

PHP MVC Starter - Boilerplate Procédural

Ce starter kit vous permet de créer rapidement des applications web avec une structure claire et des fonctionnalités de base intégrées.

Fonctionnalités

- **Architecture MVC claire** : Séparation des responsabilités
- **PHP procédural** : Pas d'orienté objet, simple à comprendre
- **Système de routing** : URLs propres et RESTful
- **Templating HTML/CSS** : Système de vues avec layouts
- **Gestion de base de données** : PDO avec fonctions utilitaires
- **Authentification** : Système de connexion/inscription sécurisé
- **Protection CSRF** : Sécurité contre les attaques cross-site
- **Messages flash** : Notifications temporaires
- **Design responsive** : Interface moderne et mobile-friendly
- **Validation de formulaires** : Côté serveur et client

Structure du projet

```
php-starter-cdpi/
├── config/                # Configuration
│   └── database.php      # Config BDD et constantes
├── controllers/          # Contrôleurs MVC
│   ├── home_controller.php
│   └── auth_controller.php
├── models/               # Modèles de données
│   └── user_model.php
├── views/                # Vues et templates
│   ├── layouts/
│   ├── home/
│   ├── auth/
│   └── errors/
├── core/                 # Système de routing et fonctions core
│   ├── database.php     # Fonctions BDD
│   ├── router.php       # Système de routing
│   └── view.php         # Système de templating
├── includes/            # Fonctions utilitaires
│   └── helpers.php
├── public/              # Point d'entrée et assets
│   ├── index.php        # Point d'entrée principal
│   ├── .htaccess        # Configuration Apache
│   └── assets/          # CSS, JS, images
├── database/            # Scripts SQL
│   └── schema.sql       # Schéma de base de données
└── bootstrap.php        # Fichier d'amorçage
```

Installation

Prérequis

- PHP 7.4 ou supérieur
- MySQL 5.7 ou supérieur
- Serveur web (Apache/Nginx)
- Extensions PHP : PDO, MySQL

Étapes d'installation

1. Cloner le projet

```
git clone https://github.com/votre-username/php-starter-cdpi.git
cd php-starter-cdpi
```

2. Configurer la base de données

- Créer une base de données MySQL
- Importer le schéma : `mysql -u root -p votre_db < database/schema.sql`
- Modifier la configuration dans `config/database.php`

3. Configuration Apache

- Pointer le DocumentRoot vers le dossier `public/`
- S'assurer que le module `mod_rewrite` est activé
- Le fichier `.htaccess` est déjà configuré

4. Configuration locale

```
// config/database.php
define('DB_HOST', 'localhost');
define('DB_NAME', 'votre_nom_de_db');
define('DB_USER', 'votre_utilisateur');
define('DB_PASS', 'votre_mot_de_passe');
define('BASE_URL', 'http://localhost/php-starter-cdpi/public');
```

Utilisation

Système de routing

Les URLs suivent le pattern : `base_url/controller/action/params`

Exemples :

- `/` → `home_controller.php` → `home_index()`
- `/auth/login` → `auth_controller.php` → `auth_login()`

- `/home/about` → `home_controller.php` → `home_about()`

Créer un nouveau contrôleur

```
<?php
// controllers/blog_controller.php

function blog_index() {
    $posts = get_all_posts(); // Appel au modèle
    $data = ['posts' => $posts, 'title' => 'Blog'];
    load_view_with_layout('blog/index', $data);
}

function blog_show($id) {
    $post = get_post_by_id($id);
    if (!$post) {
        load_404();
        return;
    }
    $data = ['post' => $post, 'title' => $post['title']];
    load_view_with_layout('blog/show', $data);
}
```

Créer un nouveau modèle

```
<?php
// models/post_model.php

function get_all_posts($limit = null) {
    $query = "SELECT * FROM posts ORDER BY created_at DESC";
    if ($limit) {
        $query .= " LIMIT $limit";
    }
    return db_select($query);
}

function get_post_by_id($id) {
    $query = "SELECT * FROM posts WHERE id = ?";
    return db_select_one($query, [$id]);
}

function create_post($title, $content, $author_id) {
    $query = "INSERT INTO posts (title, content, author_id, created_at)
VALUES (?, ?, ?, NOW())";
    if (db_execute($query, [$title, $content, $author_id])) {
        return db_last_insert_id();
    }
    return false;
}
```

Créer une nouvelle vue

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

<section class="content">
  <div class="container">
    <?php if (!empty($posts)): ?>
      <div class="posts-grid">
        <?php foreach ($posts as $post): ?>
          <article class="post-card">
            <h2><a href="<?php echo url('blog/show/' .
$post['id']); ?>">
              <?php e($post['title']); ?>
            </a></h2>
            <p><?php e(substr($post['content'], 0, 200)); ?
>...</p>
            <time><?php echo
format_date($post['created_at']); ?></time>
          </article>
        <?php endforeach; ?>
      </div>
    <?php else: ?>
      <p>Aucun article trouvé.</p>
    <?php endif; ?>
  </div>
</section>
```

Gestion des formulaires

```
// Dans le contrôleur
function blog_create() {
  if (is_post()) {
    $title = clean_input(post('title'));
    $content = clean_input(post('content'));

    if (empty($title) || empty($content)) {
      set_flash('error', 'Titre et contenu obligatoires');
    } else {
      $post_id = create_post($title, $content, current_user_id());
      if ($post_id) {
        set_flash('success', 'Article créé avec succès');
        redirect('blog/show/' . $post_id);
      }
    }
  }
}
```

```

        } else {
            set_flash('error', 'Erreur lors de la création');
        }
    }
}

load_view_with_layout('blog/create', ['title' => 'Nouvel article']);
}

```

Fonctions utilitaires

Base de données

- `db_select($query, $params)` - Exécuter une requête SELECT
- `db_select_one($query, $params)` - Une seule ligne
- `db_execute($query, $params)` - INSERT/UPDATE/DELETE
- `db_last_insert_id()` - Dernier ID inséré

Vues et templating

- `load_view($view, $data)` - Charger une vue
- `load_view_with_layout($view, $data, $layout)` - Avec layout
- `include_partial($partial, $data)` - Inclure un partial
- `escape($string) / e($string)` - Sécuriser l'affichage

Routing et URLs

- `url($path)` - Générer une URL
- `redirect($path)` - Redirection
- `is_post() / is_get()` - Type de requête

Session et sécurité

- `is_logged_in()` - Vérifier connexion
- `current_user_id()` - ID utilisateur connecté
- `csrf_token()` - Générer token CSRF
- `set_flash($type, $message)` - Message flash

Validation

- `clean_input($data)` - Nettoyer les données
- `validate_email($email)` - Valider email
- `hash_password($password)` - Hacher mot de passe

Personnalisation CSS

Le CSS utilise des variables CSS pour faciliter la personnalisation :

```
:root {  
  --primary-color: #3b82f6;  
  --secondary-color: #6b7280;  
  --success-color: #10b981;  
  --error-color: #ef4444;  
  /* ... */  
}
```

Modifiez ces variables dans `public/assets/css/style.css` pour changer l'apparence.

Sécurité

- **Protection CSRF** : Tokens automatiques dans les formulaires
- **Validation des données** : Nettoyage et validation côté serveur
- **Mots de passe** : Hachage sécurisé avec `password_hash()`
- **Sessions** : Gestion sécurisée des sessions utilisateur
- **SQL Injection** : Requêtes préparées avec PDO

Base de données

Le schéma inclut :

- Table `users` : Gestion des utilisateurs
- Table `contact_messages` : Messages de contact
- Table `sessions` : Sessions alternatives
- Table `settings` : Configuration

Contribution

1. Fork le projet
2. Créer une branche : `git checkout -b feature/ma-fonctionnalite`
3. Commit : `git commit -m 'Ajout ma fonctionnalité'`
4. Push : `git push origin feature/ma-fonctionnalite`
5. Ouvrir une Pull Request

License

Ce projet est sous licence MIT. Voir le fichier `LICENSE` pour plus de détails.

Support

Pour toute question ou problème :

- Ouvrir une issue sur GitHub
- Consulter la documentation dans le code
- Vérifier les exemples dans les contrôleurs

/**

- Section importante du code
 - =====
- */

```
## △ Gestion des erreurs

### Codes de réponse HTTP
```php
// 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;
}


```


## 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
}

```