IUT de Montpellier - Programmation Web - TD7
Authentification et BackOffice

Rémi Coletta, Romain Lebreton, Bastien Vialla

Semaine du 17 Novembre 2014

1 Modification du Modèle Utilisateur

Q 1. Nous allons commencer par modifier la table utilisateur en lui ajoutant une colonne `VARCHAR(64) mdp` stockant son mot de passe.

Plus d'explications : Étant donné que nous allons utiliser une fonction de cryptage pour stocker ce mot de passe, vous devez prévoir une taille de champs correspondant à la taille du mot de passe crypté et non de la taille du mot de passe lui-même (64 caractères pour SHA-256).

Q 2. Modifier la vue `viewCreateUtilisateur.php` pour ajouter deux champs de mot de passe au formulaire. Le deuxième champ mot de passe sert à valider le premier.

Q 3. Modifier les actions `save` puis `updated` du contrôleur `ControlleurUtilisateur.php` pour sauver dans la base le mot de passe de l'utilisateur. Vérifier que les deux champs coïncident.

Comme mentionné ci-dessus, on ne stocke jamais le mot de passe en clair dans la base, mais sa version cryptée :

```
<?php
$mot_passe_en_clair = 'apple';
$mot_passe_crypte = hash('sha256', $mot_passe_en_clair);
echo $mot_passe_crypte;
//affiche '3a7bd3e2360a3d29eea436fcfb7e44c735d117c42d1c1835420b6b9942dd4f1b'
?>
```

Pour éviter les attaques de type dictionnaire, qui permettent de retrouver le mot de passe original à partir de la chaîne de caractères cryptée, on va concaténer une chaîne de caractères aléatoire fixe à la fin de notre mot de passe.

Q 4. Recopier le code suivant dans le fichier `Conf.php`. Ce code stocke votre chaîne aléatoire dans une variable `static` qui sera accessible par la méthode statique `Conf::getSeed()`.

```
private static $seed = 'une␣chaine␣aleatoire␣fixe';

static public function getSeed() {
    return self::$seed;
}
```

Q 5. Modifier les actions `save` puis `updated` du contrôleur ControlleurUtilisateur.php pour sauver dans la base la version SHA-256 de la racine concaténée au mot de passe de l'utilisateur.

Les autres vues et actions du contrôleur ne sont pas impactées par la modification car le mot de passe n'a pas vocation à être affiché.

À l'heure actuelle, le mot de passe transite en clair dans l'URL. Vous conviendrez facilement que ce n'est pas top. Nous allons donc passer nos formulaires en méthode POST.

Il faudrait donc maintenant récupérer les variables à l'aide de `$_POST` et non `$_GET`. Cependant, nos liens internes, tels que 'Détails' ou 'Mettre à jour' fonctionnent en passant les variables dans l'URL comme un formulaire GET. Nous avons donc besoin d'être capable de récupérer les variables automatiquement dans `$_POST` ou le cas échéant dans `$_GET`.

Q 6. Créer dans le dispatcher une fonction `myGet($nomvar)` qui retournera `$_GET[$nomvar]` s'il est défini, ou `$_POST[$nomvar]` s'il est défini, ou sinon `NULL`.

Q 7. Remplacer tous les `$_GET` de `dispatcher.php`, `ControllerUtilisateur.php` et `ControllerTrajet.php` par des appels à `myGet`.

Remplacer les tests du type `isset($_GET['login'])` par `!is_null(myGet('login'))`.

Aide : Utiliser la fonction de remplacement `Ctrl+H` de NetBeans pour vous aider.

Q 8. Passer le formulaire de viewCreateUtilisateur.php en méthode POST.

2 Sécurisation d'une page avec les sessions

Pour accéder à une page réservée, un utilisateur doit s'authentifier. Une fois authentifié, un utilisateur peut accéder à toutes les pages réservées sans avoir à retaper son mot de passe.

Il faut donc faire circuler l'information "s'être authentifié" de pages en pages.

On pourrait faire ceci grâce à un champs caché dans un formulaire, mais ça ne serait absolument pas sécurisé. Nous allons donc utiliser les sessions.

2.1 Les sessions

Une session permet d'associer à une navigation (même adresse ip, même navigateur,...) un ensemble de valeurs définies de manière transparente pour le visiteur et que le serveur conserve de page en page. Le maniement des valeurs de sessions est assez simple en PHP, on peut stocker presque tout dans une variable de session : un chiffre, un texte, voir un objet (il faut utiliser `serialize($o)` lors de la mise en session de l'objet `$o`, puis `unserialize($o)` quand on le récupère de la session).

Dans toute page qui manipule les sessions

```
session_name("chaineUniqueInventeParMoi");  // Optionnel : voir section 3
session_start();
```

Attention : Il faut mettre `session_start()` avant toute écriture de code HTML dans la page.

Mettre une variable en session

```
$_SESSION['login'] = 'remi';
```

Vérifier qu'une variable existe en session

```
if (!empty($_SESSION['login'])) {//do something}
```

Destruction des variables en session

```
session_destroy();
unset($_SESSION);
```

2.2 Page de connexion

Q 9. Créer une vue `viewConnectUtilisateur.php` qui comprend un formulaire avec deux champs, l'un pour le login, l'autre pour le mot de passe. Ce formulaire appelle l'action `connected` du contrôleur de Utilisateur.

Ajouter une action `connect` qui affiche ce formulaire dans `ControlleurUtilisateur.php`.

Q 10. Démarrez la session au début du dispatcher.

Q 11. Ajouter une action `connected` dans le contrôleur de Utilisateur, qui vérifie que le couple (login / passwd) correspond à un utilisateur existant et qui, si oui, met le login de l'utilisateur en session. Affichons par défaut la vue de détail de l'utilisateur qui vient de se connecter.

Aide :

1. Utilisez la fonction `ModelUtilisateur::selectWhere($data)` qui fait un select sur les champs de `$data` (ici 'login' et 'mdp'). En effet, la fonction de base `ModelUtilisateur::select($data)` ne sert qu'à récupérer un utilisateur étant donné son login.

2. Notez que `ModelUtilisateur::selectWhere($data)` renvoie un *tableau d'utilisateurs*. Il faudra donc tester si le tableau est vide avec `count()`. De plus, la vue find a besoin que l'on initialise une variable d'utilisateur `$u` contenant l'utilisateur.
3. Enfin, n'oubliez pas de hasher le mot de passe convenablement.

Q 12. Ajouter une action `deconnect` dans le contrôleur de Utilisateur, qui détruit la session en cours. Une fois déconnecté, on renvoie l'utilisateur sur la page d'accueil du site.

Q 13. Modifier le `header.php` de sorte à ajouter :
— un lien vers la page de connexion, quand l'utilisateur n'est pas connecté (pas présent en session).
— un message de bienvenu, quand l'utilisateur est connecté, et un lien vers l'action de déconnexion.
Tester la connexion/déconnexion avec un couple login/passwd correct et un incorrect.

2.3 Sécurisation d'une page à accès réservé

On souhaite restreindre les actions de mise à jour et de suppression qu'à l'utilisateur actuellement authentifié.

Q 14. Modifier la vue de détail pour qu'elle n'affiche les liens vers la mise à jour ou la suppression que si le login concorde avec celui stocké en session.

Conseil : Pour faciliter la lecture du code, nous vous conseillons de créer une fonction `is_user()`. Pour faire les choses proprement, on va créer un fichier `config/Session.php` contenant le code suivant que l'on inclura au moment de `session_start()`.

```
class Session {
    public static function is_user($login) {
        return (!empty($_SESSION['login']) && ($_SESSION['login'] == $login));
    }
}
```

Cette modification n'est pas suffisante car un petit malin pourrait accéder à la suppression d'un utilisateur quelconque en rentrant l'action delete dans l'URL.

Q 15. Modifier les actions update et updated du contrôleur de Utilisateur de sorte que si le login pour lequel une modification est demandé ne concorde pas avec celui stocké en session, on redirige l'utilisateur sur la page de connexion.

Q 16. Sécuriser l'accès à l'action de suppression d'un utilisateur.

2.4 Super administrateur

Q 17. Ajouter un champ admin de type boolean à la table Utilisateur.

Q 18. Modifier l'action connected de `ControllerUtilisateur.php` pour enregistrer dans la session si l'utilisateur est un administrateur ou non.

Q 19. Modifier les actions update`ControllerUtilisateur.php` de sorte qu'un utilisateur de type admin ait tous les droits sur toutes les actions de tous les utilisateurs.

Conseil : Pour faciliter la lecture du code, nous vous conseillons de compléter le fichier `config/Session.php` avec la fonction `is_admin` :

```
public static function is_admin() {
    return (!empty($_SESSION['admin']) && $_SESSION['admin']);
}
```

Q 20. Modifier la vue de mise à jour pour que, si l'utilisateur authentifié est un administrateur, il puisse promouvoir à l'aide d'une checkbox l'utilisateur que l'on met à jour en administrateur.

Attention, il ne suffit pas de contrôler que l'utilisateur est un administrateur dans les vues. Un petit malin pourrait quand même accéder au actions du contrôleur et faire des dégâts.

Q 21. Sans toucher le code PHP [1], hacker votre propre site en permettant à un utilisateur non admin de se passer lui même en admin.

Q 22. Corriger cette faille de sécurité.

Indice : Modifier l'action updated dans le controlleur.

3 Le cas particulier des sessions en hébergement mutualisé

Dans le cas d'un hébergement mutualisé, (comme à l'IUT) deux répertoires différents par exemple `http://infolimon.iutmontp.univ-montp2.fr/~mon_login` et `http://infolimon.iutmontp.univ-montp2.fr/~le_login_du_voisin` sont vus comme un seul site web, alors qu'il s'agit en réalité de deux sites web différents. De ce fait, si vous utilisez exactement le même nom de variable de session, il est possible que s'authentifier sur `http://infolimon.iutmontp.univ-montp2.fr/~mon_login` vous permette de contourner l'authentification de `http://infolimon.iutmontp.univ-montp2.fr/~le_login_du_voisin`.

Afin d'éviter ces désagréments ils suffit d'utiliser des noms de variables de session qu'on ne puisse deviner, ou plus simple d'affecter un nom qu'on ne puisse pas deviner à votre session, avec l'instruction `session_name("chaineUniqueInventeParMoi");` que vous appellerez de manière systématique, avant chaque appel à `session_start();`

4 Enregistrement avec une adresse email valide

Dans beaucoup de sites Web, il est important de savoir si un utilisateur est bien réel. Pour se faire on peut utiliser une vérification de son numéro de portable, de sa carte bancaire, etc... Nous allons ici nous baser sur la vérification de l'adresse email.

Q 23. Dans votre formulaire de création d'un utilisateur, vous vérifiez actuellement le format de l'adresse email côté client avec du HTML5 (ou du JavaScript), hacker votre propre site de sorte

Vous devriez en conclure que les contrôles côté client (navigateur) offrent un confort d'affichage mais ne constituent en aucun cas, une sécurisation de votre site !

Q 24. Dans l'action create du contrôleur Utilisateur, vérifier le format de l'adresse email de l'utilisateur.

A ce stade vous savez que votre utilisateur a saisi une adresse email d'un format valide. Nous allons maintenant vérifier que cette adresse existe réellement et qu'elle appartient bien à notre utilisateur.

Pour se faire, nous allons lui envoyer un mail et ne valider l'utilisateur (l'autoriser à se connecter) que s'il a consulté le mail que nous lui avons envoyé.

Q 25. Ajouter un champs de type VARCHAR[32] à la table utilisateur.

Q 26. Modifier l'action connect du contrôleur Utilisateur, de sorte d'accepter la connexion uniquement si ce champs validation is NULL.

Q 27. Ajouter une action validate au contrôleur Utilisateur, qui récupère en GET, le login de l'utilisateur, et une chaîne de caractère nommée "validate". Si le login correspond à un utilisateur présent dans la base, et que la chaîne "validate" passée en GET correspond à celle stockée dans la base, mettre à jour le tuple correspondant dans le la table Utilisateur en mettant à NULL le champs 'validate".

Q 28. Dans l'action create du contrôleur Utilisateur, générez un identifiant unique avec la fonction uniqid() (`http://php.net/manual/fr/function.uniqid.php`) que vous stocker en base.

Puis envoyer un email à l'adresse qu'il a renseigné, contenant dans le corps du message le lien (avec les paramètres en GET) vers l'action validante du contrôleur Utilisateur.

Q 29. puis envoyez lui un mail contenant un lien de validation (utilisant cette chaîne). vérifier le format de l'adresse email de l'utilisateur.

Pour ce faire vous aller utiliser la fonction mail() de PHP. **Abuser de cette fonction serait considéré comme une violation de la charte d'utilisation des ressources informatiques de l'IUT et vous exposerait à des sanctions !**.

1. vous avez donc le droit à modifier le code HTML, ce qui peut se faire côté client !

Pour éviter d'être blacklistés des serveurs de mail, nous allons envoyer uniquement des emails dans le domaine *yopmail.com*, dont le fonctionnement est le suivant, un mail envoyé à `bob@yopmail.com` est immédiatement lisible sur `http://bob.yopmail.com`.

5 Sessions vs. Cookies

Un cookie est utilisé pour stocker une information spécifique sur l'utilisateur, comme les préférences d'un site ou le contenu d'un panier d'achat électronique. Le cookie est un fichier qui est stocké directement sur la machine de l'utilisateur, il s'agit d'une association nom/valeur. Il ne faut pas stocker de données critiques dans les cookies !

Utilisations en PHP :

— La ligne ci-dessous créée un cookie nommé "TestCookie" contenant la valeur $value et qui expire dans 1h.
```
setcookie("TestCookie", $value, time()+3600);  /* expire dans 1 heure */
```
Attention, comme les session ou la fonction header(), la fonction setcookie() doit être appelée avant tout utilisation de HTML (le protocole HTTP impose cette restriction).

— Accéder à un cookie :
```
echo $_COOKIE["TestCookie"];
```

— Effacer un cookie (il suffit de le faire expirer) :
```
setcookie ("TestCookie", "", time() - 1);
```

6 Autres sécurisations :

Le fait de crypter les mots de passe (ou les numéros de carte de crédit) dans la base de données évite qu'un accès en lecture à la base (suite à une faille de sécurité) ne permettent à l'attaquant de récupérer toutes les données de tous les utilisateurs.

On peut aussi crypter le mot de passe sur le navigateur. Dans ce cas une attaque du tiers-écouteur (type **Man in the middle**) ne permet pas d'obtenir le mot de passe en clair de l'utilisateur. Mais puisque l'authentification repose sur le mot passe crypté, le tiers peut s'authentifier avec le mot de passe crypté, qu'il a récupéré.

La seule façon fiable de sécuriser une application web est le recours au cryptage de l'ensemble des communications entre le client (browser) et le serveur, via l'utilisation du protocole `ssl` sur `http`, à savoir `https`.