Programmation Orienté Objet

PHP – Les sessions et les mots de passe

I. Les sessions

I.1. Définition

La session est un mécanisme qui permet de conserver des données entre 2 pages web, le protocole HTTP ne le permettant pas.

Par exemple, on utilise typiquement les sessions lorsqu'un utilisateur doit s'identifier via un login/mot de passe et restreindre l'accès à certaines fonctionnalités (espace client d'un site e-commerce, back-office...) selon un système de rôles et de droits.

I.2. Fonctionnement

Pour stocker des informations en session, PHP a recours à la variable `$_SESSION`. Il s'agit d'une superglobale (même type que `$_GET` ou `$_POST` etc.) qui se comporte donc comme un tableau PHP avec des paires clé/valeur.

Mais avant toute utilisation de cette variable, il est **impératif** d'utiliser la fonction `session_start()` au début de chaque fichier PHP dans lequel on manipulera cette variable et avant tout envoi de requêtes HTTP, c'est-à-dire avant tout `echo` ou quoi que ce soit d'autre : rien ne doit avoir encore été écrit/envoyé à la page web.

Ensuite, on utilise donc `$_SESSION` avec entre crochets, un nom de variable suivi de l'affectation d'une valeur :

```
<?php
session_start();
$_SESSION["login"] = "webmaster";
echo $_SESSION["login"];
```

Nous avons pour le moment en session une variable `login` qui a pour valeur `webmaster`. Dans une autre page PHP, nous pourrons désormais tester si la session existe et autoriser ou non l'utilisateur à voir le contenu d'une page, accéder à des fonctionnalités etc.

Il est possible de stocker plusieurs sessions différentes, qui seront alors différenciées par leur nom (voir ci-après) ainsi que de stocker des tableaux voire des objets. Dans ce dernier cas, on pourra faire appel au mécanisme de sérialisation.

I.3. L'identifiant de session

Le mécanisme de session repose sur un identifiant **unique** : l'identifiant de session. Celui-ci est fourni par défaut par PHP. Il s'agit d'une chaîne de caractères alphanumérique, par exemple : `jr48dn2gqsliubpq02u0cbchk5`.

La fonction `session_id()` permet d'afficher cet identifiant :

```php
<?php
session_start();

$_SESSION["login"] = "webmaster";
$_SESSION["role"] = "admin";

echo"- session ID : ".session_id();
```

`session_id()` retourne une chaîne vide s'il n'y a pas de session courante (aucun identifiant de session n'existe).

On peut cependant nommer explicitement l'ID de session, toujours via la fonction `session_id()`, en lui passant en paramètre le nom : `session_id("12345");`

Il est cependant fortement recommandé de ne pas définir explicitement une session et de laisser PHP le faire automatiquement.

I.4. Mécanisme

La création d'une session génère donc un identifiant unique stocké à la fois côté client (sur le PC de l'utilisateur, dans un fichier de type cookie) et côté serveur. Lors des échanges, le serveur compare si l'identifiant est identique des deux côtés.

Le stockage dans un cookie est la méthode par défaut. Il existe une autre possibilité, le stockage des sessions en bases de données. Cette solution est plus sécurisée mais un peu plus complexe à mettre en œuvre.

I.5. Tester la session

Une fois une session initialisée, c'est-à-dire par exemple qu'un utilisateur s'est authentifié avec un identifiant/mot de passe, il reste à contrôler si cet utilisateur peut ou non accéder à la page web. Pour cela, on va utiliser de simples conditions :

```php
<?php
session_start();

if ($_SESSION["login"])
{
echo"Vous êtes autorisé à voir cette page.";
}
else
{
echo"Cette page nécessite une identification.";
}
```

Dans la pratique, on peut réduire ce code :

```
<?php
session_start();

if ( ! isset($_SESSION["login"]) )
{
header("Location:index.php");
exit;
}

// Reste du code (PHP/HTML)
echo"Bonjour ".$_SESSION["login"]."<br>");
```

Ici, si la variable de session n'existe pas (point d'exclamation => donc si `isset() == FALSE`), on redirige (fonction `header()`, qui doit toujours être suivie de la fonction `exit`) sur une page (ici `index.php`). Si la session est initialisée, la condition ne s'exécutera pas.

En termes de sécurité, lors d'une authentification par identifiant/mot de passe, il faut bien entendu contrôler les données postées (filtrer le format, interroger la base de données pour l'identifiant et le mot de passe) et surtout ne pas affecter directement dans les variables de session le résultat des données transmises (`$_POST`) mais, lorsque cela est possible, les valeurs de la base de données.

I.6. Nommer une session

En plus d'un identifiant, une session PHP possède un nom qui peut être affiché via la fonction `session_name()`. Par défaut, le nom est `PHPSESSID` (à ne pas confondre avec l'identifiant de session).

```
session_start();
echo"- Nom de la session : ".session_name();
```

Il est possible de nommer une session de façon explicite en passant le nom souhaité en argument à la fonction `session_name()` :

```
session_start();
session_name("afpa");
echo"- Nom de la session : ".session_name();
```

Pour récupérer le nom de la session, on utilise également `session_name()`.

I.7. Détruire une session

Pour supprimer une information en session, on utilisera la fonction `unset()` avec en paramètre la variable de session à détruire. Ceci est utilisé par exemple lorsque l'on a un lien *Déconnexion* sur un site.

Exemple :

```
unset($_SESSION["login"]);
unset($_SESSION["role"]);
```

Attention, cela ne suffit pas à détruire complètement et proprement une session, même si toutes les variables contenues dans la session ont été supprimées avec `unset()`.

Voici le code pour détruire proprement une session :

```
$_SESSION["login"] = array();
$_SESSION["role"] = array();

unset($_SESSION["login"]);
unset($_SESSION["role"]);

if (ini_get("session.use_cookies"))
{
setcookie(session_name(), '', time()-42000);
}

session_destroy();
```

Explications :

- Lignes 1-2 : on « vide » les variables par l'affectation d'un tableau vide.
- Lignes 3-4 : destruction des variables de session.
- Lignes 5-8 : via la fonction `setcookie()`, on fait expirer en termes de date le cookie qui concerne le nom de la session. Ceci n'est valide que dans le cas où les sessions sont gérées par cookies (comportement par défaut de PHP), d'où la condition.
- On détruit le reste de la session, via la fonction `session_destroy()`.

I.8. Configuration avancée et sécurité

Le PHP permet une configuration (directives du fichier `php.ini`) assez fine des sessions : volume de données, durée, mode de stockage et sécurité, ce dernier point étant un sujet important : il en effet possible de détourner des sessions par vol d'identifiant (lecture du cookie contenant l'id de session). Une bonne pratique est d'utiliser la fonction `session_regenerate_id()` dès que l'on modifie quelque chose en session (ajout d'une variable, modification d'une valeur).

I.9. Exercice : formulaire d'authentification

Créer un formulaire d'authentification utilisant le principe des sessions.

- Créer un formulaire contenant 2 champs : login (qui peut être l'adresse email) et mot de passe.
- Un fichier PHP traite les données du formulaire.
- Une autre page PHP devra être accessible uniquement si une session a été initialisée, c'est-à-dire si l'utilisateur s'est authentifié correctement. Si ce n'est pas le cas, l'utilisateur devra être redirigé sur la page de connexion avec un message d'erreur du type *La connexion a échoué*.

II. La gestion des mots de passe

II.1. Introduction

Le mot de passe est sûrement l'élément auquel on pense en premier lieu en termes de sécurité informatique et en constitue aussi le maillon faible.

Pour mesurer l'ampleur du problème (ou des dégâts), consultez les ressources suivantes :

- Mots de passe les plus piratés (en 2017 mais à peu près identiques à ceux des 20 années précédentes). La liste est anglophone mais si on effectue quelques traductions cela constitue des termes français très utilisés également : *motdepasse* (*password*), *azerty* (*qwerty*), *soleil*, *vacances* (*holidays*), *sésame*…
- Un cas célèbre (mots de passe affichés derrière le gars qui est en train d'expliquer à la télé que son système informatique a été piraté la veille)
- Combien faut-il de temps pour craquer votre mot de passe ?
- Votre mot de passe a-t-il été compromis ?

II.2. Robustesse d'un mot de passe

Alors comment finalement choisir un bon de mot de passe ?

La première étape est le choix d'un mot ou d'une phrase difficile à trouver : c'est-à-dire qui n'est pas dans le dictionnaire, qui ne soit pas une suite logique (azerty, 123456…) qui ne permet pas de vous identifier :

- votre nom/prénom, surnom,
- le prénom de votre chéri-e (même si cela peut changer), de vos enfants ou encore de votre poisson rouge,
- le nom d'une star ou d'un personnage,
- le nom d'un lieu (celui de vos dernières vacances),
- de votre voiture préférée ou de la société qui vous emploie,
- votre date de naissance ou de mariage…
- oubliez aussi les fonctions/rôles (*admin*, *superadmin*, *root*, ou encore *webmaster*, *moderateur*) ou comme certains, les formules du genre *lemotdepassedeyoutube* ou incrémentales (*password1*, *password2*, *password3*…)

En fait, vous pouvez choisir une des informations évoquées mais il faudra la complexifier en mixant lettres minuscules et majuscules, chiffres et caractères spéciaux : c'est la seconde étape. Cette technique est appelée *leet speak*. Par exemple le mot de passe *vacances* (qui n'est pas une bonne idée) deviendra *v@CaNc3s* qui comporte des lettres minuscules et majuscules, au moins un nombre et un caractère spécial.

Attention, appliquer cette technique aux mots de passe les plus utilisés ne sert strictement à rien.

Enfin, il convient d'avoir un nombre de caractères suffisamment long : on voit souvent 6 à 8 caractères minimum lors d'inscription sur des sites, mais l'ANSSI (cf. paragraphe recommandations officielles) préconise 12 caractères.

Pour tester les mots de passe, utilisez des générateurs en ligne :

- Pour ceux qui ont déjà choisi un mot de passe : testeur de robustesse
- Pour les non inspirés : générateur aléatoire

II.3. Cryptographie

Un mot de passe ne doit jamais être stocké en clair : il doit être crypté à l'aide d'un algorithme de cryptage afin que sa valeur ne puisse être lue. Exemple : le mot de passe vacances sera transformé en `$2y$10$xybwIx80qUbemOsCiobdZeK4JIg2qe8BrT83vGJF1QqyJ9bnycrx6`

Dans le cadre de la formation Développeur Web et Web mobile (D2WM), vous devez connaître les bases de la cryptographie : consultez les liens suivants :

- https://www.cnil.fr/fr/comprendre-les-grands-principes-de-la-cryptologie-et-du-chiffrement
- https://www.ssi.gouv.fr/particulier/bonnes-pratiques/crypto-le-webdoc/crypto-sensu

On peut lire sur le web des exemples utilisant les algorithmes *MD5* ou *SHA1*. C'était valable il y a une dizaine d'années, mais ces algorithmes été cassés (de nombreux sites web permettent de réafficher en clair une chaîne hachée en MD5 ou SHA1).

II.4. Technique du grain de sel

La technique du grain de sel (appelée aussi salage, ou encore salt en anglais) consiste à ajouter une chaîne alphanumérique au mot de passe lui-même.

Le but est d'empêcher de retrouver le mot de passe d'origine à partir de sa chaîne hashée (appelée *hash*). Le grain de sel doit être unique pour chaque mot de passe (chaque utilisateur donc) et doit être stocké en clair dans la base de données. Il doit bien sûr ne pas être public puisqu'il est destiné à des traitements automatiques par du code.

Exemple

Aujourd'hui, les fonctions PHP telles que `password_hash()` et `password_verify()` permettent de générer et vérifier automatiquement un grain de sel.

Certains préconisent de compliquer les hachages en utilisant différentes solutions, par exemple crypter 2 fois la chaîne d'origine ou le grain de sel, ajouter un grain de sel au début et à la fin (pas forcément identiques).

Un exemple concret.

II.5. La fonction PHP password_hash()

La fonction `[password_hash())(http://php.net/manual/fr/function.password-hash.php)` permet d'utiliser des algorithmes de cryptage en PHP et donc de générer le hash d'une chaîne de caractères, grain de sel inclus. Cette fonction peut recevoir jusqu'à 3 paramètres :

```
password_hash($chaine, $algorithme, $options);
```

- 1er argument (`$chaine`) : la chaîne à crypter (un mot de passe par exemple), obligatoire.

- 2ème argument (`$algorithme`) : l'algorithme de cryptage à utiliser, obligatoire.
- 3ème argument (`$options`) : tableau (array) d'options, facultatif.

Comme l'algorithme de cryptage (2ème argument) est obligatoire, il faut donc en indiquer un, y compris si on souhaite utiliser celui proposé par défaut : dans ce cas il faut indiquer la valeur `PASSWORD_DEFAULT`. La chaîne sera alors hachée avec [bcrypt](https://fr.wikipedia.org/wiki/Bcrypt), et le hash généré aura toujours une longueur de 60 caractères (donc prévoir un type `varchar(60)` pour le champ qui stockera le mot de passe crypté en base de données) .

Exemple :

```
$password_hash = password_hash("vacances", PASSWORD_DEFAULT);
echo $password_hash;
```

Ceci affichera `$2y$10$xybwIx80qUbemOsCiobdZeK4JIg2qe8BrT83vGJF1QqyJ9bnycrx6`.
C'est cette valeur qu'il faut stocker en base de données.

II.6. La fonction PHP password_verify()

Pour vérifier si un mot de passe saisi est bien celui enregistré en base, il faut utiliser la fonction `[password_verify()](http://php.net/manual/fr/function.password-verify.php)` :

Cette fonction reçoit 2 paramètres obligatoires :

```
password_verify($chaine_saisie_en_clair, $hash_stocke_en_bdd);
```

II.7. Recommandations officielles

L'Agence Nationale de la Sécurité des Systèmes Informatiques (ANSSI) et la CNIL éditent des re-commandations officielles pour les mots de passe :

- https://www.ssi.gouv.fr/guide/mot-de-passe
- https://www.cnil.fr/fr/les-conseils-de-la-cnil-pour-un-bon-mot-de-passe

II.8. Spécifications fonctionnelles

II.8.a. Création d'un compte

- L'utilisateur renseigne un formulaire avec, pour identifiant unique, une adresse mail. L'information qui sert d'identifiant doit être strictement unique dans la base.
- Un mot de passe est saisi par l'utilisateur
- Si l'utilisateur choisit librement son mot de passe, il doit le saisir 2 fois (un second champ *Confirmation*)
- Tester la validité (comparer notamment les 2 saisies demandées), la robustesse (des scripts existent sur le net, sinon utiliser des expressions régulières), s'assurer que le mot de passe respecte les recommandations officielles.
- Une fois que tout est valide, le mot de passe doit être hashé avec un algorithme de cryptage en utilisant la technique du grain de sel.
- la version hashée du mot de passe est ensuite enregistrée en base de données.
- Ne jamais stocker un mot de passe en clair, ne jamais le mettre en session (clair ou forme hashée), envoyer un mail de confirmation sans renvoyer le mot de passe en clair comme on peut le voir parfois.

II.8.b. Authentification

- L'utilisateur saisit son mot de passe. Limiter les essais de connexion à 3 tentatives (recommandation officielle) et donc bloquer les utilisateurs qui ont atteint ce nombre (à stocker dans une colonne en base de données).
- Sur les points devant être fortement sécurisés, il est recommandé d'utiliser une double authentification, c'est-à-dire de mettre en place une seconde méthode (2ème formulaire d'authentification, envoi d'un code de sécurité par SMS etc.).
- Une fois l'utilisateur « loggué », on stocke certaines informations (identifiant, nom/prénom) en session, mais jamais le mot de passe (ni en clair ni le hash).

II.8.c. Perte du mot de passe

Lorsqu'un utilisateur ne se souvient plus (« perd ») son mot de passe, il faut bien entendu lui permettre d'en saisir un nouveau (impossible de lui renvoyer l'actuel, puisqu'il est stocké en base sous forme hashée indéchiffrable).

Pour cela :

- Placer un lien du type **mot de passe perdu** sur le formulaire de connexion, qui aboutit donc sur un second formulaire
- Le second formulaire demande l'identifiant : celui-ci demande le login ou l'adresse email de l'utilisateur
- Si l'utilisateur est trouvé en base, on lui envoie un mail (qui permet de s'assurer qu'il s'agit du bon utilisateur) avec un lien vers un formulaire de saisie d'un nouveau mot de passe.

Ne jamais envoyer un mail contenant un nouveau mot de passe (car celui-ci circulerait en clair sur le réseau).

II.9. Exercice : mot de passe

Reprendre l'exercice sur les sessions et y ajouter la gestion des mots de passe selon les consignes suivantes :

- Créer une table *users* destinée à stocker le nom, le prénom, l'adresse mail, le login, le mot de passe, les dates d'inscription et de dernière connexion des utilisateurs.
- Créer un formulaire d'inscription pour renseigner et enregistrer les informations demandées. Celles-ci doivent être vérifiées correctement avant d'être enregistrées.
- Modifier le script PHP d'authentification pour vérifier le login et le mot de passe par rapport au contenu de la table **users**.
- Vous devrez utiliser les fonctions PHP `password_hash()` et `password_verify()`.