PHP 06 - Les formulaires

Nous avons vu comment créer un formulaire en HTML et le valider avec un langage côté client, le Javascript. Cela s'arrêtait là, impossible d'en faire davantage.

Avec PHP, langage côté serveur, nous allons pouvoir :

- traiter un formulaire, c'est-à-dire valider les données
- enregistrer les valeurs saisies dans une base de données
- envoyer un fichier (document, image...) qui se trouve sur votre PC vers le serveur qui hébergé le site web

Rappel sur la balise HTML `<form>`

Un formulaire est créé via la balise HTML `<form>` et doit comporter au moins un champ `input` (type `text` ou autre) ainsi qu'un second de type `submit` (ou une balise `<button>`) permettant l'envoi :

```
<form action="script.php">
    <input type="text" name="prenom">
    <input type="text" name="nom">
    <input type="submit" value="Envoyer">
</form>
```

L'attribut `action` indique le fichier vers lequel sera envoyé le formulaire (le nom du fichier est libre).

Un second attribut, `method`, peut être omis. Dans ce cas, une valeur par défaut est attribuée : cette valeur est `GET`.

Méthode `GET`

On peut toutefois spécifier explicitement que la méthode est de type `GET`, ce qui donne :

```
// <form action="script.php"> équivaut donc à :
<form action="script.php" method="GET">
```

Avec cette méthode, les données seront transmises via l'url :

```
https://localhost/jarditou/index.html?prenom=dave&nom=loper
```

Dans le fichier *script.php*, les données seront récupérées grâce à la variable superglobale `$_GET` qui se comporte comme un tableau PHP :

```
$prenom = $_GET['prenom'];
$nom = $_GET['nom'];
```

Les index du tableau `$_GET` correspondent aux valeurs des attributs `name` des éléments `<input>` du formulaire.

> Il faut entendre `GET` au sens du protocole HTTP.

Méthode `POST`

L'attribut `method` supporte une autre valeur : `POST`. Celle-ci s'impose pour des tailles de données importantes, des valeurs non textes (non ASCII, c'est-à-dire fichiers...) ou des traitements côté serveur (enregistrement en base de données).

C'est donc la méthode à utiliser dans la plupart des cas, puisque l'on doit toujours, sur le plan de la sécurité, valider les données provenant d'un formulaire (saisies par un utilisateur) du côté serveur (côté client le Javascript peut-être désactivé).

Dans le fichier *script.php*, les données seront récupérées grâce à la variable superglobale `$_POST` :

```
$prenom = $_POST['prenom'];
$nom = $_POST['nom'];
```

> GET ou POST ?

Champs cachés

Les champs cachés (`type="hidden"`) sont eux aussi transmis et donc accessibles via `$_POST`.

```
<form action="script.php" method="post">
    Votre e-mail : <input type="text" name="email">
    <input type="hidden" name="secret" value="valeur cachée">
    <input type="submit" value="Envoyer">
</form>
```

Dans *script.php*, testez :

```
    echo $_POST["secret"];
```

Champs à valeur multiples

Pour les champs de type cases à cocher (*checkbox*) et listes déroulantes (*select*), le choix multiple est possible (via la touche *Ctrl*), on l'indique en joutant des crochets `[]` dans l'attribut `name` :

```
<form action="script.php" method="post">
    Tu utilises internet plutôt le :<br>
    <input type="checkbox" name="Fjour[]" value="Lundi">Lundi<br>
    <input type="checkbox" name="Fjour[]" value="Mardi">Mardi<br>
    <input type="checkbox" name="Fjour[]" value="Mercredi">Mercredi<br>
    <input type="checkbox" name="Fjour[]" value="Jeudi">Jeudi<br />
    <input type="checkbox" name="Fjour[]" value="Vendredi">Vendredi<br>
    <input type="submit" name="envoyer" value="Envoyer">
</form>
```

Dans *script.php*, `$_POST["Fjour"]` contiendra un tableau dont il faudra lire les valeurs à l'aide d'une boucle (testez l'exemple) :

```
<?php
echo "Tu surfes sur le web en semaine plutôt le : ";

// Lecture du tableau
foreach ($_REQUEST["Fjour"] as $jour)
{
    echo"- $jour<br>";
}
?>
```

Validation côté serveur

Pour valider les données saisies côté serveur, dans le fichier *script.php*, on va utiliser des conditions, par exemple :

```
if (empty($_POST["nom"]))
{
    echo "Le nom doit être renseigné";
}
```

Fonctions utiles :

- `empty()`
- `is_null()`
- `strlen()`
- `preg_match()`
- `isset()`

Regardez la documentation officielle de ces différentes fonctions pour comprendre leur usage.

Exercice

1. Reprenez le formulaire que vous aviez realisé pour l'évaluation HTML / Javascript du TBDA (formulaire de contact Jarditou).

2. Modifiez ce formulaire pour valider les données côté serveur.

3. Un message devra être affiché en cas d'erreur de saisie.

On se trouve ici confronté à un problème : les fichiers PHP du formulaire et de son traitement sont distincts et il n'y a pas de retour possible vers le formulaire. La seule solution (en l'état actuel de vos connaissances), est de relier les 2 pages avec un lien dans lequel on passe un numéro d'erreur.

Dans le fichier *script.php* tout d'abord :

```
if (empty($_POST["nom"]))
{
    header("Location:formulaire.php?erreur=nom");
    exit;
}

if (empty($_POST["prenom"]))
{
    header("Location:formulaire.php?erreur=prenom");
    exit;
}
```

La fonction `header()` redirige automatiquement vers le formulaire via l'url indiquée. On passe dans cette url un paramètre *erreur* qui a pour valeur *nom* : cela indiquera qu'il y a une erreur sur ce champ. Cette fonction appelle la page de l'url et sort donc du script (*script.php*).

Dans le fichier du formulaire maintenant :

Attention, votre formulaire va désormais comprendre du PHP, vous devez donc le renommer avec une extension en *.php*, par exemple *contact.php* au lieu de *contact.html*.

Une condition teste la présence du paramètre *erreur* (récupéré avec `GET` puisque dans l'url) et la valeur de celui-ci; la condition est placée là où on souhaite afficher dans la page :

```
<form action="script.php">
    <input type="text" name="prenom">

    <?php
    if (isset($_GET["erreur"]) && $_GET["erreur"] == "prenom")
    {
        echo"Le prénom doit être renseigné.";
    }
    ?>

    <input type="text" name="nom">

    <?php
    if (isset($_GET["erreur"]) && $_GET["erreur"] == "nom")
    {
        echo"Le nom doit être renseigné.";
    }
    ?>

    <input type="submit" value="Envoyer">
</form>
```

Il faut répéter la redirection avec header et la condition pour l'affichage autant de fois qu'il y a de tests de verification à réaliser.

4. Une fois tous les champs validés, l'ensemble des données saisies devra être affiché.

Il n'y a ici pas non plus de solution (encore une fois à ce moment précis de votre apprentissage) pour passer les valeurs envoyées à script.*php*.
On pourrait utiliser ici aussi la méthode des paramètres dans l'url pour réafficher les données, mais cela s'avère vite fastidieux.

Exemple :

```
if (strlen($_POST["nom"]) > 12)
{
    header("Location:formulaire.php?erreur=nom&valeur=".$_POST["nom"]);
    exit;
}
```

Côté formulaire, on réaffichera la valeur dans l'attribue `value` du champ en mettant directement une condition PHP dans cet attribut :

```
<form action="script.php">
<input type="text" name="prenom" value="<?php if (isset($_GET["erreur"]) && $_GET["erreur"] == "prenom") { echo

<?php
if (isset($_GET["erreur"]) && $_GET["erreur"] == "prenom")
{
    echo"Le prénom doit être renseigné.";
}
?>
```

> NB: les frameworks côté serveur (CodeIgniter, Symfony etc.) apportent des mécanismes pour faciliter la gestion des problématiques de formulaire vues ici.