Leçon complète : Notions PHP Fondamentales

Table des matières

1 PDO (PHP Data Objects)

1.1 Qu'est-ce que PDO ?

PDO est l'acronyme de **PHP Data Objects**. C'est une extension de PHP qui fournit une interface **uniforme** et **cohérente** pour accéder à différentes bases de données. Imaginez-le comme un adaptateur universel pour parler à plusieurs types de systèmes de gestion de bases de données (SGBD).

— **Extension native de PHP** : Cela signifie qu'elle est généralement incluse avec PHP et ne nécessite pas d'installation complexe. Elle agit comme un **pilote unique** (une couche d'abstraction) permettant à votre code PHP d'interagir avec divers SGBD tels que MySQL, PostgreSQL, SQLite, Oracle, SQL Server, etc., sans avoir à changer fondamentalement votre code d'accès aux données pour chaque SGBD.

— **API Orientée Objet** : L'interaction avec PDO se fait principalement en utilisant des objets et des méthodes. Par exemple, vous créez un objet `PDO` pour la connexion, un objet `PDOStatement` pour une requête. Cette approche est considérée comme plus moderne, plus structurée et plus maintenable que les anciennes API procédurales (comme les fonctions `mysql_*` aujourd'hui obsolètes, ou même `mysqli_*` en mode procédural). L'avantage majeur est que la **syntaxe reste la même** quel que soit le moteur de base de données utilisé en arrière-plan (seule la chaîne de connexion change).

— **Support des requêtes préparées** : C'est l'un des atouts majeurs de PDO. Les requêtes préparées améliorent considérablement la **sécurité** de vos applications en prévenant les attaques par injection SQL. Elles peuvent également améliorer les **performances** si la même requête est exécutée plusieurs fois avec des données différentes.

1.2 Connexion minimale à une base de données avec PDO

Pour se connecter à une base de données via PDO, vous avez besoin de trois informations principales : le DSN, le nom d'utilisateur et le mot de passe.

Listing 1 – Connexion PDO à une base MySQL

```php
// DSN (Data Source Name) : chane spcifiant le type de SGBD, l'hte,
// le nom de la base de donnes et le jeu de caractres.
$dsn  = "mysql:host=localhost;dbname=etudiant;charset=utf8mb4";
$user = "root"; // Nom d'utilisateur pour la base de donnes
$pass = "";    // Mot de passe (laisser vide si pas de mot de passe)

try {
    // Tentative de cration d'une nouvelle instance de l'objet PDO
    $pdo = new PDO($dsn, $user, $pass, [
        // Options de configuration pour PDO :
        // PDO::ATTR_ERRMODE: Configure le mode de rapport d'erreurs.
        // PDO::ERRMODE_EXCEPTION: PDO lvera des exceptions PDOException en cas d'erreur
            ↪ SQL.
        // C'est la mthode recommande pour grer les erreurs.
        PDO::ATTR_ERRMODE          => PDO::ERRMODE_EXCEPTION,

        // PDO::ATTR_DEFAULT_FETCH_MODE: Dfinit le mode de rcupration par dfaut des
            ↪ rsultats.
        // PDO::FETCH_ASSOC: Rcupre les lignes comme des tableaux associatifs (cls = noms
            ↪  de colonnes).
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
    ]);
    // Si la connexion russit, $pdo contient l'objet de connexion.
```

```php
    // echo "Connexion  la base de donnes  russie  !"; // Optionnel

} catch (PDOException $e) {
    // Si une exception PDOException est leve (chec de connexion ou autre erreur PDO),
    // le bloc catch l'intercepte. $e est un objet contenant les dtails de l'erreur.
    // En production, il est prfrable de logger cette erreur et d'afficher un message
    //  gnrique     l'utilisateur.
    error_log("Erreur de connexion PDO : " . $e->getMessage());
    die("Connexion  la base de donnes impossible. Veuillez ressayer plus tard.");
    // 'die()' arrte l'excution  du script.
}
```

Explication des éléments :

- `$dsn` (Data Source Name) : Une chaîne qui décrit la source de données. Pour MySQL, elle commence par `mysql:`.
  - `host=localhost` : L'adresse du serveur de base de données (souvent 'localhost' en développement).
  - `dbname=etudiant` : Le nom de la base de données à laquelle se connecter.
  - `charset=utf8mb4` : Spécifie le jeu de caractères pour la connexion. 'utf8mb4' est recommandé pour une prise en charge complète d'Unicode (y compris les emojis).
- `$user` et `$pass` : Vos identifiants de connexion à la base de données.
- `new PDO(...)` : Crée une instance de la classe PDO, établissant ainsi la connexion.
- `try...catch` : Un bloc essentiel pour la gestion des erreurs. Si la connexion échoue, une `PDOException` est levée et attrapée par le bloc `catch`, empêchant le script de crasher brutalement et vous permettant de gérer l'erreur proprement.
- `PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION` : Configure PDO pour qu'il lève des exceptions en cas d'erreur SQL. Cela simplifie grandement la détection et la gestion des erreurs.
- `PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC` : Indique que, par défaut, les résultats des requêtes seront retournés sous forme de tableaux associatifs (où les clés sont les noms des colonnes).

1.3 Lecture simple de données

Une fois connecté, vous pouvez exécuter des requêtes. La méthode `query()` de l'objet PDO est utilisée pour les requêtes simples qui ne nécessitent pas de lier des paramètres (comme un `SELECT` sans clause `WHERE` dynamique).

Listing 2 – Exécution d'un SELECT basique avec PDO

```php
// $pdo est l'objet de connexion obtenu prcdemment.
// La mthode  query() excute directement la requte SQL.
// Elle retourne un objet PDOStatement si la requte est valide.
$statement = $pdo->query("SELECT id_classe, nom_classe FROM classes ORDER BY nom_classe
    ↪ ASC");

// On peut itrer directement sur l'objet PDOStatement avec un foreach
// pour  rcuprer  chaque ligne du rsultat.
```

```php
// Par dfaut (avec PDO::FETCH_ASSOC), $row sera un tableau associatif.
if ($statement) { // Bonne pratique de vrifier si la requte a russi
    foreach ($statement as $row) {
        echo htmlspecialchars($row['id_classe']) . "  " . htmlspecialchars($row['
            ↪ nom_classe']) . "<br>";
    }
} else {
    echo "Erreur lors de l'excution de la requte.";
}
```

Explication :

- `$pdo->query("...")` : Exécute la requête SQL fournie. Pour un `SELECT`, cette méthode retourne un objet `PDOStatement` qui représente l'ensemble des résultats.
- `foreach ($statement as $row)` : On peut itérer directement sur l'objet `PDOStatement`. À chaque itération, `$row` contiendra une ligne de résultat sous forme de tableau associatif (grâce à `PDO::FETCH_ASSOC`).
- `htmlspecialchars()` : Il est crucial d'utiliser cette fonction lors de l'affichage de données provenant de la base de données dans du HTML pour prévenir les failles de sécurité XSS (Cross-Site Scripting).

2 Les Requêtes Préparées avec PDO

2.1 Pourquoi utiliser les requêtes préparées ?

Les requêtes préparées sont un mécanisme fondamental en matière de sécurité et d'efficacité lors de l'interaction avec une base de données.

- **Prévention des injections SQL** : C'est l'avantage le plus important. Une injection SQL est une attaque où du code SQL malveillant est inséré dans les entrées d'une application (par exemple, un champ de formulaire). Si ces entrées sont directement concaténées dans une requête SQL, l'attaquant peut manipuler la base de données (lire des données sensibles, modifier, supprimer). Avec les requêtes préparées, la structure de la requête SQL est envoyée au SGBD séparément des données. Les données (paramètres) sont ensuite envoyées et traitées comme de simples valeurs, et non comme une partie exécutable de la requête. Ainsi, même si une valeur contient du code SQL, il ne sera pas interprété comme tel. **Principe clé** : Les valeurs fournies par l'utilisateur ne sont **jamais directement concaténées** dans la chaîne de la requête SQL.
- **Performance (potentielle)** : La structure de la requête (le "plan d'exécution") est envoyée au SGBD et **pré-compilée une seule fois**. Si vous exécutez ensuite la même requête plusieurs fois avec des valeurs différentes, le SGBD peut réutiliser ce plan pré-compilé, ce qui peut être plus rapide que de recompiler la requête à chaque fois. Cet avantage est plus notable pour les requêtes exécutées en boucle.

2.2 Exemple d'insertion sécurisée avec une requête préparée

Le processus se déroule en deux étapes principales : `prepare` puis `execute`.

Listing 3 – INSERTION sécurisée avec requête préparée

```php
// $pdo est l'objet de connexion.
// $nomSaisi, $prenomSaisi, $idClasse sont des variables contenant les donnes
// (par exemple, issues d'un formulaire $_POST).

// tape 1 : Prparation de la requte SQL
// On utilise des marqueurs nomms (ex: :nom) ou des marqueurs interrogatifs (?)
// pour indiquer o les valeurs seront insres.
$sql = "INSERT INTO etudiants (nom_etudiant, prenom_etudiant, id_classe_etudiant)
        VALUES (:nom, :prenom, :classe)";
// La mthode prepare() envoie la structure de la requte au SGBD.
// Elle retourne un objet PDOStatement.
$stmt = $pdo->prepare($sql);

// tape 2 : Excution de la requte prpare avec les donnes
// La mthode execute() prend un tableau associatif o les cls correspondent
// aux marqueurs nomms de la requte prpare.
// PDO s'occupe de lier et d'chapper correctement les valeurs.
try {
    $resultatExecution = $stmt->execute([
        ':nom'    => $nomSaisi,    // Lie la valeur de $nomSaisi au marqueur :nom
        ':prenom' => $prenomSaisi, // Lie la valeur de $prenomSaisi au marqueur :prenom
        ':classe' => $idClasse     // Lie la valeur de $idClasse au marqueur :classe
                                   // PDO gre correctement les valeurs NULL si $idClasse
                                       ↪ est null.
    ]);

    if ($resultatExecution) {
        echo "tudiant ajout avec succs. ID : " . $pdo->lastInsertId();
        // lastInsertId() retourne l'ID de la dernire ligne insre (pour les colonnes
            ↪ AUTO_INCREMENT).
    } else {
        echo "Erreur lors de l'ajout de l'tudiant.";
    }
} catch (PDOException $e) {
    // Grer les erreurs spcifiques l'excution (ex: violation de contrainte UNIQUE)
    error_log("Erreur PDO lors de l'insertion : " . $e->getMessage());
    echo "Une erreur technique est survenue lors de l'ajout.";
    // Afficher $e->getCode() ou $e->errorInfo peut aussi tre utile pour le dbogage.
}
```

*L'approche en deux étapes « **`prepare`** » puis « **`execute`** » est cruciale car elle* ***sépare la structure logique de la requête SQL des données variables****. C'est cette séparation qui assure la protection contre les injections SQL.*

Alternativement à `bindParam()` (qui lie une variable par référence), on peut passer directement un tableau de valeurs à `execute()` comme montré ci-dessus, ce qui est souvent plus concis.

3 SESSION, COOKIE, LocalStorage : Mécanismes de persistance côté client/serveur

Ces trois mécanismes permettent de stocker des informations, mais ils diffèrent par leur lieu de stockage, leur taille, leur durée de vie et leur accessibilité.

3.1 Comparatif rapide

Caractéristique	**SESSION (PHP)**	**COOKIE (HTTP)**	**LocalStorage (JS)**
Lieu de stockage	Côté **serveur**	Côté **navigateur**	Côté **navigateur**
Taille maximale	Pratiquement **illimitée**	Environ **4 Ko**	Environ **5 à 10 Mo**
Persistance des données	**Oui** (si cookie de session)	**Oui** (jusqu'à expiration)	**Oui** (pas d'expiration auto)
Accessible par PHP	`$_SESSION`	`$_COOKIE`, `setcookie()`	**Non** directement
Accessible par JavaScript (JS)	**Indirectement**	`document.cookie`	API `localStorage`
Cas d'usage typiques	Authentification, panier	Préférences, suivi simple	Cache offline, état d'app

TABLE 1 – Comparaison des mécanismes de stockage SESSION, COOKIE et LocalStorage.

3.2 Gestion basique d'une SESSION PHP

Les sessions PHP permettent de stocker des informations spécifiques à un utilisateur à travers plusieurs requêtes HTTP. Le serveur génère un identifiant de session unique (Session ID) qui est généralement stocké dans un cookie côté client. Ce cookie est renvoyé par le navigateur à chaque requête, permettant au serveur de retrouver les données de session correspondantes.

Listing 4 – Cycle de vie typique d'une session PHP

```php
// 1. Dmarrer ou reprendre une session existante.
// DOIT tre  appel  AVANT toute sortie HTML vers le navigateur.
session_start();

// 2. Stocker des donnes dans la session.
// On utilise la superglobale $_SESSION comme un tableau associatif.
$_SESSION['user_id'] = 42;
$_SESSION['username'] = "UtilisateurExemple";
$_SESSION['role'] = "admin";

// 3. Accder aux donnes  stockes  dans la session.
```

```php
if (isset($_SESSION['username'])) {
    echo "Bonjour, " . htmlspecialchars($_SESSION['username']) . "!<br>";
    echo "Votre ID utilisateur est : " . $_SESSION['user_id'] . "<br>";
} else {
    echo "Vous n'tes pas  connect .<br>";
}

// 4. Modifier une donne de session.
$_SESSION['role'] = "editor";

// 5. Supprimer une donne spcifique de la session.
unset($_SESSION['role']);

// 6. Dtruire  toutes les donnes de la session (dconnexion).
// Ceci supprime les variables de la superglobale $_SESSION.
session_unset(); // Optionnel, session_destroy() s'en charge souvent.
session_destroy(); // Dtruit toutes les donnes enregistres dans une session.

// Pour une dconnexion  complte , il est aussi recommand de dtruire le cookie de session
// si vous voulez que l'utilisateur soit immdiatement dconnect et
// qu'une nouvelle session ne soit pas reprise avec l'ancien cookie.
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]
    );
}
echo "Session dtruite et utilisateur  dconnect .";
```

Points importants sur les sessions PHP :

- `session_start()` : Essentiel. Doit être appelé au début de chaque script qui a besoin d'accéder aux données de session.
- `$_SESSION` : Superglobale PHP (tableau associatif) où les données de session sont stockées et accessibles.
- `session_destroy()` : Supprime toutes les données associées à la session courante sur le serveur. Ne supprime pas le cookie de session du navigateur par défaut, ce qui peut parfois prêter à confusion. L'ajout du code pour supprimer le cookie est une bonne pratique pour une déconnexion "complète".

3.3 Utilisation des COOKIES PHP

Les cookies sont de petits fichiers texte stockés par le navigateur web sur l'ordinateur de l'utilisateur. Ils sont envoyés avec chaque requête HTTP au serveur pour lequel ils ont été créés.

3.3.1 Créer et modifier un cookie avec PHP

La fonction `setcookie()` est utilisée pour créer ou modifier un cookie. Elle doit être appelée **avant toute sortie HTML**.

Listing 5 – Créer/Modifier un cookie en PHP

```
<?php
// Nom du cookie
$nom_cookie = "preference_utilisateur";
// Valeur du cookie
$valeur_cookie = "theme_sombre";
// Date d'expiration (ici, dans 1 heure)
$expiration = time() + (60 * 60); // 60 secondes * 60 minutes

//  Cre  le cookie. Tous les paramtres aprs la valeur sont optionnels.
setcookie($nom_cookie, $valeur_cookie, $expiration, "/", "", false, true);
// Paramtres :
// 1. Nom du cookie
// 2. Valeur du cookie
// 3. Expiration (timestamp Unix)
// 4. Chemin sur le serveur o le cookie sera disponible ('/' = tout le site)
// 5. Domaine (vide = domaine courant)
// 6. Secure (true = seulement envoy sur HTTPS)
// 7. HttpOnly (true = non accessible par JavaScript, scurit accrue contre XSS)

if(!isset($_COOKIE[$nom_cookie])) {
    echo "Cookie '" . $nom_cookie . "' n'est pas encore dfini (sera dfini au prochain
        ↪ rechargement).<br>";
} else {
    echo "Cookie '" . $nom_cookie . "' est dfini !<br>";
    echo "Valeur : " . htmlspecialchars($_COOKIE[$nom_cookie]) . "<br>";
}

// Pour modifier un cookie, il suffit d'appeler setcookie()  nouveau avec le mme nom
// et les nouvelles valeurs (valeur, expiration, etc.).
// setcookie($nom_cookie, "theme_clair", time() + (3600 * 24), "/");
?>
(*@\textit{Le code HTML/PHP qui suit doit tre plac aprs l'appel  setcookie()}@*)
```

3.3.2 Lire un cookie avec PHP

Les cookies envoyés par le navigateur sont accessibles via la superglobale `$_COOKIE`.

Listing 6 – Lire un cookie en PHP

```
<?php
// Nom du cookie que l'on souhaite lire
$nom_cookie = "preference_utilisateur";
```

```php
if(isset($_COOKIE[$nom_cookie])) {
    // Important : toujours chapper la valeur du cookie avant de l'afficher
    $preference = htmlspecialchars($_COOKIE[$nom_cookie]);
    echo "Votre  prfrence  enregistre  est : " . $preference . "<br>";

    if ($preference === "theme_sombre") {
        // Appliquer le thme sombre...
        echo "Application du thme sombre.<br>";
    }
} else {
    echo "Aucune  prfrence  de  thme  n'a  t  trouve  dans les cookies.<br>";
    // Peut- tre  dfinir  une valeur par dfaut ou proposer de choisir
}
?>
```

3.3.3 Supprimer un cookie avec PHP

Pour supprimer un cookie, on utilise `setcookie()` avec le même nom, une valeur vide (ou quelconque), et une date d'expiration dans le passé.

Listing 7 – Supprimer un cookie en PHP

```php
<?php
$nom_cookie_a_supprimer = "preference_utilisateur";

// Mettre la date d'expiration dans le pass (ex: il y a 1 heure)
setcookie($nom_cookie_a_supprimer, "", time() - 3600, "/");

echo "Tentative de suppression du cookie '" . $nom_cookie_a_supprimer . "'.<br>";
echo "Il sera effectivement supprim du navigateur lors de la prochaine requte.<br>";

// Pour vrifier la suppression, il faudra recharger la page.
// unset($_COOKIE[$nom_cookie_a_supprimer]); // Ceci supprime seulement la variable du
    ↪ tableau $_COOKIE pour le script courant,
                                    // mais pas le cookie lui-mme du navigateur.
?>
```

3.4 Utilisation de LocalStorage avec JavaScript

LocalStorage est une API du navigateur web qui permet aux sites et applications web de stocker des données (paires clé/valeur) dans le navigateur de l'utilisateur de manière persistante (elles ne sont pas supprimées à la fermeture du navigateur). Il est accessible uniquement via JavaScript côté client.

3.4.1 Stocker des données dans LocalStorage

La méthode `localStorage.setItem(key, value)` est utilisée. Les valeurs sont stockées sous forme de chaînes de caractères. Si vous voulez stocker des objets ou des tableaux, vous devez les

convertir en JSON (avec `JSON.stringify()`) avant de les stocker.

Listing 8 – Stocker des données dans LocalStorage (JS)

```
// Stocker une chane simple
localStorage.setItem('username', 'JeanDupont');
localStorage.setItem('themePreference', 'dark');

// Stocker un nombre (il sera converti en chane)
localStorage.setItem('userScore', '100');

// Stocker un objet (doit tre converti en chane JSON)
const userSettings = {
  notifications: true,
  language: 'fr'
};
localStorage.setItem('settings', JSON.stringify(userSettings));

console.log("Donnes stockes dans LocalStorage.");
// Vous pouvez vrifier dans l'onglet "Application" -> "LocalStorage" des outils de
    ↪ dveloppement du navigateur.
```

3.4.2 Lire des données depuis LocalStorage

La méthode `localStorage.getItem(key)` est utilisée. Si la clé n'existe pas, elle retourne `null`. Si vous avez stocké un objet JSON, vous devrez le reconvertir avec `JSON.parse()`.

Listing 9 – Lire des données depuis LocalStorage (JS)

```
// Lire une chane simple
const storedUsername = localStorage.getItem('username');
if (storedUsername) {
  console.log('Nom d\'utilisateur rcupr :', storedUsername);
  // alert('Bienvenue, ' + storedUsername);
} else {
  console.log('Aucun nom d\'utilisateur trouv dans LocalStorage.');
}

// Lire un nombre (sera une chane, convertir si besoin)
const storedScore = localStorage.getItem('userScore');
if (storedScore) {
  const scoreAsNumber = parseInt(storedScore, 10); // Convertir en nombre entier
  console.log('Score rcupr :', scoreAsNumber);
}

// Lire un objet JSON
const storedSettingsString = localStorage.getItem('settings');
if (storedSettingsString) {
  const userSettingsObject = JSON.parse(storedSettingsString);
```

```
  console.log('Paramtres   rcuprs   :', userSettingsObject);
  if (userSettingsObject.language === 'fr') {
    console.log('La langue  prfre  est le franais.');
  }
}
```

3.4.3 Supprimer des données de LocalStorage

— `localStorage.removeItem(key)` : Supprime un item spécifique par sa clé.
— `localStorage.clear()` : Supprime tous les items stockés pour le domaine courant.

Listing 10 – Supprimer des données de LocalStorage (JS)

```
// Supprimer un item spcifique
localStorage.removeItem('userScore');
console.log('userScore a  t  supprim  du LocalStorage.');

// Vrifier si l'item a bien  t  supprim
if (localStorage.getItem('userScore') === null) {
  console.log('La cl userScore n\'existe plus.');
}

// Supprimer tous les items pour ce domaine ( utiliser avec prcaution)
// localStorage.clear();
// console.log('Toutes les donnes LocalStorage pour ce domaine ont t effaces.');
```

Important pour LocalStorage :

— Les données sont stockées par origine (protocole + domaine + port). Un site sur ‘http ://example.com‘ n’a pas accès au LocalStorage de ‘https ://example.com‘ ou ‘http ://autre.com‘.
— Les données sont stockées indéfiniment jusqu’à ce qu’elles soient explicitement supprimées par le code JavaScript, ou par l’utilisateur via les outils de son navigateur.
— Il n’est pas adapté pour stocker des données sensibles car elles sont accessibles par n’importe quel script JavaScript exécuté sur la page.

4 Fonctions et Instructions PHP Clés (utiles dans un contexte web)

Voici un rappel de quelques fonctions et constructions PHP fréquemment utilisées dans le développement web, notamment avec le projet de gestion de notes.

`require_once __DIR__.'/../config/database.php'` — `require_once` : Inclut et évalue le fichier spécifié. Si le code du fichier a déjà été inclus, il ne sera pas inclus à nouveau (évite les redéfinitions de fonctions/classes). Si le fichier n’est pas trouvé, cela produit une erreur fatale et arrête le script.

— `__DIR__` : Une "constante magique" PHP qui retourne le chemin absolu du répertoire du fichier où elle est utilisée. C’est très utile pour construire des chemins de fichiers de manière fiable, indépendamment de l’endroit d’où le script est exécuté.

— `'/../config/database.php'` : Chemin relatif depuis le répertoire courant (`__DIR__`). `..` signifie "remonter d'un niveau dans l'arborescence des dossiers".

`session_start()` — Démarre une nouvelle session ou reprend une session existante. Cette fonction doit être appelée **avant toute sortie HTML** (avant tout `echo`, espace, ou balise HTML en dehors des blocs PHP). Elle lit l'identifiant de session (généralement via un cookie) ou en crée un nouveau.

`$pdo = db_connect();` — Supposant que `db_connect()` est une fonction personnalisée (comme définie dans `config/database.php`) qui retourne un objet `PDO` représentant la connexion à la base de données.

`trim($chaine)` — Supprime les espaces (ou d'autres caractères spécifiés) en début et en fin d'une chaîne de caractères. Utile pour nettoyer les entrées utilisateur avant de les traiter ou de les stocker.

`header('Location: autre_page.php'); exit;` — `header('Location: ...')` : Envoie un entête HTTP brut au client pour le rediriger vers une autre URL. Doit être appelé **avant toute sortie HTML**.

— `exit;` (ou `die();`) : Termine immédiatement l'exécution du script courant. Il est crucial de l'appeler après un `header('Location: ...')` pour s'assurer quaucun autre code n'est exécuté après la redirection, ce qui pourrait causer des comportements inattendus ou des failles de sécurité.

— C'est le pattern **Post/Redirect/Get (PRG)** : après une soumission de formulaire (POST) qui modifie des données, on redirige l'utilisateur vers une page (GET) pour éviter la resoumission du formulaire si l'utilisateur actualise la page.

`isset($variable)` — Fonction de langage qui vérifie si une variable est définie et si sa valeur n'est pas `NULL`. Retourne `true` si la variable existe et a une valeur autre que `NULL`, sinon `false`. Très utile pour vérifier si une clé existe dans un tableau (`isset($_POST['champ'])`) avant d'essayer d'y accéder.

`empty($variable)` — Fonction de langage qui vérifie si une variable est considérée comme "vide". Une variable est vide si elle n'existe pas, ou si sa valeur équivaut à `false`. Les valeurs suivantes sont considérées comme vides : `""` (chaîne vide), `0` (entier), `0.0` (flottant), `"0"` (chaîne), `NULL`, `FALSE`, `array()` (tableau vide).

`htmlspecialchars($chaine, ENT_QUOTES, 'UTF-8')` — Convertit les caractères spéciaux en entités HTML. C'est une mesure de sécurité essentielle pour prévenir les attaques XSS (Cross-Site Scripting) lorsque vous affichez des données fournies par l'utilisateur (ou provenant de la base de données) dans une page HTML.

— Caractères convertis par défaut :
  — `&` (esperluette) devient `&`
  — `"` (guillemet double) devient ¨ (seulement si `ENT_NOQUOTES` n'est pas défini)
  — `'` (guillemet simple) devient `&#039;` ou ´ (seulement si `ENT_QUOTES` est défini)
  — `<` (inférieur à) devient `<`
  — `>` (supérieur à) devient `>`

— `ENT_QUOTES` : Convertit à la fois les guillemets doubles et simples.

— `'UTF-8'` : Spécifie l'encodage à utiliser, important pour la cohérence.

5 Mini-exercice : Formulaire de pseudo et affichage

Cet exercice simple illustre l'utilisation des sessions pour passer une information d'une page à une autre, et la redirection après soumission d'un formulaire.

5.1 Fichier `index.php` (Formulaire de saisie)

Ce script affiche un formulaire. Lorsque l'utilisateur soumet son pseudo, celui-ci est stocké dans la session et l'utilisateur est redirigé vers `bienvenue.php`.

Listing 11 – index.php - Formulaire de pseudo

```php
<?php
// Inclut le fichier de configuration de la base de donnes.
// Bien que non utilis directement dans ce script prcis, c'est une bonne habitude
// si d'autres parties de l'application en ont besoin.
require_once __DIR__.'/config/database.php';

// Dmarre la session pour pouvoir stocker/lire des donnes de session.
session_start();

// Vrifie si la requte HTTP est de type POST (formulaire soumis).
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Vrifie si le champ 'pseudo' a t envoy dans le formulaire POST.
    if (isset($_POST['pseudo'])) {
        // Nettoie le pseudo (enlve les espaces au dbut/fin) et le stocke dans la session.
        $_SESSION['pseudo'] = trim($_POST['pseudo']);

        // Redirige l'utilisateur vers la page bienvenue.php.
        header('Location: bienvenue.php');
        // Termine l'excution du script pour s'assurer que la redirection a lieu.
        exit;
    } else {
        // Grer le cas o 'pseudo' n'est pas dfini, si ncessaire
        // echo "Veuillez entrer un pseudo.";
    }
}
// Si la requte n'est pas POST, ou si le champ pseudo n'est pas post,
// le script continue et affiche le formulaire HTML ci-dessous.
?>
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Saisie du Pseudo</title>
</head>
```

```
<body>
    <form method="post" action="index.php"> (*@\textit{action="index.php" est facultatif,
        ↪ par dfaut le formulaire se soumet  la page courante}@*)
      <label for="pseudo_id">Votre pseudo :</label>
      <input type="text" id="pseudo_id" name="pseudo" placeholder="Entrez votre pseudo ici
          ↪ " required>
      <button type="submit">OK</button>
    </form>
</body>
</html>
```

5.2 Fichier `bienvenue.php` (Page d'accueil)

Ce script récupère le pseudo stocké dans la session et l'affiche.

Listing 12 – bienvenue.php - Affichage du pseudo

```
<?php
// Dmarre  la session pour accder aux donnes  stockes .
session_start();

// Prpare  le message d'accueil.
$message = "Bonjour ";

// Vrifie  si la variable de session 'pseudo' existe et n'est pas vide.
// L'oprateur de coalescence nulle (??) est utilis ici :
// $_SESSION['pseudo'] ?? 'inconnu' retourne la valeur de $_SESSION['pseudo'] si elle
    ↪ existe et n'est pas NULL,
// sinon, il retourne 'inconnu'.
$pseudo = $_SESSION['pseudo'] ?? 'cher inconnu';

// chappe  les caractres  spciaux  du pseudo avant de l'afficher pour la scurit (prvention
    ↪  XSS).
$message .= htmlspecialchars($pseudo);
$message .= " !";
?>
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Bienvenue</title>
</head>
<body>
    <h1>Page d'accueil</h1>
    <p><?php echo $message; ?></p>
    <p><a href="index.php">Retourner  la saisie</a></p>
    (*@\textit{Optionnel : Bouton de dconnexion}@*)
    (*@\textit{<form method="post" action="deconnexion.php">}@*)
```

```
    (*@\textit{ <button type="submit">Dconnexion</button>}@*)
    (*@\textit{</form>}@*)
</body>
</html>
```

Explication du fonctionnement :

1. L'utilisateur ouvre `index.php` et voit le formulaire.
2. Il saisit un pseudo et clique sur "OK". Le formulaire est soumis en méthode POST à `index.php` lui-même.
3. Dans `index.php`, la condition `$_SERVER['REQUEST_METHOD'] === 'POST'` est vraie. Le pseudo est récupéré de `$_POST['pseudo']`, nettoyé avec `trim()`, et stocké dans `$_SESSION['pseudo']`.
4. Le script effectue ensuite une redirection HTTP vers `bienvenue.php` grâce à `header('Location: bienvenue.php'); exit;`.
5. Le navigateur de l'utilisateur charge `bienvenue.php`.

6. Dans `bienvenue.php`, `session_start()` reprend la session existante. Le script accède à `$_SESSION['pseudo']` et affiche le message de bienvenue personnalisé. L'opérateur ' ? ¿ (coalescence nulle) fournit une valeur par défaut si '$_SESSION['pseudo']'n'estpasdfini, cequiestunebonnepratique.

À retenir : Bonnes Pratiques Essentielles

- **PDO avec requêtes préparées** : C'est le standard moderne et sécurisé pour interagir avec les bases de données en PHP. **Ne jamais** construire des requêtes SQL en concaténant directement des variables utilisateur.
- **Sessions PHP (`$_SESSION`)** : Utiles pour maintenir l'état d'un utilisateur entre plusieurs pages (par exemple, pour l'authentification, un panier d'achat). L'identifiant de session est géré côté client (généralement via un cookie), mais les données réelles sont stockées côté **serveur**, ce qui est plus sécurisé pour les informations sensibles.
- **Toujours valider et assainir les entrées utilisateur** :
  - Utilisez `trim()` pour supprimer les espaces superflus.
  - Utilisez `isset()` ou `empty()` pour vérifier l'existence et le contenu des variables (surtout pour `$_POST`, `$_GET`, `$_SESSION`).
  - Utilisez `htmlspecialchars()` (ou des bibliothèques de templating qui le font automatiquement) avant d'afficher toute donnée provenant de l'utilisateur ou de la base de données dans du HTML, pour prévenir les failles XSS.
  - Validez le type et le format des données (ex : un email doit ressembler à un email, un nombre doit être un nombre).
- **Redirection après POST (Pattern PRG)** : Après une action qui modifie des données (typiquement une soumission de formulaire en POST), redirigez l'utilisateur vers une autre page (souvent la page de confirmation ou la liste mise à jour) en utilisant `header('Location: nouvelle_page.php'); exit;`. Cela évite les problèmes de resoumission de formulaire si l'utilisateur actualise la page ou utilise le bouton "Précédent" du navigateur.

— **Gestion des erreurs** : Utilisez des blocs `try...catch` pour les opérations PDO. Loggez les erreurs techniques côté serveur et affichez des messages conviviaux aux utilisateurs.