Cours : Automatiser les tests PHP avec PHPUnit

Introduction : Les tests unitaires et l'automatisation

Les **tests unitaires** en PHP permettent de vérifier que chaque fonction ou méthode fonctionne correctement de manière isolée.

L'**automatisation des tests** signifie que ces tests sont exécutés automatiquement, souvent via des outils comme PHPUnit. Cela garantit la qualité du code, facilite la maintenance et détecte rapidement les régressions.

Pourquoi utiliser PHPUnit ?

- PHPUnit est le framework de test le plus populaire en PHP.
- Il est simple à installer avec Composer.
- Il offre une syntaxe claire pour écrire des tests (classes, assertions).
- Il intègre des outils pour exécuter automatiquement les tests et générer des rapports.
- Compatible avec les projets PHP natifs comme avec les frameworks (Symfony, Laravel, etc.).

---

1. Installer PHPUnit

La méthode recommandée est via Composer.

1. Installer PHPUnit en dépendance de développement :

```
composer require --dev phpunit/phpunit
```

2. Vérifier l'installation :

```
./vendor/bin/phpunit --version
```

3. Créer un raccourci dans le fichier **composer.json** (optionnel) :

```
{
    "scripts": {
        "test": "./vendor/bin/phpunit tests"
    }
}
```

---

2. Structure de base d'un test PHPUnit

Un test PHPUnit est une classe PHP qui hérite de `PHPUnit\Framework\TestCase`. Chaque méthode publique de cette classe qui commence par `test` est considérée comme un test.

Exemple simple dans `tests/MathTest.php` :

```
<?php
use PHPUnit\Framework\TestCase;

class MathTest extends TestCase
{
    public function testAdd()
    {
        $result = 2 + 3;
        $this->assertEquals(5, $result);
    }
}
?>
```

---

3. Exécuter les tests

Pour lancer tous les tests dans le dossier `tests/` :

```
./vendor/bin/phpunit tests
# Ou si vous avez créé le raccourci :
composer test
```

PHPUnit va afficher un résumé des tests passés, échoués ou ignorés.

---

4. Exemple concret avec une classe à tester

Imaginons une classe `Calculator.php` dans `src/` :

```
<?php
class Calculator
{
    public function add($a, $b)
    {
        return $a + $b;
    }

    public function divide($a, $b)
    {
        if ($b === 0) {
            throw new InvalidArgumentException("Division by zero");
```

```
        }
        return $a / $b;
    }
}
?>
```

Le fichier de test `tests/CalculatorTest.php` :

```php
<?php
use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../src/Calculator.php';

class CalculatorTest extends TestCase
{
    private $calculator;

    protected function setUp(): void
    {
        // Instanciation avant chaque test
        $this->calculator = new Calculator();
    }

    public function testAdd()
    {
        $this->assertEquals(7, $this->calculator->add(3, 4));
    }

    public function testDivide()
    {
        $this->assertEquals(2, $this->calculator->divide(6, 3));
    }

    public function testDivideByZero()
    {
        $this->expectException(InvalidArgumentException::class);
        $this->calculator->divide(5, 0);
    }
}
?>
```

5. Exemple avec integration de HTML

Fichier `csrf.php` (fonction à tester)

```php
<?php
session_start();
```

```php
/**
 * Génère un token CSRF, le stocke en session, et affiche un input hidden
 */
function setCSRF(): void
{
    if (session_status() !== PHP_SESSION_ACTIVE) {
        session_start();
    }

    // Générer un token aléatoire
    $token = bin2hex(random_bytes(16));
    // Stocker le token en session
    $_SESSION['csrf_token'] = $token;

    // Afficher l'input hidden avec le token
    echo '<input type="hidden" name="csrf_token" value="' . $token . '">';
}
?>
```

Fichier `CSRFTest.php` (test PHPUnit)

```php
<?php
use PHPUnit\Framework\TestCase;

// Inclure la fonction à tester
require_once 'csrf.php';

class CSRFTest extends TestCase
{
    protected function setUp(): void
    {
        // Démarrer une session pour chaque test
        if (session_status() !== PHP_SESSION_ACTIVE) {
            session_start();
        }

        // Vider la session avant chaque test
        $_SESSION = [];
    }

    public function testSetCSRFOutputAndSession()
    {
        // Démarrer la capture de sortie
        ob_start();

        // Appeler la fonction qui fait echo et set la session
        setCSRF();

        // Récupérer le contenu affiché
```

```
        $output = ob_get_clean();

        // Vérifier que la sortie contient un input hidden avec name csrf_token
        $this->assertStringContainsString('<input type="hidden"', $output);
        $this->assertStringContainsString('name="csrf_token"', $output);

        // Vérifier que la valeur est un token hexadécimal de 32 caractères
        preg_match('/value="([a-f0-9]{32})"/', $output, $matches);
        $this->assertNotEmpty($matches, "Le token CSRF n'a pas été trouvé ou est
incorrect.");

        $tokenFromInput = $matches[1];

        // Vérifier que le token est bien stocké en session
        $this->assertArrayHasKey('csrf_token', $_SESSION);
        $this->assertEquals($tokenFromInput, $_SESSION['csrf_token']);
    }
}
?>
```

Explications

- La fonction `setCSRF()` démarre la session si elle n'est pas déjà active.
- Elle génère un token, le stocke en `$_SESSION['csrf_token']`.
- Elle produit un input HTML avec ce token.
- Dans le test, on démarre aussi la session (via `setUp()`).
- On vide la session avant chaque test pour éviter les interférences.
- On capture la sortie pour vérifier le HTML.
- On vérifie que le token dans l'input correspond bien à celui stocké en session.

Avec ce pattern, tu assures que la protection CSRF est bien mise en place côté serveur et visible côté client.

---

6. Assertions courantes

Voici quelques assertions utiles :

Méthode	Description
`assertEquals(\$expected, \$actual)`	Vérifie que les valeurs sont égales
`assertTrue(\$condition)`	Vérifie que la condition est vraie
`assertFalse(\$condition)`	Vérifie que la condition est fausse
`assertNull(\$variable)`	Vérifie que la variable est nulle
`assertInstanceOf(\$class, \$object)`	Vérifie que l'objet est une instance de la classe donnée
`expectException(\$exceptionClass)`	S'attend à ce qu'une exception soit levée

6. Organiser ses tests

- Place les tests dans un dossier `tests/` à la racine du projet.
- Structure les tests par rapport aux namespaces ou dossiers sources.
- Utilise `setUp()` et `tearDown()` pour initialiser et nettoyer avant/après chaque test.
- Nomme tes méthodes avec un préfixe `test` suivi de ce que tu testes.

---

7. Intégration continue et rapports

- PHPUnit peut générer des rapports en XML (ex : pour Jenkins, GitHub Actions).
- Intègre PHPUnit dans un pipeline CI pour lancer les tests automatiquement.
- Utilise `--coverage` pour mesurer la couverture du code.

---

8. Ressources utiles

- Documentation officielle PHPUnit
- Guide PHPUnit sur Symfony
- Composer
- Exemple de configuration PHPUnit XML

> Avec PHPUnit, tu peux mettre en place une base solide de tests automatisés pour tes projets PHP, améliorer la fiabilité du code et faciliter sa maintenance.