# Sujet 8 - Technologies du Web

mars 2020

## Sessions - Authentification

**Exercice 1 :**
Le but de cet exercice est de construire un système de contrôle d'accès et d'authentification par login et mot de passe. Nous construirons pas à pas un script php qu'il suffira d'inclure en tête des pages PHP à protéger.

**Q 0 .** *Mise en place*

1. Exécutez sur le serveur Postgresql le fichier `table_users.sql` fourni, il créée une table `s8_users`. Consultez cette table (sa structure, ses contraintes, son contenu) via phpPgAdmin.

   > **Attention** : la table contient (pour l'instant) les mots de passe en clair, ce qui est totalement déconseillé. Par souci de simplicité, c'est pourtant ce que nous ferons pour ce premier exercice. Nous verrons par la suite comment procéder différemment.

2. Sur webtp, créez un dossier dédié à l'exercice et installez-y l'arborescence de fichiers fournie.
   Il s'agit d'un site comportant une page principale `index.php`. Quand le site sera opérationnel, `index.php` devra afficher une page d'accueil personnalisée si l'utilisateur est authentifié, ou un formulaire de connexion (« login ») dans le cas contraire. Pour l'instant l'authentification n'est pas mise en place et `index.php` produit un résultat non souhaité (vérifiez).
   Examinez notamment le contenu des fichiers `index.php`, `views/pageAccueil.php`.
   D'autres pages du site resteront ouvertes à tous, comme la page `register.php` (nous ne nous intéresserons à cette page qu'à partir de la question 7).

**Q 1 .** Écrivez une classe `DataLayer` sur le modèle des exercices précédents. Assurez-vous que la connexion utilise les options
`[PDO::ATTR_ERRMODE=> PDO::ERRMODE_EXCEPTION, PDO::ATTR_DEFAULT_FETCH_MODE=> PDO::FETCH_ASSOC]`
Vous créerez une méthode `authentifier($login, $password)`.

- s'il existe dans `s8_users` un utilisateur avec le login ET le mot de passe fournis, alors le résultat est une instance de la classe `Identite` avec les informations concernant cet utilisateur.
- sinon le résultat est NULL.

**Q 2 .** Prenez en compte les deux paramètres HTTP suivants :

- `login` : une chaîne (argument **optionnel**, chaîne vide par défaut)
- `password` : une chaîne (argument **obligatoire si login** $\neq$ `""`, ignoré sinon)

Sauvegardez les valeurs reçues dans de variables globales `$login` et `$password`

**Q 3 .** Nous allons maintenant mettre en place le mécanisme de session. Il faut tout d'abord choisir une clé dans le tableau `$_SESSION` qui témoignera du fait que l'utilisateur s'est déjà authentifié au cours de la session. Ce sera pour nous la clé `'ident'`. Le principe à suivre sera :

> `$_SESSION['ident']` doit être défini **si, et seulement si,** l'utilisateur s'est déjà authentifié au cours de la session. La valeur associée à `$_SESSION['ident']` est alors un objet `Identité` représentant cet utilisateur.

Vous allez écrire un script `lib/watchdog.php` (destiné à être inclus depuis un script public) qui aura le comportement suivant :

- exécute `session_start()`
- si l'utilisateur est déjà authentifié (c'est à dire si `$_SESSION['ident']` est défini), alors le script se termine normalement, sans rien faire.
- sinon, si des arguments `login` et `password` ont été transmis **ET SI** ces identifiants sont corrects, alors on crée l'objet `$_SESSION['ident']` et le script termine normalement.

— dans **TOUS** les autres cas, le script doit inclure la page `views/pageLogin.php` et **terminer par `exit()`**

Ce script jouera donc le rôle de **sentinelle** : pour un utilisateur authentifié (soit préalablement, soit au cours de son exécution) il se termine normalement, et sans effet visible (aucun texte produit). Dans le cas contraire, il inclut la page de login et met fin à l'exécution du script qui l'appelé.
En résumé, il laisse passer, ou pas, selon que le visiteur s'est authentifié ou non.
Le script `watchdog`, sera appelé (via `require`) au tout début de toute page dont on veut contrôler l'accès.
**Q 4 .** Ajoutez l'inclusion de la sentinelle `watchdog.php` au début de `index.php` . Testez le site qui maintenant doit être correctement protégé (connectez vous avec l'un des comptes existants).
**Q 5 .** Créez une page de déconnexion `logout.php` qui permettra de mettre fin à la session. Comme précédemment, `logout.php` commencera par inclure `watchdog.php`. Testez le fonctionnement.
**Q 6 .** On souhaite que la page de connexion affiche un message à l'utilisateur en cas d'échec de connexion. Vous allez modifier `watchdog.php` pour faire en sorte qu'en cas d'echec de connexion (login ou mot de passe incorrect) on définisse `$_SESSION['echec']` avec la valeur TRUE. (et en cas de succès de connexion, on supprime `$_SESSION['echec']` ). Testez.
**Q 7 .** Une page de demande de création de compte vous est fournie (`register`). Il reste à écrire le script qui va traiter cette demande.

1. dans la classe `DataLayer` ajoutez une méthode `createUser($login,$password,$nom,$prenom)` qui crée un utilisateur. La méthode renvoie un booléen indiquant si l'insertion s'est bien passée. Notez que l'attribut `login` est la clé primaire de la table, et que des contraintes interdisent aux attributs d'être des chaînes vides. Une tentative de création invalide provoquera donc une erreur SQL. Pour détecter une erreur, le plus simple est de récupérer la PDOException qui se produira.
2. Prenez en compte les paramètres HTTP **obligatoires** `login`, `password`, `nom`, `prenom` (chaînes non vides) . Sauvegardez les valeurs reçues dans des variables globales de même nom.
3. Écrivez le script `createUser.php` qui reçoit ces 4 arguments et réalise la création de l'utilisateur, si aucun utilisateur de ce `login` n'existait déjà.

**Exercice 2 :**
Un site ne devrait **jamais** stocker les mots de passe. Tout d'abord l'administrateur de site n'a pas à connaître les mots de passe des usagers (c'est une donnée privée et souvent les utilisateurs emploient le même mot de passe pour plusieurs sites ou applications). Par ailleurs, en cas de vol de données, tous les mots de passe des utilisateurs peuvent se retrouver divulgués.
**Le principe de l'empreinte**
L'**empreinte** d'une donnée est le résultat d'un calcul effectué par un algorithme cryptographique (une fonction de hachage).
Si l'algorithme cryptographique est robuste, on considère qu'il est « impossible » (dans un laps de temps à l'échelle humaine) de retrouver à partir de l'empreinte ni la donnée d'origine ni une autre donnée qui aurait la même empreinte.
Au lieu de mémoriser le mot de passe, on va mémoriser son empreinte.

— **création/changement du mot de passe** : l'empreinte du mot de passe est calculée puis stockée dans la table des utilisateurs, à la place du mot de passe lui-même.
— **tentative de connexion** : l'empreinte du mot de passe proposé est calculée. Si elle est identique à l'empreinte stockée, le mot de passe proposé est considéré comme correct.

**Calcul d'empreinte en PHP**

1. **création/changement du mot de passe** : une empreinte est calculée par la fonction

   `password_hash($motDePasse,$algo)`

   La fonction peut appliquer plusieurs algorithmes ; nous utiliserons BLOWFISH. L'appel sera ainsi : `password_hash($motDePasse,CRYPT_BLOWFISH)`

   Dans le cas de BLOWFISH, le résultat est une chaîne de 60 caractères dont le préfixe (entre les signes $) indique le code de l'algo appliqué (BLOWFISH : `$2y$10$`). Viennent ensuite 22 caractères qui contiennent la « semence » utilisée (une valeur qui sert à initialiser l'algorithme et

que password_hash a généré aléatoirement). Les 31 derniers caractères constituent l'empreinte du mot de passe proprement dite .

Par abus de langage nous appellerons « empreinte » l'ensemble de cette chaîne de 60 caractères qui est à mémoriser en entier dans la table `users`, en remplacement du mot de passe.

2. **tentative de connexion** : la vérification du mot de passe proposé utilise la fonction

```
crypt($motDePasseProposé, $salt)
```

La valeur de salt à utiliser est l'empreinte (60 caractères) stockée dans la table, afin d'assurer que le calcul utilise le même algo et la même semence que pour le mot de passe d'origine.

Le mot de passe proposé est exact si

```
crypt($motDePasseProposé, $empreinteStockee) == $empreinteStockee
```

**Q 0 .** *Passage à la version « empreinte »*
En utilisant phpPgAdmin, **exécutez les commandes du fichier fourni : `hash_password.sql`** ce qui a pour effet

— de ne conserver dans la table `users` que les 2 utilisateurs qui s'y trouvaient à l'origine.

— de remplacer les mots de passe par leurs empreintes :

| | |
|---|---|
| `animal` | `$2y$10$m09t6nVdVgjw/qD7hkowFOd4AWtQI5jukA73Cq0D2mfZM0chMPda2` |
| `vapo` | `$2y$10$//IwI6O9e1YbiSp4W//6v.8s6AOo7w0hqQLhC6PqjSr.dC6.1XmOi` |

**Q 1 .** Modifiez la méthode `authenfier($login,$password)` de la classe `DataLayer`. La spécification de la méthode ne change pas.
La méthode doit maintenant d'abord aller chercher dans la base de donnée l'ensemble des informations (empreinte incluse) concernant l'utilisateur, s'il existe, puis vérifier la validité du mot de passe proposé (cf plus haut les explications sur la fonction `crypt(..)` ).
Vérifiez que l'authentification fonctionne de nouveau.
**Q 2 .** Modifiez la méthode `createUser()` de la classe `DataLayer` de manière à stocker l'empreinte du mot de passe et non le mot de passe.
Testez la création de nouveaux comptes et l'authentification.

**Exercice 3 :**
Cet exercice porte sur la prise en charge de données binaires (ici une image) : comment envoyer une image sur le serveur (upload), comment la stocker dans la base de données, comment concevoir une script qui permettra de la lire.
**Q 0 .** En utilisant phpPgAdmin, ajoutez à la table `users` les attributs

— `avatar` de type `bytea`

— `mimetype` de type `character varying (30)`

**Q 1 .**

1. créez un script sentinelle `lib/watchhdog_service.php` quasi identique à l'autre, excepté qu'en cas d'échec, au lieu de d'inclure la page de login, il effectuera :
`echo json_encode(['status'=>'error', 'message'=>'Échec de l\'authentification']);exit();`
2. lisez le document concernant l'upload de fichier et l'enregistrement dans une base Postgres (points 1 et 2 du document)
3. ajoutez à la classe `DataLayer` une méthode

```
/*
 * Enregistre un avatar pour l'utilisateur $login
 * paramètre $imageSpec : un tableau associatif contenant deux clés :
 *   'data' : flux ouvert en lecture sur les données à stocker
 *   'mimetype' : type MIME (chaîne)
 * résultat : booléen indiquant si l'opération s'est bien passée
 */
function storeAvatar($imageSpec, $login)
```

4. créez un script `uploadAvatar.php` qui reçoit en mode POST une image nommée `'image'`. Cette image sera rangée telle quelle dans la base de données. L'invocation de ce script est réservée à un utilisateur préalablement identifié. le script commencera donc par inclure la sentinelle ci-dessus.

   Ce script produira du JSON. En cas de succès il produira `{"status":"ok"}` et en cas d'échec : `{"status":"error", "message":"...."}`

   NB : pour l'instant l'upload renvoie donc une page JSON. C'est pas très « user friendly », mais c'est provisoire ;-)
5. testez en utilisant le formulaire prévu (vérifiez avec phpPgAdmin que des données ont bien été enregistrées).

**Q 2 .**

1. lisez les points 3 et 4 du document consacré aux données binaires.
2. ajoutez à la classe `DataLayer` une méthode

```
/*
 * Récupère l'avatar de l'utilisateur $login
 * résultat :  un tableau associatif contenant deux clés :
 *   'data' : flux ouvert en lecture sur les données
 *   'mimetype' : type MIME (chaîne)
 * ou FALSE en cas d'échec
 */
function getAvatar($login)
```

3. Écrire un script `getAvatar.php` qui reçoit en argument un identifiant (nommé `login`). Ce script produit une image, qui est l'avatar de l'utilisateur. Si l'utilisateur n'a pas d'avatar défini dans la base de données (la valeur de l'attribut `avatar` vaut NULL), le script renvoie l'avatar par défaut. NB : ce script ne nécessite pas d'authentification.
4. testez.
5. modifiez `views/pageAccueil.php` pour afficher le bon avatar.

**Q 3 .** Nous allons modifier le formulaire d'upload de fichier, de façon à ne plus renvoyer à une page JSON.
Éditez le fichier `views/pageFormUpload.php` pour inclure les fichiers javascript (en commentaires).
Consultez le contenu du fichier `formUpload.js` pour en comprendre le fonctionnement.
Testez.