Tutoriel MVC PHP (PDO / MySQL)

1. Objectif du TP

- **À la fin de ce TP, l'apprenant sera capable de :**
  - Mettre en place une **CRUD complète** (Create, Read, Update, Delete) sans framework.
  - Utiliser **PDO** (PHP Data Objects) pour interagir de manière sécurisée avec MySQL.
  - Structurer une application PHP proprement selon le pattern **MVC** (Modèle – Vue – Contrôleur).
  - Créer un **modèle `User`** relié à une table MySQL.
  - Utiliser un **contrôleur** pour gérer les actions CRUD.
  - Afficher les données dans des **vues** propres et sécurisées.
  - Comprendre le **cheminement complet d'une requête** dans une architecture MVC.
- **Rappel – Qu'est-ce que le MVC ?**

  Le pattern **MVC** sépare l'application en trois couches distinctes :
  - **Modèle (Model)**: accès aux données, SQL, logique métier.
  - **Vue (View)**: affichage HTML, aucune logique métier.
  - **Contrôleur (Controller)**: réception des requêtes, appel du modèle, choix de la vue.

2. Création de la base de données

- Nous allons créer une base de données `mvc_app` contenant une table `users` pour notre CRUD.
- Exécutez ce script SQL dans **phpMyAdmin** ou votre terminal MySQL :

```
-- Création de la base de données
CREATE DATABASE mvc_app CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- Sélection de la base
USE mvc_app;

-- Création de la table users
CREATE TABLE users (
```

apcpedagogie.com

```
    id    INT AUTO_INCREMENT PRIMARY KEY,  -- Identifiant unique auto-incrémenté
    name  VARCHAR(100) NOT NULL,           -- Nom de l'utilisateur (obligatoire)
    email VARCHAR(150) NOT NULL UNIQUE     -- Email unique (obligatoire)
);

-- Insertion de données de test
INSERT INTO users (name, email) VALUES
    ('Alice Martin', 'alice@example.com'),
    ('Bob Dupont',   'bob@example.com');
```

- **Remarque :** L'attribut `UNIQUE` sur `email` empêche les doublons au niveau de la base de données.

3. Architecture complète du projet MVC (CRUD Users)

- Voici l'arborescence complète du projet à créer **manuellement**:

```
mvc-users/
│
├── app/                          ← Code applicatif (MVC)
│   ├── controllers/
│   │   └── UserController.php    ← Contrôleur : gère les actions CRUD
│   │
│   ├── models/
│   │   └── User.php              ← Modèle : requêtes SQL sur la table users
│   │
│   └── views/
│       └── users/
│           ├── index.php         ← Vue : liste des utilisateurs
│           ├── create.php        ← Vue : formulaire d'ajout
│           └── edit.php          ← Vue : formulaire de modification
│
├── core/                         ← Classes de base du framework maison
│   ├── Model.php                 ← Connexion PDO partagée
│   ├── Controller.php            ← Classe parente des contrôleurs
│   └── Router.php                ← Routeur URL → contrôleur/action
│
├── config/
│   └── database.php              ← Paramètres de connexion BDD
│
├── routes/
│   └── web.php                   ← Déclaration des routes de l'application
```

apcpedagogie.com

```
│
├── public/
│   └── index.php                ← Point d'entrée unique (Front Controller)
│
├── .htaccess                    ← Réécriture d'URL vers public/index.php
└── README.md                    ← Documentation du projet
```

- **? Tous ces fichiers doivent être créés manuellement.**
- **Principe du Front Controller :** toutes les requêtes HTTP passent par `public/index.php`, qui délègue ensuite au bon contrôleur via le routeur.

4. Fichiers

1. **public/index.php – Point d'entrée unique (Front Controller)**
   - **Chemin :** `public/index.php`

```php
<?php

// Chargement du routeur central
require_once '../core/Router.php';

// Instanciation du routeur
$router = new Router();

// Chargement des routes déclarées dans routes/web.php
require_once '../routes/web.php';

// Démarrage du routeur : analyse l'URL et dispatche vers le bon contrôleur
$router->run();
```

   - **Rôle :** démarre l'application MVC – c'est le seul fichier appelé directement par le navigateur.
   - ***Toutes les requêtes passent obligatoirement par ce fichier grâce au .htaccess.***

2. **.htaccess – Réécriture d'URL (à la racine du projet)**
   - **Chemin :** `.htaccess` (à la racine du projet)

apcpedagogie.com

```
RewriteEngine On

# Si le fichier ou dossier demandé n'existe pas physiquement sur le serveur
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

# Redirige toutes les requêtes vers public/index.php
RewriteRule ^(.*)$ public/index.php [QSA,L]
```

- **Sert à :**
  - Masquer les fichiers internes (modèles, config, etc.) du navigateur.
  - Forcer toutes les URLs à passer par le Front Controller MVC.
  - Permettre des URLs propres comme `/users` ou `/users/edit/3`.
- ⚠ **Prérequis :** le module `mod_rewrite` doit être activé dans Apache.

3. **config/database.php – Configuration de la base de données**

- **Chemin :** `config/database.php`

```
<?php

// Retourne un tableau de configuration pour la connexion PDO
// Ce fichier est chargé par core/Model.php
return [
    'host'     => 'localhost',   // Hôte MySQL (souvent localhost en dev)
    'dbname'   => 'mvc_app',     // Nom de la base de données créée
précédemment
    'user'     => 'root',        // Utilisateur MySQL
    'password' => ''             // Mot de passe (vide par défaut avec
XAMPP/WAMP)
];
```

- **Bonne pratique :** aucun SQL ici, uniquement la configuration. En production, ne commitez jamais ce fichier dans Git (ajoutez-le à `.gitignore`).

4. **core/Model.php – Classe de base PDO (parent de tous les modèles)**

- **Chemin :** `core/Model.php`

```
<?php
```

apcpedagogie.com

```
/**
 * Classe Model – Classe parente de tous les modèles de l'application.
 * Elle centralise la connexion PDO pour éviter de la répéter dans chaque
modèle.
 */
class Model
{
    // Propriété partagée : instance PDO accessible dans toutes les sous-
classes
    protected $db;

    public function __construct()
    {
        // Chargement de la configuration depuis config/database.php
        $config = require '../config/database.php';

        // Création de la connexion PDO avec le DSN MySQL
        $this->db = new PDO(

"mysql:host={$config['host']};dbname={$config['dbname']};charset=utf8mb4",
            $config['user'],
            $config['password']
        );

        // Mode d'erreur : lance une exception en cas d'erreur SQL (facilite
le débogage)
        $this->db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        // Retourne les résultats sous forme de tableaux associatifs par
défaut
        $this->db->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE,
PDO::FETCH_ASSOC);
    }
}
```

- **Tous les modèles héritent de cette classe** via `class User extends Model`.
- **Avantage :** la connexion est instanciée une seule fois par modèle, centralisée et facile à maintenir.

5. **core/Router.php – Routeur URL**

- **Chemin :** `core/Router.php`

apcpedagogie.com

```php
<?php

/**
 * Classe Router — Analyse l'URL et dispatche vers le bon contrôleur et la bonne action.
 */
class Router
{
    // Tableau des routes enregistrées : ['METHOD /chemin' => callable]
    private $routes = [];

    /**
     * Enregistre une route GET.
     * @param string   $path     L'URL à matcher (ex: '/users')
     * @param callable $callback La fonction ou méthode à exécuter
     */
    public function get($path, $callback)
    {
        $this->routes['GET ' . $path] = $callback;
    }

    /**
     * Enregistre une route POST.
     */
    public function post($path, $callback)
    {
        $this->routes['POST ' . $path] = $callback;
    }

    /**
     * Démarre le routeur : compare l'URL courante aux routes enregistrées.
     */
    public function run()
    {
        // Récupère la méthode HTTP (GET ou POST)
        $method = $_SERVER['REQUEST_METHOD'];

        // Récupère le chemin URL sans les paramètres de requête
        $uri = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);

        foreach ($this->routes as $route => $callback) {
            // Convertit les segments dynamiques comme {id} en regex
```

apcpedagogie.com

```
            [$routeMethod, $routePath] = explode(' ', $route, 2);
            $pattern = preg_replace('/\{[a-z]+\}/', '([^/]+)', $routePath);
            $pattern = '#^' . $pattern . '$#';

            if ($routeMethod === $method && preg_match($pattern, $uri,
$matches)) {
                // Supprime le match complet, garde uniquement les paramètres
                array_shift($matches);
                // Appelle le callback avec les paramètres extraits de l'URL
                call_user_func_array($callback, $matches);
                return;
            }
        }

        // Aucune route correspondante → erreur 404
        http_response_code(404);
        echo "<h1>404 – Page non trouvée</h1>";
    }
}
```

6. **routes/web.php – Déclaration des routes**

   - **Chemin :** `routes/web.php`

```
<?php

// Chargement du contrôleur utilisateur
require_once '../app/controllers/UserController.php';

$controller = new UserController();

// Affiche la liste des utilisateurs
$router->get('/users', function() use ($controller) {
    $controller->index();
});

// Affiche le formulaire de création
$router->get('/users/create', function() use ($controller) {
    $controller->create();
});

// Traite le formulaire de création (POST)
```

apcpedagogie.com

```
$router->post('/users/store', function() use ($controller) {
    $controller->store();
});

// Affiche le formulaire de modification pour l'utilisateur {id}
$router->get('/users/edit/{id}', function($id) use ($controller) {
    $controller->edit($id);
});

// Traite le formulaire de modification (POST)
$router->post('/users/update/{id}', function($id) use ($controller) {
    $controller->update($id);
});

// Supprime l'utilisateur {id}
$router->get('/users/delete/{id}', function($id) use ($controller) {
    $controller->delete($id);
});
```

7. **app/models/User.php – Modèle User (toutes les opérations SQL)**

   - **Chemin :** `app/models/User.php`

```
<?php

require_once '../core/Model.php';

/**
 * Classe User – Modèle représentant la table `users`.
 * Contient UNIQUEMENT les opérations SQL (jamais de HTML ici).
 */
class User extends Model
{
    /**
     * Récupère tous les utilisateurs de la table.
     * @return array Tableau de tous les utilisateurs
     */
    public function getAll()
    {
        // query() suffit pour une requête sans paramètre dynamique
        return $this->db
                    ->query("SELECT * FROM users ORDER BY id DESC")
```

apcpedagogie.com

```
                    ->fetchAll(PDO::FETCH_ASSOC);
    }

    /**
     * Récupère un utilisateur par son identifiant.
     * @param int $id L'identifiant de l'utilisateur
     * @return array|false Les données de l'utilisateur ou false si non trouvé
     */
    public function getById($id)
    {
        // prepare() + execute() = requête préparée → protège contre les
injections SQL
        $stmt = $this->db->prepare("SELECT * FROM users WHERE id = ?");
        $stmt->execute([$id]);
        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * Insère un nouvel utilisateur.
     * @param string $name  Le nom de l'utilisateur
     * @param string $email L'adresse email
     * @return bool true en cas de succès
     */
    public function create($name, $email)
    {
        $stmt = $this->db->prepare(
            "INSERT INTO users (name, email) VALUES (?, ?)"
        );
        return $stmt->execute([$name, $email]);
    }

    /**
     * Met à jour les données d'un utilisateur existant.
     * @param int    $id    L'identifiant de l'utilisateur à modifier
     * @param string $name  Le nouveau nom
     * @param string $email Le nouvel email
     * @return bool true en cas de succès
     */
    public function update($id, $name, $email)
    {
        $stmt = $this->db->prepare(
            "UPDATE users SET name = ?, email = ? WHERE id = ?"
        );
```

apcpedagogie.com

```
        // Attention à l'ordre des paramètres : name, email, puis id
        return $stmt->execute([$name, $email, $id]);
    }

    /**
     * Supprime un utilisateur par son identifiant.
     * @param int $id L'identifiant de l'utilisateur à supprimer
     * @return bool true en cas de succès
     */
    public function delete($id)
    {
        $stmt = $this->db->prepare("DELETE FROM users WHERE id = ?");
        return $stmt->execute([$id]);
    }
}
```

- **Règle d'or :** SQL uniquement ici – jamais de HTML dans un modèle.
- **Sécurité :** toutes les requêtes avec données utilisateur utilisent des **requêtes préparées** (`prepare` + `execute`) pour éviter les injections SQL.

8. **app/controllers/UserController.php – Contrôleur User**

   - **Chemin :** `app/controllers/UserController.php`

```
<?php

require_once '../app/models/User.php';

/**
 * Classe UserController – Contrôleur gérant toutes les actions CRUD sur les utilisateurs.
 * Le contrôleur fait le lien entre le modèle (données) et la vue (affichage).
 * Il ne contient ni SQL, ni HTML brut.
 */
class UserController
{
    /**
     * Action index – Affiche la liste de tous les utilisateurs.
     * Correspond à la route : GET /users
     */
    public function index()
    {
        $model = new User();
```

```
        $users = $model->getAll();           // Récupère les données via le
modèle
        require '../app/views/users/index.php'; // Délègue l'affichage à la
vue
    }

    /**
     * Action create — Affiche le formulaire de création d'un utilisateur.
     * Correspond à la route : GET /users/create
     */
    public function create()
    {
        require '../app/views/users/create.php';
    }

    /**
     * Action store — Traite le formulaire de création (méthode POST).
     * Correspond à la route : POST /users/store
     */
    public function store()
    {
        // Récupération et nettoyage des données du formulaire
        $name  = trim($_POST['name']  ?? '');
        $email = trim($_POST['email'] ?? '');

        // Validation basique avant insertion
        if (!empty($name) && filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $model = new User();
            $model->create($name, $email);
        }

        // Redirection vers la liste après enregistrement
        header("Location: /users");
        exit; // Important : stoppe l'exécution après un header redirect
    }

    /**
     * Action edit — Affiche le formulaire de modification d'un utilisateur.
     * Correspond à la route : GET /users/edit/{id}
     * @param int $id L'identifiant de l'utilisateur à modifier
     */
    public function edit($id)
    {
```

apcpedagogie.com

```
        $model = new User();
        $data  = $model->getById($id); // Récupère les données actuelles de
l'utilisateur

        if (!$data) {
            // Si l'utilisateur n'existe pas, redirection vers la liste
            header("Location: /users");
            exit;
        }

        require '../app/views/users/edit.php'; // Passe $data à la vue
    }

    /**
     * Action update — Traite le formulaire de modification (méthode POST).
     * Correspond à la route : POST /users/update/{id}
     * @param int $id L'identifiant de l'utilisateur à mettre à jour
     */
    public function update($id)
    {
        $name  = trim($_POST['name']  ?? '');
        $email = trim($_POST['email'] ?? '');

        if (!empty($name) && filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $model = new User();
            $model->update($id, $name, $email);
        }

        header("Location: /users");
        exit;
    }

    /**
     * Action delete — Supprime un utilisateur.
     * Correspond à la route : GET /users/delete/{id}
     * @param int $id L'identifiant de l'utilisateur à supprimer
     */
    public function delete($id)
    {
        $model = new User();
        $model->delete($id);
        header("Location: /users");
        exit;
```

apcpedagogie.com

```
    }
}
```

- 💡 **Bonne pratique :** toujours ajouter `exit;` après un `header("Location: ...")` pour éviter que le script continue à s'exécuter.

5. Vue : Liste des utilisateurs

- **Chemin :** `app/views/users/index.php`
- La variable `$users` est transmise par le contrôleur. La vue ne fait qu'afficher les données.

```
<!-- Lien vers le formulaire de création -->
<a href="/users/create">➕ Ajouter un utilisateur</a>

<table border="1">
    <thead>
        <tr>
            <th>ID</th>
            <th>Nom</th>
            <th>Email</th>
            <th>Actions</th>
        </tr>
    </thead>
    <tbody>

    <?php if (empty($users)): ?>
        <!-- Cas où la table est vide -->
        <tr>
            <td colspan="4">Aucun utilisateur trouvé.</td>
        </tr>
    <?php else: ?>

        <?php foreach ($users as $user): ?>
        <tr>
            <td><?= $user['id'] ?></td>

            <!-- htmlspecialchars() protège contre les attaques XSS -->
            <td><?= htmlspecialchars($user['name'],  ENT_QUOTES, 'UTF-8') ?></td>
            <td><?= htmlspecialchars($user['email'], ENT_QUOTES, 'UTF-8') ?></td>

            <td>
                <a href="/users/edit/<?= (int)$user['id'] ?>">✏️ Modifier</a>
```

apcpedagogie.com

```
                <!-- Idéalement, la suppression devrait passer par un formulaire
POST -->
                <a href="/users/delete/<?= (int)$user['id'] ?>"
                   onclick="return confirm('Confirmer la suppression ?')">
                   ?️ Supprimer
                </a>
            </td>
        </tr>
        <?php endforeach; ?>

    <?php endif; ?>
    </tbody>
</table>
```

- **Sécurité :** `htmlspecialchars()` convertit les caractères spéciaux HTML (<, >, etc.) en entités HTML pour éviter les attaques **XSS**.
- **Sécurité :** `(int)$user['id']` force le transtypage en entier pour éviter toute injection dans l'URL.

6. Vue : Création d'un utilisateur

- **Chemin :** `app/views/users/create.php`

```
<h2>Ajouter un utilisateur</h2>

<!-- action pointe vers la route POST /users/store -->
<form method="post" action="/users/store">

    <label for="name">Nom :</label><br>
    <input type="text"
           id="name"
           name="name"
           placeholder="Ex : Alice Martin"
           required
           maxlength="100">
    <br><br>

    <label for="email">Email :</label><br>
    <input type="email"
           id="email"
           name="email"
```

apcpedagogie.com

```
            placeholder="Ex : alice@example.com"
            required
            maxlength="150">
    <br><br>

    <button type="submit">? Enregistrer</button>
    <a href="/users">Annuler</a>

</form>
```

- **Note :** les attributs `required` et `type="email"` offrent une première validation côté navigateur. La validation côté serveur (dans le contrôleur) reste indispensable.

7. Vue : Modification d'un utilisateur

- **Chemin :** `app/views/users/edit.php`
- La variable `$data` contient les informations actuelles de l'utilisateur, transmises par le contrôleur.

```
<h2>Modifier l'utilisateur</h2>

<!-- L'id est intégré dans l'URL de l'action -->
<form method="post" action="/users/update/<?= (int)$data['id'] ?>">

    <label for="name">Nom :</label><br>
    <!-- value pré-remplit le champ avec la valeur actuelle —
         htmlspecialchars protège contre le XSS -->
    <input type="text"
           id="name"
           name="name"
           value="<?= htmlspecialchars($data['name'], ENT_QUOTES, 'UTF-8') ?>"
           required
           maxlength="100">
    <br><br>

    <label for="email">Email :</label><br>
    <input type="email"
           id="email"
           name="email"
           value="<?= htmlspecialchars($data['email'], ENT_QUOTES, 'UTF-8') ?>"
           required
```

apcpedagogie.com

```
            maxlength="150">
    <br><br>

    <button type="submit">✏ Mettre à jour</button>
    <a href="/users">Annuler</a>

</form>
```

8. Schéma du flux MVC – Comment fonctionne une requête ?

- Voici le cheminement complet d'une requête dans notre application MVC :

```
Navigateur
   │
   │  GET /users
   ▼
.htaccess
   │  Redirige tout vers public/index.php
   ▼
public/index.php  (Front Controller)
   │  Instancie le Router et charge les routes
   ▼
core/Router.php
   │  Analyse l'URL, trouve la route correspondante
   ▼
app/controllers/UserController.php → action index()
   │  Appelle le modèle pour récupérer les données
   ▼
app/models/User.php → getAll()
   │  Exécute la requête SQL via PDO
   │  Retourne un tableau de résultats
   ▼
UserController::index()
   │  Passe les données ($users) à la vue
   ▼
app/views/users/index.php
   │  Génère le HTML avec les données
   ▼
Navigateur  (affichage de la page)
```

apcpedagogie.com

9. À retenir

- Le **Modèle** contient tout le SQL – rien d'autre.
- Le **Contrôleur** gère la logique : reçoit la requête, appelle le modèle, choisit la vue.
- La **Vue** affiche les données – aucune logique métier, aucun SQL.
- Toujours utiliser **des requêtes préparées** (PDO `prepare` + `execute`) pour sécuriser les requêtes SQL.
- Toujours utiliser **`htmlspecialchars()`** dans les vues pour protéger contre le XSS.
- Toujours ajouter **`exit;`** après `header("Location: ...")`.
- Pas de SQL dans les vues.
- Pas de HTML dans les modèles.
- Pas de logique métier complexe dans les vues.
- **MVC = structure claire, maintenable et professionnelle.**

Cours PHP

Source : https://apcpedagogie.com/tutoriel-mvc-php-pdo-mysql · téléchargé le 07/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com