JS : Vérification d'un formulaire

Le JavaScript est un bon moyen d'effectuer une validation des données utilisateur avant l'envoi sur le serveur, c'est-à-dire côté client. Il faudra refaire une validation côté serveur, mais cela est une autre histoire dont vous ne vous occupez pas pour le moment.

RÉCUPERATION DES ÉLÉMENTS

L'attribut **`name`** est indispensable dans la validation d'un formulaire. C'est sur cet attribut que l'on va se baser pour récupérer nos éléments.

Nous allons mettre un attribut « **`name='nameForm'`** » à notre formulaire. Côté HTML on aura alors :

```html
<form name="nameForm">
```

Côté JavaScript, vous y accéderez comme ceci :

```js
var myForm = document.nameForm;
```

Il est aussi possible d'utiliser les sélecteurs vus précédemment comme **`getElementById`**, **`querySelector`**, etc.

Pour récupérer la valeur d'un champ (**`input`**, **`select`**, etc.) on va aussi se baser sur son attribut **`name`**. Donc, si on considère un champ avec l'attribut « **`name='nameInput'`** » on accède à sa valeur comme ceci :

```js
var myValue = document.nameForm.nameInput.value;
```

LES ÉVÈNEMENTS LIÉS AUX CHAMPS

Avant de valider notre formulaire, en plus des éléments déjà contrôlés grâce aux attributs HTML, on va vérifier certaines informations dynamiquement lors du remplissage des champs. Et avertir l'utilisateur en cas d'erreur. On va utiliser des écouteurs d'événements, rappelez-vous lors d'un cours précédent on a utilisé un événement au clique sur un élément.

Pour les formulaires on peut utiliser un événement au « focus » d'un champ. C'est-à-dire tant que le champ reste sélectionné. Avec la fonction « **`input`** » :

```js
nameInput.addEventListener("input", function(){
    console.log(this.value)
});
```

Ici on va donc vérifier la valeur du champ dans la console tant que celui-ci garde le « focus ».

Une autre possibilité est de vérifier cette valeur lorsque le champ perd le « focus ». Avec la fonction « **`change`** » :

```js
nameInput.addEventListener("change", function(){
    console.log(this.value)
});
```

LES EXPRESSIONS RÉGULIÈRES

Maintenant nous voulons contrôler plus en détail les données que l'utilisateur renseigne. Pour les champs dont on ne peut pas mettre d'attribut HTML, par exemple, qu'un champ contient uniquement des chiffres. Pour cela, on utilise les expressions régulières : **`regex`**.

Il faut alors déclarer une variable, dont la valeur est ce que l'on souhaite vérifier. On définit une expression régulière en plaçant son motif entre deux caractères « **`/`** ».

```
var regex = /@/; // la chaîne de caractère doit contenir le caractère @
```

Sa méthode **`test`** détecte la présence d'une correspondance avec le motif dans la chaîne de caractères passée en paramètre et renvoie **`true`** ou **`false`** selon le cas. Par exemple :

```
console.log(regex.test("adressemail.fr")); // renvoie false
console.log(regex.test("adresse@mail.fr")); // renvoie true
```

Voici une petite liste d'exemple de **motif** possible :

- **`abc`** La chaîne contient "abc"
- **`[abc]`** La chaîne contient soit "a", soit "b", soit "c"
- **`[a-z]`** La chaîne contient n'importe quelle lettre minuscule de l'alphabet
- **`[0-9]`** La chaîne contient un chiffre
- **`a.c`** La chaîne contient "a" suivi d'un caractère (n'importe lequel) suivi de "c"
- **`a\.c`** La chaîne contient "a.c"
- **`a.+c`** La chaîne contient "a" suivi d'un ou plusieurs caractères (n'importe lesquels) suivi de "c"
- **`a.*c`** La chaîne contient "a" suivi de zéro ou plusieurs caractères (n'importe lesquels) suivi de "c"

Il existe de nombreux autres types de motif possible.

Voici une explication de comment fonctionne ces motifs :

- Les crochets « **`[]`** » définissent un intervalle de caractères. Toute chaîne contenant au moins un caractère dans cet intervalle correspondra au motif. Les motifs **`[a-z]`** et **`[A-Z]`** permettent de rechercher n'importe quelle lettre de l'alphabet, respectivement en minuscules ou en majuscules.
- Le caractère point « **`.`** » permet de remplacer n'importe quel caractère.
- Le caractère « **`\`** » ("antislash" ou "backslash") indique que le caractère qui suit doit être recherché en tant que tel.
- Le caractère « **`+`** » permet de rechercher une ou plusieurs occurrences de l'expression qui le précède.
- Le caractère * permet de rechercher zéro ou plusieurs occurrences de l'expression qui le précède.

On peut donc obtenir un motif complexe comme : **`/.+@.+\..+/`**

Décodons ce motif :

- Commence par un ou plusieurs caractères (**`.+`**)
- Contient ensuite le caractère « **`@`** » (**`@`**)
- Contient ensuite un ou plusieurs caractères (**`.+`**)
- Contient ensuite le caractère « **`.`** »(**`\.`**)
- Finit par un ou plusieurs caractères (**`.+`**)

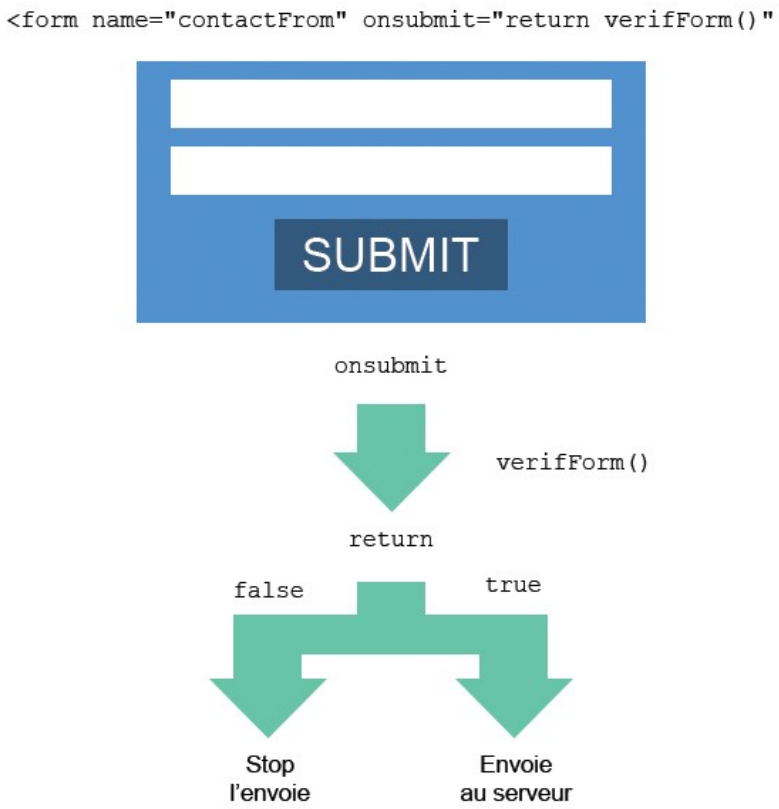
Il est donc pratique pour vérifier une adresse mail.

VÉRIFICATION DU FORMULAIRE

Au clique sur le bouton « envoyer » de notre formulaire, on peut faire une vérification de celui-ci, avant d'envoyer les données. On aura donc besoin d'un nouvel attribut **`onsubmit`** que l'on va placer directement sur notre formulaire. Du côté HTML on écrit alors :

```
<form name="contactForm" onsubmit="return verifForm()">
```

La fonction **`return`** va nous permettre de stopper l'envoi du formulaire si une information est inexacte ou une condition non remplie. Comme le montre ce schéma :



Ensuite, dans notre JavaScript on va créer la fonction de validation qui est appelée par notre attribut **`onsubmit`** :

```
function verifForm() {
   // instructions avant validation
}
```

C'est dans cette fonction qu'on mettra, lors d'une erreur rencontrée :

```
return false;
```

Dès que cette instruction est donnée, le formulaire va stopper l'envoi des informations.

Par exemple :

```
if(champObligatoires.value==""){
      return false;
}
```

MESSAGE DE VALIDATION

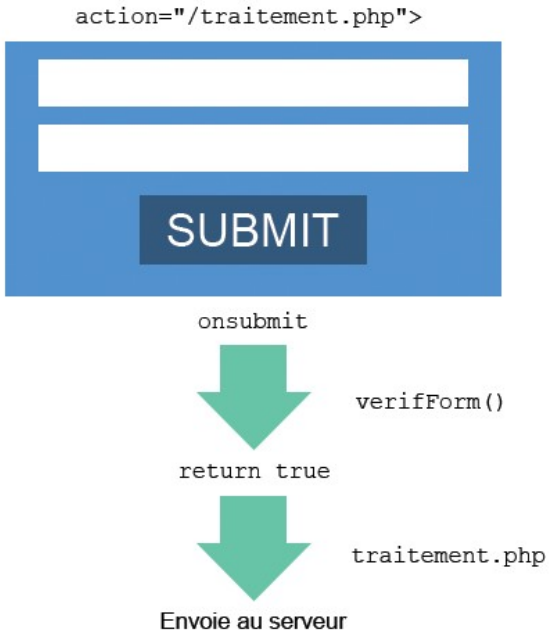
Tout comme pour les erreurs il est important de notifier à l'utilisateur que le formulaire a bien été rempli et envoyé. Pour cela on va ajouter dans notre fonction, une nouvelle condition avec la méthode `checkValidity()` par exemple :

```
if (contactForm.checkValidity()){
      // instructions d'affichage du message de validation
 }
```

TRAITEMENT DES DONNÉES

Pour le traitement des données et l'envoi au serveur, on va avoir besoin de rediriger vers une autre page. On aura besoin de l'attribut `action` qui donne le chemin du nouveau fichier (en pHp) qui traitera les données. Côté HTML on va donc rajouter :

```
<form name="contactForm" onsubmit="return verifForm()" action="/traitement.php">
```



<u>À noter</u> : Nous ne traiterons pas les données dans ce cours, ni lors de la formation. Vous pouvez donc vous arrêter à la méthode précédente ou alors la remplacer par celle-ci en appelant une page HTML à la place, avec votre message de validation.