TD 8 : Formulaires

2 décembre 2015

Objectif: Dans ce TD, nous allons introduire les formulaires en HTML. Pour pouvoir les utiliser on abordera aussi les plug-ins jQuery (en particulier jQuery UI) ainsi que l'API WebStorage.

1 Formulaires HTML

Les formulaires HTML sont utilisés pour collecter les entrées utilisateur.

Un formulaire HTML est caractérisé par une balise `<form>` et possède un certain nombre d'éléments, de plusieurs types. Dans la suite on va étudier les deux types les plus courantes : `<input>` et `<select>`.

1.1 Input

L'élément de formulaire la plus importante est `<input>`, utilisé pour entrer des données. Son attribut le plus important est `type` qui permet de définir sa fonctionalité.

Par exemple :

- `<input type="text">` crée un champ de texte (au plus 20 caractères) ;
- `<input type="number">` crée un champ pour entrer un nombre ;
- `<input type="number">` crée un champ pour entrer un adresse email ;
- `<input type="number">` crée un champ pour entrer un mot de passe (les caractères sont masqués) ;
- `<input type="radio">` crée un case d'option, qui permet à ul'utilisateur de sélectionner l'un d'un nombre limité de choix (une seule) ;
- `<input type="checkbox">` crée une case à cocher, qui permet à ul'utilisateur de sélectionner l'un d'un nombre limité de choix (plusieurs choix sont possibles) ;
- `<input type="button">` crée un bouton sur lequel on peut cliquer (par exemple, pour activer un script) [1] ;
- `<input type="submit">` définit un bouton pour soumettre le formulaire à un gestionnaire des formulaires, l'adresse duquel est donné comme valeur à l'attribut `action` de `<form>` (voir ci-dessous) ;
- `<input type="image" src="monImage.png">` comme `submit` mais utilisant une image à la place du bouton ;
- `<input type="reset">` définit un bouton pour réinitialiser toutes les valeurs du formulaire ;

1. En gros, il a le même comportement que l'élément `<button>` que vous avez déjà rencontré.

– etc.

Par exemple :

```
<form action="javascript:maFonctionPourLeFormulaire()">
  Prénom: <br>
  <input type="text" name="prenom" value="Moumin"><br> // La valeur par défaut
      sera "Moumin".
  Nom:<br>
  <input type="text" name="nom" value="Peikko"><br> // La valeur par défaut sera "
      Peikko".
  <input type="radio" name="sexe" value="homme" checked>Homme<br>  // La case sera
       cochée par défaut.
  <input type="radio" name="sexe" value="femme">Femme<br>
  <br>
  <input type="submit" value="Soumettre">
</form>
```

Un gestionnaire de formulaires est généralement une page sur un server avec un script pour le traitement des données d'entrée. Le format le plus utilisé est le `.php`.

L'adresse de cette page est spécifié dans l'attribut `action` de l'élément `<form>`, par exemple `<form action="maPageDynamique.php" method="post">`. Dans ce cours on ne vas pas apprendre comment un serveur traite des données, c'est-à-dire, vous n'allez pas envoyer les donnée du formulaire à un script PHP, mais les utiliser "en locale" à l'aide de JavaScript : avec un bouton ayant l'attribut `onclick` ou en précisant comme adresse de `action` une fonction du script (par exemple, `<form action="javascript:maFonctionAMoi()">`).

1.2 Select

Un autre élément du formulaire est `<select>`. Celui-ci définit une liste déroulante. Les options possibles sont contenus dans l'élément `<option>`.

Par exemple :

```
<form>
  Mon caractère Moomin favori:<br>
  <select name="caractere">
    <option value="moomin">Moomin</option>
    <option value="shuka">La demoiselle Shuka</option>
    <option value="pipo">Pipo</option>
    <option value="mie">Joliemie</option>
  </select>
</form>
```

▸ **Exercice 1:** Créez un formulaire sur votre page HTML. Celui-ci devra permettre à l'utilisateur de saisir :

- son nom ;
- son prénom ;
- son titre de civilité (une seule choix entre « M. », « Mme » et « Mlle ») ;
- son adresse e-mail ;
- sa date de naissance ;
- son métier ;
- sa nationalité (liste déroulante) ;
- la façon dont il a entendu parler du site, lui proposant un certain nombre de réponses possibles (plusieurs choix possibles) ;

– une case d'option lui demandant s'il désire recevoir une newsletter.

Comme pour tous les éléments HTML, il est possible accéder à la valeur d'un élément dans un formulaire en JavaScript en utilisant (par exemple) son identifiant.
Par exemple, si dans un `<form>` on a défini l'`input` suivant :

```
Mot de passe : <input type="password" name="motDePasse" id="monInputSecret">
```

on peut accéder à l'`input` avec le code :

```
document.getElementById("monDiv").innerHTML = document.getElementById("
    monInputSecret").value;
```

ou, en utilisant jQuery :

```
$("#monDiv").html($("#monInputSecret").val());
```

▸ **Exercice 2:** En utilisant le formulaire de l'exercice précédent, affichez sur la même page un titre de niveau 1 centré contenant le titre de civilité, le prénom et le nom que l'utilisateur a saisi. Si l'utilisauter est un homme, le titre sera coloré en rose et s'il s'agit d'une femme le titre sera bleu.

2 Plugins jQuery: jQuery UI

N'importe quel développeur peut proposer ses modules à la communauté : c'est ce qu'on appelle un plug-in.
Le gros avantage des plug-ins, c'est qu'ils nous permettent d'utiliser autant de composants externes que l'on souhaite, pour nous éviter de coder quelque chose que quelqu'un aurait déjà fait ailleurs (et sûrement en mieux, sans offense).
Outre le fait que les plug-ins nous permettent de créer des fonctionnalités complètes en quelques lignes, ceux-ci sont en général bien documentés et entièrement personnalisables.
Comme vous pouvez le constater, vous allez réaliser en une ligne ce qui vous auriez mis en place en deux semaines. Vous ne ferez plus jamais votre lessive à la main, et vous allez adorer les plug-ins jQuery.

Grâce à une communauté de développeurs très active, un ensemble de plug-ins orientés « graphique » a été créé. Il s'agit de **jQuery UI** (pour « User Interface »).
Vous pourrez y trouver des fonctionnalités de drag & drop, des nouveaux effets de transitions ou encore des widgets divers et variés (barre de progression, onglets, sélecteur de couleur...).
Commencez par télécharger jQuery UI. Son contenu étant varié, il est possible de le télécharger module par module, afin d'éviter de charger trop de données inutiles sur sa page web.
Ici, nous n'aurons par exemple pas besoin de tous les nouveaux « effets » proposés dans UI.

Pour commencer, allez sur la site

```
http://jqueryui.com/
```

Sur la site, choisissez « Custom Download ».
Vous êtez alors rédiges sur une nouvelle page « Download Builder ».
Décochez tous (« Toggle All » sous « Components »).
Ensuite, cochez « Toggle All » de « UI Core », et « Autocomplete » et « Datepicker » de « Widgets ».

Parcourez les fichiers téléchargés. En plus des CSS, JS et images nécessaires au fonctionnement des plug-ins, vous avez à votre disposition une documentation complète.

Dans ce TD, vous allez utiliser les fichiers `jquery-ui.js` et `jquery-ui.css`.

Joignez-les à votre page HTML (attention à l'ordre d'inclusion). Vous êtes maintenant prêts à utiliser tous les plug-ins UI à votre disposition !

2.1 `DatePicker`: Sélecteur de date

Dans jQuery UI vous trouverez `DatePicker`, un widget qui permet d'afficher un calendrier qui évite à l'utilisateur de saisir une date entièrement.

Le `DatePicker` peut être lié à un champ de texte standard dans un formulaire. Quand vous cliquez sur le champ de texte, le calendrier interactif s'ouvre, et quand la date est choisie le feedback est affichée sur le champ de texte.

Pour ajouter un `DatePicker` à un élément HTML, vous faites comme l'exemple suivant :

```
$( "#monElement").datepicker({
  // liste des options sépares par virgules
});
```

Une liste d'options peut être, par exemple :

```
changeMonth: true,
changeYear: true,
yearRange: "-10:+10",
maxDate:"+2M",
```

Dans l'exemple précédente, l'option « `changeMonth` » affiche dans le calendrier une liste déroulante qui permet à l'utilisateur de choisir le mois directement, tandis que « `changeYear` » fait le même chose, mais pour l'année. « `yearRange` » montre les années permis par rapport à l'année actuelle, et « `maxDate` » accepte une chaine de caractères, par exemple `"+2M+2Y"`. Utilisez « `D` » pour le jours, « `W` » pour les semaines, « `M` » pour les mois et « `Y` » pour les années.

Pour une liste d'options complète, voir la documentation à :

```
http://api.jqueryui.com/datepicker/
```

▸ **Exercice 3:** Dans vôtre formulaire, ajoutez un `DatePicker` au champ « `Date de naissance` ». Cette date doit pouvoir commencer à l'année 1900, mais ne doit pas dépasser la date d'aujourdhui.

Prendrez soin à écrire la date en format français: jour/mois/année (pensez à utiliser l'option « `dateFormat` »).

2.2 Autocomplétion

Qui n'a jamais joué avec Google Suggests ? Vous tapez quelques lettres dans le champ de recherche, et plusieurs termes vous sont proposés. Ce procédé s'appelle **auto-complétion**.

Il est assez utilisé dans les formulaires... et est possible grâce à jQuery UI !

Supposons, par exemple, de vouloir simplifier la saisie du métier de l'utilisateur dans le formulaire de l'Exercice 1. Pour cela, nous allons établir une petite « base de données » des métiers

les plus courants, et lui proposer de compléter le champ de texte s'il commence à écrire le nom d'un métier prédéfini.

Un exemple simple est le suivant.

```
$("#langages").autocomplete({
  source: [ "c++", "java", "php", "coldfusion", "javascript", "asp", "ruby" ]
});
```

Notez que dans l'exemple ci-dessus, si on écrit un lettre dans l'élément avec ID `monID`, l'autocompletion va nous donner toutes les langages qui contiennent ce lettre. Donc en tapant « `c` », on va obtenir comme suggestions « `coldfusion` », « `c++` » et « `javascript` ».

Pour une liste d'options complète, voir la documentation à :

```
http://api.jqueryui.com/autocomplete/
```

▸ **Exercice 4:** Modifiez le champs « métier » de votre formulaire pour qu'il permet l'autocomplétion. Pour avoir quelque chose d'utile, definissez d'abord 10 métiers dans un array `source`.

3 Web Storage

Le **Web Storage** permet au navigateur de stocker des informations de façon locale, sans avoir à communiquer avec un quelconque serveur. Il s'agit-là d'une grande nouveauté, introduite par HTML5.

En effet, jusqu'alors, il était compliqué de stocker des informations complexes. L'utilisation des cookies, par exemple, n'était pas idéale : avec une taille limitée, ils n'étaient pas adaptés au stockage local. La solution la plus pratique était de stocker ces données en ligne, grâce à des appels AJAX : pratique lourde en ressources !

Désormais, le *Web Storage* met en place deux espaces de stockage. Le premier, *Session Storage*, est destiné à la mémorisation de données ayant une faible durée de vie : celles-ci seront supprimées à la fermeture du navigateur. *Local Storage*, quant à lui, bénéficie d'une durée de vie plus longue (les données ne sont pas supprimées à la fermeture du navigateur et restent disponibles (théoriquement) pour toujours) et possède une portée étendue à toutes les pages d'un même domaine. Au sein d'un même site, on peut donc mettre en place un vrai espace de stockage.

En plus, contrairement aux cookies, la limite de stockage est assez grande (au moins 5 Mb) et les informations ne sont jamais trasnférés vers le serveur.

Comme tous les navigateurs ne supportent pas le nouvelles fonctionnalités d'HTML5, la première chose à faire est de vérifier que le navigateur supporte le *Web Storage* :

```
if(typeof(Storage) !== "undefined") {
    // Code pour le Web Storage.
} else {
    alert("Désolé, mais le Web Storage n'est pas suppoté");
}
```

Pour stocker nos données, on utilisera l'objet `localStorage` pour le *Local Storage* et l'objet `sessionStorage` pour le *Session Storage*. La syntaxe est la même pour les deux.

Les méthodes « `setItem()` » and « `getItem()` » nous permettent respectivement de stocker et de récupérer le contenu de l'objet.

Dans la première on spécifie le nom et la valeur de la donnée (`.setItem("nom","valeur")`), tandis que `getItem` nécessite seulement du nom (la syntaxe est `.getItem("nom")`).

Par exemple :

```
localStorage.setItem("ville", "Rovaniemi");
$("#monDiv").html(localStorage.getItem("ville"));
```

Une syntaxe alternative pour stocker et récuperer les données sans utiliser `setItem` et `getItem` est la suivante :

```
localStorage.nom = "valeur";
```

Par exemple :

```
sessionStorage.ville = "Rovaniemi";
sessionStorage.nom = "Père Noel";

$("#monDiv").html(sessionStorage.nom + " habite à " + sessionStorage.ville);
}
```

Remarquez que les noms et les valeurs sont toujours stockés comme des chaînes de caractères.

▸ **Exercice 5:** Créez un compteur de visites d'une page web. Vous devrez afficher le nombre de venues (c'est-à-dire le nombre d'actualisations de la page) de l'utilisateur. Pensez à utiliser `Number()` pour convertir l'objet en nombre.

Le résultat est-il le même si on utilise `localStorage` ou `sessionStorage` ?

▸ **Exercice 6:** À l'aide d'un formulaire, demandez à l'utilisateur la première fois qu'il entre dans votre site (et que la première fois), son titre de civilité, son prénom et son nom. Modifiez le message de l'exercice précédente pour afficher aussi ces informations (en affichant, par exemple, " Bonjour M. Matti Meikäläinen, vous avez visité cette page 42 fois.").

Même si en principe le *Web Storage* permet de stocker uniquement des chaînes de caractères, avec une petite astuce on peut mémoriser des objets entiers. Pour cela, nous allons utiliser la syntaxe JSON, en transformant des objets en chaîne de caractères, grâce à la méthode `JSON.stringify()`, pour après la ré-parser, grâce à `JSON.parse()`.

Par exemple :

```
// Construction d'un objet JSON
var album = {}

// Remplissage de l'objet
album.artiste = 'Carola';
album.titre = 'I denna natt blir varlden ny';
album.annee = '2007';
album.chansons = [ { "titre" : "Stilla natt, heliga natt" }, { "titre" : "Lyss
    till anglasangens ord" }, { "titre" : "Glans over sjo och strand" }, { "titre"
     : "I denna natt blir varlden ny" } ];

// On place l'objet en mémoire en transformant le JSON en chaine de caracteres
localStorage.album = JSON.stringify(album);

// Pour le recuperer, on transforme la chaine en JSON
var recupalbum = JSON.parse(localStorage.album);
```

▸ **Exercice 7: (à rendre sur la plateforme avant le 8 décembre)** En vous appuyant sur cette technique, vous allez créer une application web de « to-do list ».

Les tâches à réaliser seront mémorisées dans le navigateur (en *Local Storage*) et listées sur la page, accompagnées d'un formulaire permettant à l'utilisateur d'en ajouter.

- Commencez par créer un formulaire pour l'ajout de tâche. Il sera composé du titre de la tâche, de la date à respecter pour sa réalisation (utilisez un *datePicker*), d'une description plus complète (utilisez un `<textarea>`) et d'un état d'avancement (0-100%) (utilisez un `<input>` de type `range`).
- Traitez ce formulaire grâce à jQuery. À la soumission de ce dernier, récupérez-en les valeurs pour les stocker dans le *Local Storage*, sous forme d'objets JSON.
- Listez les tâches déjà ajoutées. Vous pourrez utiliser du CSS3 pour un rendu en forme de post-its, par exemple. Et pourquoi pas des animations au survol ?
- Faites en sorte de pouvoir modifier l'état d'avancement d'une tâche en cliquant sur un bouton. Une fois effectuée, son affichage devra être modifié (contenu barré ? Changement de couleur ?)
- Ajoutez un bouton pour chaque tâche à fin de la supprimer.