Les formulaires

Table des matières

1. Les requêtes http

En HTML, un formulaire permet d'interagir avec le serveur en définissant une **méthode**. Les plus courantes sont **POST** et **GET**. Pour cela, on va définir un attribut **method** dans notre balise **<form>**.

L'attribut **action**, lui, définit la **destination** du formulaire. Dans notre exemple, le formulaire envoie ses données vers la page contact.php.

```
<form method="POST" action="contact.php">
```

Dans le formulaire, on peut avoir plusieurs types d'**input**, que l'on va ranger en deux catégories : Les inputs **texte**, et les inputs de **sélection**. Chaque input a un attribut **name** qui permet de définir le nom de la clé que l'on va retrouver dans le tableau **$_POST** ou **$_GET**.

Pour provoquer l'envoi du formulaire, il faut mettre une balise button dans le formulaire. La balise button doit avoir l'attribut **type** défini sur **submit**.

```
<button type="submit">Envoyer</button>
```

Un formulaire complet pourrait ressembler à ça :

```
<form method="POST" action="connexion.php">
    <input type="email" name="email" placeholder="E-mail"/>
    <input type="password" name="password" placeholder="Mot de passe"
minlength="10"/>
    <button type="submit">Envoyer</button>
</form>
```

2. Les inputs TEXTE

Pour définir le comportement de notre **input**, on va utiliser l'attribut **type**. On peut lui donner plusieurs valeurs :

- text
- password
- number
- email

et enfin la balise <textarea>

Voici un exemple :

```
<input type="text" name="prenom" placeholder="Prénom"/>
```

Les inputs de type texte comme **text**, **email**, **password**, peuvent aussi prendre des attributs **minlength** et **maxlength** pour contrôler la longueur de la chaîne de caractères admise. Exemple :

```
<input type="text" name="nom" placeholder="Nom" maxlength="10"/>
```

Pour les inputs de type **number**, on va plutôt utiliser les attributs **min** et **max**, comme ici :

```
<input type="number" name="note" min="0" max="20" placeholder="Note"/>
```

Enfin, pour les longs textes, on peut utiliser la balise <textarea> pour afficher un input large avec une barre de scroll.

```
<textarea name="mon_texte" rows="6" cols="50"></textarea>
```

3. Les inputs SELECT

Les input de type **select** permettent de forcer l'utilisateur à faire un choix parmi une **sélection donnée**. On en dénombre trois :

- Les <select>
- Les input de type radio
- Les input de type checkbox

Les <select> permettent de choisir une option parmi une liste. On lui donne des balises <option> avec un attribut **value** qui portera la valeur que l'on retrouvera dans **$_POST**. On l'écrit ainsi :

```
<select name="departement">
    <option value="">Choisissez votre département</option>
    <option value="42">Loire</option>
    <option value="03">Allier</option>
    <option value="63">Puy-de-Dôme</option>
    <option value="43">Haute-Loire</option>
    <option value="23">Creuse</option>
    <option value="38">Isère</option>
</select>
```

Les inputs de type radio permettent d'imposer un et **un seul** choix parmi un nombre défini. On donne à chaque input de type radio le même **name** pour que le navigateur soit capable de forcer le choix d'un seul élément. Voici la syntaxe :

```
Sexe : <br>
<input type="radio" name="sexe" value="H">
<label for="H">Homme</label><br>
<input type="radio" name="sexe" value="F">
<label for="F">Femme</label><br>
```

Et enfin, on a les inputs de type **checkbox** qui permettent de choisir **plusieurs** éléments parmi une liste. Pour que **$_POST** récupère bien les multiples choix, on lui donne un nom avec des accolades [], pour qu'il comprenne que l'on a affaire à un tableau. Voici la syntaxe :

```
Comment êtes-vous habillés ? : <br>
<input type="checkbox" name="habits[]" value="Chemise">
<label for="Chemise">Chemise</label><br>
<input type="checkbox" name="habits[]" value="Pantalon">
<label for="Pantalon">Pantalon</label><br>
<input type="checkbox" name="habits[]" value="Chaussettes">
<label for="Chaussettes">Chaussettes</label><br>
<input type="checkbox" name="habits[]" value="Chaussures">
<label for="Chaussures">Chaussures</label><br>
```

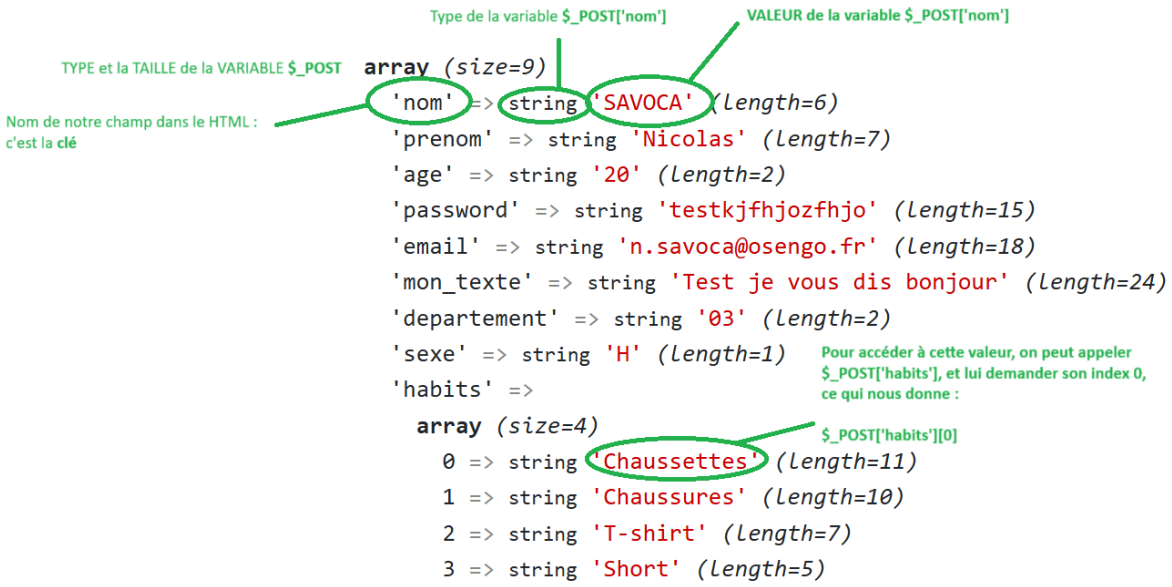
4. Traiter les données avec $_POST

Une fois que le formulaire est envoyé, si son attribut **method** est défini sur « **POST** », alors on peut récupérer les données du formulaire dans la variable PHP **$_POST.**

Cette variable, tout le temps accessible, se présente sous forme de tableau associatif faisant le lien entre la **clé** (l'attribut name) et la **valeur** (l'attribut value dans les input de sélection, ou alors la saisie utilisateur dans les input textes).

Si l'on a saisi plusieurs choix dans notre liste de checkbox par exemple, alors la variable **$_POST** nous donnera un **tableau** contenant toutes les cases cochées.

Voici le contenu de **$_POST** après avoir envoyé le formulaire réalisé ici :



5. Traiter les données avec $_GET

Quand l'attribut **method** du formulaire est défini sur **GET**, le comportement est légèrement différent de **POST**. En effet, les valeurs saisies dans le formulaire vont être envoyées à l'URL de **destination** (l'attribut **action**) et seront donc **visibles dans l'URL**. On parle parfois de **paramètres**. Exemple :

```
http://www.domain.com/index.html?name1=value1
```

Après l'adresse complète, on note un **?** qui va délimiter le début de notre liste de paramètres, puis on voit notre première **clé**, name1, et notre première **valeur**, value1.

Il est tout à fait possible de faire passer dans l'URL plusieurs paramètres, comme ceci :

```
http://www.domain.com/index.html?name1=value1&name2=value2
```

Après le premier paramètre, on ajoute les autres avec le caractère **&**

Une fois le formulaire envoyé, on peut également accéder aux valeurs saisies, mais cette fois non pas avec **$_POST**, mais **$_GET**.

Exemple :

Pour l'URL :
http://localhost/contact.php?nom=SAVOCA&prenom=Nicolas&age=5&password=monmotdepasse&email=email%40domaine.com

On obtient **$_GET** comme ceci :

```
array (size=9)
  'nom' => string 'SAVOCA' (length=6)
  'prenom' => string 'Nicolas' (length=7)
  'age' => string '5' (length=1)
  'password' => string 'monmotdepasse' (length=13)
  'email' => string 'email@domaine.com' (length=17)
```

Si on a un formulaire avec plusieurs choix, ils apparaîtront comme ceci dans l'URL, séparés par %5B%5D, un caractère spécial encodé dans l'URL :

http://localhost/contact.php?habits%5B%5D=Chemise&habits%5B%5D=Pantalon&habits%5B%5D=Chaussettes

Et apparaîtront ainsi dans le **var_dump** de **$_GET** :

```
array (size=1)
  'habits' =>
    array (size=3)
      0 => string 'Chemise' (length=7)
      1 => string 'Pantalon' (length=8)
      2 => string 'Chaussettes' (length=11)
```

6. Pourquoi choisir $_GET ? Ou $_POST

La visibilité des valeurs dans l'URL est le principal point de décision. Dans certains cas, on veut que les paramètres soient visibles car l'URL contenant ces paramètres peut être **référencée** par les moteurs de recherche. Par exemple, dans le cas d'une page « catalogue de produit », on peut imaginer qu'une url comme celle-ci peut être intéressante car elle met en valeur la **catégorie** et la **collection** :

http://localhost/catalogue.php?categorie=mobilier-jardin&collection=printemps

Mais dans d'autres cas, pour des raisons de sécurité, il vaut mieux cacher les valeurs qui transitent par le formulaire, par exemple les formulaires de connexion et d'inscription, évidemment.

En gros on peut considérer que tout ce qui relève des données utilisateurs doit absolument être caché, et donc transiter avec $_POST, et tout ce qui peut servir au référencement et au partage d'URL réutilisables transitera par $_GET.

Mais bien sûr, comme à chaque fois, il existe des exceptions.

7. Et les fichiers dans tout ça ?

Si l'on veut envoyer des fichiers dans notre formulaire, il n'est pas nécessaire de changer la méthode du formulaire. En effet, tout fonctionnera comme d'habitude avec **$_GET** ou **$_POST**

En revanche on va avoir besoin d'un nouveau type d'input, l'input de type **file**. C'est cet input qui va nous générer le bouton « Parcourir » que l'on connaît tous, qui va nous permettre de chercher le fichier à uploader dans notre ordinateur.

```
<input type="file" name="photo_de_profil"/>
```

Et enfin, afin que ce fichier soit bien transmis, il faut également rajouter le type des données envoyées dans la balise form, avec le **enctype** ceci :

```
<form method="POST" action="/add" enctype="multipart/form-data">
```

Une fois ces ajouts effectués, le fichier uploadé sera récupéré non pas dans **$_POST** mais dans **$_FILES**, une autre variable superglobale de PHP.

Nous allons alors devoir récupérer le fichier dans la mémoire de PHP grâce à **$_FILES**, et le **déplacer** vers un chemin définitif de notre choix afin de le stocker de manière permanente.

Pour cela nous utilisons **move_uploaded_file()**, comme ceci :

```
$fichierEnMemoire = $_FILES['photo_de_profil']['tmp_name'];
$cheminFinal = "/images/" . $_FILES['photo_de_profil']['name'];
move_uploaded_file($fichierEnMemoire, $cheminFinal);
```