

# Module 6 — Qualité et Sécurité

Jour 2 — Audits, tests et conformité

The Bearded Bear

Février 2026



## Table des matières

<b>Module 6 : Qualité et Sécurité</b>	<b>2</b>
Objectifs . . . . .	2
1. Pre-commit Checklist . . . . .	2
Les 13 Sections de Validation . . . . .	2
Automatisation avec Git Hooks . . . . .	2
Vérification manuelle avec Claude . . . . .	3
2. Tests TDD/BDD avec Claude . . . . .	3
Cycle TDD . . . . .	3
Demander les tests d'abord . . . . .	3
Implémenter pour passer les tests . . . . .	4
Refactorer . . . . .	4
BDD avec Behat . . . . .	4
3. Sécurité OWASP Top 10 . . . . .	4
A01 : Broken Access Control . . . . .	4
A03 : Injection . . . . .	5
A07 : XSS (Cross-Site Scripting) . . . . .	5
Audit avec Claude . . . . .	5
Correction assistée . . . . .	5
4. Git Workflow et Conventional Commits . . . . .	6
GitHub Flow . . . . .	6
Format Conventional Commits . . . . .	6
Types autorisés . . . . .	6
Exemples . . . . .	7
Validation avec Commitlint . . . . .	7
5. Exercice Pratique . . . . .	7
Objectif . . . . .	7
Étapes . . . . .	7
Critères de succès . . . . .	8
Points Clés à Retenir . . . . .	8

## Module 6 : Qualité et Sécurité

### Objectifs

À la fin de ce module, vous serez capable de : - Utiliser la pre-commit checklist - Appliquer le TDD/BDD avec Claude - Identifier et corriger les vulnérabilités OWASP - Intégrer la qualité dans le workflow Git

### 1. Pre-commit Checklist

#### Les 13 Sections de Validation

PRE-COMMIT CHECKLIST
<ul style="list-style-type: none"><li><input type="checkbox"/> 1. Code compiles sans erreur</li><li><input type="checkbox"/> 2. Tests unitaires passent</li><li><input type="checkbox"/> 3. Tests d'intégration passent</li><li><input type="checkbox"/> 4. Couverture &gt;= 80%</li><li><input type="checkbox"/> 5. PHPStan level 8 sans erreur</li><li><input type="checkbox"/> 6. PHP-CS-Fixer appliqué</li><li><input type="checkbox"/> 7. Pas de TODO/FIXME non documentés</li><li><input type="checkbox"/> 8. Pas de code commenté</li><li><input type="checkbox"/> 9. Documentation à jour</li><li><input type="checkbox"/> 10. Pas de secrets/credentials</li><li><input type="checkbox"/> 11. Validation sécurité OWASP</li><li><input type="checkbox"/> 12. Conventional Commit message</li><li><input type="checkbox"/> 13. PR template rempli</li></ul>

#### Automatisation avec Git Hooks

```
# .husky/pre-commit ou .git/hooks/pre-commit
#!/bin/bash
```

```
echo "[?] Running pre-commit checks..."
```

```
# 1. PHPStan
vendor/bin/phpstan analyse --level=8
if [ $? -ne 0 ]; then
    echo "[KO] PHPStan failed"
    exit 1
fi
```

```
# 2. CS-Fixer
vendor/bin/php-cs-fixer fix --dry-run --diff
if [ $? -ne 0 ]; then
    echo "[K0] Code style issues found"
    exit 1
fi

# 3. Tests
vendor/bin/phpunit --testsuite=unit
if [ $? -ne 0 ]; then
    echo "[K0] Tests failed"
    exit 1
fi

echo "[OK] All checks passed!"
```

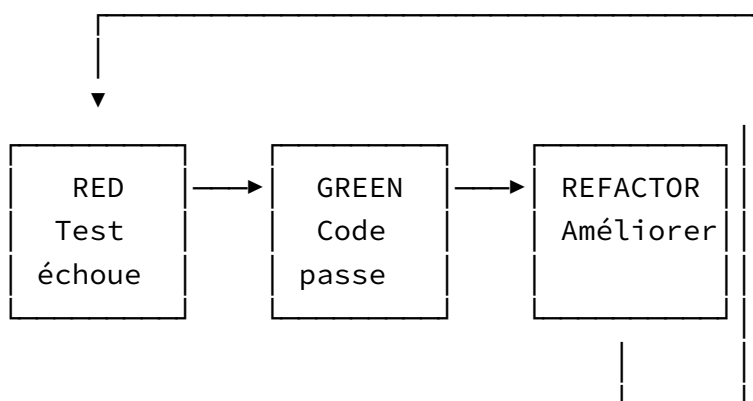
### Vérification manuelle avec Claude

/common:pre-commit-check

*# Claude vérifie toutes les sections et génère un rapport*

## 2. Tests TDD/BDD avec Claude

### Cycle TDD



### Demander les tests d'abord

@tdd-coach "Je dois implémenter un service de calcul de remise.

Règles:

- 10% si total > 100€

- 20% si total > 500€
- 5% supplémentaire si client fidèle

Écris les tests PHPUnit d'abord."

```
# Claude génère:  
# - testNoDiscountUnder100()  
# - testTenPercentOver100()  
# - testTwentyPercentOver500()  
# - testLoyaltyBonus()  
# - testCombinedDiscounts()
```

### Implémenter pour passer les tests

"Maintenant implémente DiscountCalculator pour faire passer tous les tests"

```
# Claude implémente le minimum nécessaire
```

### Refactorer

"Refactore DiscountCalculator en appliquant le pattern Strategy pour les  
↪ règles de remise"

```
# Claude améliore le code en gardant les tests verts
```

### BDD avec Behat

@tdd-coach "Écris les scénarios Gherkin pour le processus de commande"

```
# Claude génère:  
# Feature: Order Process  
#   Scenario: Customer places an order  
#     Given I am a logged in customer  
#     And I have items in my cart  
#     When I proceed to checkout  
#     Then I should see my order confirmation
```

---

## 3. Sécurité OWASP Top 10

### A01 : Broken Access Control

```
// [KO] MAUVAIS - Pas de vérification de propriété  
public function editOrder(int $id): Response  
{  
    $order = $this->orderRepository->find($id);
```

```
// N'importe qui peut modifier n'importe quelle commande !
}

// [OK] BON - Vérification du propriétaire
public function editOrder(int $id): Response
{
    $order = $this->orderRepository->find($id);
    if ($order->getCustomer() !== $this->getUser()) {
        throw new AccessDeniedException();
    }
}
```

### A03 : Injection

```
// [KO] MAUVAIS - SQL Injection
$query = "SELECT * FROM users WHERE email = '$email'";

// [OK] BON - Requête paramétrée
$query = $this->em->createQuery(
    'SELECT u FROM User u WHERE u.email = :email'
)->setParameter('email', $email);
```

### A07 : XSS (Cross-Site Scripting)

```
{# [KO] MAUVAIS - XSS possible #}
{{ user.bio|raw }}
```

```
{# [OK] BON - Échappement automatique #}
{{ user.bio }}
```

```
{# [OK] BON - Échappement explicite si besoin de HTML #}
{{ user.bio|striptags('<p><br>')|raw }}
```

### Audit avec Claude

```
/symfony:check-security

# Identifie automatiquement les vulnérabilités
# Propose des corrections
```

### Correction assistée

```
"Corrige la vulnérabilité SQL Injection dans ProductRepository ligne 34"

# Claude analyse, corrige, et explique la correction
```

## 4. Git Workflow et Conventional Commits

### GitHub Flow

```
main (production-ready)
├─> feature/add-discount-system
│   ├── feat(order): add discount calculation
│   ├── test(order): add discount tests
│   └── refactor(order): extract discount strategy
│       └─> Pull Request → Review → Merge
└─> main (updated)
```

### Format Conventional Commits

<type>(<scope>): <description>

[optional body]

[optional footer]

### Types autorisés

Type	Usage
feat	Nouvelle fonctionnalité
fix	Correction de bug
docs	Documentation
style	Formatage (pas de changement de code)
refactor	Refactoring
perf	Amélioration de performance
test	Ajout/correction de tests
build	Changements de build
ci	Changements CI/CD
chore	Maintenance

## Exemples

```
# Feature
feat(auth): add JWT refresh token endpoint

# Bug fix
fix(cart): correct total calculation with discounts

# Refactoring
refactor(order): extract payment logic to PaymentService

# Tests
test(discount): add edge cases for loyalty bonus
```

## Validation avec Commitlint

```
// commitlint.config.js
module.exports = {
  extends: ['@commitlint/config-conventional'],
  rules: {
    'scope-enum': [2, 'always', ['auth', 'order', 'cart', 'user', 'product']]
  }
};
```

---

## 5. Exercice Pratique

### Objectif

Exécuter un audit qualité complet et corriger les 3 findings les plus critiques.

### Étapes

#### 1. Lancer l'audit

```
/symfony:check-compliance
```

#### 2. Identifier les issues critiques

- Sécurité (priorité 1)
- Tests manquants (priorité 2)
- Code quality (priorité 3)

#### 3. Corriger issue 1 : Sécurité

```
"Corrige [description de l'issue sécurité]"
# Puis ajouter le test de non-régression
```



#### 4. Corriger issue 2 : Tests

```
@tdd-coach "Ajoute les tests manquants pour [composant]"
```

#### 5. Corriger issue 3 : Qualité

```
"Refactore [méthode/classe] pour réduire la complexité"
```

#### 6. Vérifier

```
/symfony:check-compliance  
# Vérifier que les scores ont augmenté
```

#### Critères de succès

- ☐ 3 issues critiques corrigées
  - ☐ Tests de non-régression ajoutés
  - ☐ Score sécurité amélioré
  - ☐ Score qualité amélioré
- 

#### Points Clés à Retenir

1. **Pre-commit** : 13 sections à valider avant chaque commit
  2. **TDD** : Tests d'abord, implémentation ensuite
  3. **OWASP** : Les 10 principales vulnérabilités à connaître
  4. **Conventional Commits** : Format standardisé pour l'historique Git
  5. **Automatisation** : Hooks Git pour garantir la qualité
- 

**Durée estimée** : 1h **Prochain module** : Agents Spécialisés