POLYTECH MARSEILLE Aix*Marseille Université **Informatique – 4A**

Applications web et mobiles

TP n°2 : Backend – Authentification

Étape 6 – Création du script de connexion à la base de données

Dans votre backend, écrivez le script PHP de création de l'instance PDO, ainsi que le script de configuration contenant les paramètres de connexion (le nom du host, de la base, *etc.*). Personnellement, j'appelle traditionnellement ces scripts `mysqlConnect.php` et `config.php`.

Étape 7 – Test de la base de données

Écrivez un script PHP se connectant à votre base de données et affichant la liste de tous les utilisateurs. Les fonctions `fecthAll`, `fetchColumn` et `fetch`, *etc.* de PDO sont très bien documentées dans l'URL suivante : https://phpdelusions.net/pdo.

Testez votre script via l'application `postman` (n'oubliez pas de démarrer votre serveur Apache).

Modifiez votre script afin qu'il récupère via un POST un champ « login » et qu'il affiche également ce champ. Retestez votre script avec `postman`.

Étape 8 – Authentification – partie 1

Écrivez un script PHP `auth.php` qui démarre une nouvelle session ou en restaure une ancienne via l'instruction `session_start(['cookie_samesite' => 'Lax'])` [1].

Dans ce script, affichez le contenu du tableau `$_SESSION` via la fonction PHP `print_r()`, puis rajoutez un élément dont la clef est `test` et la valeur la chaîne « `test des sessions` ». Avec `postman`, accédez via un POST à `auth.php`. Vous devriez voir en bas de la fenêtre de `postman`, dans le *body*, un tableau vide. Vous devriez également voir dans la rubrique *Cookies*, un *cookie* dont le nom est `PHPSESSID`. C'est votre cookie de session.

Dans `postman`, recliquez sur le bouton `Send` afin de réaccéder à votre script `auth.php`. Maintenant, dans le *body*, vous devriez voir ce que vous aviez ajouté dans la variable `$_SESSION`.

Les variables de session sont stockées par défaut une trentaine de minutes sur votre serveur. Pendant cette demi-heure, si vous vous reconnectez, vos scripts PHP retrouvent toutes les informations stockées

1. Rappel : les sessions permettent de stocker sur le serveur des informations concernant le client. Pour cela, il suffit de lire ou d'écrire dans le tableau superglobal `$_SESSION`. L'option `['cookie_samesite' => 'Lax']` sert à éviter un problème de CORS du cookie de session (cf. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite).

dans `$_SESSION`. L'idée de réaliser les authentifications via des variables de session consiste donc à rajouter dans `$_SESSION` des informations quand l'utilisateur a réussi à s'authentifier. Par la suite, quand l'utilisateur continue à surfer sur votre site, il suffit de vérifier que `$_SESSION` contient bien les informations en question, ce qui assure qu'il s'est bien authentifié.

Étape 9 – Authentification – partie 2

Supprimez du fichier `auth.php` toutes les instructions que vous avez écrites excepté le `session_start()`. Maintenant, écrivez deux fonctions :

1. une fonction `authenticate()` qui :
   (a) vérifie que la variable `$_POST` contient des champs `login` et `password` (cf. les fonctions `array_key_exists` et `isset` de PHP, (https://www.php.net/manual/fr/funcref.php, sous-sections concernant les « tableaux » et « Extensions relatives aux variables »).
   (b) si les champs existent, vérifiez que votre base de données contient bien ce couple (login, password).
   (c) Si c'est le cas, votre fonction doit placer dans la variable `$_SESSION` des informations permettant d'identifier l'utilisateur lors de ses connexions futures, puis elle retourne la valeur `true` pour indiquer que l'identification s'est bien passée.
   (d) S'il manque dans le `$_POST` le login et/ou le password, ou bien si votre base de données ne les contient pas, la fonction ne modifie pas `$_SESSION` et retourne la valeur `false`.
2. une fonction `isAuthenticated()` qui renvoie `true` si les clefs des informations que vous avez ajoutées à `$_SESSION` en cas d'authentification réussie existent bien, et renvoie `false` sinon.

Testez vos fonctions via postman. Pour cela, il vous suffit simplement de rajouter en bas de votre fichier PHP un appel à la fonction `authenticate()` ou bien à `isAuthenticated()` et d'afficher un message en fonction de la valeur retournée. Quand vos tests sont OK, supprimez ces appels.

Étape 10 – Helper

Téléchargez le script `helper.php` sur l'URL https://pageperso.lis-lab.fr/christophe.gonzales/teaching/mobile/ressources/fr-FR/helper.php et placez-le dans votre répertoire `backend`. Ce script contient :

1. des instructions qui modifient les headers des pages que retournera votre *backend* afin de contourner une restriction de sécurité appelée CORS (cross-origin resource sharing), cf. https://developer.mozilla.org/fr/docs/Web/HTTP/CORS [2].
2. deux fonctions `sendMessage` et `sendError` qui simplifient le transfert de données vers *Angular*. Quand on transfère des informations à *Angular*, on le fait en principe en utilisant le format JSON. Ces fonctions créent le format en question. Quand vous souhaitez renvoyer un message d'erreur à *Angular*, il suffit d'appeler `sendError` en lui passant en paramètre une chaîne de caractères précisant l'erreur qui s'est produite. Si tout s'est bien passé, en revanche, utilisez `sendMessage` en lui passant en paramètre les données que vous souhaitez renvoyer à *Angular*. Notez que la dernière instruction de ces fonctions est `die`. Cela permet de stopper le script PHP : après un `die`, aucune autre instruction PHP n'est exécutée.
3. des instructions qui utilisent les fonctions que vous avez écrites dans les deux étapes précédentes : on y précise que si le script PHP auquel l'utilisateur essaye d'accéder ne s'appelle

2. Dans vos TPs, vous allez tester votre *frontend Angular* sur le port 4200 de votre machine et votre serveur Apache est, lui, sur le port 8080 en TP ou sur le port 80 chez vous. Par conséquent, vous utilisez deux serveurs différents et cela induit le CORS, qui empêche à Apache et *Angular* de converser entre eux. Les headers de `helper.php` suppriment cette restriction.

pas `checkLogin.php` (cf. l'étape suivante) et si l'utilisateur n'est pas authentifié, on renvoie un message d'erreur JSON et on stoppe l'exécution de PHP (`die`), sinon on ne fait rien. Par conséquent, si vous incluez `helper.php` en début de tous vos scripts (cf. `require_once`), ceux-ci exécuteront leur code si et seulement si l'utilisateur est bien authentifié.

Étape 11 – Authentification – checkLogin.php

Créez un script `checkLogin.php` que l'on contactera via un POST. Ce script PHP vérifiera que les champs `login` et `password` du POST correspondent bien à un utilisateur référencé dans votre base de données. Le cas échéant, il renverra un message JSON contenant une chaîne vide. Sinon, il renverra un message d'erreur JSON précisant que le login/password est invalide. Évidemment, pour cela, vous vous appuierez sur les fonctions du fichier `helper.php` ainsi que sur votre fonction `authenticate()`.

Testez votre script `checkLogin.php` via `postman`.