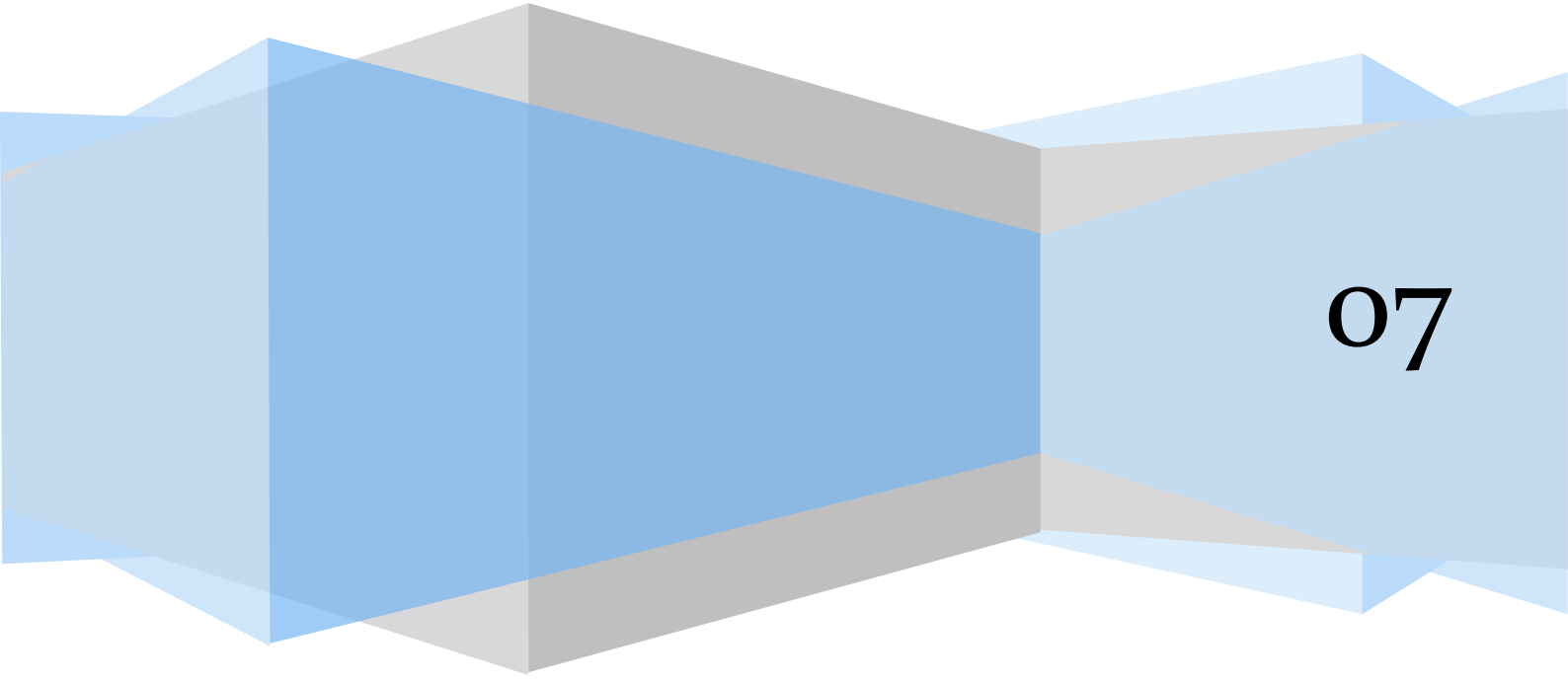
HEC Lausanne

dojo the javascript toolkit

# Projet RAT

## Remarkable Ajax Tutorial

Grenon Benoît & Fritscher Boris



07

## Table des matières


Introduction ..... 3

Prérequis ..... 3

Qu'est-ce qu'AJAX? ..... 3

A quoi sert AJAX? ..... 3

Comment fonctionne AJAX? (adaptivepath) ..... 4

Utilité d'un toolkit JavaScript comme DOJO ..... 7

Les étapes du tutorial ..... 9

Etape 1: Hello World ..... 9

Mise en place ..... 9

Widget qu'est-ce que c'est? ..... 11

Etape 2: Layout Widget ..... 12

Mise en page facilité ..... 12

Chargement partiel sans frais ..... 14

Etape 3: Form Widget ..... 15

Formulaire facilité ..... 15

Validation rapide ..... 17

Etape 4: Communication avec le serveur ..... 18

Autocomplete avec un serveur ..... 18

Code qui connecte ..... 19

Etape 5: drag & drop et communication avancée ..... 21

Drag & Drop ..... 21

Communication ..... 22

Conclusion ..... 25

Ressources ..... 26

Références ..... 26

# Introduction

## Prérequis

Pour pouvoir tester, modifier ou encore implémenter les concepts démontrés dans ce tutorial, il vous faut une installation serveur de type LAMP ou un accès à un hébergement offrant PHP et MySQL 5.

Il est également fortement recommandé de posséder des bonnes notions dans les langages de mise en forme utilisés pour la création de pages web comme: HTML et CSS. De plus, il est recommandé d'avoir des notions de base en JavaScript ainsi qu'en PHP. Nous avons choisi d'utiliser le PHP comme language de traitement côté serveur, car il est un des plus répandus sur la toile. Mais il est également possible d'en utiliser d'autres qui vous seront peut-être plus familiers.

Votre serveur LAMP nécessite la version 5.2 ou supérieure de PHP, car notre tutoriel nécessite les nouvelles fonctions JSON ainsi que la version 5 du serveur MySQL pour des raisons de prise en charge de caractères.

## Qu'est-ce qu'AJAX?

Un produit de nettoyage?, un club de football néerlandais? Non plus! Comme tout le monde en parle, voici ce que cela signifie vraiment.

AJAX: Asynchronous JavaScript and XML

D'après la signification de l'acronyme, vous remarquerez rapidement qu'AJAX n'est pas une technologie en soi, mais un «regroupement de technologies déjà existantes pour créer des applications web découplé client-serveur». (openweb.eu.org):

- une présentation utilisant XHTML et CSS ;
- la manipulation dynamique des pages à travers DOM;
- la manipulation des données avec XML et XSLT ;
- l'échange des données de manière asynchrone avec XMLHttpRequest ;
- le tout étant assemblé avec du Javascript

En 2005, le terme AJAX est employé et apparu sur la toile. Il s'est vite doté d'une bonne réputation et fait actuellement sa place comme méthode informatique de développement d'application Web. Cependant, ses composants sont plus anciens, par exemple, l'objet XMLHttpRequest est l'héritier d'un objet ActiveX développé par Microsoft et introduit dans Internet Explorer 5 en 1999. Il a été amélioré sous Internet Explorer 6 pour finalement devenir un standard quelques années plus tard dans les autres browsers comme FireFox, Safari, etc.

## A quoi sert AJAX?

Voici la définition d'après le site openweb.eu.org: «Cette technologie permet de faire des requêtes HTTP afin de récupérer des données au format XML qui pourront être intégrées à un document». Cela peut être très utile pour mettre à jour des données sans pour autant recharger la page. En somme, AJAX va-t-il changer ma vie développeur? Certainement oui, mais sûrement l'expérience de navigation de l'internaute. AJAX augmente considérablement

l'ergonomie ainsi que la facilité de la navigation des sites Internet. Il permet ainsi aux applications Web 2.0 de ressembler et d'être de plus en plus compétitives face aux applications qui fonctionnent sur des systèmes d'exploitation.

Les avantages possibles:

- Diminution importante de l'utilisation de la bande passante: seules les données dynamiques sont chargées et non plus tout le document;
- Interactivité accrue: plus de rechargement de la page;
- Rationalisation du code: des routines (de vérification par exemple) n'ont plus à être écrites et maintenues dans deux langages (côté client et côté serveur).
- Diminution du nombre de pages dynamiques nécessaires pour effectuer une même ou un groupe d'actions.

Les inconvénients possibles:

- Ne fonctionne pas sans JavaScript, ni dans les navigateurs les plus anciens;
- Ne fonctionne qu'avec HTTP: il est impossible de récupérer des données sur un disque local (ce qui est normal);
- Peut gêner les habitudes des internautes, car certaines fonctionnalités considérées comme standard ne sont plus disponibles, à savoir:
    - Les marques-pages et liens vers la page;
    - L'enregistrement des pages;
    - Le bouton retour.

## Comment fonctionne AJAX? (adaptivepath)

Commençons par expliquer le modèle d'une application Web classique. Le client informe le serveur quelle page il désire consulter, ce dernier fait ses calculs et la lui renvoie. Ensuite, l'utilisateur clique sur un nouveau lien ou effectue une nouvelle action, ce qui déclenche à nouveau une phase d'échanges entre le client et le serveur durant laquelle l'utilisateur doit attendre.

Ces temps d'attentes (pendant lesquels l'information est traitée par le serveur) résultent de la communication synchrone entre le client et le serveur, ce qu'Ajax permet d'éviter. Voici un exemple concret pour être un peu plus explicite:

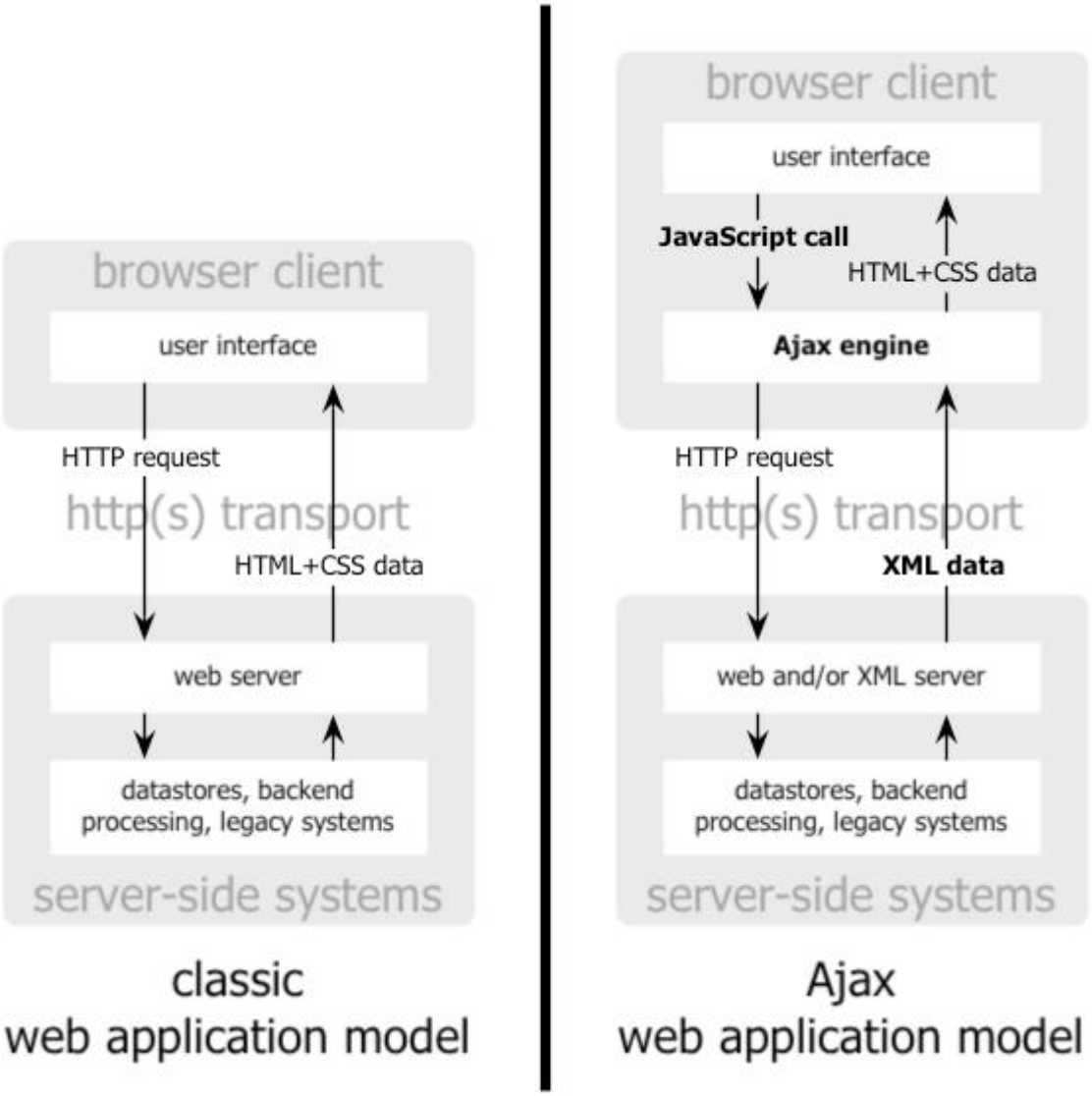



**Figure 1 Le modèle classique d'application web (gauche) comparé au modèle AJAX (droite) [adaptivepath]**

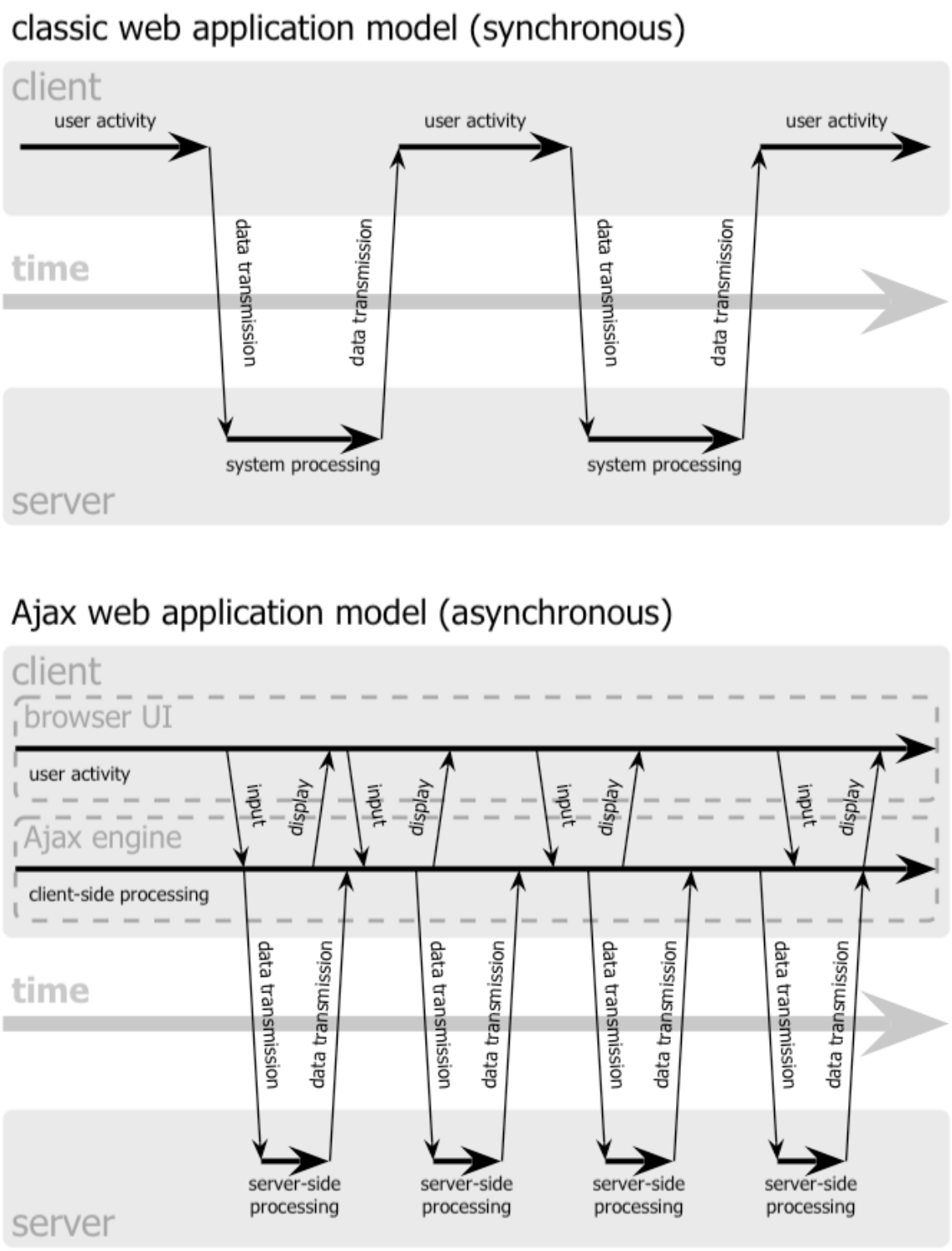



**Figure 2 Le modèle synchrone des applications web classiques (haut) comparé au modèle asynchrone des applications AJAX (bas) [adaptivepath]**

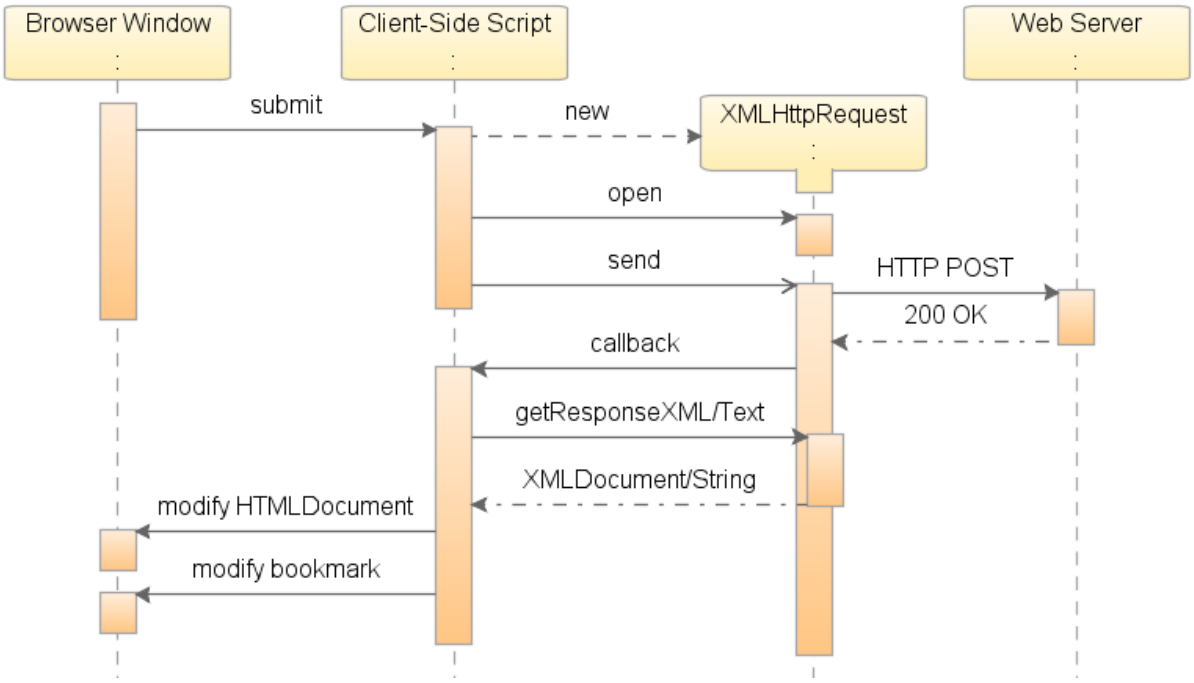



**Figure 3 Une vue plus détaillée d'une interaction Ajax [java.net]**

Le déroulement d'une requête quelconque sous AJAX est plus compliqué du fait que la communication est asynchrone. Mais il permet de garantir à l'utilisateur une interactivité accrue, car il n'aura pas besoin d'attendre un rechargement de la page entière pour disposer des nouvelles informations qu'il attend.

Une fois que la page est chargée avec ses composants JavaScript, une action de l'utilisateur va déclencher une communication en arrière-plan de la part du script JavaScript grâce à un objet XML avec le serveur. Il peut donc continuer à effectuer des actions sur la page. Dès que le serveur répond de nouveau par un objet XML le script le récupère, et va mettre à jour la partie de la page web qui est concernée par les nouvelles informations.

## Utilité d'un toolkit JavaScript comme dojo

Le toolkit DOJO est un cadriciel (framework) Open Source qui peut se résumer en un ensemble de bibliothèques qui a pour but de permettre le développement rapide d'applications Web abouties en JavaScript. Elle permet d'implémenter facilement de l'AJAX dans un site Web. Par ailleurs, elle se targue d'être un outil d'édition d'application Web professionnelle. Mais qui est derrière DOJO? Un peu de culture informatique générale…

DOJO Foundation est une organisation à but non lucratif qui a pour but de promouvoir ses projets en Open Source. Leur philosophie se résume en quatre points:

- Encourage adoption
- Discourage political contention
- Encourage collaboration and integration with other projects
- Be transparent

Elle est parrainée par Ibm corporation, Sun Microsystems, AOL, et bien d'autres encore. Cette fondation généreuse met à disposition son toolkit sous licence duale, à savoir Academic Free License v2.1 et BSD License. Ce qui vous autorise de l'utiliser, de créer des logiciels sans rétribuer de royalties à DOJO Fondation que ce soit de manière monétaire ou en développement du toolkit et finalement reconnaît l'entièreté de vos droits quant à la future application que vous allez créez grâce au toolkit DOJO.

L'avantage d'utiliser un toolkit comme DOJO ( http://dojotoolkit.org ) est que le développeur peut profiter de libraires et *templates* préfabriqués. Cela permet de créer une application riche plus rapidement, sans devoir écrire tous les bouts de code de bas niveau qui sont nécessaires à la communication client-serveur, ou pour l'animation, ou le remplacement de balises HTML.

## Les étapes du tutorial


Le but de ce tutorial est de créer un site rudimentaire d'e-commerce qui s'appellera «*Shopalot*», en 5 étapes seulement! Pour cela, nous commencerons par l'affichage d'un simple HelloWorld que nous étendrons jusqu'à obtenir un site e-shop entièrement fonctionnel. Cependant, nous ne couvrons qu'une petite partie de tous les modules disponibles de l'énorme toolkit qu'est DOJO. Il vous restera donc beaucoup de choses à explorer par vous-même, l'important est de comprendre la base de développement sous AJAX. Si quelque chose ne vous paraît pas clair, il est toujours conseillé d'aller consulter l'API http://dojotoolkit.org/api/ ou directement la source des fichiers DOJO.

# Etape 1: Hello World

## Mise en place

La première étape consiste à télécharger le toolkit Dojo (si ce n'est pas encore fait) http://dojotoolkit.org/download/ et de l'extraire dans le dossier de votre projet. Ensuite, créez un fichier shopalot.html dans le répertoire source de votre projet. Vous devriez avoir:

```
/
|--dojo/ <- les fichiers dojos décompressés
|--shopalot.html
```

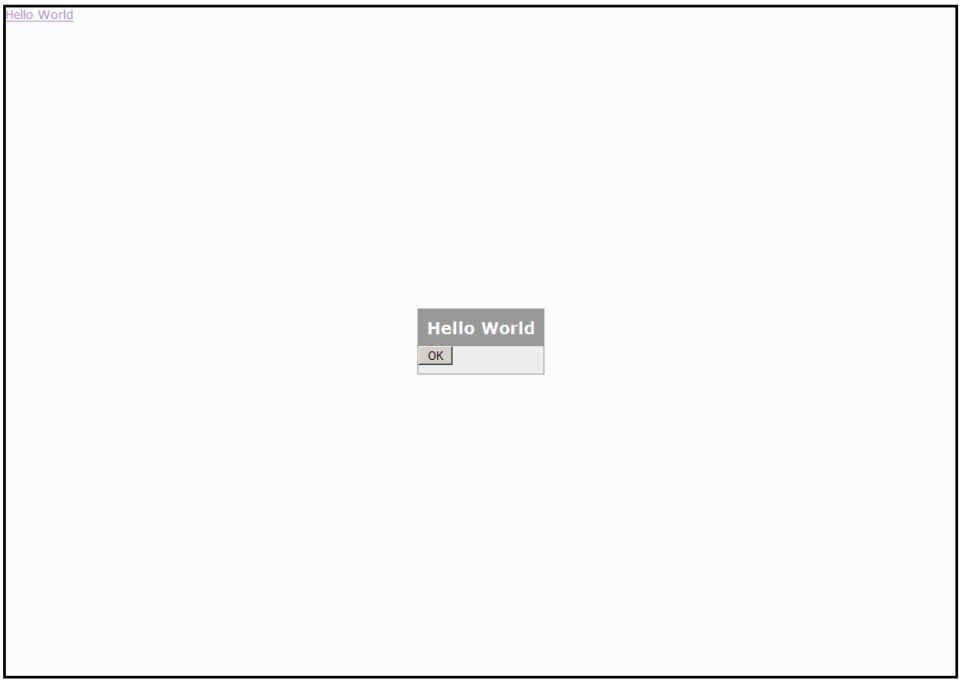
En ayant construit la structure de base et téléchargé tous les fichiers nécessaires, nous allons enfin pouvoir commencer à concevoir shopalot.html.

Remarque préliminaire; vous avez la possibilité de télécharger un fichier zip contenant tous les fichiers sources du tutorial, ainsi que des éléments graphiques complémentaires utilisés par celui-ci. Donc, si vous ne préférez pas copier/coller le code de ce document et d'étendre votre propre fichier, il vous sera toujours indiqué quels sont les fichiers de référence de chaque étape du tutorial.

Copiez/collez le code ci-dessous dans votre nouveau fichier shopalot.html ou servez vous du fichier etape1.html de nos sources. Il faut également récupérer le fichier style.css si vous voulez récupérer notre mise en page.

```
<html>
<head>
<title>Shopalot: We have everything you will ever need to buy</title>
<script type="text/javascript" src="dojo/dojo.js"></script>
<script type="text/javascript">
dojo.require("dojo.widget.*" );
dojo.require("dojo.widget.Dialog");
dojo.require("dojo.event.*");
//all dojo.require above this line
dojo.addOnLoad(init); //function to execute when the page starts

// executed at start
function init(e) {
      //dialogHelp
      var dlgHelp = dojo.widget.byId("dialogHelp");
      var btn = document.getElementById("hider0");
      dlgHelp.setCloseControl(btn);

      //connect links
      var linkQuestion = dojo.byId("linkQuestion");
      dojo.event.connect(linkQuestion,'onclick',dlgHelp,"show");
}
</script>
</head>
<body>
<a href="#" id="linkQuestion">Hello World</a>
<div dojoType="dialog" id="dialogHelp" bgColor="white" bgOpacity="0.5"
toggle="fade" toggleDuration="250">
      <h1>Hello World</h1>
      <form onsubmit="return false" action="">
      <input type="button" id="hider0" value="OK" /></form>
</div>
</body>
</html>
```

Le résultat attendu de ce code devrait ressembler à ceci:



## Widget qu'est-ce que c'est?

Avant de regarder en détail le code JavaScript, observons d'abord les propriétés de la balise  de notre code. On voit qu'elle comporte un attribut «dojoType», c'est ce que dojo appelle un «widget», c'est-à-dire une balise qui, lors de l'exécution du code, sera remplacée par du code plus complexe avec une mise en page et une fonctionnalité avancée. Pour savoir quels sont les paramètres de ce «widget» (s'il n'est pas encore documenté), le plus simple c'est de le retrouver dans l'API. Par exemple pour notre «widget» de type «dialog» on trouve:

```
// focusElement: String
// provide a focusable element or element id if you need to
// work around FF's tendency to send focus into outer space on hide
focusElement: "",

// bgColor: String
//      color of viewport when displaying a dialog
bgColor: "black",

// bgOpacity: Number
// opacity (0~1) of viewport color (see bgColor attribute)
bgOpacity: 0.4,

// followScroll: Boolean
// if true, readjusts the dialog (and dialog background) when the user
moves the scrollbar
followScroll: true,

// closeOnBackgroundClick: Boolean
// clicking anywhere on the background will close the dialog
closeOnBackgroundClick: false,
```

Revenons au code JavaScript:

Avant de pouvoir utiliser DOJO, il faut charger le script principal du toolkit, c'est ce que fait la première balise <*script*>.

Dans la seconde balise <*script*> nous insérons notre propre code. Il faut toutefois indiquer quels sont les modules que nous allons utiliser grâce à la commande «*dojo.require()*». On peut également spécifier tout comme éléments à charger dans la catégorie si l'on met *, mais cela va ralentir le chargement de la page.

Ensuite, nous spécifions la fonction qui sera exécuter au chargement de la page grâce à *dojo.addOnLoad(nom_de_notre_fonction)* que nous avons appelé dans notre cas «*init*».

Vous pouvez constater que nous avons utilisé trois manières différentes de récupérer la référence d'un objet HTML portant un id et de l'assigner dans une variable. Il est donc important de donner un attribut id aux balises HTML sur lesquelles on veut interagir avec du code JavaScript. Cependant, il n'y a en fait que deux fonctions, car «*dojo.byId*» n'est qu'un raccourci de la fonciton JavaScript «*document.getElementById*». Il ne reste donc plus que «*dojo.widget.byId*» qui lui ne récupère non seulement les informations HTML de l'objet portant le nom indiqué, mais aussi tout l'objet «widget» avec ses fonctions spéciales. C'est ainsi que l'on peut par la suite exécuter des fonctions comme par exemple «*setCloseControl*»

qui va assigner le bouton HTML que l'on vient de récupérer dans la variable «btn» comme étant celui qui va désactiver la boîte de dialogue.

Il ne reste plus qu'à expliquer ce que fait la commande «*dojo.event.connect*». Celle-ci permet, comme s'on nom l'indique, de connecter des actions de différents objets. Dans notre cas nous voulons que lorsque l'utilisateur clique sur le lien qui porte le nom «linkQuestion», cela affiche la boîte de «dialog» crée par le «widget» DOJO. Nous spécifions donc l'objet source, la fonction qui écoute et nécessaire au déclanchement, l'objet «destination» et quelle fonction de celui-ci sera appelée.

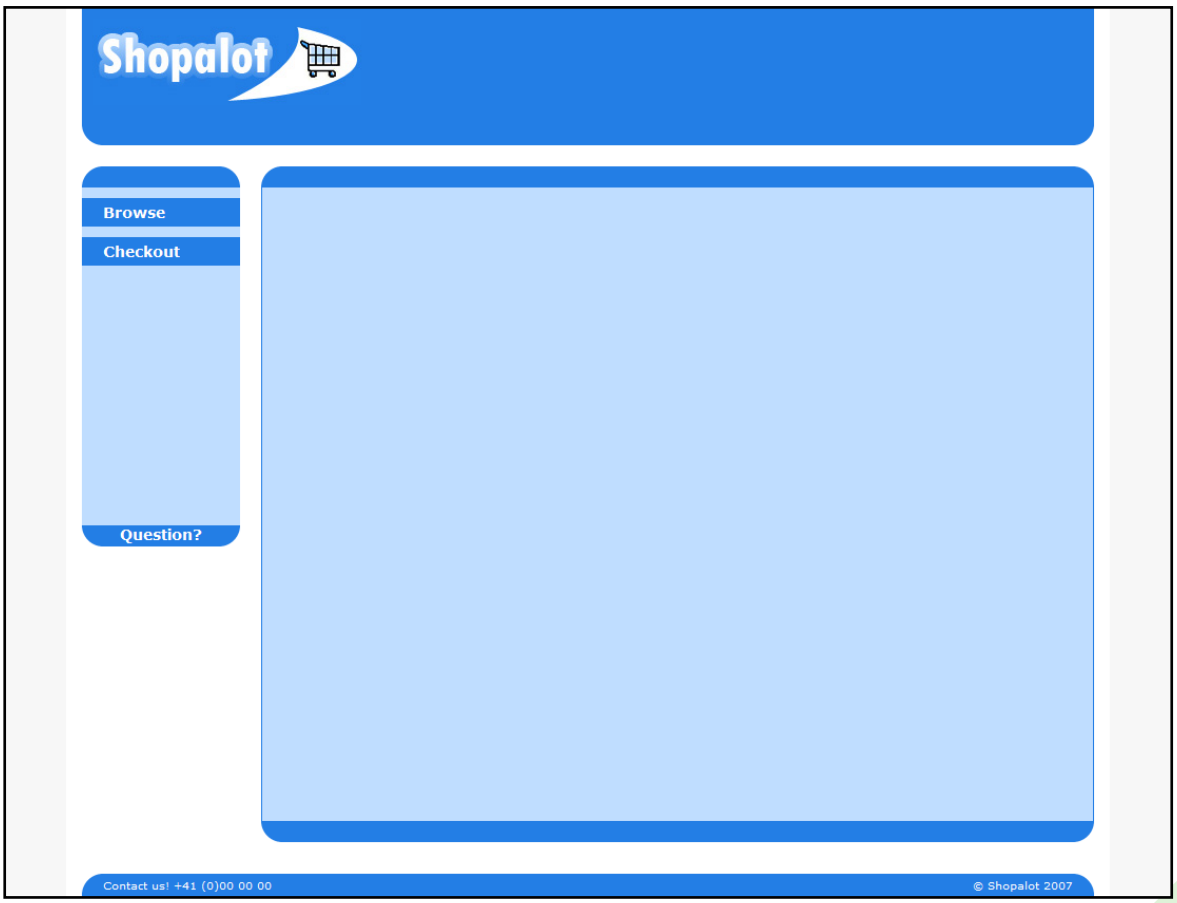
L'avantage de connecter les éléments de cette manière est d'avoir plus de flexibilité et de contrôle que si le code était écrit au niveau de chaque balise à travers tout le texte HTML.

Remarque: le code CSS n'est pas expliqué, mais si cela vous intéresse, observez le fichier style. Il faut toutefois savoir que certains *templates* DOJO peuvent être modifiés par CSS en spécifiant *body. «dojoValidateEmpty»*, si cela ne devait pas fonctionner normalement.

## Etape 2: Layout Widget

### Mise en page facilité

DOJO n'a pas uniquement des fonctionnalités pour gérer facilement l'interaction sur un site web, mais aussi sa mise en page. Voici la mise en page que nous allons obtenir, sans devoir utiliser d'astuces avec des propriétés css telles que «float, margin»...



Insérez ce code dans vote fichier shoplalot.html (après le tag «body») ou simplement référez-vous au fichier etape2.html. Pour les graphismes, logo et bords arrondis, copiez le répertoire images et son contenu dans votre dossier de travail.

```
<div dojoType="LayoutContainer" layoutChildPriority='none' id="main">
 <!-- top -->
 <div dojoType="ContentPane" layoutAlign="top" id="top">
  <div><img src="images/shopalot.jpg" alt="Shopalot" /></div>
 </div>
 <!-- bottom -->
 <div dojoType="ContentPane" layoutAlign="bottom" id="bottom">
  <div><span>Contact us! +41 (0)00 00 00</span> &copy; Shopalot 2007</div>
 </div>
 <!--  menu -->
 <div dojoType="ContentPane" layoutAlign="left" id="left">
  <!-- menu rounded corner top -->
  <div class="box_top"><div> </div></div>
  <!-- menu content -->
  <div id="menu_content">
   <a href="#" id="linkBrowse">Browse</a>
   <a href="#" id="linkCheckout">Checkout</a>
  </div>
  <!-- menu rounded corner bottom -->
  <div class="box_bottom">
   <div><a href="#" id="linkQuestion"> Question?</a></div>
  </div>
 </div> <!-- fin menu -->
<!-- main content -->
<div dojoType="ContentPane" layoutAlign="client" id="contentPane"
cacheContent="false">
 <!-- main content rounded corner top -->
 <div class="box_top"><div> </div></div>
 <!-- container that loads the other pages -->
 <div  dojoType="ContentPane"  id="content" cacheContent="false"></div>
 <!-- main content rounded corner bottom -->
 <div class="box_bottom"><div> </div></div>
 </div>
</div>
```

Dans ce code nous utilisons deux nouveaux «widgets» DOJO: «LayoutContainer» et «ContentPane».

Le «LayoutContainer», comme son nom l'indique, n'a pas vraiment de fonction autre que celle de permettre le positionnement des éléments qui se trouvent à l'intérieur de celui-ci. D'où son seul attribut «*layoutChildPriority*» qui peut prendre les valeurs suivantes: «none, top-bottom, left-right» qui définissent l'ordre de priorité dans lequel les éléments sont alignés.

Le «ContentPane» est un «div» spécial qui, s'il est placé à l'intérieur d'un «LayoutContainer», tout comme d'autres «widgets», prend en plus des ses propres attributs, un attribut de positionnement «*layoutAlign*» qui peut contenir les valeurs, «left, right, bottom, top et client». Cependant, dans toute la page le «ContentPane» n' a le droit d'avoir qu'un seul élément avec la valeur client, étant donné que celui-ci remplit l'espace non occupé par les autres.

Pour notre site nous utilisons simplement un «top» et un «bottom» pour l'en-tête et le pied de page, un «left» pour le menu et le client pour la zone principale.

## Chargement partiel sans frais

Le «ContentPane» de DOJO offre, en plus de ses attributs de positionnement, la possibilité de changer aisément et dynamiquement son contenu ou de charger une page tierce sans avoir recours à un «iframe» (ce qui peut éviter des problèmes de droits d'accès lors de «cross-site scripting», que nous ne couvrons pas dans ce tutorial).

Voici quelques attributs intéressants de l'API.

```
// href: String
// The href of the content that displays now
// Set this at construction if you want to load externally,
// changing href after creation doesnt have any effect, see setUrl
href: "",

// extractContent Boolean: Extract visible content from inside of <body>
.... </body>
extractContent: true,

// parseContent Boolean: Construct all widgets that is in content
parseContent:  true,

// cacheContent Boolean: Cache content retreived externally
cacheContent:  true,

// loadingMessage: String
// Message that shows while downloading
 loadingMessage: "Loading...",
```

Nous constatons que la page a chargé une page HTML complète et dont seule la party body sera recopiée par défaut.

Pour tester le chargement dynamique, créez une page form.html avec juste un titre noté «checkout» à l'intérieur, comme dans notre fichier etape2_form.html.

Ensuite, ajoutez à notre fonction «init» les deux lignes de code qui figurent ci-dessous. D'après l'étape 1, vous devriez savoir le rôle qu'ont ces deux lignes, c'est-à-dire d'appeler la fonction «checkout» quand on clique sur le lien «checkout» de note menu.

Par ailleurs, recopiez la fonction «checkout», ou simplement utilisez le fichier etape2.html.

```
function init(e) {
      ... ...
      var linkCheckout = dojo.byId("linkCheckout");
      dojo.event.connect(linkCheckout,'onclick','checkout');
}
function checkout(){
      var contentPane = dojo.widget.byId("content");
      contentPane.setUrl('form.html');
}
```

Dans cette fonction «checkout», nous récupérons l'objet «widget» dans lequel nous voulons charger la nouvelle page et en exécutant la fonction «setUrl» avec comme paramètre l'emplacement du nouveau fichier.

Si nous avions voulu afficher directement la page form.html au chargement de la page nous aurions pu utiliser l'attribut «href» sur le «widget» «ContentPane».

```
<div dojoType="ContentPane"  id="content" cacheContent="false"
href="form.html"></div>
```

## Etape 3: Form Widget

### Formulaire facilité

DOJO contient aussi des «widgets» facilitant la création de formulaire plus intuitif. Par exemple, nous pouvons dénombrer une aide à la sélection de date, un éditeur de texte riche, un contrôle du contenu d'un champ.

Voici le formulaire que nous allons réaliser.

Remplacez le code dans votre fichier form.html par celui-ci ou récupérer le fichier etape3_form.html.

```
<form id="myForm" action="showPost.php" method="post">
 <h1>Checkout</h1>
 <div class="formFrame">
  <div class="formRow">
   <label for="lastname">Last Name</label>
   <input id="lastname" type="text" name="lastname"
   dojoType="ValidationTextbox" trim="true" ucfirst="true" required="true"
   missingMessage="* Your last name is required." />
  </div>
  <div class="formRow">
   <label for="firstname">First Name</label>
   <input id="firstname" type="text" name="firstname"
   dojoType="ValidationTextbox" trim="true" ucfirst="true" required="true"
   missingMessage="* Your first name is required." />
  </div>
  <div class="formRow">
   <label for="email">Email Adress</label>
   <input type="text" name="email" id="email" dojoType="EmailTextbox"
   trim="true"    required="true" invalidMessage="Invalid Email Address." />
  </div>
  <div class="formRow">
   <label for="birthday">Birthday</label>
   <input type="text" name="birthday" id="birthday" dojoType="dropdowndatepicker"
   displayFormat="dd.MM.yyyy" value="1980-01-01" />
  </div>
  <div class="formRow">
   <label for="shipping">Shipping</label>
   <select  name="shipping" id="shipping" dojoType="ComboBox">
    <option value="1">Very Fast</option>
    <option value="2" selected>Normal</option>
    <option value="3">Slow</option>
   </select>
  </div>
  <div class="formRow">
   <label for="extras[]">Extras</label>
    <input dojotype="checkbox" type="checkbox" name="extras[]" value="1"
/>Printed Catalog<br />
    <input dojotype="checkbox" type="checkbox" name="extras[]" value="2"
    checked="checked" disabled="disabled" /> Spam Me<br />
    <input dojotype="checkbox" type="checkbox" name="extras[]" value="3"
    checked="checked" /> Give my address to 3rd parties<br />
    <input dojotype="checkbox" type="checkbox" name="extras[]" value="4" />Send
me a gift on my birthday</div>
  <div class="formRow" style="text-align: center;">
   <button dojoType="button" onclick="submit();">
    <div style="height: 20px; width: 150px;">Confirm</div>
   </button>
  </div>
 </div>
</form>
```

Comme vous pouvez le voir, nous avons surtout utilisé un «widget» qui s'appelle «*ValidationTextbox*», mais il en existe bien d'autres encore: «CurrencyTextbox», «DateTextbox», «emailTextbox», «IntegerTextbox», «IpAddressTextbox»,

«RealNumberTextbox», «TimeTextbox»,… Consultez la page de validation dans le dossier demo du toolkit pour visualiser leurs différents effets.

Revenons à notre «ValidationTextbox», si nous regardons dans l'API, nous nous rendons compte que nous pouvons spécifier un message unique dans trois situations: message d'invite (promptMessage), message si le champ est invalide (invalidMessage) et si le champ est vide (missingMessage).

Cependant, il n'y a pas de référence à «trim», ni «ucFrist» que nous avons utilisé pour effacer les espaces vides après le contenu ou pour mettre en majuscule la première lettre de chaque mot entré.

Pour trouver d'où viennent ces attributs il vous faut vous rendre à la section «*Inheritance Chain*» de l'API et vous constaterez que «ValidationTextbox» à comme parent Textbox. Et c'est bien dans les infos de celui-ci que vous y retrouverez ces attributs.

```
//trim: Boolean
//Removes leading and trailing whitespace if true.  Default is false.
trim: false,

//uppercase: Boolean
//Converts all characters to uppercase if true.  Default is false.
uppercase: false,

//lowercase: Boolean
//Converts all characters to lowercase if true.  Default is false.
lowercase: false,

//ucFirst: Boolean
//Converts the first character of each word to uppercase if true.
ucFirst: false,

//digit: Boolean
//Removes all characters that are not digits if true.  Default is
false.
digit: false,
```

Un autre composant que nous avons utilisé est le «*dropdowndatepicker*». Nous y avons juste spécifié le format d'output désiré et laissé les autres paramètres par défaut, mais soyez attentif au fait que vous avez la possibilité de configurer énormément d'autres variables.

Les autres éléments que nous avons utilisés comme «ComboBox, CheckBox, Button» n'ont pas vraiment de fonctions spéciales, ils permettent juste d'obtenir un graphisme plus attrayant sans aucune modification préalable. Cependant, si vous voulez créer vos propres graphismes avancés, il vous faudra allez bien plus loin que ce que nous proposons dans ce tutorial. Un bon point de départ est la démo sur les boutons disponible dans le toolkit.

### Validation rapide

Au niveau du code JavaScript nous avons uniquement crée une fonction «submit()» appelée par le bouton du formulaire. Cette fonction récupère les références vers les objets champs puis vérifie si ceux-ci sont soit rempli ou soit valide d'après le type de champ, grâce aux différentes

fonctions préprogrammées des différents «widgets». (Code disponible dans le fichier etape3.html)

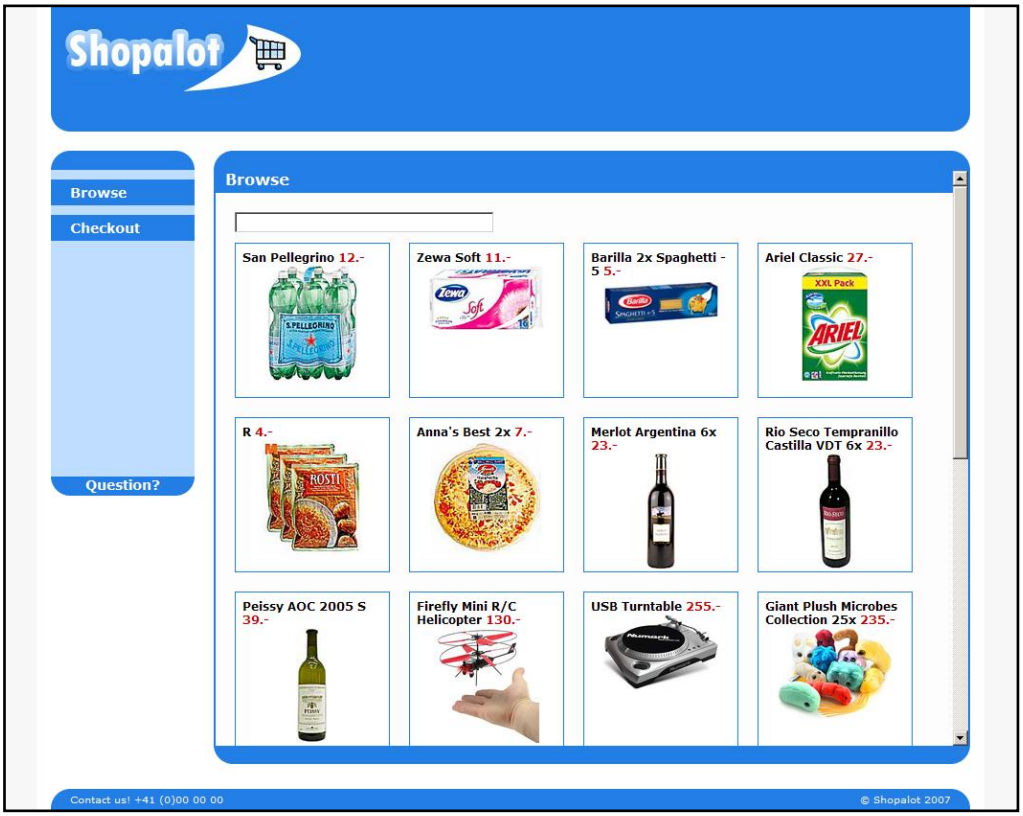
```
function submit(){
 var myForm = dojo.byId("myForm");
 var firstname = dojo.widget.byId("firstname");
 var lastname = dojo.widget.byId("lastname");
 var email = dojo.widget.byId("email");

 if(!firstname.isMissing() & !lastname.isMissing() & email.isValid()){
  myForm.submit();
 } else{
  alert("There are some required fields missing!");
 }
}
```

Copiez le fichier showPost.php dans votre dossier de travail, si tout est en ordre. Le formulaire envoie ses informations au fichier PHP showpost.php par méthode POST. Celui-ci affiche uniquement les valeurs qu'il reçoit par la méthode POST, à vous d'étendre ses fonctionnalités comme vous le désirez ou de rediriger le formulaire sur un autre script. Toutefois, n'oubliez pas qu'il est toujours une bonne idée de revalider le contenu au niveau du serveur, afin d'éviter des tentatives malicieuses d'injecter du code.

## Etape 4: Communication avec le serveur

### Autocomplete avec un serveur

Maintenant, il est temps d'afficher les produits que nous voulons vendre:



Dans cette section nous allons nous intéresser à la base de données. Chargez le script-requête shopalot.sql (attention celui-ci est au format «utf8», cependant la table crée ne l'est pas)

Copiez également le dossier «items» qui contient les images des produits, ainsi que le script getitems.php. N'oubliez pas d'ajuster les informations de connexion à votre base de données dans ce script. Nous reviendrons sur son contenu ultérieurement.

Comme nous l'avons fait pour le fichier form.html, nous allons maintenant créer un fichier browse.html qui affiche la liste des produits que nous vendons.

```
<div id="myForm">
<h1>Browse</h1>
<form class="formFrame">
<input type="textbox" name="search" id="search" value=""
onkeyup="getItems(this.value);"/>
<ul id="catalog"></ul>
<br style="clear:both;"/>
</form>
```

Comme vous pouvez le constater, ce n'est qu'un simple formulaire avec un champ unique qui appelle une fonction lorsqu'un caractère est entré et lui renvoie son contenu. La deuxième chose à remarquer est la liste vide que nous appelons «catalog».

## Code qui connecte

Comme d'habitude vous pouvez compléter votre fichier shopalot.html ou directement regarder le résultat dans le fichier etape4.html

Nous allons créez une fonction «browse» comme nous l'avions fait pour «checkout».

```
function browse(){
 var contentPane = dojo.widget.byId("content");
 //populate list from db on load
 dojo.event.connect(contentPane, 'onLoad', getItems(''));
 contentPane.setUrl('browse.html');
}
```

Ajouter browse(); à note fonction init(e) pour que la page contenant la liste de produits s'affiche au chargement de la page.

Ce que «browse» fait de plus que «checkout», est de connecter la fonction «onLoad» avec une nouvelle fonction «getItems()» que nous allons créer. Cette fonction a pour but de charger la liste (catalog) de tous les éléments de la base de données lorsque la page browse.html a été chargée dans le «contentPane».

Avant d'aller plus loin dans les détails de la fonction, précisons une des actions que fait le script getitems.php. Si nous appelons ce script avec comme paramètre «action» la valeur «search» et comme second paramètre «keyword» avec une valeur représentant un mot-clé, celui-ci nous retourne les objets correspondant à ce mot-clé se trouvant dans note base de données. Le format dans lequel ces valeurs sont retournées est appelé JSON (JavaScript Object Notation) qui permettra de facilement retraiter ces données dans nos fonctions.

```
function getItems(keyword){
 var bindArgs = {
  url: "getitems.php",
  handler: updateCatalog,
  mimetype: "text/json",
  content: {"action":"search","keyword":keyword}
  };
  dojo.io.bind(bindArgs);
}
```

Ce trick magique est effectué par «*dojo.io.bind*» et des quelques paramètres que nous attribuons à cette fonction. Dans notre cas, le paramètre «*url*» contient le nom du fichier à contacter, «*content*» les paramètres à lui transmettre, tandis que «*mimetype*» spécifie le format de retour auquel nous nous attendons et «*handler*» la fonction à exécuter lorsqu'on reçoit une réponse.

Remarque. Ce que nous ne couvrons pas: Io permet aussi de soumettre un formulaire en asynchrone http://manual.dojotoolkit.org/WikiHome/DojoDotBook/Book108

Maintenant que le serveur nous répond, il faut créer la fonction «updateCatalog» que nous avons spécifié dans la communication. Par défaut, le connecteur «bind» va toujours nous retourner une variable «type» qui contient «load» ou «error» pour faire du traitement d'erreur, «data» qui contient le contenu de la réponse du serveur, en plus de l'événement standard de JavaScript.

```
//create catalog from results provided by the server
function updateCatalog(type, data, evt){
 var catalog = dojo.byId("catalog");
 //empty the list
 dojo.dom.removeChildren(catalog);
 if (data) {
  for(var i=0;i<data.length; i++){
   //create the item
   var newLi = document.createElement("li");
   newLi.innerHTML = '<h2 style="font-size: 12px;margin:0;padding:
2px;text-align:left">'+data[i].name+' <span
style="color:#c00;">'+data[i].price+'.-</span></h2><img
src="items/'+data[i].id+'.jpg" alt="'+data[i].name+'"/>';
   newLi.id = data[i].id;
   newLi.style.listStyleType="none";
   //add the item
   catalog.appendChild(newLi);
  }
 }else{
  //case not result add an item as message
  var newLi2 = document.createElement("li");
  newLi2.innerHTML = '<h2 style="font-size: 12px;margin:0;padding:
2px;text-align:left; color:#c00;">Not matching item found!</h2>';
  newLi2.style.listStyleType="none";
  catalog.appendChild(newLi2);
  }
}
```

Même si cette fonction à l'air complexe, tout ce qu'elle fait est de supprimer l'entier des éléments de la liste catalog, pour ensuite y ajouter des nouveaux éléments créés d'après la

réponse du serveur. Si le serveur n'a rien trouvé, un élément contenant un message d'erreur est ajouté à la liste.

Presque tout le code est du JavaScript standard, notons juste la présence d'une fonction d'aide DOJO pour facilement ôter l'ensemble des éléments d'une liste, «*dojo.dom.removeChildren*». Nous ne pouvons donc que vivement recommander que vous jetiez un œil aux différents utilitaires de traitement des nœuds «DOM» que propose DOJO.

## Etape 5: drag & drop et communication avancée

### Drag & Drop

A ce stade, le client peut voir les produits, mais il ne dispose pas encore de caddy pour stocker ses achats.

Rajoutons donc ces trois lignes dans le menu de notre fichier shopalot.html pour avoir la version finale de ce document.

```
<div id="menu_content">
... ...
 <img src="images/cart.png" alt="cart" id="cart" />
 <ul id="cartList"></ul>
 <div id="totalRow">Total: <span id="total">0.00</span></div>
</div>
```

Tout ce qu'il nous faut maintenant concevoir, est le fait que le client puisse glisser & déplacer les produits dans son caddy pour que ceux-ci y soient ajoutés.

Si vous le voulez, vous pouvez continuer à ajouter le code dans votre document shopalot.html, ou, pour mieux séparer code et mise en page, collez la totalité du code JavaScript dans un nouveau fichier jscode.js et ensuite ajoutez une balise «<script>» dans votre fichier HTML afin qu'il charge jscode.js.

<script type="text/JavaScript" src="jscode.js"></script>

La première chose que nous allons faire est de rendre mobiles les différents produits que nous ajoutons à la liste «catalog» dans «*updateCatalotog*». Pour cela, ajoutez ce bout de code après «*catalog.appendChild(newLi)*».

```
//make the item draggable
var dnd = new dojo.dnd.HtmlDragCopySource(newLi, "*", true);
//add a tooltip to the item
var toolTip = dojo.widget.createWidget("Tooltip",
{caption:data[i].description,connectId:data[i].id,showDelay:100});
document.body.appendChild(toolTip.domNode);
```

D'une part, nous rendons le nouvel objet créé déplaçable de catégorie * car nous pouvons isoler les différents objets déplaçables dans des catégories (ce que nous n'utilisons pas), et

spécifions qu'il ne peut y avoir de copie de l'élément (sans importance dans note cas vu que nous détruisons l'objet une fois qu'il est lâché sur le caddy).

D'autre part, nous profitons d'ajouter une bulle d'aide affichant la description du produit. Ceci surtout pour montrer la capacité de DOJO à créer des «widgets» en code aussi bien qu'en HTML.

Maintenant que les produits sont déplaçables, il faut encore que nous puissions les déposer dans l'image du caddy. Dans notre fonction init(e), il faut encore y ajouter ces lignes de code.

```
//drag & drop: drop target
var cart = dojo.byId("cart");
var target = new dojo.dnd.HtmlDropTarget(cart, ["*"]);
//redefine onDrop function of our target
target.onDrop = function(e){
  //desactivate drag object
  this.onDragOut(e);
  var id = e.dragSource.dragObject.id;
  addItemToCart(id);
};
```

Nous créons un objet DOJO «DropTarget» avec «cart» qui est le nom de l'image du caddy que nous avons ajouté juste auparavant. Mais vu que nous ne souhaitions pas utiliser l'action par défaut, lorsqu'un produit est lâché dessus, nous redéfinissons la fonction «onDrop» pour qu'elle récupère l'id de l'élément lâché et le transmette à note fonction «*addItemToCart*».

## Communication

Voici notre dernière fonction pour que le puzzle soit au complet.

```
function addItemToCart(itemId) {
 var bindArgs = {
  url: "getitems.php",
  handler: function(type, data, evt){
      var total = dojo.byId("total");
     if (data) {
      //update total
      total.innerHTML =  (parseFloat(data.price) +
parseFloat(total.innerHTML)).toFixed(2);
      //create new item from the data
      var newLi = document.createElement("li");
      newLi.innerHTML = dojo.string.summary(data.name,15);
      var cartList = dojo.byId("cartList");
      //keep the list small
      if(cartList.getElementsByTagName("li").length >=5){
       dojo.dom.removeNode(cartList.firstChild);
      }
      //add item to the list
      cartList.appendChild(newLi);
     }
      },
  mimetype: "text/json",
    content: {"action":"additem","itemid":itemId}
   };
  //make the connection
  dojo.io.bind(bindArgs);
}
```

Avant d'expliquer ce que fait cette fonction, il faut rapidement introduire deux fonctions supplémentaires que propose note script getitems.php. Si on l'appelle avec comme paramètre «action» et valeur «additem», celui-ci attend un deuxième paramètre «itemid» qui aura comme valeur le nom du produit à ajouter au caddy. Il cherchera ce produit dans la base de données, l'ajoute dans le caddy (au niveau de la session PHP) et ensuite répond avec les informations sur l'objet en question. (C'est ici que l'on pourrait améliorer le tutorial et par exemple directement retourner le caddy que l'on réaffichera).

La deuxième fonction à expliquer et que si nous appelons le script sans paramètre, celui-ci nous retourne une liste en HTML avec le contenu du caddy.

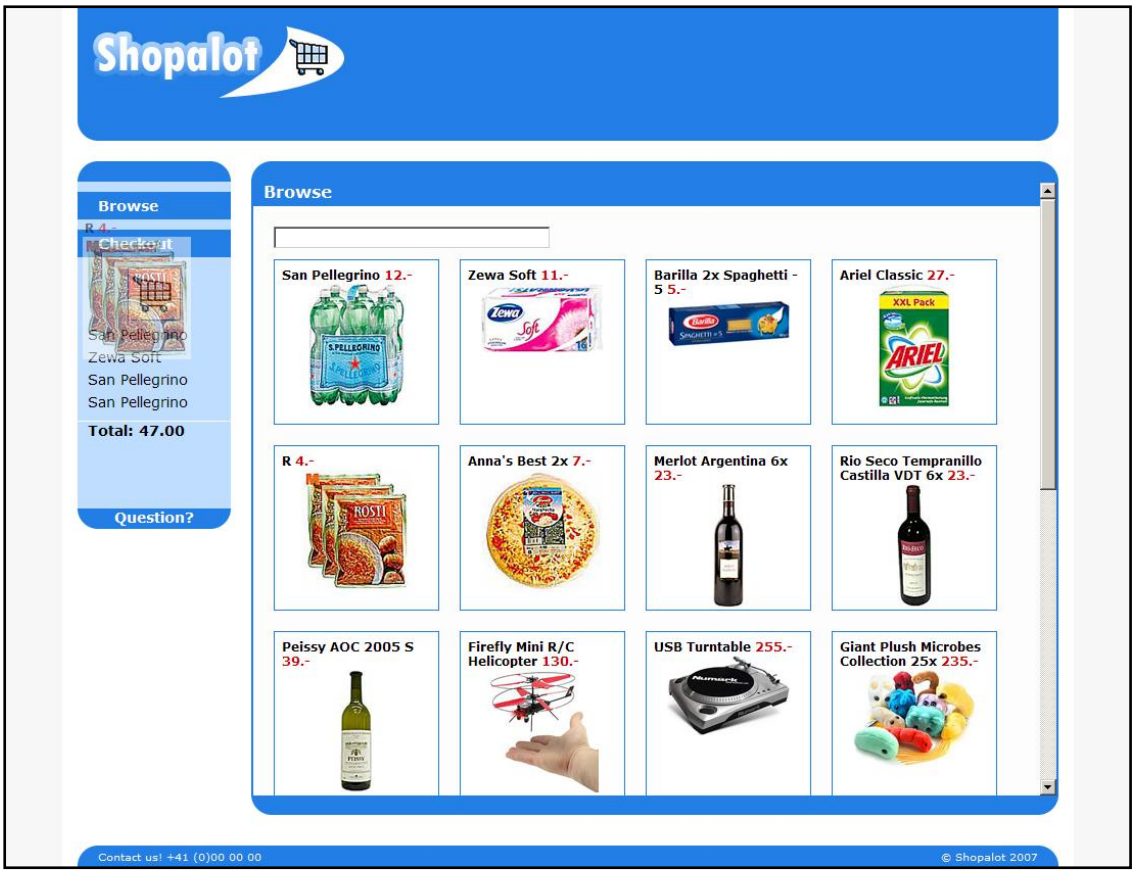
Pour utiliser cette option, nous devons ajouter cette ligne dans notre page form.html

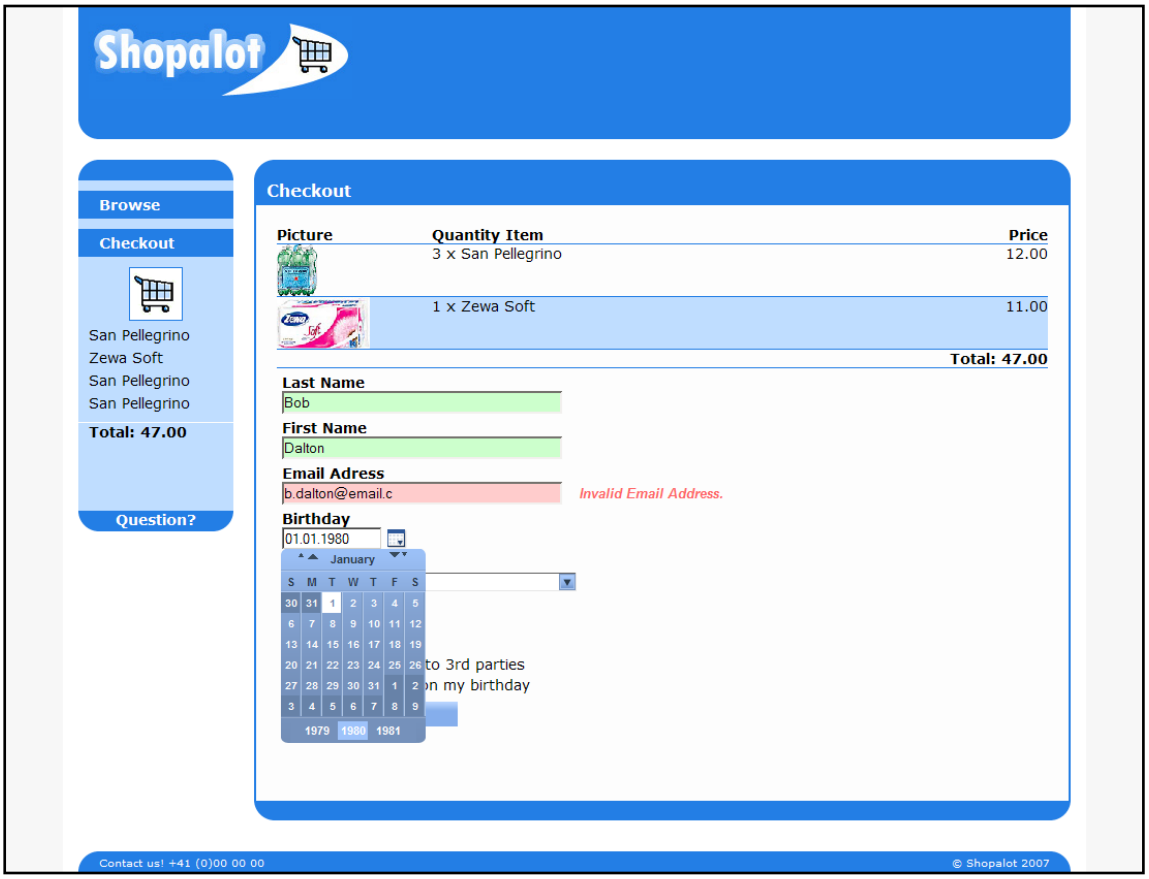
 

Pour ce qui est de la fonction «addItemToCart» l'explication est quasi identique à celle de la fonction «getItems». La différence majeure est au niveau de l'attribut «*handler*» qui ne contient pas un nom d'une fonction à exécuter, mais directement la fonction. Celle-ci met à jour le total, ainsi que la liste des cinq derniers produits ajoutés.

Nous pouvons voir, qu'en plus des fonctions d'aide DOM, DOJO propose aussi des fonctions d'aide pour la manipulation de texte (String) comme «*dojo.string.summary*» qui nous aide à limiter le nombre de caractères à 15 maximum.

Voici le résultat final de notre shop!


Shopalot
Browse
Checkout
San Pellegrino
Zewa Soft
San Pellegrino
San Pellegrino
Total: 47.00
Question?
Browse
San Pellegrino 12.-
Zewa Soft 11.-
Barilla 2x Spaghetti - 5 5.-
Ariel Classic 27.-
R 4.-
Anna's Best 2x 7.-
Merlot Argentina 6x 23.-
Rio Seco Tempranillo Castilla VDT 6x 23.-
Peissy AOC 2005 S 39.-
Firefly Mini R/C Helicopter 130.-
USB Turntable 255.-
Giant Plush Microbes Collection 25x 235.-
Contact us! +41 (0)00 00 00
© Shopalot 2007


Shopalot
Browse
Checkout
San Pellegrino
Zewa Soft
San Pellegrino
San Pellegrino
Total: 47.00
Question?
Checkout
Picture
Quantity Item
Price
3 x San Pellegrino
12.00
1 x Zewa Soft
11.00
Total: 47.00
Last Name
Bob
First Name
Dalton
Email Adress
b.dalton@email.c
Invalid Email Address.
Birthday
01.01.1980
January
to 3rd parties
n my birthday
Contact us! +41 (0)00 00 00
© Shopalot 2007

## Conclusion

Maintenant c'est à vous d'aller plus loin en vous aidant de l'API et du manuel. Tous les exemples disponibles dans les répertoires «demos» et «test» du toolkit peuvent être une grande source d'inspiration!

Pour quelques pistes supplémentaires, allez voir par exemple dans le manuel la section création de vos propres «widgets», ou encore essayez d'ajouter des fonctions supplémentaires à notre application, comme par exemple des catégories pour les produits, un historique de commandes, gestion de stock, etc.

N'oubliez pas que la manière la plus simple d'apprendre est d'essayer!

## Ressources

Le tookit Dojo http://dojotoolkit.org

L'API http://dojotoolkit.org/api/

Manuel http://manual.dojotoolkit.org/WikiHome/DojoDotBook

Navigateur firefox http://www.mozilla.com/firefox/

Extension firefox pour debugger facilement http://getfirebug.com/

IDE eclipse http://www.eclipse.org/

Addon AJAX Toolkit Framework pour eclipse http://www.eclipse.org/atf/

## Références

(openweb.eu.org) http://openweb.eu.org/articles/objet_xmlhttprequest/

(java.net) https://d2.dev.java.net/doc/introduction.html

(adaptivepath) http://www.adaptivepath.com/publications/essays/archives/000385.php

http://dojo.jot.com/Tutorials/HelloWorld

http://today.java.net/pub/a/today/2006/04/27/building-ajax-with-dojo-and-json.html

Source des images : http://www.leshop.ch/, http://www.thinkgeek.com, http://www.amazon.com