Smartphones, connexions par réseaux GSM
    - Ressources client réduites : matérielles et logicielles
    - Fonctionnalités et usages spécifiques : contextualisation, géolocalisation…

Quelques références

- Ressources
  - Une liste de frameworks : http://www.ajaxprojects.com/
  - Une liste de plein de choses : http://ajaxpatterns.org/
  - En particulier, quelques outils de conception et de développement
    - Frameworks
      - openAjax (IBM) : Dojo
      - Ruby / Ruby on Rails (RoR)
      - Plugins Eclipse : Rich Ajax Platform, Direct Web Remoting
      - PHP : http://ajaxpatterns.org/PHP_Ajax_Frameworks
    - Librairies
      - Jquery : http://jquery.com/
      - Google Web Toolkit (AJAXSLT…)