

## TP1 – Gestion de Stock en PHP avec PDO et PHPUnit

### Objectifs pédagogiques

- Créer un projet PHP structuré (sans framework)
- Manipuler une base MySQL avec PDO
- Implémenter un CRUD complet (Créer, Lire, Mettre à jour, Supprimer)
- Écrire et exécuter des **tests unitaires** avec PHPUnit
- Comprendre la séparation entre **code métier, accès base et interface**

### 1. Structure du projet

Crée un dossier :

```
stock-manager/
├── composer.json
├── phpunit.xml
└── vendor/
    └── src/
        ├── Database.php
        ├── Produit.php
        └── StockManager.php
    └── public/
        ├── index.php
        ├── add.php
        ├── edit.php
        ├── delete.php
        └── style.css
    └── tests/
        ├── ProduitTest.php
        └── StockManagerTest.php
    └── database/
        ├── stock_db.sql
        └── produits.json
```

### 2. Initialisation du projet Composer

Dans le terminal :

```
mkdir stock-manager
```

Test-Logiciel

```
cd stock-manager
```

```
composer init
```

Réponds :

- **Nom du projet** : stock-manager
- **Description** : Mini gestion de stock PHP + PDO + PHPUnit
- **Auteur** : Ton nom
- **Autoload** : App\\ → src/

Installe PHPUnit :

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

Puis génère l'autoload :

```
composer dump-autoload
```

### 3. Fichier composer.json

```
{
  "name": "etud/stock-manager",
  "description": "Mini TP gestion de stock avec PDO et PHPUnit",
  "autoload": {
    "psr-4": {
      "App\\": "src/"
    }
  },
  "require-dev": {
    "phpunit/phpunit": "^11.0"
  },
  "authors": [
    {
      "name": "Ton Nom",
      "email": "ton.email@example.com"
    }
  ]
}
```

### 4. Configuration PHPUnit

```
phpunit.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit bootstrap="vendor/autoload.php" colors="true">
    <testsuites>
        <testsuite name="StockManager Test Suite">
            <directory>./tests</directory>
        </testsuite>
    </testsuites>
</phpunit>
```

## 5. Création de la base de données

database/stock\_db.sql

```
CREATE DATABASE stock_db CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci;
USE stock_db;

CREATE TABLE produits (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    prix DECIMAL(10,2) NOT NULL,
    quantite INT DEFAULT 0
);
```

Crée la base :

```
mysql -u root -p < database/stock_db.sql
```

## 6. Connexion à MySQL (PDO)

- src/Database.php

```
<?php  
namespace App;  
use PDO;  
use PDOException;  
class Database  
{  
    private static ?PDO $pdo = null;  
    public static function getConnection(): PDO  
    {  
        if ($self::$pdo === null) {  
            $host = '127.0.0.1';  
            $db = 'stock_db';  
            $user = 'root';  
            $pass = '';  
            $charset = 'utf8mb4';  
            $dsn = "mysql:host=$host;dbname=$db;charset=$charset";  
            try {  
                $self::$pdo = new PDO($dsn, $user, $pass, [  
                    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,  
                    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,  
                ]);  
            } catch (PDOException $e) {  
                die("Erreur de connexion : " . $e->getMessage());  
            }  
        }  
        return $self::$pdo;  
    }  
}
```

## 7. Classe Produit (modèle métier)

- src/Produit.php

```
<?php
namespace App;
use App\Database;
use Exception;
class Produit
{
    private ?int $id;
    private string $nom;
    private float $prix;
    private int $quantite;
    public function __construct(string $nom, float $prix, int $quantite = 0, ?int $id = null)
    {
        $this->id = $id;
        $this->nom = $nom;
        $this->prix = $prix;
        $this->quantite = $quantite;
    }
    public function getId(): ?int { return $this->id; }
    public function getNom(): string { return $this->nom; }
    public function getPrix(): float { return $this->prix; }
    public function getQuantite(): int { return $this->quantite; }
    public function ajouterStock(int $quantite): void
    {
        if ($quantite < 0) throw new Exception("Quantité invalide");
        $this->quantite += $quantite;
    }
}
```

```
public function retirerStock(int $quantite): void
{
    if ($quantite > $this->quantite) throw new Exception("Stock insuffisant");
    $this->quantite -= $quantite;
}

public function save(): void
```

```
{  
    $pdo = Database::getConnection();  
    if ($this->id) {  
        $stmt = $pdo->prepare("UPDATE produits SET nom=?, prix=?, quantite=? WHERE id=?");  
        $stmt->execute([$this->nom, $this->prix, $this->quantite, $this->id]);  
    } else {  
        $stmt = $pdo->prepare("INSERT INTO produits (nom, prix, quantite) VALUES (?, ?, ?)");  
        $stmt->execute([$this->nom, $this->prix, $this->quantite]);  
        $this->id = (int)$pdo->lastInsertId();  
    }  
}  
  
public static function all(): array  
{  
    $pdo = Database::getConnection();  
    $stmt = $pdo->query("SELECT * FROM produits");  
    $result = [];  
    while ($row = $stmt->fetch()) {  
        $result[] = new self($row['nom'], (float)$row['prix'], (int)$row['quantite'],  
            (int)$row['id']);  
    }  
    return $result;  
}  
  
public static function find(int $id): ?self  
{  
    $pdo = Database::getConnection();  
    $stmt = $pdo->prepare("SELECT * FROM produits WHERE id = ?");  
    $stmt->execute([$id]);  
    $row = $stmt->fetch();  
    if (!$row) return null;
```

```
        return new self($row['nom'], (float)$row['prix'], (int)$row['quantite'], (int)$row['id']);  
    }  
  
    public static function delete(int $id): void  
    {  
        $pdo = Database::getConnection();  
        $stmt = $pdo->prepare("DELETE FROM produits WHERE id = ?");  
        $stmt->execute([$id]);  
    }  
}
```

## 8. Classe StockManager

src/StockManager.php

```
<?php  
namespace App;  
  
class StockManager  
{  
    public function ajouterProduit(Produit $produit): void  
    {  
        $produit->save();  
    }  
  
    public function getProduits(): array  
    {  
        return Produit::all();  
    }  
  
    public function getProduit(int $id): ?Produit  
    {  
        return Produit::find($id);  
    }  
  
    public function supprimerProduit(int $id): void
```

```
{  
    Produit::delete($id);  
}  
  
public function totalStock(): float  
{  
    $total = 0;  
    foreach (Produit::all() as $p) {  
        $total += $p->getPrix() * $p->getQuantite();  
    }  
    return $total;  
}  
}
```

## 9. Tests unitaires avec PHPUnit

### tests/ProduitTest.php

```
<?php  
use PHPUnit\Framework\TestCase;  
use App\Database;  
use App\Produit;  
  
class ProduitTest extends TestCase  
{  
    protected function setUp(): void  
    {  
        Database::getConnection()->exec("DELETE FROM produits");  
    }  
  
    public function test_creation_produit()  
    {  
        $p = new Produit("Stylo", 1000, 5);  
        $p->save();  
  
        $this->assertNotNull($p->getId());  
        $this->assertEquals("Stylo", $p->getNom());  
    }  
}
```

```
$this->assertEquals(5, $p->getQuantite());  
}  
  
public function test_ajout_stock()  
{  
    $p = new Produit("Crayon", 500, 10);  
    $p->ajouterStock(5);  
    $this->assertEquals(15, $p->getQuantite());  
}  
}
```

### tests/StockManagerTest.php

```
<?php  
use PHPUnit\Framework\TestCase;  
use App\Database;  
use App\Produit;  
use App\StockManager;  
  
class StockManagerTest extends TestCase  
{  
    protected function setUp(): void  
    {  
        Database::getConnection()->exec("DELETE FROM produits");  
    }  
  
    public function test_total_stock()  
    {  
        $sm = new StockManager();  
        $sm->ajouterProduit(new Produit("Stylo", 1000, 2));  
        $sm->ajouterProduit(new Produit("Crayon", 500, 4));  
  
        $this->assertEquals(1000*2 + 500*4, $sm->totalStock());  
    }  
}
```

## 10. Exécution

Lancer le serveur web :

```
php -S localhost:8000 -t public
```

→ Accède sur : <http://localhost:8000>

Lancer les tests :

```
vendor/bin/phpunit
```

Résultat attendu :

PHPUnit X.X.X by Sebastian Bergmann and contributors.

.. 2 / 2 (100%)

OK (2 tests, 4 assertions)

## Résumé du TP

Étape	Objectif
1	Création du projet et structure
2	Configuration Composer / PHPUnit
3	Création de la base MySQL
4	Connexion PDO
5	Classe Produit (métier)
6	Classe StockManager (gestionnaire)
7	Interface CRUD
8	Tests unitaires
9	Exécution et validation

## 1 Préparer une base de test

1. Crée une base dédiée pour les tests : stock\_db\_test

```
CREATE DATABASE stock_db_test CHARACTER SET utf8mb4 COLLATE  
utf8mb4_general_ci;  
USE stock_db_test;
```

```
CREATE TABLE produits (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    nom VARCHAR(100) NOT NULL,  
    prix DECIMAL(10,2) NOT NULL,  
    quantite INT DEFAULT 0  
);
```

2. Modifie temporairement Database.php pour pouvoir passer le nom de la base, ou crée une **fonction utilitaire** pour la base de test :

```
public static function getTestConnection(): PDO  
{  
    $host = '127.0.0.1';  
    $db = 'stock_db_test';  
    $user = 'root';  
    $pass = '';  
    $charset = 'utf8mb4';  
    $dsn = "mysql:host=$host;dbname=$db;charset=$charset";  
    return new PDO($dsn, $user, $pass, [  
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,  
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,  
    ]);  
}
```

## 2 Exemple de test d'intégration

tests/IntegrationTest.php

```
<?php  
use PHPUnit\Framework\TestCase;  
use App\Produit;  
use App\StockManager;
```

```
use App\Database;

class IntegrationTest extends TestCase
{
    protected function setUp(): void
    {
        $pdo = Database::getTestConnection();
        $pdo->exec("DELETE FROM produits");
    }

    public function test_ajout_et_retrieval_produit()
    {
        $sm = new StockManager();

        // Ajouter un produit
        $produit = new Produit("Gomme", 300, 3);
        $sm->ajouterProduit($produit);

        // Récupérer le produit via StockManager
        $retrieved = $sm->getProduit($produit->getId());

        $this->assertNotNull($retrieved);
        $this->assertEquals("Gomme", $retrieved->getNom());
        $this->assertEquals(3, $retrieved->getQuantite());

        // Vérifier totalStock
        $this->assertEquals(300*3, $sm->totalStock());
    }

    public function test_suppression_produit()
    {
        $sm = new StockManager();
        $p = new Produit("Stylo", 1000, 2);
        $sm->ajouterProduit($p);
```

```
$sm->supprimerProduit($p->getId());  
  
$this->assertNull($sm->getProduit($p->getId()));  
$this->assertEquals(0, $sm->totalStock());  
}  
}  
}
```

Ce test vérifie :

- Ajout d'un produit
- Récupération depuis la base
- Calcul du stock total
- Suppression d'un produit

### 3 Exemple de test d'acceptation simple

- Simule un **workflow complet** comme un utilisateur : ajouter → modifier → supprimer
- tests/AcceptanceTest.php

```
<?php  
use PHPUnit\Framework\TestCase;  
use App\Produit;  
use App\StockManager;  
use App\Database;  
  
class AcceptanceTest extends TestCase  
{  
    protected function setUp(): void  
    {  
        Database::getTestConnection()->exec("DELETE FROM produits");  
    }  
  
    public function test_workflow_complet()  
    {  
        $sm = new StockManager();  
  
        // 1 Ajouter produit
```

```
$p1 = new Produit("Cahier", 1500, 5);
$sm->ajouterProduit($p1);

$this->assertEquals(1500*5, $sm->totalStock());

// 2 Ajouter un autre produit
$p2 = new Produit("Crayon", 500, 10);
$sm->ajouterProduit($p2);

$this->assertEquals(1500*5 + 500*10, $sm->totalStock());

// 3 Modifier stock
$p1->ajouterStock(5);
$p1->save();

$this->assertEquals(1500*10 + 500*10, $sm->totalStock());

// 4 Supprimer un produit
$sm->supprimerProduit($p2->getId());
$this->assertEquals(1500*10, $sm->totalStock());
}
```

Ce test simule un **scénario utilisateur complet**.

---

## 4 Exécution

vendor/bin/phpunit --testsuite "StockManager Test Suite"

•

### 1 Créer la base de test

Ouvre ton terminal MySQL et tape :

```
mysql -u root -p
```

Puis dans MySQL :

```
CREATE DATABASE stock_db_test CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci;
USE stock_db_test;
```

```
CREATE TABLE produits (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    prix DECIMAL(10,2) NOT NULL,
    quantite INT DEFAULT 0
);
EXIT;
```

## 2 Préparer ton projet PHP

Assure-toi que tu es dans le dossier racine de ton projet (stock-manager/) et que composer est installé.

Si ce n'est pas déjà fait, installe PHPUnit :

```
composer require --dev phpunit/phpunit
composer dump-autoload
```

## 3 Créer les fichiers de test

- tests/IntegrationTest.php → contenu que je t'ai fourni
- tests/AcceptanceTest.php → contenu que je t'ai fourni

Vérifie que dans tes tests, tu utilises Database::getTestConnection() pour la base de test.

## 4 Lancer les tests

Dans ton terminal, à la racine du projet :

```
vendor/bin/phpunit
```

ou pour être sûr de lancer la suite spécifique :

```
vendor/bin/phpunit --testsuite "StockManager Test Suite"
```

## 5 Résultat attendu

Tu devrais voir quelque chose comme :

```
PHPUnit X.X.X by Sebastian Bergmann and contributors.
```

```
.....
```

5 / 5 (100%)

```
OK (5 tests, XX assertions)
```