TP3 – AJAX, JSON & XML

1 Formulaire de contact en AJAX

a) Partie client

Assurez-vous que vous avez la dernière version fonctionnelle. On rappelle l'url du dépôt git :

```
https://github.com/changuelsami/bloggy_v1.git
```

Nettoyer le fichier contact.php en supprimant tout le code PHP sauf l'appel au header et au footer. Remplacer le code PHP supprimé par ceci :

```
<div class="alert alert-success" id="notif"></div>
```

Ajouter une règle CSS permettant de cacher ce div au chargement de la page.

Dans le fichier contact_action.php, dans la dernière ligne remplacer « header » par « print » pour avoir le résultat suivant :

```
print("Location: contact.php?retour=$retour&sujet=$sujet");
```

Créer le fichier js/bloggy.js, dans le fichier contact.php ajouter en dernière ligne un appel à ce fichier. Voici le contenu de js/bloggy.js :

```
$(document).ready(function() {
    $("form").submit(function(eve){
        eve.preventDefault();
        alert("Et après ???")
    });
});
```

Sauvegarder tous les fichiers et ouvrir la page de contact sur le navigateur, répondre aux questions suivante :

1. la vérification des champs obligatoire fonctionne-t-elle toujours ?
2. en déduite si l'évènement submit est lancé ou pas
3. la validation du formulaire prend-elle en compte l'attribut « action » du formulaire ? Argumenter

Réponses :

1. oui, les champs requirés sont toujours fonctionnel !
2. Oui il est bien lancé puisque les contrôles sont exécutés lors du submit
3. non la page ne passe plus les paramètres à « contact_action.php » ! C'est parcequ'en JS on a pris en charge la gestion de l'évènement submit ensuite on a annulé le traitement par défaut.

Supprimer l'alerte et insérer le code ci-dessous (à compléter), tester via le navigateur en s'assurant que la console est bien active (F12 sous chrome ou firefox) ATTENTION aux virgules

```
$.ajax({
            method: "...",
            url : "contact_action.php",
            data : {
```

```
            email   : $("...").val(),
            sujet   : ...,
            ...     : ...,
            ...     : ...
            },
        success : function(data) {
            console.log(data)
        }
    });
```

AIDE : jQuery propose la méthode `.prop('checked')` qui renvoi « *true* » si la case est cochée, « *false* » sinon.

Vérifier que les points suivants :

- Aucune erreur jaavscript dans la console
- La page ne se recharge pas
- Les données sont bien enregistrées dans la BdD

b) Partie serveur

Supprimer la dernière ligne et la remplacer par le code suivant (on donne l'algorithme) :

- créer un tableau associatif contenant les 2 varaibles de retour
- définir l'en-tête HTTP suivante : Content-Type: application/json
- transformer ce tableau en objet JSON et l'afficher

Lire les documentations suivantes :

- http://php.net/manual/fr/language.types.array.php
- http://php.net/manual/fr/function.header.php
- http://php.net/manual/fr/function.json-encode.php

c) Le client - bis

A partir de maintenant le client récupère un objet JSON qui facilite la manipulation des données retournées par le serveur. Dans la fonction « success » saisir le code permettant de :

- créer une variable « msg » qui contient le bon message selon la valeur du retour (1 ou -1)
- affecter ce message au div #notif
- afficher le div en question

Rappel : une variable JSON est comme un objet, on accède aux propriétés comme en C++ ou en Java : obj.prop

2 XML

Dans la classe Contact.php, vérifier que la méthode liste() renvoi bien toutes les données de la table « contact »

a) Export

Créer la page « admin/export_xml.php » qui permet d'exporter le contenu de la table sous forme d'un fichier XML et de proposer à l'admin de télécharger le fichier généré. Voici quelques détails :

- Entête d'un fichier XML : `<?xml version='1.0' encoding='utf-8'?>`
- Forcer le téléchargement d'un flux de données (à insérer dans l'en-tête) : `Content-disposition: attachment; filename="export.xml"`

Voici le format XML souhaité :

```
-<contacts>
 +<contact></contact>
 -<contact>
    <email> foo@bar.com </email>
    <sujet> Hi there </sujet>
    <message> Bonjour tout le monde, ceci est un test </message>
    <newsletter>Oui</newsletter>
    <date> 2016-10-16 16:35:17 </date>
  </contact>
</contacts>
```

L'algorithme est le suivant :

- Créer une chaine de caractère initialisée avec l'en-tête (CONTACTS)
- Faire une boucle sur le contenu de la table et concaténer la chaine au fure et à mesure
- Fermer le balise ouverte lors de la remière étape
- Définir l'en-têtes HTML pour forcer le téléchargement
- Afficher le flux XML

Dans la page d'accueil de l'admin créer alors un bouton qui permet de lancer l'export.

b) Import

Dans la page d'accueil de l'admin ajouter un bloc permettant l'import des données XML, ce bloc contiendra deux éléments. La classe suivante sera utilie :

```php
class Util
{
    public function upload($file_field_name, $path)
    {
        $newFileName = $_FILES[$file_field_name]['name'];
        if( !move_uploaded_file( $_FILES[$file_field_name]['tmp_name'],
                                 $path.'/'.$newFileName) )
            return ""; //Erreur d'upload
        return  $newFileName ;
    }
}
```

Pour mieux comprendre la notion d'upload de fichier, lire ceci :

`http://www.w3schools.com/php/php_file_upload.asp`

`http://php.net/manual/fr/features.file-upload.post-method.php`

Une fois le fichier importé (uplodé) il faut insérer les données dans la table (parsing du XML et insertion dans la BdD via la calsse Contact) en utilisant la bibliothèque SimpleXML. Pour comprendra comment on peut faire le parsing tester le code suivant :

```
<pre>
<?php

$xml = <<<XML
<?xml version='1.0' standalone='yes'?>
<movies>
 <movie>
  <title>PHP: Behind the Parser</title>
  <rating>7/10</rating>
 </movie>
</movies>
XML;
$films = new SimpleXMLElement($xml) or die("Error: Cannot create object");

var_dump($films);

echo $films->movie[0]->rating;
```

Sources :
https://api.jquery.com/submit/
http://php.net/manual/ro/pdostatement.debugdumpparams.php
http://php.net/manual/fr/simplexml.examples-basic.php