

Ordre de développement MVC avec Repository/Services

Principe : Du plus bas niveau vers le plus haut niveau

Database → Models → Repository Interfaces → Repositories → Services → Controllers → Views

Étapes détaillées

1 Base de données et configuration

```
sql

-- Créer les tables d'abord
CREATE TABLE users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  email VARCHAR(255) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  active BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```
php

// Config/Database.php
class Database {
  public static function getConnection(): PDO {
    return new PDO(
      "mysql:host=localhost;dbname=myapp",
      "username",
      "password"
    );
  }
}
```

2 Models/Entités (Structures de données)

```
php
```

```
// Models/User.php
class User {
    private ?int $id = null;
    private string $email;
    private string $password;
    private bool $active = false;
    private DateTime $createdAt;

    // Constructeur, getters, setters...

    public function getId(): ?int {
        return $this->id;
    }

    public function setPassword(string $password): void {
        $this->password = password_hash($password, PASSWORD_ARGON2ID);
    }

    // ... autres méthodes
}
```

3 Repository Interfaces (Contrats)

```
php
// Repositories/Interfaces/UserRepositoryInterface.php
interface UserRepositoryInterface {
    public function findById(int $id): ?User;
    public function findByEmail(string $email): ?User;
    public function save(User $user): User;
    public function delete(int $id): bool;
}
```

4 Repository Implementations (Accès données)

```
php
```

```
// Repositories/UserRepository.php
class UserRepository implements UserRepositoryInterface {
    private PDO $pdo;

    public function __construct(PDO $pdo) {
        $this->pdo = $pdo;
    }

    public function findById(int $id): ?User {
        $stmt = $this->pdo->prepare("SELECT * FROM users WHERE id = ?");
        $stmt->execute([$id]);
        $data = $stmt->fetch(PDO::FETCH_ASSOC);

        return $data ? $this->hydrate($data) : null;
    }

    private function hydrate(array $data): User {
        $user = new User();
        $user->setId($data['id']);
        $user->setEmail($data['email']);
        // Ne pas setter le password hashé directement
        return $user;
    }
}
```

5 Services (Logique métier)

php

```
// Services/UserService.php
class UserService {
    private UserRepositoryInterface $userRepository;

    public function __construct(UserRepositoryInterface $userRepository) {
        $this->userRepository = $userRepository;
    }

    public function createUser(string $email, string $password): User {
        // Validation métier
        if ($this->userRepository->findByEmail($email)) {
            throw new Exception("Email déjà utilisé");
        }

        // Création
        $user = new User();
        $user->setEmail($email);
        $user->setPassword($password);
        $user->setCreatedAt(new DateTime());

        return $this->userRepository->save($user);
    }
}
```

6 Controllers (Orchestration des requêtes)

php

```
// Controllers/UserController.php
class UserController {
    private UserService $userService;

    public function __construct(UserService $userService) {
        $this->userService = $userService;
    }

    public function register(): void {
        try {
            $email = $_POST['email'] ?? '';
            $password = $_POST['password'] ?? '';

            $user = $this->userService->createUser($email, $password);

            echo json_encode([
                'success' => true,
                'user_id' => $user->getId()
            ]);
        } catch (Exception $e) {
            http_response_code(400);
            echo json_encode([
                'success' => false,
                'message' => $e->getMessage()
            ]);
        }
    }
}
```

7 Container/DI (Assemblage)

```
php
// Config/Container.php
class Container {
    public function getUserController(): UserController {
        $pdo = Database::getConnection();
        $userRepository = new UserRepository($pdo);
        $userService = new UserService($userRepository);

        return new UserController($userService);
    }
}
```

8 Router/Index (Point d'entrée)

php

```
// public/index.php
require_once './vendor/autoload.php';

$container = new Container();

$uri = $_SERVER['REQUEST_URI'];
$method = $_SERVER['REQUEST_METHOD'];

if ($uri === '/users/register' && $method === 'POST') {
    $controller = $container->getUserController();
    $controller->register();
}
```

9 Views (Si nécessaire)

php

```
// Views/user/register.php
<form method="POST" action="/users/register">
    <input type="email" name="email" required>
    <input type="password" name="password" required>
    <button type="submit">S'inscrire</button>
</form>
```

Ordre pour les tests

1. Tests unitaires des Models

php

```
class UserTest extends PHPUnit\Framework\TestCase {
    public function testPasswordHashing() {
        $user = new User();
        $user->setPassword('plaintext');

        $this->assertTrue(password_verify('plaintext', $user->getPassword()));
    }
}
```

2. Tests d'intégration des Repositories

php

```

class UserRepositoryTest extends PHPUnit\Framework\TestCase {
    public function testSaveAndFind() {
        $pdo = $this->getTestDatabase();
        $repo = new UserRepository($pdo);

        $user = new User();
        $user->setEmail('test@example.com');

        $savedUser = $repo->save($user);
        $foundUser = $repo->findById($savedUser->getId());

        $this->assertEquals('test@example.com', $foundUser->getEmail());
    }
}

```

3. Tests unitaires des Services (avec mocks)

```

php

class UserServiceTest extends PHPUnit\Framework\TestCase {
    public function testCreateUser() {
        $mockRepo = $this->createMock(UserRepositoryInterface::class);
        $mockRepo->method('findByEmail')->willReturn(null);

        $service = new UserService($mockRepo);
        $user = $service->createUser('test@example.com', 'password');

        $this->assertEquals('test@example.com', $user->getEmail());
    }
}

```

✅ Avantages de cet ordre

Développement incrémental

- Chaque étape s'appuie sur la précédente
- Possibilité de tester à chaque niveau

Détection d'erreurs précoce

- Les problèmes de conception sont visibles rapidement
- Validation des concepts dès les premières étapes

Feedback rapide

- Tests unitaires possibles très tôt
 - Validation des Repository avant les Services
-

Erreurs à éviter

Commencer par les Controllers

- Risque de mélanger logique métier et présentation
- Difficile de tester sans les couches inférieures

Oublier les interfaces

- Couplage fort entre les couches
- Tests compliqués

Tout développer en même temps

- Bugs difficiles à localiser
 - Architecture fragile
-

Conseil pratique

Développez feature par feature :

1. Une table + son Model
2. Son Repository + Interface
3. Son Service avec la logique métier
4. Son Controller pour l'exposition
5. Tests à chaque étape

Cette approche garantit une architecture solide et testable !