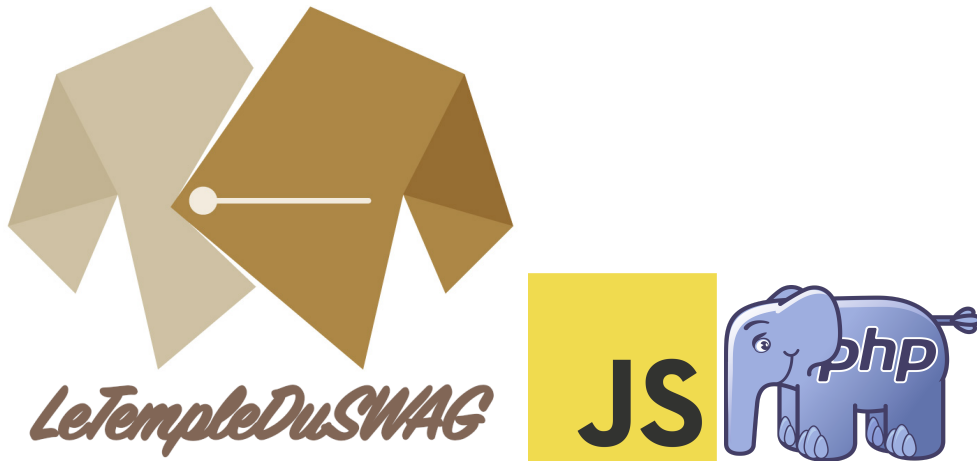


Web avancé – L2 MIASHS

TD 11 – LeTempleDuSwag – les bases des utilisateurs



Votre site du temple du Swag permet désormais de filtrer les vêtements par cible du produit ainsi que d'ajouter et supprimer des articles. Dans ce TD, nous allons voir comment limiter l'accès à notre super site à l'aide de la connexion des utilisateurs. Attention toutefois, nous allons limiter l'accès, mais pas mettre en place des rôles d'utilisateurs pour définir un accès plus précis (administrateur, visiteur, etc.).

Voici notre planning très simplifié :

Feuille de route du TempleDuSwag :

- ☒ ~~Mise en place du parcours des articles~~
- ☒ ~~Ajout d'un filtre par genre en AJAX~~
- ☒ ~~Ajout des fonctionnalités de suppression et d'ajout avec formulaire dédié~~
- ☐ Ajout d'une inscription et d'une connexion de l'utilisateur (ce TD 😊)

Pour ce TD et gérer les utilisateurs, nous aurons besoin des éléments suivants :

- une page avec un formulaire de connexion / inscription classique (pas en AJAX)
- une gestion de la redirection en fonction de l'authentification (par session)
- une table dédiée dans la BDD avec mot de passe crypté

Objectif : gérer les utilisateurs (inscription / connexion / session)

Cible : les étudiants à l'aise avec PHP et les requêtes en base de données MySQL. Le Javascript n'est pas vraiment nécessaire pour ce TD.

Durée : très variable, en fonction de votre aisance dans l'usage combiné des différentes notions, et surtout de PHP.

Un conseil important : même si ce TD peut, à certains points, vous sembler facile, ne sautez pas d'étape !

Chaque exercice se distingue par ce logo  !

Dans ce TD, le code est intégré sous forme d'image afin que vous ne puissiez pas simplement le copier-coller.

Les requêtes SQL vous seront données, il sera à votre charge de les adapter dans votre code PHP. Enfin, certaines explications données dans les TDs précédents ne seront pas systématiquement répétées !

L'objectif du TD est de ne permettre l'accès à notre site que si l'utilisateur est authentifié. Pour cela, nous devons lui permettre de créer un compte et de se connecter / déconnecter. Un utilisateur non connecté sera alors automatiquement renvoyé vers la page de connexion.

Création de la table des utilisateurs

La première étape est évidente : il nous faut créer une représentation des utilisateurs dans la base de données. Pour cela, créons une table minimaliste qui ne contient que l'adresse email en clé primaire et le mot de passe de l'utilisateur. En voici la requête SQL à lancer dans phpMyAdmin :

```
1 CREATE TABLE `produits`.`users` (`email` VARCHAR(255) NOT NULL , `mdp` VARCHAR(255) NOT NULL , PRIMARY KEY (`email`)) ENGINE = InnoDB;
```

Cette requête vous créera alors une table "users" (signifiant "utilisateurs") dans la base de données **produits**. Nous pourrions créer une autre base de données pour gérer les utilisateurs, cela dépend des besoins et de la taille de notre application. Vous verrez que nous ne verrons pas de relation entre les utilisateurs et les produits. Il est évident que pour une application à plus grande échelle du SQL plus avancé sera nécessaire pour une représentation plus propre des données.

Page de connexion

C'est dans la page de connexion que nous allons mettre l'essentiel des nouveautés. Pour cela :



- Créez un fichier **login.php**

- Mettez-y le code HTML suivant

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>TempleDuSWAG</title>
    <link rel="icon" href="img/logo.png" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46Mgn0M80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.css">
  </head>
  <body>
    <div class="container">
      <div class="row">
        <div class="form-group">
          <form action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
            <div class="form-group">
              <input type="email" name="email" class="form-control" aria-describedby="emailHelp" placeholder="Adresse email">
            </div>
            <div class="form-group">
              <input type="password" name="mdp" class="form-control" placeholder="Mot de passe">
            </div>
            <div class="form-group form-check">
              <input type="checkbox" name="inscription" class="form-check-input" id="modeInscription">
              <label class="form-check-label" for="modeInscription">Inscription</label>
            </div>
            <button type="submit" class="btn btn-primary">Envoyer</button>
          </form>
        </div>
      </div>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-kenU1KFdBIE4zVF0s0G1M5b4hcxpyD9F7jL+jjXkk+Q2h455rYXK/7HAuoJl+0I4" crossorigin="anonymous"></script>
  </body>
</html>
```

Comme vous le voyez, ce code est similaire à notre page **index.php** à l'exception que cette page ne contient qu'un seul formulaire. C'est d'ailleurs précisément ce formulaire que vous devez créer, le reste vous pouvez le copier du fichier **index.php**. Ce qui nous donne le résultat suivant pour la page **localhost/login.php** (ou équivalent selon votre installation) :

☐ Inscription

Vous remarquerez également que dans l'attribut action du formulaire, nous faisons un lien direct même la même page.

```
<form action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
```

Cela va nous permettre de récupérer le résultat de ce formulaire par la méthode POST. Chaque clé sera égale à l'attribut **name** donné pour chaque balise **input**.

Fonctions PHP et gestion des données

Maintenant, passons aux choses sérieuses 😊 ! Créez une balise php après le formulaire. Nous allons y mettre des fonctions et un contrôle sur la requête POST. Cela vous donnera une organisation similaire à :

```

<?php
function connect() { ...
}

function addUser($conn, $email, $mdp) { ...
}

function checkIfExists($conn, $email) { ...
}

function checkUser($conn, $email, $mdp) { ...
}

$conn = connect();

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (count($_POST)) { ...
    }
}

$conn->close();
?>

```

Créez donc la balise php et mettez-y :



- une fonction de connexion à la base de données (**connect()**) comme vu lors des TDs précédents
- une fonction d'ajout d'un utilisateur **addUser()** qui accepte l'objet de connexion, l'email et le mot de passe. Cette fonction doit :

- Crypter le mot de passe avec la fonction **hash()** comme ceci

```
hash('sha384', $mdp)
```

- exécuter la requête SQL suivante en donnant l'email et le mot de passe crypté. Par exemple :

```
"INSERT INTO users(`email`, `mdp`) VALUES ('gael@lycos.fr', 'mdp crypté')"
```

À vous de bien préparer la requête SQL.

Cela ne suffit pas. Il nous faut deux autres fonctions, une pour vérifier si l'utilisateur existe déjà dans la base de données afin de ne pas permettre d'écraser un utilisateur existant et une fonction pour vérifier les identifiant/motdepasse données avec ce qu'il y a dans la base de données.

- Créez une fonction **checkIfExists()** qui accepte en arguments l'objet de connexion et l'email de l'utilisateur.

- Cette fonction doit exécuter la requête suivante (à adapter vous-même)

```
"SELECT (email) FROM users WHERE email = gael@lycos.fr AND mdp = youhouu ;"
```

Comme vous le voyez, pas besoin de récupérer tous les utilisateurs, juste compter sur la base de la clé primaire (donc il y aura soit 0 en retour, soit 1). Attention ! Quand vous allez parcourir le résultat, l'objet **\$row** sera alors un simple numérique ! (soit 0, soit 1)

- Cette fonction doit retourner le nombre d'éléments présents (donc le résultat numérique de la requête SQL)
- Créez une fonction **checkUser()** qui vérifie si l'email et le mot de passe crypté sont bien présents dans la base de données. Une vérification des identifiants en somme



- Cette fonction accepte trois arguments : l'objet de connexion, l'email et le mot de passe.
- Cette fonction doit crypter le mot de passe comme vu plus haut.
- Elle exécute la requête SQL suivante (à adapter) et récupère le tableau associatif représentant les infos de l'utilisateur.

```
"SELECT (email) FROM users WHERE email = 'gael@lycos.fr' AND mdp = 'youhouu' ;"
```

Encore une fois, vous remarquerez que nous ne prenons que l'email. Ou, pour être plus exact, nous ne récupérons pas le mot de passe. À quoi bon le récupérer puisque notre requête SQL fait déjà la comparaison du mot de passe crypté.

- Cette fonction doit retourner le tableau associatif représentant l'utilisateur.

Contrôle du résultat du formulaire

Maintenant, il ne reste plus qu'à contrôler l'accès via la méthode POST (celle de notre formulaire qui renvoie vers cette même page).

Vous devez donc :



- Vérifiez si la méthode de requête du serveur utilisée est bien la méthode POST.
- Si c'est le cas, vérifiez si la balise **\$_POST** possède plusieurs clés
- À partir de la balise **\$_POST** récupérez l'email et le mot de passe que vous mettrez dans une variable chacune. Le nom des clés est égal à l'attribut **name** dans le formulaire HTML.
- Récupérez le statut du bouton à cocher pour indiquer le mot "inscription" ou non. Pour cela, si le bouton n'est pas coché, la variable **\$_POST** ne possède pas la clé correspondante. S'il est coché, elle possède cette clé. Utilisez **isset()** pour faire de cette logique un simple booléen à mettre dans une variable 🤖.

Maintenant, nous allons gérer le mode inscription. Donc si le bouton est coché alors :



- Vérifiez l'existence de l'utilisateur dans la base de données à l'aide de la fonction **checkIfExists()**.
 - S'il existe, affichez simplement un paragraphe HTML indiquant par exemple "l'utilisateur existe déjà ! 😞".

- S'il n'existe pas, utilisez la fonction `addUser()` puis affichez un paragraphe indiquant que l'utilisateur a bien été créé et qu'il faut désormais se connecter.

Grâce à nos super fonctions, la logique est assez simple à suivre n'est-ce pas ? C'est tout l'intérêt d'un code bien organisé 😊 Et quoi de mieux que des fonctions pour bien organiser un code ? ... des classes, mais ce sera pour une prochaine fois 😊

Maintenant, nous allons gérer le mode de connexion (c'est-à-dire quand le bouton d'inscription n'est pas coché). Pour cela, nous allons devoir prendre en compte les sessions.



- Vérifiez les identifiants fournis avec ceux présents dans la base de données à l'aide de notre super fonction **`checkUser()`**
- Ajoutez une clé "email" à la super globale `$_SESSION`. Mettez y l'email de l'utilisateur.
- Faites une redirection vers la page d'index à l'aide du header suivant

```
header("Location: index.php");
```

- Utilisez la fonction **`die()`** ou **`exit()`** pour s'assurer de l'arrêt du script à ce moment-là, même si la redirection n'a pas fonctionné.

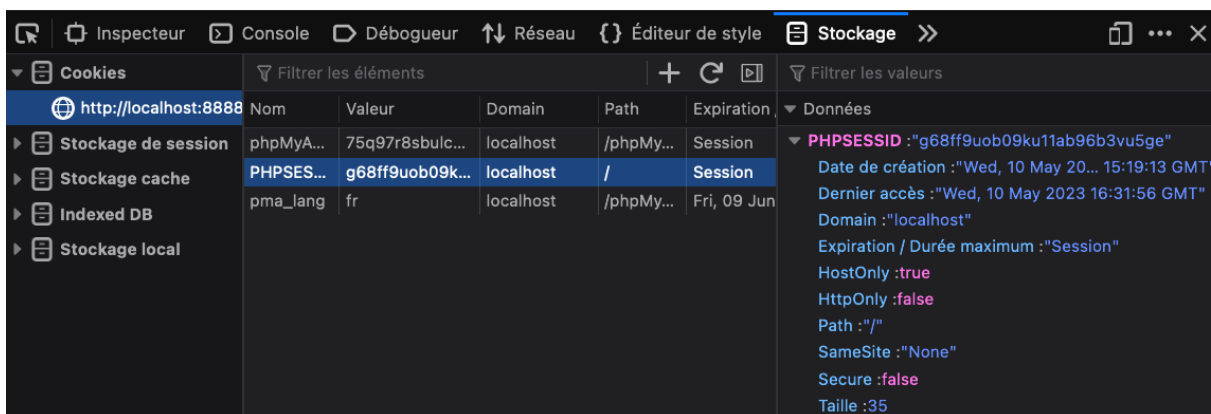
Maintenant il ne nous reste plus qu'à ajouter l'activation des sessions dans ce script PHP.



Pour cela, tout en haut du fichier, à avant la moindre balise, ajoutez `session_start()`

```
1  <?php
2  // permet de démarrer la session
3  // et donc les activer
4  session_start();
5  ?>
```

Et voilà ! 🎉 Vous pouvez désormais tester votre fichier **login.php** et bien voir votre inscription, puis votre connexion qui doit vous renvoyer vers **index.php**. Si vous regardez bien dans la console du navigateur, vous verrez alors dans l'onglet "storage" les différents cookies présents. L'utilisation de la session PHP a créé un cookie automatiquement comme visible ci-dessous :



Nom	Valeur	Domain	Path	Expiration
phpMyA...	75q97r8sbulc...	localhost	/phpMy...	Session
PHPSES...	g68ff9uob09k...	localhost	/	Session
pma_lang	fr	localhost	/phpMy...	Fri, 09 Jun

Données

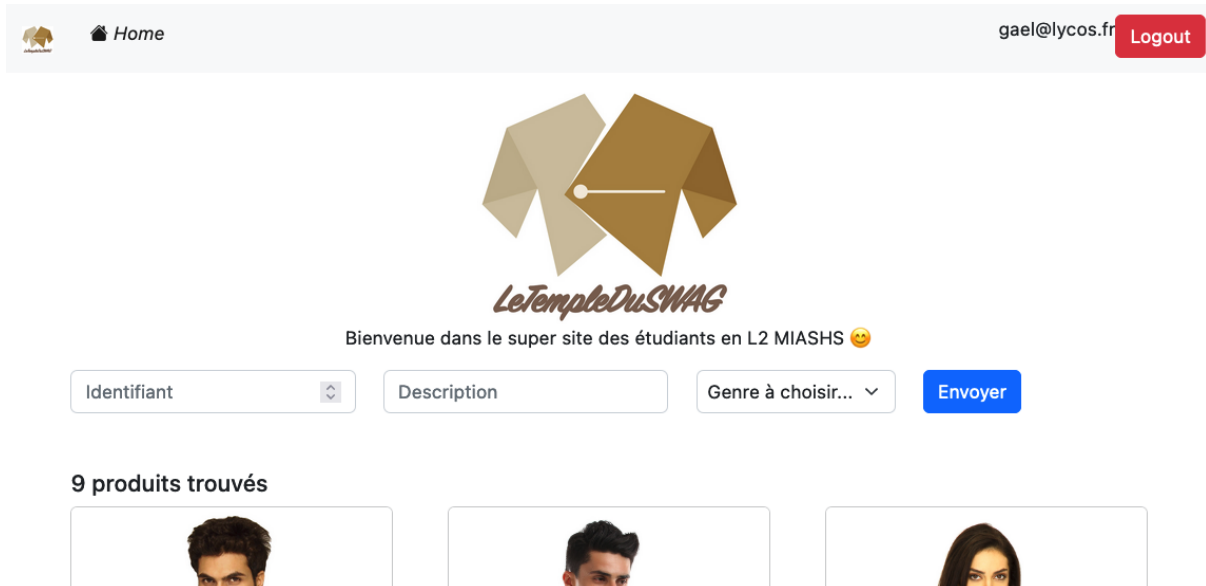
- PHPSESSID :** "g68ff9uob09ku11ab96b3vu5ge"
- Date de création : "Wed, 10 May 20... 15:19:13 GMT"
- Dernier accès : "Wed, 10 May 2023 16:31:56 GMT"
- Domain : "localhost"
- Expiration / Durée maximum : "Session"
- HostOnly : true
- HttpOnly : false
- Path : "/"
- SameSite : "None"
- Secure : false
- Taille : 35

Pas mal n'est-ce pas ?

Il manque cependant une chose... si nous pouvons librement accéder à `index.php`, à quoi bon instaurer un système de connexion ? 😞 Nous allons de ce pas y remédier ! 😊

Limiter l'accès à la page d'index

Nous devons limiter l'accès à **index.php**. Pour cela, nous devons vérifier si dans la session il y a un utilisateur authentifié et ainsi l'afficher dans la barre de navigation aux côtés d'un bouton de déconnexion.



Commençons par contrôler l'accès à cette page :



- Créez de nouveau du code PHP pour activer les sessions avec `session_start()`
- Juste après, vérifiez si la clé "email" est présente dans l'objet `$_SESSION` (utilisez `isset()`). Si ce n'est pas le cas, utilisez `header()` pour rediriger vers `login.php`. N'oubliez pas de quitter le script ensuite avec `exit()`

Vous pouvez désormais essayer d'accéder à cette page sans être connecté (pour vous déconnecter, supprimez les cookies à l'aide du panneau des développeurs de votre navigateur).

Affichons le nom de l'utilisateur connecté et permettons sa déconnexion.



- Dans la barre de navigation, créez une nouvelle liste non ordonnée après la première et donnez lui les classes "navbar-nav" et "navbar-right" (pour que son contenu soit placé à droite).
 - Mettez-y un élément de liste qui possède, à l'aide de PHP, l'email provenant de la session.
 - Mettez-y un autre élément de liste, qui possède un formulaire dont l'action redirige vers ce même fichier (voir ce que nous avons fait plus haut) par la méthode POST et qui possède un bouton HTML de classes "btn" et

“btn-danger” (pour un rouge vif), ainsi que l’attribut **name** de valeur “logout” et l’attribut **type** de valeur “submit” (pour envoyer le formulaire).

- Dans la balise PHP du tout début du fichier
 - Si l’accès se fait via une méthode POST et que la variable **\$_POST** possède la clé “logout”, alors utilisez [session_destroy\(\)](#) pour détruire la session, puis redirigez vers la page **login.php**.

Tout est parfait !!! C’est magnifaïk 🥳 ! Vous pouvez désormais vous amuser à vérifier l’accès sans être connecté, à vérifier l’inscription et la connexion. 🎉

Bien entendu, tout n’est pas parfait, et il nous faudrait une implémentation plus précise de l’authentification des requêtes pour le fichier **api.php**. Ces authentifications par AJAX ne sont pas les mêmes et utilisent généralement des jetons donnés par [cookies sécurisés](#) (uniquement en HTTPS). Toutefois, nous n’irons pas jusque là dans le cadre de ce cours car nous n’avons pas réellement mis en place une API REST. 😊

Vous avez désormais des bases en PHP et devriez être plus à l’aise en Javascript, notamment en AJAX. Maintenant, nous pouvons accéder au plaisir ULTIME nommé : “la famosa liste de complétion terminée” ! Regardez, admirez, contemplez !!!

Feuille de route du TempleDuSwag :

- ☒ ~~Mise en place du parcours des articles~~
- ☒ ~~Ajout d’un filtre par genre en AJAX~~
- ☒ ~~Ajout des fonctionnalités de suppression et d’ajout avec formulaire dédié~~
- ☒ ~~Ajout d’une inscription et d’une connexion de l’utilisateur~~

YEAH ! C’est fini !



Liens utiles

- <https://www.php.net/manual/fr/>
- <https://www.w3schools.com/php/default.asp>
- <https://emojipedia.org/>