Projet Web & Données
TP de prise en main de Php

Sophie Rousseau et Jonas Mézière

ESEO DIS 2022-23 - S6 PWND - BDR

Introduction

Ce TP a pour objectif principal de vous permettre de vous lancer sur le projet avec une base commune à toutes les équipes.

Ce TP se présente sous la forme d'un tutoriel à suivre pour appréhender les différents concepts devant vous servir dans la suite de vos développements.

Ainsi, à la fin de ce TP, vous devriez avoir une première partie du projet mettant en uvre les concepts suivants :
- construction d'une page d'accueil avec gestion de la connexion des utilisateurs,
- gestion de l'inscription d'un nouvel utilisateur,
- utilisation d'une base de données,
- utilisation de fonctions javascript.

1 Outils de développement et déploiement

Pour effectuer vos développements vous pouvez utiliser :
- un éditeur de text type notepad, notepad++
- Visual Studio Code

Les fichiers résultats de vos développements doivent être envoyés sur le serveur pour tester votre application web.

Si vous utilisez la machine virtuelle, vous allez devoir envoyer vos différents fichiers dessus afin de pouvoir tester vos développements.

Pour ce faire, le protocole devant être utilisé est SCP.

Ainsi, selon vos systèmes d'exploitation, vous allez pouvoir utiliser des outils divers :
- un terminal et des commandes.
  - la commande pour envoyer le fichier *test.php* à la racine du dossier de l'utilisateur pwnd :
    ```
    scp test.php pwnd@192.168.56.80:/var/www/html/pwnd
    ```
  - la commande pour envoyer plusieurs fichiers ayant la même extension (par exemple extension php) :

```
scp *.php pwnd@192.168.56.80:/var/www/html/pwnd
```

— la commande pour envoyer un dossier *exemple* entier et le copier avec la même arborescence sur le serveur :
```
scp -r exemple pwnd@192.168.56.80:/var/www/html/pwnd/exemple
```

— FileZilla, extension de VSCode, autre outil que vous maîtrisez et qui vous permet d'envoyer vos fichiers sur le serveur.

2 Première étape - 1 page

L'objectif de ce TP est de vous faire mettre en uvre une première version de la page *index.php*.

Lorsque vous prendrez la main pour votre partie de l'application, vous pourrez reprendre vos travaux préparatoires réalisés durant la première séance de projet et qui vous a permis de produire 2 fichiers : *index.html* et *css.css*.

Dans un premier temps, le look de la page n'est pas notre priorité. L'objectif est de construire une page et comprendre le langage php.

1-On commence une page

Créez un dossier sur votre ordinateur dans lequel vous allez pouvoir ranger tous les fichiers du projet web.

Une fois ce dossier créé,

— si vous avez choisi la solution lignes de commandes pour l'envoi des fichiers, ouvrez un terminal partir de ce dossier ;

— si vous utilisez Visual Studio Code, ouvrez le dossier créé.

Dans le dossier, crééun nouveau fichier *index.php* dans lequel vous ajoutez le code suivant :

```
 <!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="utf-8">
    <title>Accueil projet RS ESEO</title>
</head>

<body>
    <div>
        <h2>Bienvenue sur RS ESEO</h2>

    </div>
</body>

</html>
```

Une fois sauvegardé, vous envoyez ce fichier sur le serveur et vous accédez l'application via l'url :

http ://192.168.56.80/pwnd

Vous devriez avoir pour résultat une page dont l'onglet à pour titre *Accueil projet RS ESEO* et donc le contenu est un titre *Bienvenue sur RS ESEO*.

Pour l'instant, rien de bien compliqué et surtout, que du code HTML. Donc, pourquoi nommer ce fichier *index.php* ?

Patience ! On y vient !

2-Affichage de texte avec Php

Pour ajouter du code Php dans une page, on utilise les balises

```
<?php

?>
```

Entre ces balises, on peut ajouter notre code Php.
Par exemple, pour afficher un texte, on utilise **echo**
Dans notre page, on va remplacer le texte présent dans la balise title par les informations suivantes :

```
 <title><?php echo 'Accueil projet RS ESEO' ?></title>
```

Vous sauvegardez, vous envoyez le fichier sur le serveur, et vous rafraichissez la page web.
Vous devriez ne voir aucune différence avec la version précédente.
Donc, s'il n'y a aucune différence, pourquoi complexifier le code ?
Et bien, parce qu'on va pouvoir également afficher la valeur de variables grace à *echo*.
On va donc définir une variable qui va contenir le titre de la page et afficher la valeur de cette variable.
En tout début de page, avant la balise *DOCTYPE*, vous ajoutez :

```
<?php
    $_TITRE_PAGE = 'Accueil projet RS ESEO';
?>
```

Puis vous remplacez le texte affiché par :

```
 <title><?php echo $_TITRE_PAGE ?></title>
```

On sauvegarde et on envoie sur le serveur et, on a, le même résultat que précédemment.
On va faire un petit arrêt ici pour aller étudier le code source de la page reçu par votre navigateur.
Un clic-droit dans la page devrait vous permettre d'afficher le code source.
Et, vous devriez constater que le code affiché est le même code que dans la toute premiere version du fichier.
Mais pourquoi donc ?
Et bien, comme vu durant le cours, le code Php est interprété par le moteur Php présent sur le serveur avant d'être envoyé au client (votre navigateur) dans un format html.

3- Un premier formulaire

Après le titre h2, on ajoute le code suivant permettant de créer un formaulaire de connexion.

```
<div>
    <form>
<p>Connexion 2</p>
<p>
    <label for="mail">Email</label>
    <input id="mail" name="mail" type="text">
</p>
<p>
    <label for="defaultLoginFormPassword">Mot de passe</label>
    <input name="password" type="password" id="defaultLoginFormPassword">
</p>
<button name="connexion_submit" value="1" type="submit">Connexion</button>
    </form>
</div>
```

On sauvegarde, on envoie sur le serveur et on rafraichit la page.
Et pour tester, on entre *toto* dans le champ *Email* et *1234* dans le champ *Mot de passe*.
Après avoir cliqué sur le bouton *Connexion*, vous devriez constater le résultat suivant :
— La page est rafraichie et les champs texte sont vides ;

— l'adresse de la page a été modifiée et contient les informations suivantes :
`http://192.168.56.80/pwnd/?mail=toto&password=1234&connexion_submit=1`

On se rend compte ici que, bien que nous ayons utilisé un champ text de type password, lors de la validation du formulaire, la valeur de chaque entrée du formulaire - ici mail, password et la valeur du bouton - est transmise à la page en clair dans l'url après le symbol ?.

Pour éviter ce mode de fonctionnement et permettre un premier niveau de sécurité et éviter la transmission en clair d'informations de connexion, on ajoute à la balise form un attribut method avec la valeur post.

```
<form method="post">
```

On sauvegarde, on envoie sur le serveur et on accède à la page http ://192.168.56.80/pwnd/

Si on entre les mêmes valeurs que précédemment, le clic sur le bouton Connexion a pour résultat :

— les champs texte sont vides ;

— l'url n'est pas modifiée.

Mais que sont devenues les valeurs entrées ?

Et bien toutes les valeurs du formulaire sont stockées dans une variable de type tableau **$_POST**.

Nous allons donc pouvoir récupérer les valeurs saisies dans le formulaire et, par exemple, les afficher.

Entre les balise php du début du fichier, on ajoute le code suivant permettant un affichage des valeurs présentes dans le tableau $_POST.

```
echo $_POST['connexion_submit'].'</br>';
echo $_POST['mail'].'</br>';
echo $_POST['password'];
```

echo affichage

$_POST['mail'] récupération de la valeur de l'entrée ayant un attribut name avec la valeur mail.

. opérateur de concaténation entre 2 chaînes de caractères.

On sauvegarde, on envoie sur le serveur et on accède à la page.

Vous devriez remarquer que le titre de la page est plus bas et qu'une zone blanche est présente.

C'est normal, les 3 echo ajoutés affichent des chaînes vide car, le formulaire n'ayant pas été validé, aucune valeur n'est présente dans le tableau POST.

Si vous renseignez des valeurs pour les 2 champs et que vous validez, vous devriez voir s'afficher en haut de la page quelque chose comme :

```
1
toto@mail.fr
1234
```

Ceci valide l'envoi des valeurs des champs du formulaire.

Il va donc être possible d'effectuer un traitement en utilisant ces valeurs, par exemple, vérifier que les informations saisies permettent la connexion ou non.

Et dans le cas de la validation d'une connexion, on devrait pouvoir conditionner les éléments à afficher à l'utilisateur.

4- Suis-je connecté ?

L'idée ici est d'afficher le formulaire lorsque l'utilisateur n'est pas connecté ou d'afficher un message de bienvenue s'il l'est.

Dans la section php en haut de la page, on définit une variable *compte* initialisée avec la valeur *false*.

Ajoutez le code php dans le corps de la page permettant d'afficher le formulaire lorsque la variable compte a la valeur false, et affiche un message de bienvenue lorsque la variable a la valeur true ;

Dans la section php en haut de la page, on va détecter l'envoi du formulaire, et, si le mot de passe est *network*, on fait en sorte de considérer l'utilisateur comme connecté.

Vous envoyez votre fichier sur le serveur et vous testez.

Ce que vous derviez constater, c'est que, si vous réaccédez à la page, vous n'êtes plus connecté.

Ceci vient du fait que notre variable *compte* a une durée de vie liée à la requête envoyée. Si la page est rechargée, une nouvelle requête est envoyée et donc, la variable reprend sa valeur initiale.

Vous devriez avoir une page dont le code est le suivant :

```
 <?php
    $_TITRE_PAGE = 'Accueil projet RS ESEO';

    $compte = false;

    if($_POST['connexion_submit'] == 1){
        if($_POST['password'] == 'network'){
            $compte = true;
        }
    }

?>
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="utf-8">
    <title><?php echo $_TITRE_PAGE ?></title>
</head>

<body>
    <div>
        <h2>Bienvenue sur RS ESEO</h2>

        <?php
        if(!$compte){
        ?>
        <div>
  <form method="post">
      <p>Connexion</p>
      <p>
  <label for="idmail">Email</label>
  <input id="idmail" name="mail" type="text">
      </p>
      <p>
  <label for="defaultLoginFormPassword">Mot de passe</label>
  <input name="password" type="password" id="defaultLoginFormPassword">
      </p>
      <button name="connexion_submit" value="1" type="submit">Connexion</button>
  </form>
        </div>
        <?php
        }else{
        ?>
        <div>
            <h2>Vous êtes connecté !</h2>
        </div>

        <?php
```

```
        }
        ?>
    </div>
</body>

</html>
```

L'idée lorsque l'on se connecte à une application est de rester connecté sans avoir à resaisir ses identifiants à chaque action.

Pour cela, on va utiliser l'objet session liée à l'application : $_SESSION. Cette variable est un tableau qui a pour rôle de contenir l'ensemble des valeurs devant être conservées sur toute la durée de la session liée à l'application.

Pour utiliser les variables de session, il faut activer la session. Ceci se fait par l'appel à la fonction *session_start()*.

L'accès aux valeurs stockées dans le tableau *$_SESSION* se fait alors par le nom de la variable recherchée.

Dans notre cas, on modifie la section php du début de la page par :

```
<?php
    $_TITRE_PAGE = 'Accueil projet RS ESEO';

    session_start();


    if($_POST['connexion_submit'] == 1){ //on détecte que l'action
                                         //vient du bouton de la connexion
        if($_POST['password'] == 'network'){
            $_SESSION['compte'] = true;
        }
    }

?>
```

L'appel à *session_start()* active la session.

$_SESSION['compte'] permet de récupérer la valeur correspondant au nom *compte*.

Une fois la page mise à jour pour fonctionner et le fichier envoyé sur le serveur, vous devriez pouvoir vous connecter et une fois ceci fait, rafraichir la page ne vous fait pas perdre cette connexion.

Il faut donc maintenant permettre la déconnexion.

5-Je me déconnecte

Ajoutons un lien dans la page qui ne soit accessible que pour les utilisateurs connectés.

```
<a href="http://192.168.56.80/pwnd?logout=1">Se déconnecter</a>
```

Ce lien permet d'être dirigé vers la racine du site avec un paramètre dans l'url qui sera testé dans le code php pour déconnecter l'utilisateur.

Ensuite, pour la gestion de cette déconnexion, ajoutez le code suivant dans la section php du début de la page, après le *if* de la gestion de la connexion. Les explications arrivent après.

```
if($_GET['logout'] == 1) {
    unset($_SESSION['compte']);
    header("Location: ./");
}
```

Le paramètre *logout* étant directement ajouté dans l'url, ce paramètre est stockée dans le tableau $_GET.

On le récupère donc et si la valeur vaut 1, on réalise les actions permettant la déconnexion.

La méthode *unset* détruit une variable.

documentation : `https://www.php.net/manual/fr/function.unset.php`

Ici, on détruit la variable *compte* présente dans le tableau $_SESSION.
Ensuite, il faut recharger la page afin de ne plus avoir le paramètre *logout* disponible dans la variable $_GET.
On réalise cette action par l'appel de la fonction *header*.

documentation : `https://www.php.net/manual/fr/function.header.php`

6-Quelques fonctions Php

Pendant que l'on est dans l'utilisation de fonctions php, en voici deux autres que nous vous invitons fortement à utiliser :

isset() détermine si une variable est déclarée et est différente de *null*
documentation : `https://www.php.net/manual/fr/function.isset.php`

empty() détermine si une variable est vide, c'est-à-dire si elle n'existe pas ou si sa valeur équivault à *false*.
documentation : `https://www.php.net/manual/fr/function.empty.php`

On peut donc modifier la page index.php comme suit :

— Dans le corps de la page, le test de la variable de session compte permettant d'afficher ou non le formulaire donne :

```
<?php
   if(empty($_SESSION['compte'])){
?>
```

— Dans la section php du haut de la page, dans la gestion de la connexion, avant de tester la valeur de la variable *connexion_submit*, on vérifie qu'elle soit déclarée.

```
if(isset($_POST['connexion_submit']) && $_POST['connexion_submit'] == 1){
```

7- Une première connexion à une base de données

C'est bien tout ça mais, pour vérifier une connexion, il faut pouvoir aller chercher les informations où elles sont stockées.

Dans la section Php du début du fichier nous allons donc, dans la condition qui permet de détecter et traiter la demande de connexion, ajouter les étapes suivantes :

Dans le cas où le mail et le mot de passe ont bien été renseignés :

1. se connecter à la base de données
2. créer la requête permettant de récupérer l'identifiant de l'étudiant lorsqu'il existe
3. exécuter la requête
4. vérifier la présence d'un résultat et si oui, on affecte la valeur de l'identifiant de l'étudiant dans la variable de session *compte*.
5. fermeture de la connexion

Avant de se lancer dans ces opérations, nous allons faire en sorte de voir apparaître les erreurs directement dans la page web ce qui vous permettra de comprendre d'où pourront provenir les éventuels disfonctionnements de votre application.

En tout début de la section Php du début de la page, vous ajouter :

```
error_reporting(E_ALL);
ini_set('display_errors', 'On');
ini_set('display_startup_errors', 'On');
```

La première fonction permet de définir le niveau d'erreur devant être affiché si l'affichage est activé.

La fonction *ini_set* permet de modfier une option de configuration du fonctionnement de Php. Elle a comme paramètres : le nom de l'option à modifier et la nouvelle valeur de cette option.

Les options :

display_errors détermine si les erreurs doivent être affichées à l'écran ou non.

display_startup_errors ăaffichage des erreurs pouvant survenir lors du démarrage.

Ces 2 options sont utiles et à activer pendant la phase de développement de l'application.
Bien entendu, il faut que ces options soient désactivées avant une mise en production de l'application finale.

connexion à la base de données

Dans un premier temps, il faut définir les informations nécessaires à la connexion : l'adresse du serveur, le login et le mot de passe de l'utilisateur de la base de données et enfin, le nom de la base de données.
On crée donc une variable *infoBdd* contenant ses informations :

```
$infoBdd = ['server' => 'localhost',
            'login' => 'pwnd',
            'password' => 'network',
            'db_name' => 'projet', ];
```

Ainsi, la valeur de l'adresse du serveur, ici *localhost* sera accessible dans *$infoBdd['server']*.
Il ne devrait pas y avoir de problème de compréhension particulier concernant les valeurs de *login*, *password* et *db_name*.
Par contre, quelques mots concernant l'adresse du serveur.
L'adresse attendue est l'adresse du serveur contenant le SGBD dans lequel votre base de données *projet* a été créée.
On pourrait s'attendre à devoir entrer la valeur *192.168.56.80*.
Et pourtant, la valeur, dans le context de notre projet et dans le cas où vous utilisez la machine virtuelle est bien d'affecter la valeur *localhost*.
Pourquoi ?
Et bien parce que votre application est déployée sur le même serveur que votre SGBD. Donc, du point de vue sur serveur d'application, la base de données est bien locale à l'application web.
Les informations de connexion étant fixées, on peut créer une connexion à la base :

```
$mysqli = new mysqli($infoBdd['server'], $infoBdd['login'],
                           $infoBdd['password'],$infoBdd['db_name']);

if ($mysqli->connect_errno) {
  exit('Problème de connexion à la BDD');
}
```

La condition permet, par l'appel à la fonction *connect_errno*, de vérifier que la connexion a bien été établie. Dans le cas contraire, l'exécution est arrêtée.
La fonction *connect_errno* renvoie un code erreur correspondant à la dernière tentative de connexion. La valeur zéro signifie qu'il n'y a pas eu d'erreur.
Par exemple, si le mot de passe renseigné dans les informations de connexion n'est pas le bon, vous pourriez avoir l'affichage suivant :

```
Warning: mysqli::__construct(): (HY000/1045): Access denied for user 'pwnd'@'localhost'
(using password: YES) in /var/www/html/pwnd/index.php on line 19
Problème de connexion à la BDD
```

Les 2 premières lignes correspondent à l'affichage activé grace à l'option *display_errors*.
La 3e ligne correspond au message en paramètre de la fonction *exit*.

création de la requête

La requête SQL permettant d'obtenir l'identifiant de l'étudiant en train de se connecter à partir des valeurs du formulaire mail et mot de passe va être stockée dans une variable *$sql*.

```
$sql = "SELECT idEtu
        FROM Etudiant
        WHERE email = '".$_POST['mail']."'
        AND motDePasse = '".$_POST['password']."'";
```

Une amélioration de la requête serait :

```
$sql = "SELECT idEtu
        FROM Etudiant
        WHERE email = '".trim($_POST['mail'])."'
        AND motDePasse = '".trim($_POST['password'])."'";
```

trim(chaîne) fonction qui supprime les espaces situés en début et fin de chaîne.

exécuter la requête

Pour exécuter une requête SQL, on fait appel à la fonction *query* de mysqli.
Cette fonction a comme paramètre la requête SQL devant être exécutée.

```
$result = $mysqli->query($sql);
if (!$result) {
    exit($mysqli->error);
}
```

Dans le cas où une erreur survient lors de l'exécution de la requête, le resultat n'existe pas. Ainsi, le test présent dans le code proposé permet de stopper l'exécution du script Php en affichant le message d'erreur envoyé par la base de données.

C'est le moment de parler un peu de sécurité et de se protéger contre une des attaques possibles : l'injection SQL.

Notre requête contenant les valeurs de variables dont la valeur dépend d'un utilisateur de l'application, les données doivent être formattées et les chaînes de caractères doivent être échappées.

La fonction *real_escape_string*, fonction de mysqli, protège les caractères spéciaux d'une chaîne pour en permettre l'utilisation dans une requête SQL.

```
$mail_escaped = $mysqli->real_escape_string(trim($_POST['mail']));
$password_escaped = $mysqli->real_escape_string(trim($_POST['password']));

$sql = "SELECT idEtu
FROM Etudiant
WHERE email  = '".$mail_escaped."'
AND motDePasse = '".$password_escaped."'";
```

Ainsi, dans le morceau de code ci-dessus, le mail et le mot de passe sont récupérés du formulaire puis nettoyés des espaces pouvant être situés en début et fin de chaîne.

Enfin, chaque chaîne est encodée selon le jeu de caractères défini au niveau du serveur et devient donc une chaîne dite échappée qui peut être utiliser dans la construction d'une chaîne contenant une requête SQL.

Pour plus de détails sur l'injection SQL :
`https://www.php.net/manual/fr/security.database.sql-injection.php`

test de la présence de résultat et action en conséquence

Dans notre cas, la recherche de la présence d'un étudiant dans la base peut avoir 2 résultats : soit l'étudiant existe, soit non.

Donc, le résultat de la requête est soit vide, soit il ne contient qu'une ligne.

On va donc, dans le cas où il y a une ligne, récupérer la valeur de l'identifiant pour le stocker dans la variable de session *compte*.

Ce qui donne le code suivant :

```
$nb = $result->num_rows;

 if ($nb) {
     //récupération de l'id de l'étudiant
     $row = $result->fetch_assoc();

     $_SESSION['compte'] = $row['idEtu'];
 }
```

num_rows fonction de mysqli qui renvoie le nombre de lignes contenu dans la variable *$result*.

fetch_assoc fonction de mysqli qui permet d'accéder à la ligne suivante d'un résultat d'exécution d'une requête. Le résultat de l'appel à cette fonction est un tableau associatif ou **null** s'il n'y a plus de ligne.

Dans le cas de notre code, la variable *$row* est un tableau associatif qui contient les colonnes selectionnées dans la clause SELECT de la requête. La requête contenant ici une seule colonne, le tableau *$row* contient une seule case.

La valeur de cette case est récupérée pour être stockée dans la variable de session *compte*.

fermer la connexion

Une fois l'utilisation de la connexion terminée, il faut fermer cette dernière en utilisant la fonction *close*.

```
$mysqli->close();
```

L'appel à la fonction close() se fait dans une balise php ajoutée à la balise footer de la page.

3 Deuxième étape - on s'organise pour capitaliser

Logiquement, vous avez globalement compris les bases.

Par contre, tout est dans la page *index.php* et il y a un certain nombre d'informations ou de fonctionnements qui seront surement utilisés dans d'autres pages. Par exemple, le fait de devoir se connecter à la base de données.

On va donc construire des scripts Php utilitaires qui pourront par la suite être utiliser dans l'ensemble des pages de votre application.

Création du fichier core.php

Ce fichier php a pour rôle de contenir l'ensemble des éléments qui seront utilisés dans l'ensemble des pages de l'application.

Ce fichier est ensuite inclus dans les pages.

Dans ce fichier, vous ouvrez une balise php et vous ajoutez les éléments suivants :

— les 3 instructions de gestion de l'affichage des erreurs ;
— l'instruction de démarrage de la session ;
— la gestion de la création de la connexion à la base de données ;
— la gestion de la déconnexion ;

Un plus peut être d'ajouter une gestion de la récupération des informations utilisateur si ce dernier est connecté : la requête SQL devant être produite doit permettre de récupérer l'ensemble des informations de l'étudiant connecté et le nom de son année scolaire.

Un affichage qui pourrait permettre de valider ce fonctionnement serait d'afficher le nom et le prénom de l'étudiant dans le cas où ce dernier est bien connecté à l'application.

Inclure le fichier core.php dans index.php

La première instruction php du fichier index.php devient :

```
include 'core/core.php';
```

Et vous retravaillez le code du fichier index.php pour qu'il ne contienne plus les instructions présentes maintenant dans core.php.

4 Troisième étape - Vous ajoutez le formulaire d'inscription

En vous basant sur ce qui a été fait précédemment, vous ajoutez la fonctionnalité d'inscription d'un utilisateur.