Développement Web 2
TP/TD - N°2

Rappel HTML

Pour rappel, et au vu de la difficulté de beaucoup d'entre vous d'isoler l'en-tête (head) dans l'exercice 6 du dernier TP, voici la structure de base d'une page HTML :

```
1. <!DOCTYPE html>
2. <html>
3.     <head>
4.         <title>Titre</title>
5.     </head>
6.
7.     <body>
8.
9.     </body>
10. </html>
```

Le header d'une page html est bien composé du texte compris entre les balises $< head >$. Dans l'exercice 6 du dernier TP, une avec l'id «header» ($< div\ id = "header" >$) était contenu dans le body. La valeur d'une id d'une balise est un choix individuel relatif au créateur de la page HTML et affecté une valeur d'id «header» à une balise n'en fait en aucun cas l'entête du document.

Exercice 1 – Factoriser un site (suite)

- Récupérer votre projet de l'exercice 6 du dernier TP ou la correction présente sur le site du cours afin de continuer le TP.

- Modifiez le code PHP partiellement factorisé afin d'organiser le code avec un point d'entrée unique selon l'exemple vu au cours 1.

Exercice 2 – Fichiers .htaccess de configuration d'Apache

Cet exercice nécessitant une configuration d'Apache qui ne nous est pas accessible à salle de TP. Nous allons juste le survoler rapidement afin d'expliquer quelques fonctionnalités du fichier .htaccess. N'hésitez pas à essayer de réaliser cet exercice sur votre ordinateur portable si vous l'avez avec eu (Je vous expliquerais la configuration à réaliser pour cela).
Vous avez eu en cours que le fichier .htaccess pouvait permettre la réécriture des URL. Ce fichier de configuration d'Apache permet beaucoup plus de fonctionnalités.

Lorsque vous réalisez votre site en PHP, vous serez souvent amenés par exemple à créer une zone « Admin » où l'accès est limité. Afin d'empêcher les utilisateurs lambda d'y accéder, il est courant d'utiliser les fichiers .htaccess afin de créer une protection par login/mot de passe qui empêche l'accès à tous les fichiers du dossier.

Pour cela, il est nécessaire de créer deux fichiers :

- Le fichier .htaccess contiendra l'adresse du.htpasswd et des instructions pour le serveur :

```
1. AuthUserFile "absolute_path/.htppasswd"
2. AuthName "Page avec accès protégé"
3. AuthType Basic
4. Require valid-user
```

AuthName est le texte qui invitera l'utilisateur à inscrire son login et son mot de passe.
AuthUserFile est le chemin absolu vers le fichier .htpasswd (que vous mettrez dans le même répertoire que le.htaccess).

- Le fichier .htpasswd contiendra une liste de logins/mots de passe des personnes autorisées sous le format login:mot_de_passe_crypté :

```
1. mateo21:$1$MEqT//cb$hAVid.qmmSGFW/wDlIfQ81
2. ptipilou:$1$/lgP8dYa$sQNXcCP47KhP1sneRIZoO0
3. djfox:$1$lT7nqnsg$cVtoPfe0IgrjES7Ushmoy.
4. vincent:$1$h4oVHp3O$X7Ejpn.uuOhJRkT3qnw3i0
```

Dans cet exemple, il y a quatre personnes autorisées à accéder au dossier : mateo21, ptipilou, djfox et vincent.
Leurs mots de passes sont cryptés .htpasswd dans le fichier et cela est très important. En effet, si quelqu'un vient un jour à lire votre fichier .htpasswd (quelqu'un qui utilise le même PC que vous par exemple), il ne verra que le mot de passe crypté. Et là, il y n'a aucun risque qu'il ne retrouve votre mot de passe : ce cryptage est indéchiffrable car il est à sens unique.

Pour le cryptage, il existe une fonction PHP $crypt()$ prenant en entrée votre mot de passe en clair et retournant le mot de passe crypté.
Nous implémenterons cette fonction dans l'exercice suivant pour une introduction au formulaire.

Pour en savoir en plus sur les fonctionnalités (Protection de répertoire, Redirection, Gestion des erreurs,…) du fichier .htaccess :
http://httpd.apache.org/docs/2.4/fr/howto/htaccess.html
https://openclassrooms.com/courses/le-htaccess-et-ses-fonctionnalites

Exercice 3 – Mon premier formulaire

Cet exercice a pour but de vous faire réaliser votre premier formulaire. Considérez la structure du code ci-dessous :

```php
<?php
if (isset($_POST['login']) AND isset($_POST['pass']))
{
    //Cryptage du mot de passe avec la fonction crypt()
    //Affichage du login et du mot de passe crypté
}

else // On n'a pas encore rempli le formulaire
{
?>

<p>Entrez votre login et votre mot de passe pour le crypter :</p>

<!-- Insérer votre formulaire ici -->

<?php
}
?>
```

Ce code est constitué de deux parties à compléter :

- Si les variables $_POST['login'] et $_POST['pass'] existent, alors c'est qu'on vient de valider le formulaire.
  On crypte le mot de passe qu'on a entré, et on affiche le login et le mot de passe crypté.

- Sinon, si les variables $_POST['login'] ou $_POST['pass'] n'existent pas, on affiche le formulaire pour demander d'entrer un login et un mot de passe.

Le formulaire recharge la même page car il n'y a pas d'attribut « action » dans la balise $< form >$.
Lors du rechargement de la page, les variables $_POST['login'] et $_POST['pass'] existeront puisque vous venez de confirmer le formulaire.
Le login et le mot de passe crypté seront alors affichés.

- ➢ Complétez le code.

Exercice 5 – Gestion de base de données

Cet exercice a pour but de vous faire comprendre le flot de traitement d'un formulaire et de savoir utiliser $filter\_input$, $filter\_var$, etc …

- Ecrivez un script sql qui crée une table $VOITURE$ avec une deux colonnes, $Immatriculation$ et $Propriétaire$, toutes deux de type $TEXT$. (Vous pouvez ajouter une colonne id si vous le souhaitez, mais cela n'est pas important pour l'exercice).

- Exécutez ce script pour créer la base de données $sqlite$.

- Écrivez une page (ou un « template ») $HTML$ contenant un formulaire demandant l'immatriculation et le nom du propriétaire de la voiture (préférez, pour le propriétaire, une balise $textarea$ au lieu qu'une balise input, vous allez comprendre pourquoi dans la suite du TP).

- Le formulaire activera un deuxième script (dont on vous demande d'écrire le code) ; Ce script aura deux objectifs :
  - ajouter la voiture dans la base de données $sqlite$, sans filtrer les données du $\$\_POST$, et sans utiliser de requêtes préparés PDO.
  - afficher ensuite toutes les voitures contenues dans la base de données sans échapper la sortie (sans utiliser la fonction $htmlentities$ ou d'autres variantes des cette fonction).

- Ajoutez au deuxième script le filtrage des entrées : utilisez $filter\_input\_array$ pour vérifier que le champ $Proprietaire$ du $\$\_POST$ ne soit pas vide, et que le champ $Immatriculation$ contienne bien une chaîne de caractères représentant le numéro d'immatriculation d'une voiture. Utilisez ainsi $FILTER\_VALIDATE\_REGEXP$ avec l'expression régulière suivante : $/\hat{}[A-Z]\{2\}-[0-9]\{3\}-[A-Z]\{2\}\$/$ pour valider le champ $Immatriculation$.

- Optionnel : ajoutez les validations coté client dans le formulaire, disponibles avec html5.
  Aide : https://www.alsacreations.com/tuto/lire/1391-formulaire-html5-placeholder-required-pattern.html

  Est-ce suffisant de valider un formulaire seulement du coté client ? En supposant que les validations coté serveur ne soient pas implémentées, proposez une méthode pour rentrer dans la base de données une voiture dont le numéro d'immatriculation n'est pas reconnu par l'expression régulière $/\hat{}[A-Z]\{2\}-[0-9]-\{3\}-[A-Z]\{2\}\$/$, en contournant les validations coté client.

Exercice 6 – Injection sql

Cet exercice a pour but de vous faire injecter du code dans des requêtes sql et de savoir utiliser les requêtes préparées PDO pour prévenir ces injections. Cet exercice utilisera les scripts de l'exercice précédent.

En sachant que le script d'insertion d'une voiture n'utilise pas les requêtes préparés PDO, amusez-vous à injecter du code sql dans le formulaire afin de :

- vider la table VOITURE de la base de données de toutes les lignes ;
- Après avoir utilisé le formulaire pour ajouter la voiture de M. Macron à la table $VOITURE$ et possiblement d'autres voitures, faite disparaître la voiture de M. Macron (et seulement la sienne) de la table.
- Utilisez une injection sql afin de détruire complément la table entière.

Enfin :

- Corrigez le script d'insertion en utilisant les requêtes préparées PDO.
- Vérifiez qu'il n'est plus possible de manipuler la base de données comme fait auparavant.

Exercice 7 – Faille XSS

Cet exercice a pour but de vous faire simuler une attaque XSS par injection de code $javascript$ dans la table. Cet exercice utilisera les scripts des exercices précédents.

- Injectez du code $javascript$ dans la base de données : lors de l'affichage des données de la base, le code injecté ouvrira en boucle une fenêtre d'alerte prétendant une rançon pour débloquer le navigateur. L'injection de code sera faite en écrivant le code dans la $textarea$ du formulaire demandant le nom du propriétaire.
- Corrigez vos « templates » : chaque affichage sera échappé.
  Par exemple, chaque paire de balises ouvrantes/fermantes de la forme $<?= \$variable\ ?>$ sera substitué par $<?= htmlentities(\$variable)\ ?>$.
- Vérifiez qu'il n'est plus possible d'injecter du code $javascript$.