Composant jQuery EasyAutocomplete

1. Présentation

EasyAutocomplete est un composant permettant d'associer à un champ de type texte une zone d'auto-complétions.

Url	http://easyautocomplete.com/
Documentation	http://easyautocomplete.com/guide
Download	http://easyautocomplete.com/download

2. Chargement des ressources – balise <head>

```
<link rel="stylesheet"
      href="//cdnjs.cloudflare.com/ajax/libs/easy-autocomplete/1.3.5/easy-
autocomplete.min.css">
<script src="//cdnjs.cloudflare.com/ajax/libs/easy-autocomplete/1.3.5/jquery.easy-
autocomplete.min.js"></script>
```

```
<link rel="stylesheet" href="//cdnjs.cloudflare.com/ajax/libs/easy-autocomplete/1.3.5/easy-
autocomplete.themes.min.css">
```

3. Mise en place

Il faut associer le composant au champ de type texte

```
let $nom = $("#nom")
$nom.easyAutocomplete(options);
```

option est un objet javascript contenant un ensemble de propriétés permettant de définir la source de données. Au minimum il faut définir la propriété **data** ou url pour indiquer la source de données qui va servir à alimenter la zone d'autocomplétion.

```
let options = { data: ["valeur 1", "valeur 2", … , "Valeur n]};
```

```
let options = { url : "ajax/ville.json", getValue: "nom",};
```

Si le champ est placé dans une fenêtre modale, la taille du champ doit être définie dans l'attribut style

```
<input id="nomPrenom"
        style = "width: 250px"
       type="text"
       placeholder="Nom du membre"
      class="form-control">
```

4. Les principales propriétés

Propriété	Rôle	Valeur
data:	Permet de définir la source des données stockées en mémoire (tableau javascript)	['rouge', 'vert' …]
url:	Permet de définir une source externe de données : un fichier json ou un script php contenant une requête Sql retournant des donnant au format Json.	"ajax/fichier.json"
getvalue	Permet d'indiquer la propriété utilisée pour l'auto-complétions lorsque la source de données contient des objets ayant plusieurs propriétés. Peut correspondre à une fonction si la donnée à afficher se compose de plusieurs propriétés de la liste (le nom et le prénom par exemple)	"nomPrenom" **function**(element) { **return** element.**nom** + **' '** + element.**prenom**; }
list	Contient un ensemble de propriétés permettant de paramétrer le fonctionnement du composant	

Liste complète : http://easyautocomplete.com/guide

5. Les propriétés de la propriété list

Propriété	Rôle
showAnimation	Contient un ensemble de propriétés permettant de paramétrer l'animation à l'ouverture à la fermeture de la zone d'auto-complétion.
hideAnimation	Contient un ensemble de propriétés permettant de paramétrer l'animation à la fermeture de la zone d'auto-complétion.
Exemple	*list: { showAnimation: {* *type: "fade", //normal|slide|fade* *time: 400, callback: function() {}* *},* *hideAnimation: {* *type: "slide", //normal|slide|fade* *time: 400, callback: function() {}* *}* *}*
match	Permet de limiter les valeurs affichées à celles qui correspondent à la valeur saisie.
Exemple	*list: { match: {enabled: true} }*
maxNumberOfElements	Définit le nombre maximum de valeurs affichées
Exemple	*list: { maxNumberOfElements: 10}*
onSelectItemEvent	Action réalisée sur le survol d'une valeur de la liste
onChooseEvent	Action réalisée lorsque l'utilisateur clique sur une valeur ou enfonce la touche entrée **dans la liste**
onClickEvent	Action réalisée lorsque l'utilisateur clique sur une valeur
onKeyEnterEvent	Action réalisée lorsque l'utilisateur enfonce la touche entrée **dans la liste**
onShowListEvent	Action réalisée lorsque la liste s'affiche
onLoadEvent	Action réalisée lorsque la liste se charge. Permet à l'aide de la méthode getItems() de récupérer les valeurs de la liste et par exemple de les comptabiliser
Exemple	*list: { onChooseEvent: function () {* *let id = $nomPrenom.getSelectedItemData().id;* *afficher(id);* *},*

Liste des événements : http://easyautocomplete.com/guide#sec-trigger-event
Attention : La mise en place de ce composant inhibe l'événement keypress du champ de saisie. Si l'on veut savoir si l'utilisateur a enfoncé la touche Entrée dans le champ de saisie (pas dans la liste associée) il faut utiliser l'événement keydown.
Source : https://github.com/pawelczak/EasyAutocomplete/issues/195

Petite astuce : Si vous souhaitez réduire la taille de la police utilisée dans la liste

Dans la feuille de style rechercher font-size et réduire la valeur

6. Les principales méthodes

Les méthodes doivent être appelées sur l'objet jQuery, qui représente le champ de texte en entrée.

Méthode	Rôle
getSelectedItemIndex()	Retourne l'index de l'élément sélectionné dans la liste affichée (pas dans l'ensemble des valeurs)
getSelectedItemData()	Retourne la valeur de l'élément sélectionné (chaîne, objet)
getItemData(index)	Retourne la valeur de l'élément spécifié par l'index
getItems()	Retourne un tableau des valeurs présentes dans la liste

7. Exemple

Exemple 1

Soit un champ 'nomPrenom' associé au système d'auto-complétion.
La source provient d'un fichier json contenant des objets avec les propriétés id, nom et prenom.
L'objectif est de récupérer l'id de la personne afin d'afficher les données de cette personne contenues dans la base.

```
let $nomPrenom = $("#nomPrenom")
let option = {
    data: "ajax/personne.json",
    getValue: function(element) { return element.nom + ' ' + element.prenom; },
    list: {
        match: { enabled: true },
        onChooseEvent: function() {
            let id = $nomPrenom.getSelectedItemData().id;
            rechercher(id);
        }
    }
}
$nomPrenom.easyAutocomplete(option);
```

Si on souhaite que la comparaison s'effectue uniquement à partir des premières lettres, il faut ajouter la propriété method dans les propriétés de match :

```
list: {
    match: {
            enabled: true ,
            method:  function(element, phrase) { return element.indexOf(phrase) === 0 ; }
    },
    …
}
```

Attention : Si la source de données provient d'une base de données, le champ utilisé pour la recherche ne doit pas contenir de valeur null sinon la méthode indexOf va échouer.

Exemple 2

Soit un champ ville associé au système d'autocomplétion.
La source provient d'un fichier 'ville.json' contenant des objets avec les propriétés id, nom et codePostal.
L'objectif est d'assister la saisie du champ ville et d'alimenter le champ 'codePostal' lorsque l'utilisateur sélectionne une valeur dans la liste

```
let $ville;
window.onload = init;

function init() {
   $ville = $('#ville');
   // mise en place de l'autocomplétion sur le champ ville
   let option = {
      url : "ajax/ville.json",
      // valeur alimentant la zone d'auto-complétion
      getValue: "nom",
      list: {
         match: {
            enabled: true,
            method: function (element, phrase) {
               return element.indexOf(phrase) === 0;
            }
         },
         onChooseEvent: function () {
            // récupération du code postal
            codePostal.value = $ville.getSelectedItemData().codePostal
         }
      }
   }
   $ville.easyAutocomplete(option);
}
```

Exemple 3

Soit un champ ville associé au système d'autocomplétion.
La source provient d'un fichier 'ville.json' contenant des objets avec les propriétés id, nom et codePostal.
L'objectif est d'assister la saisie du champ ville et d'alimenter une variable 'codePostal' lorsque l'utilisateur sélectionne une valeur dans la liste.
Cette fois ci, on veut obliger l'utilisateur à sélectionner une ville dans la liste.
On va donc :

Gérer l'affichage du message d'erreur dès qu'il n'y a plus de propositions correspondant à la saisie
Attribuer automatiquement la valeur au champ de saisie lorsqu'il ne reste plus qu'une proposition.

Pour le savoir on définit la propriété onLoadEvent qui contient dans une fonction anomyme les actions à réaliser à chaque fois que la liste est rafraichie. A ce moment la méthode getItems() permet de récupérer les éléments composant la liste. Il est alors possible de les comptabiliser (propriété length). S'il ne reste plus de valeur on peut afficher un message d'erreur, si n'en reste qu'une on peut alimenter automatiquement le champ de saisie et la variable codePostal. Il faut aussi initialiser la variable codePostal à " afin de détecter l'absence de sélection d'une valeur

```
let $ville;
window.onload = init;
function init() {
   codePostal = "
   $ville = $('#ville');
   let option = {
      url : "ajax/ville.json",
      getValue: "nom",
      list: {
         match: {
            enabled: true,
            method: function (element, phrase) { return element.indexOf(phrase) === 0;}
         },
         onChooseEvent: function () {
            codePostal = $ville.getSelectedItemData().codePostal;
            messageVille.innerHTML = '';
         },
         onLoadEvent: function () {
            // récupération des valeurs proposées
            let lesValeurs = $ville.getItems();
            // Si aucune valeur n'est proposée
            if (lesValeurs.length === 0) {
               codePostal = "";
               messageVille.innerHTML = "Vous devez sélectionner une ville dans la liste";
            } else {
               // s'il ne reste qu'une valeur on la valide automatiquement
               if (lesValeurs.length === 1) {
                  codePostal = lesValeurs[0].codePostal;
                  $ville.val(lesValeurs[0].nom);
                  // On donne le focus au champs suivant qui n'est pas encore renseigné
                  bntAjouter.focus();
               }
            }
         }
      }
   }
   $ville.easyAutocomplete(option);
}
```

8. Rafraichissement de la zone d'autocomplétion

Il peut arriver que la source de données du composant soit modifiée au cours du traitement.
Par exemple un système d'autocomplétion pour sélectionner une personne à supprimer. La zone d'autocomplétion ne doit plus afficher cette personne après la suppression.

A chaque fois que le composant affiche les lignes correspondant à la saisie de l'utilisateur, il interroge la source et donc la source est immédiatement mise à jour. **Le rafraichissement est donc automatique, il faut cependant effacer le contenu du champ de recherche pour l'obliger à rafraichir la liste.**
Cela va cependant engendrer par la suite des appel ajax répétés déclenchés à chaque saisie d'un caractère.
Si la source est un tableau, il faut penser à mettre à jour ce tableau en cas de modification.

Pour cela la solution consiste à rappeler la méthode easyAutocomplete.
Dans ce cas, il faut donc que la variable option soit globale.

Si plusieurs systèmes d'autocomplétion sont utilisés sur la page, on utilise plusieurs variables pour stocker les options.

9. Eléments avancés

Si la source de données au format json est complexe, il faut utiliser la propriété listLocation pour définir l'éléments dans lequel la liste prendra sa source

```
{
  - results: [
      - {
            ville: "Amiens",
            cp: 80090
        },
      - {
            ville: "Amiens",
            cp: 80000
        }
    ],
    nbr: 2
}
```

```
listLocation: "results",
getValue: "ville",
```

Il est possible d'ajouter dans la zone de liste à côté de la valeur une description (par exemple ici le code postal) à l'aide de la propriété template.

```
template: {  type: "description", fields: { description: "cp"  }},
```

EasyAutomplete peut prendre des données non seulement à partir de fichiers json / xml, mais peut également être connecté à une api. La propriété url prend alors la forme suivante :

```
url: function(phrase) { return url + phrase + "&format=json"; },
```

Le paramètre phrase contient la valeur saisie dans le champ associé au système d'autocomplétion ici le champ ville.