# Normes de Codage

**Auteur**: REICHHART Damien - BENAD Kilian
**Date de création**: 23/04/2024
**Dernière modification**: 23/04/2024

## Sommaire



## Introduction

Ce document définit les normes de codage à respecter pour le développement de l'application. Ces normes visent à assurer la cohérence, la lisibilité et la maintenabilité du code.

## Conventions de Nommage

### 1. Fichiers

- Utiliser le PascalCase pour les noms de classes
- Utiliser le snake_case pour les autres fichiers
- Extension `.php` pour les fichiers PHP
- Extension `.twig` pour les templates

Exemples :

```
UserController.php
auth_middleware.php
login.twig
```

### 2. Classes

- Utiliser le PascalCase
- Nommer selon le pattern `[Nom]Controller`, `[Nom]Service`, etc.

Exemples :

```
class UserController
class DockerService
class SecurityMiddleware
```

### 3. Méthodes

- Utiliser le camelCase
- Commencer par un verbe
- Être descriptif

Exemples :

```
public function getUserById(int $id): User
public function validateContainerConfig(array $config): bool
```

### 4. Variables

- Utiliser le camelCase
- Être descriptif
- Éviter les abréviations

Exemples :

```
$userList = [];
$containerConfig = [];
$isValid = true;
```

### 5. Constantes

- Utiliser le UPPER_SNAKE_CASE
- Préfixer par le nom de la classe si nécessaire

Exemples :

```
const MAX_CONTAINERS = 10;
const DEFAULT_TIMEOUT = 30;
```

## Structure du Code

### 1. Organisation des Fichiers

- Un fichier = une classe
- Regrouper les fichiers par fonctionnalité
- Suivre la structure MVC

### 2. Formatage

- Indentation : 4 espaces
- Longueur maximale de ligne : 120 caractères

- Espaces autour des opérateurs
- Accolades sur nouvelle ligne pour les classes et méthodes

Exemple :

```
class UserController
{
    public function getUserById(int $id): User
    {
        if ($id <= 0) {
            throw new InvalidArgumentException('ID must be positive');
        }

        return $this->userService->findById($id);
    }
}
```

### 3. Ordre des Déclarations

1. Constantes de classe
2. Propriétés
3. Constructeur
4. Méthodes publiques
5. Méthodes protégées
6. Méthodes privées

## Documentation

### 1. Documentation des Classes

```
/**
 * Gestion des utilisateurs
 *
 * @package App\Controller
 */
class UserController
{
    // ...
}
```

### 2. Documentation des Méthodes

```
/**
 * Récupère un utilisateur par son ID
 *
 * @param int $id Identifiant de l'utilisateur
 * @return User
```

```
     * @throws UserNotFoundException Si l'utilisateur n'existe pas
     */
    public function getUserById(int $id): User
    {
        // ...
    }
```

### 3. Documentation des Propriétés

```
    /**
     * @var UserService Service de gestion des utilisateurs
     */
    private UserService $userService;
```

## Bonnes Pratiques

### 1. Sécurité

- Valider toutes les entrées
- Échapper les sorties
- Utiliser des requêtes préparées
- Implémenter CSRF protection

### 2. Performance

- Éviter les requêtes N+1
- Utiliser le lazy loading
- Mettre en cache quand nécessaire

### 3. Maintenabilité

- Respecter le principe DRY
- Écrire des tests unitaires
- Utiliser des types stricts
- Gérer les erreurs proprement

### 4. Tests

- Un test par cas d'utilisation
- Tests unitaires pour chaque méthode
- Tests d'intégration pour les workflows