

Normes de Code - PHP MVC Starter

Ce document décrit les conventions et normes de code utilisées dans ce projet PHP MVC développé en approche procédurale.



Table des matières

- [Architecture générale](#)
- [Conventions de nommage](#)
- [Structure des fichiers](#)
- [Conventions PHP](#)
- [Conventions de base de données](#)
- [Conventions Frontend](#)
- [Sécurité](#)
- [Documentation](#)
- [Gestion des erreurs](#)



Architecture générale

Pattern MVC Procédural

- **Modèle** : Fonctions de gestion des données (pas de classes)
- **Vue** : Templates PHP avec système de layout
- **Contrôleur** : Fonctions de logique métier et coordination

Séparation des responsabilités

```
|— config/           # Configuration (constantes, paramètres)
|— controllers/      # Logique de contrôle
|— models/          # Accès aux données
|— views/           # Présentation
|— core/            # Système de base (routing, templating, DB)
|— includes/        # Utilitaires et helpers
|— public/          # Point d'entrée et assets statiques
```



Conventions de nommage

Fichiers et répertoires

- **Fichiers PHP** : **snake_case** avec suffixes descriptifs
 - Contrôleurs : **{nom}_controller.php**
 - Modèles : **{nom}_model.php**
 - Configuration : **{nom}.php**
- **Vues** : **snake_case.php**
- **Assets** : **kebab-case** pour CSS/JS

```
// ✅ Correct
controllers/home_controller.php
models/user_model.php
views/auth/login.php

// ❌ Incorrect
controllers/HomeController.php
models/userModel.php
views/Auth/Login.php
```

Variables et fonctions

- **Variables** : `snake_case`
- **Fonctions** : `snake_case` avec préfixes organisationnels
- **Constantes** : `UPPER_CASE`

```
// ✅ Variables
$user_name = 'John Doe';
$current_user_id = 123;
$database_connection = null;

// ✅ Fonctions avec préfixes
function home_index() { }           // Contrôleur home, action index
function get_user_by_id($id) { }    // Modèle user, récupération
function create_user($data) { }     // Modèle user, création
function db_select($query) { }      // Système DB, sélection

// ✅ Constantes
define('DB_HOST', 'localhost');
define('APP_NAME', 'PHP MVC Starter');
define('ROOT_PATH', __DIR__);
```

Structure des fichiers

Contrôleurs

- Une fonction par action
- Préfixe : `{contrôleur}_{action}`
- Validation des données en entrée
- Chargement des vues avec layout

```
<?php
// controllers/home_controller.php

/**
 * Page d'accueil
```

```

*/
function home_index() {
    $data = [
        'title' => 'Accueil',
        'message' => 'Bienvenue'
    ];

    load_view_with_layout('home/index', $data);
}

/**
 * Traitement de formulaire
 */
function home_contact() {
    if (is_post()) {
        $name = clean_input(post('name'));
        // Validation et traitement...
    }

    load_view_with_layout('home/contact', $data);
}

```

Modèles

- Fonctions CRUD avec préfixes standardisés
- Requêtes préparées obligatoires
- Gestion des erreurs

```

<?php
// models/user_model.php

/**
 * Récupération d'un utilisateur
 */
function get_user_by_id($id) {
    $query = "SELECT * FROM users WHERE id = ? LIMIT 1";
    return db_select_one($query, [$id]);
}

/**
 * Création d'un utilisateur
 */
function create_user($name, $email, $password) {
    $hashed_password = hash_password($password);
    $query = "INSERT INTO users (name, email, password, created_at)
VALUES (?, ?, ?, NOW())";

    if (db_execute($query, [$name, $email, $hashed_password])) {
        return db_last_insert_id();
    }
}

```

```
        return false;
    }
```

Vues

- Utilisation du système de layout
- Échappement systématique des données
- Séparation logique/présentation

```
<!-- views/home/index.php -->
<div class="page-header">
    <h1><?php e($title); ?></h1>
</div>

<section class="content">
    <p><?php e($message); ?></p>

    <?php if (!empty($features)): ?>
        <ul>
            <?php foreach ($features as $feature): ?>
                <li><?php e($feature); ?></li>
            <?php endforeach; ?>
        </ul>
    <?php endif; ?>
</section>
```

Conventions PHP

Structure des fonctions

```
/**
 * Description de la fonction
 *
 * @param string $param Description du paramètre
 * @return bool Description du retour
 */
function function_name($param) {
    // Validation des paramètres
    if (empty($param)) {
        return false;
    }

    // Logique principale
    $result = process_data($param);

    // Retour
```

```
    return $result;
}
```

Gestion des sessions

```
// Démarrage de session dans bootstrap.php
if (session_status() == PHP_SESSION_NONE) {
    session_start();
}

// Utilisation des helpers pour les données de session
if (is_logged_in()) {
    $user_id = current_user_id();
}
```

Configuration


- Toutes les constantes dans `config/database.php`
- Pas de magic numbers dans le code
- Variables d'environnement pour la configuration sensible

```
// config/database.php
define('DB_HOST', 'localhost');
define('DB_NAME', 'php_mvc_app');
define('BASE_URL', 'http://localhost/php-starter-cdpi/public');
define('ROOT_PATH', dirname(__DIR__));
```

Conventions de base de données

Nommage des tables

- **Tables** : pluriel, `snake_case`
- **Colonnes** : `snake_case`
- **Clés primaires** : `id`
- **Clés étrangères** : `{table}_id`

```
--  Correct
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
    CURRENT_TIMESTAMP
```

```
);

CREATE TABLE posts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    content TEXT,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Requêtes

- Toujours utiliser des requêtes préparées
- Fonctions wrapper pour PDO
- Gestion des erreurs

```
// ✅ Correct – Requête préparée
$user = db_select_one("SELECT * FROM users WHERE email = ?", [$email]);

// ❌ Incorrect – Injection SQL possible
$user = db_select("SELECT * FROM users WHERE email = '$email'");
```

Conventions Frontend

CSS

- **Variables CSS** : `--kebab-case`
- **Classes** : `kebab-case` ou BEM
- **Responsive first**
- **Utilisation de flexbox/grid**

```
/* Variables CSS */
:root {
    --primary-color: #3b82f6;
    --text-color: #1f2937;
    --border-radius: 0.375rem;
}

/* Classes BEM */
.nav-menu { }
.nav-menu__item { }
.nav-menu__item--active { }

/* Classes utilitaires */
.container { }
.btn-primary { }
.alert-error { }
```

HTML

- **Sémantique** : utilisation correcte des balises HTML5
- **Accessibilité** : attributs `alt`, `aria-*`
- **Formulaires** : labels associés, validation

```
<!-- ✅ Structure sémantique -->
<header class="header">
  <nav class="navbar" role="navigation" aria-label="Navigation
principale">
    <ul class="nav-menu">
      <li><a href="/" aria-current="page">Accueil</a></li>
    </ul>
  </nav>
</header>

<main class="main-content">
  <!-- Contenu principal -->
</main>
```

JavaScript

- **Vanilla JS** privilégié
- **Fonctions nommées**
- **Event delegation**

```
// ✅ Conventions JavaScript
document.addEventListener('DOMContentLoaded', function() {
  initializeApp();
});

function initializeApp() {
  setupFormValidation();
  setupNavigation();
}

function setupFormValidation() {
  // Logique de validation
}
```

Sécurité

Protection XSS

- **Échappement systématique** des données d'affichage

- **Fonctions helpers** : `esc()`, `e()`

```
// ✅ Correct
<h1><?php e($title); ?></h1>
<p><?php echo esc($user_input); ?></p>

// ❌ Dangereux
<h1><?php echo $title; ?></h1>
```

Protection CSRF

- **Token CSRF** pour tous les formulaires
- **Vérification** côté serveur

```
<!-- Vue -->
<form method="POST">
    <input type="hidden" name="csrf_token" value="<?php echo
csrf_token(); ?>">
    <!-- autres champs -->
</form>

// Contrôleur
if (is_post()) {
    if (!verify_csrf_token(post('csrf_token'))) {
        set_flash('error', 'Token CSRF invalide');
        return;
    }
    // Traitement sécurisé
}
```

Mots de passe

- **Hachage sécurisé** avec `password_hash()`
- **Vérification** avec `password_verify()`

```
// Création
$hashed = hash_password($password);

// Vérification
if (verify_password($password, $stored_hash)) {
    // Connexion autorisée
}
```



Documentation


```
/**
 * Description courte de la fonction
 *
 * Description longue si nécessaire, expliquant
 * le comportement et les cas particuliers.
 *
 * @param string $email L'adresse email à valider
 * @param int $max_length Longueur maximale autorisée
 * @return bool True si valide, false sinon
 * @throws InvalidArgumentException Si l'email est vide
 */
function validate_email($email, $max_length = 255) {
    // Implémentation
}
```

Commentaires de code

```
// Commentaire explicatif pour une logique complexe
if ($user_attempts >= MAX_LOGIN_ATTEMPTS) {
    // Bloquer temporairement après trop de tentatives
    block_user_temporarily($user_id);
}

/**
 * Section importante du code
 * =====
 */
```

⚠ Gestion des erreurs

Codes de réponse HTTP

```
// 404 - Page non trouvée
function load_404() {
    http_response_code(404);
    load_view('errors/404');
}

// 403 - Accès interdit
function require_login() {
    if (!is_logged_in()) {
        http_response_code(403);
        redirect('auth/login');
    }
}
```

```
}  
}
```

Messages flash

```
// Types standardisés  
set_flash('success', 'Opération réussie');  
set_flash('error', 'Une erreur est survenue');  
set_flash('warning', 'Attention à...');  
set_flash('info', 'Information importante');
```

Validation des données

```
function validate_user_data($data) {  
    $errors = [];  
  
    if (empty($data['name'])) {  
        $errors[] = 'Le nom est obligatoire';  
    }  
  
    if (!validate_email($data['email'])) {  
        $errors[] = 'Email invalide';  
    }  
  
    if (strlen($data['password']) < 8) {  
        $errors[] = 'Mot de passe trop court';  
    }  
  
    return $errors;  
}
```



Bonnes pratiques

Performance

- **Autoloader** pour les fichiers nécessaires uniquement
- **Requêtes optimisées** avec LIMIT
- **Cache** des résultats fréquents

Maintenabilité

- **Fonctions courtes** et spécialisées
- **Séparation des responsabilités**
- **Réutilisabilité** des composants

Tests

- **Fonctions de test** dans `bootstrap.php`
- **Isolation** des tests
- **Données de test** séparées

```
// Fonction de test
function create_test_user($name = 'Test User', $email =
'test@example.com') {
    return create_user($name, $email, 'password123');
}

// Configuration de test
if (defined('TESTING')) {
    define('DB_NAME', 'php_mvc_app_test');
}
```

Résumé des règles essentielles

1. **Nommage** : `snake_case` pour PHP, `kebab-case` pour CSS
2. **Sécurité** : Toujours échapper les données, utiliser les requêtes préparées
3. **Structure** : Une responsabilité par fonction, séparation MVC claire
4. **Documentation** : Commentaires PHPDoc obligatoires pour les fonctions publiques
5. **Validation** : Toujours valider les données en entrée
6. **Erreurs** : Gestion explicite avec messages utilisateur appropriés

Ces normes garantissent un code cohérent, sécurisé et maintenable pour l'ensemble du projet.