











Les sessions

Sommaire

I. Principe de fonctionnement des sessions	3
II. Mise en œuvre des sessions : accès à l'administration	5
A. Page login.php : formulaire d'identification et traitement de la saisie.....	6
B. Vérification permanente de l'utilisateur dans les pages de l'administration.....	8

Crédits des illustrations:
© DR.

Les repères de lecture

 Retour au chapitre	 Définition	 Objectif(s)	 Espace Élèves	 Vidéo / Audio
 Point important / À retenir	 Remarque(s)	 Pour aller plus loin	 Normes et lois	 Quiz

I. Principe de fonctionnement des sessions

Les sessions PHP sont un moyen de propager des informations d'une page à une autre. Elles permettent un fonctionnement similaire aux variables externes mais sans utiliser de valeurs dans l'URL. Les sessions permettent l'enregistrement d'un nombre illimité de variables qui sont ainsi préservées durant la navigation par un internaute.

Pour comprendre la mise en œuvre des sessions, nous allons créer un répertoire « sessions » puis créer deux fichiers `page_a.php` et `page_b.php`.

Le mécanisme des sessions repose en premier lieu sur la déclaration `session_start()` qui va indiquer à PHP que nous souhaitons utiliser les variables de session.

Page A : initialisation des sessions et création d'une variable de session « test ». La syntaxe de la variable de session est `$_SESSION['test']`. La syntaxe à base de crochets indique qu'il s'agit d'un tableau de données `$_SESSION` et nous pouvons alors créer une variable en ajoutant simplement l'index 'test'.

```
<?php
session_start(); // initialise les sessions
$_SESSION['test'] = 'une valeur de test'; // crée une variable de
session 'test'
?>
<a href="page_b.php">Lien vers la page B</a>
```

Fig. 1

Page B : grâce à la fonction `session_start()`, nous récupérons simplement en l'écrivant la variable de session `$_SESSION['test']`.

```
<?php
session_start(); // initialise les sessions
echo $_SESSION['test'] ;
?>
<br>
<a href="page_a.php">Lien vers la page A</a>
```

Fig. 2

Nous constatons que l'information est bien propagée (transmise) de la page A vers la page B et que cette information est récupérée par la variable de session.

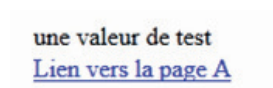


Fig. 3 Affichage sur la page B de la variable créée en page A

Le mécanisme des sessions est donc simple, il repose sur l'initialisation des sessions avec la fonction `session_start()` et sur la déclaration des variables de sessions, de type tableaux de données, dont la syntaxe est `$_SESSION`.

Nous pouvons créer autant de variables de sessions que nous le souhaitons. Par exemple, sur la page A, nous pouvons créer les variables suivantes :

```
$_SESSION['couleur'] = 'rouge' ;
$_SESSION['prenom'] = 'Bob';
$_SESSION['metier'] = 'webmaster;
```

Fig.4

Il suffit ensuite dans la page B de les écrire pour les afficher.

```
echo $_SESSION['couleur'] ; //affiche rouge
echo $_SESSION['prenom'] ; // affiche Bob
echo $_SESSION['metier'] ; // affiche webmaster
```

Fig.5

Puisqu'il s'agit de variables, nous pouvons les affecter à d'autres variables, comme par exemple :

```
$couleur = $_SESSION['couleur'] ;
echo $couleur ; //affiche rouge
```

Fig.6

Il s'agit donc de données qui se comportent comme des variables ordinaires. Nous pouvons également les afficher au moyen d'un `print_r` ou d'un `var_dump`, puisqu'il s'agit d'un tableau de données.

```
Array
(
    [test] => une valeur de test
    [couleur] => rouge
    [prenom] => Bob
    [metier] => webmaster
)
```

Fig.7

Nous pouvons aussi les supprimer si besoin avec les fonctions PHP dédiées aux sessions.

Exemple de suppression d'une session :

```
session_start();
session_unset() ; //Détruit toutes les variables d'une session
session_destroy() ; // Détruit une session
```

Fig.8



Note: à partir du moment où nous utilisons `session_start()`, un identifiant de session est automatiquement créé par PHP qui est accessible par la fonction `session_id()`.

```
echo session_id(); // affiche un4h980oev1dr0v7mov70c1t31
```

Fig.9

Cet identifiant de session, créé pour la session courante, n'est pas accessible par le tableau `$_SESSION` et la fonction `setcookie()` doit être utilisée pour le supprimer si besoin.

Le manuel PHP traite des sessions et cette page liste l'ensemble des fonctions utiles.

L'adresse de la page du manuel est : <http://fr2.php.net/manual/fr/ref.session.php>

II. Mise en œuvre des sessions : accès à l'administration

Si nous reprenons l'exemple du gestionnaire de contenus, nous avons vu qu'il était souhaitable que l'accès à l'interface d'administration nécessite la saisie d'un identifiant et d'un mot de passe pour identifier la ou les personnes autorisées. Pour des raisons de sécurité, ce besoin d'identification doit être permanent tout au long de l'utilisation des fonctionnalités de création, modification ou suppression offertes par l'administration. Afin de ne pas demander à chaque page de saisir un identifiant et un mot de passe, nous allons utiliser les sessions pour contribuer à la sécurisation de notre interface d'administration.

Nous aurons donc besoin des éléments suivants :

- une table nommée user (utilisateur en anglais, sous-entendu utilisateur autorisé de l'administration) contenant l'identifiant et le mot de passe du Webmaster ;
- un formulaire permettant la saisie de l'identifiant et du mot de passe afin de vérifier si cet identifiant et ce mot de passe sont dans la table user ;
- insérer le code PHP que nous consacrons à la sécurité d'accès et qui repose sur le mécanisme des sessions dans les pages de l'administration : admin.php, ajout.php, edition.php et suppression.php.

Dans la base projet_villes, nous créons avec PhpMyAdmin la table user comme cela est stipulé dans le code MySQL suivant :

```
CREATE TABLE IF NOT EXISTS `user` (  
  `user_id` tinyint(4) NOT NULL AUTO_INCREMENT,  
  `user_login` varchar(20) NOT NULL,  
  `user_password` varchar(34) NOT NULL,  
  PRIMARY KEY (`user_id`)  
);
```

Fig. 10

- user_id est la clef primaire, et possède l'option AUTO_INCREMENT.

Son type est tinyint (nombre entier inférieur à 127):

- user_login est le champ destiné à stocker l'identifiant (ou login) de la personne. Le type est VARCHAR sur 20 caractères. Il s'agira dans notre exemple de l'email de la personne;
- user_password est le champ destiné à enregistrer le mot de passe de l'utilisateur.

Il s'agit d'un VARCHAR de 34 caractères. Le mot de passe sera crypté avec la fonction crypt() qui retourne une chaîne de caractères de 34 caractères.

L'email de l'utilisateur est : user@projetville.com et son mot de passe est : FR@3m:We8!

Pour crypter le mot de passe, nous utilisons la fonction crypt() comme suit dans un fichier PHP temporaire afin d'afficher le mot de passe crypté que nous pouvons copier depuis la page web et coller ensuite dans la table.

```
$mdp = 'FR@3m:We8!';  
$mdpcrypt = crypt($mdp);  
echo $mdpcrypt; // affiche par exemple : $1$7Q..  
Co4.$t2qZZb7UvsVOInHbL/xii/
```

Fig. 11

Nous enregistrons donc (avec PhpMyAdmin) dans la table user les valeurs suivantes:

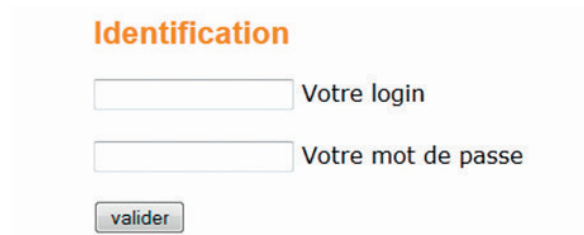
- user_login: user@projetville.com;
- user_password: \$1\$7Q..Co4.\$t2qZZb7UvsVOInHbL/xii/.

Notons que la valeur de \$mdcrypt peut varier et donc être différente de celle qui est imprimée dans notre exemple. Cela n'a pas d'incidence sur le fonctionnement et nous saisissons simplement le résultat qui est affiché dans notre page PHP, la fonction crypt() s'occupera de gérer la reconnaissance du mot de passe.

A. Page login.php: formulaire d'identification et traitement de la saisie

Le formulaire contient deux parties:

1. Le formulaire HTML destiné à saisir le login et le mot de passe de la part de l'utilisateur et dont les deux champs auront comme attribut name respectifs: user_input_login et user_input_password.



The image shows a web form titled "Identification" in orange text. It has two text input fields. The first field is labeled "Votre login" and the second is labeled "Votre mot de passe". Below these fields is a button with the text "valider".

Fig. 12 Le formulaire d'identification

2. La deuxième partie contient le code de traitement de la saisie utilisateur : récupération des variables externes de type POST et vérification dans la base.

Après avoir classiquement récupéré les variables externes et les avoir affectées à des variables simples nous avons vérifié si elles sont vides ou non. Puis nous vérifions que le login qui a été saisi dans le formulaire correspond à une valeur existant dans la base. Nous posons donc la requête avec la clause WHERE portant sur le login. Rappelons que l'égalité en MySQL repose sur le simple signe égal.

```
$result = $mysqli->query('SELECT user_login, user_password FROM user
                           WHERE user_login = "' . $user_input_login ."'');
$row = $result->fetch_array();
```

Fig. 13

Si `$row['user_login']` n'existe pas alors le login saisi par l'utilisateur n'est pas dans la base, sinon nous pouvons passer à l'étape de vérification du mot de passe avec la fonction `crypt()` qui a servi à générer le mot de passe crypté. Nous passons en argument la saisie utilisateur qui n'est pas cryptée (`$user_input_password`) et le mot de passe crypté qui provient de la base (`$user_password`).

La fonction `crypt()` va ainsi pouvoir vérifier que le mot de passe issu du formulaire est identique à celui stocké dans la base.

```
if (crypt($user_input_password, $user_password) != $user_password)
```

Fig. 14

Si le mot de passe est bien celui qui est associé dans la base au login de l'utilisateur, alors l'utilisateur est reconnu, son login et son mot de passe sont bien ceux de la base de données.

Nous créons alors une variable de sessions `$_SESSION['login']` qui contiendra le login utilisateur et servira de support au script dédié à la vérification que nous intégrerons à toutes les pages qui constituent l'espace d'administration.

```
session_start();
$_SESSION['user_login'] = $user_login;
```

Fig. 15

Nous redirigeons ensuite l'utilisateur vers la page d'accueil de l'administration avec la fonction `header()` à laquelle nous passons en argument l'URL de cette page.

```
header('location:admin.php');
```

Fig. 16

Code complet hors formulaire du fichier login.php:

```
// récupération des variables
if(isset($_POST['submit_form']))
{
    $user_input_login = $_POST['user_input_login'];
    $user_input_password = $_POST['user_input_password'];
    // vérification si les variables sont vides
    if((empty($user_input_login)) OR empty($user_input_password))
    {
        $message = '<p class="error">Vous devez saisir les informations demandées.</p>';
    }
    else
    {
        /* le login saisi correspond-il à une valeur existant la base ?
        Nous posons la requête avec la clause WHERE portant sur le login */
        $result = mysqli->query('SELECT user_login, user_password
        FROM user WHERE user_login = " . $user_input_login . "'');
        $row = $result->fetch_array();
        if(!isset($row['user_login']))
        {
            // la requête ne retourne aucun résultat pour ce login
            $message = '<p class="error">Erreur d\'identification.<br>Vous n\'avez pas accès à cette page</p>';
        }
        else
        {
            /* la requête retourne un résultat, le login existe dans la
            base. Vérifions avec la fonction crypt que le mot de passe saisi
            correspond à celui de la base.*/
            $user_login = $row['user_login'];
            $user_password = $row['user_password'];
            if (crypt($user_input_password, $user_password) != $user_
            password)
            {
                $message = '<p class="error">Erreur d\'identification.<br>
                Vous n\'avez pas accès à cette page</p>';
            }
            else
            {
                /*l'utilisateur est reconnu.
                Nous créons une variable de session 'user_login' puis redirigeons
                l'utilisateur vers la page d'accueil de l'administration
                avec la fonction header à laquelle nous passons en argument
                'location:admin.php'.
                La variable de session 'user_login' sera ainsi transmise à la page
                admin.php */
                session_start();
                $_SESSION['user_login'] = $user_login;
                header('location:admin.php');
            }
        }
    }
}
```

Fig.17

B. Vérification permanente de l'utilisateur dans les pages de l'administration

Fichier inc_identification_user.php

```
<?php
session_start();
if (!isset($_SESSION['user_login']))
{
    echo 'Vous n\'avez pas les droits d\'accès à cette page';
    echo '<br><a href="index.php">retour vers le site</a>';
    exit;
}
$user_login = $_SESSION['user_login'] ;
```

Fig.18

Ce fichier contient le code qui vérifie si la variable de session \$_SESSION['user_login'] existe.

Si elle n'existe pas, un message est affiché avec un lien retour et nous arrêtons le traitement PHP avec la fonction exit(). Si la variable de session existe, nous affectons sa valeur à la variable \$user_login que nous utiliserons dans le reste des pages de l'espace de gestion, notamment dans le menu pour afficher le login de connexion de l'utilisateur.

Cependant, nous pouvons avoir affaire à un utilisateur malintentionné qui a réussi à créer une session sans pour autant détenir le bon couple login/mot de passe. Nous allons donc ajouter à ce code la vérification que le login est bien celui de la base, ce qui sera une précaution supplémentaire utile, en matière de sécurité sur Internet, nous ne sommes jamais assez prudent. Nous ajoutons donc la requête que nous adaptons à celle présente dans le fichier login.php et qui vérifie que le login est bien dans la base. S'il ne l'est pas, le message est affiché avec un lien retour et nous arrêtons le traitement PHP avec la fonction exit(). Si l'utilisateur est reconnu, le système de vérification est terminé et l'utilisateur peut travailler à l'intérieur de l'espace de gestion du site.

Le code suivant effectue la requête permettant de vérifier que le login existe dans la base :

```
session_start();
if (!isset($_SESSION['user_login']))
{
    echo 'Vous n\'avez pas les droits d\'accès à cette page';
    echo '<br><a href="index.php">retour vers le site</a>';
    exit;
}
$user_login = $_SESSION['user_login'] ;
require('inc_connexion.php');
$result = mysqli->query('SELECT user_login FROM user WHERE user_login = " . $user_login . "'');
$row = $result->fetch_array();
if(!isset($row['user_login']))
{
    echo 'Vous n\'avez pas les droits d\'accès à cette page';
    echo '<br><a href="index.php">retour vers le site</a>';
    exit;
}
```

Fig. 19

Nous intégrons ensuite ce fichier avec require() dans tous les fichiers utiles à l'administration afin que l'identification de l'utilisateur soit permanente.

```
<?php require('inc_identification_user.php'); ?>
```

Fig. 20

Les fichiers de l'administration sont: admin.php, ajout.php, edition.php, suppression.php.

Remarquons que nous commençons à avoir un certain nombre de fichiers qui servent soit à afficher le site, soit à constituer l'espace d'administration. Il conviendrait de créer un répertoire dédié à accueillir les fichiers de l'administration. Nous allons donc créer un répertoire nommé « admin » dans lequel nous allons placer nos fichiers. Nous renommons le fichier admin.php en index.php afin que celui-ci soit la page d'accueil du répertoire admin et nous modifions bien sûr la redirection du fichier afin de bien pointer le nouvel emplacement de la page comme ceci :

```
header('location:admin/index.php');
```

Fig.21

Nous modifions également tous les liens des fichiers d'administration, notamment dans le fichier admin_menu.php ainsi que les appels de feuille de style et les inclusions des fichiers de connexion :

```
<?php require('../inc_connexion.php'); ?>
```

Fig.22

Comme nous le voyons, une telle réorganisation, si elle est facilitée par le mécanisme de l'inclusion, nécessite pour autant un effort et une attention qui ne sont pas à minimiser. Il convient donc de bien réfléchir à la structure des fichiers que nous voulons avant de démarrer le développement.

Après avoir effectué le déplacement des fichiers, nous vérifions le bon fonctionnement du site puis continuons le développement de l'administration du site. Après que l'utilisateur ait effectué son travail, il est pertinent d'envisager qu'il puisse se déconnecter afin que personne ne puisse utiliser la session administrateur qu'il a créé en se connectant. Nous proposons donc très classiquement un lien permettant de se déconnecter et qui, une fois activé, supprimera toutes les sessions créées.

Nous ajoutons donc dans le menu, le lien « déconnexion » que nous faisons pointer vers la page logout.php (déconnexion en anglais). Le fichier logout.php contient le script que nous avons vu, permettant de supprimer les sessions :

```
<?php
session_start();
session_unset();
session_destroy();
header('location:../index.php');
```

Fig.23

Après avoir unset puis destroy les sessions, l'utilisateur est redirigé vers la page d'accueil du site qui est situé dans un niveau de répertoire supérieur (et donc nous utilisons les doubles points pour remonter dans l'arborescence des répertoires).

Revenons à l'interface front et posons-nous la question de savoir si nous ne pouvons pas optimiser l'accès à l'administration. Actuellement, le lien « administration » proposé pointe vers le formulaire d'identification. Mais si l'utilisateur est connecté, il serait utile lorsqu'il revient sur le site de ne pas avoir à s'identifier à nouveau. Nous utilisons donc le mécanisme des sessions pour vérifier sur la page du formulaire si la session utilisateur existe. Si oui, nous redirigeons directement l'utilisateur vers la page d'accueil de l'administration, sinon nous affichons le formulaire.

Nous ajoutons donc simplement le code suivant au début du fichier login.php :

```
<?php
session_start();
if (isset($_SESSION['user_login']))
header('location:admin/index.php');
?>
```

Fig. 24

Nous avons donc maintenant un site et un espace de gestion complets et fonctionnels dont voici l'architecture :

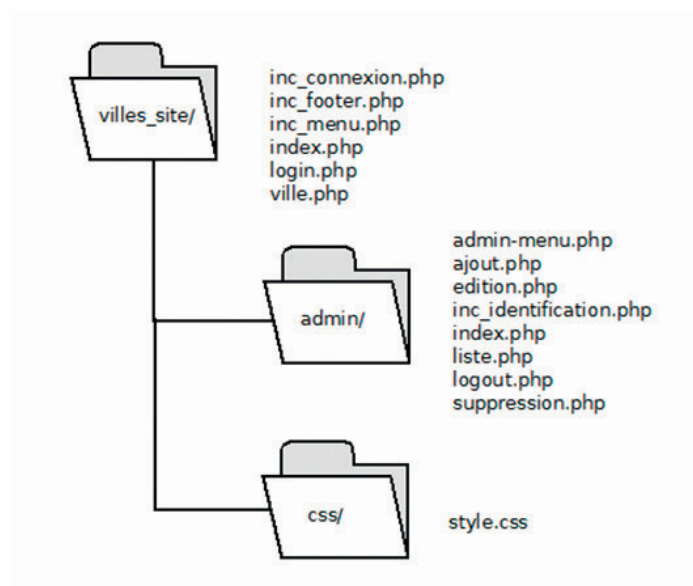


Fig. 25 Architecture des fichiers du site

