TP Symfony - Création d'une Application de Gestion de Vélos

Objectifs pédagogiques

À la fin de ce TP, vous saurez :

- Créer un projet Symfony complet
- Comprendre l'architecture MVC de Symfony
- Créer et manipuler des entités avec Doctrine
- Utiliser les fixtures pour les données de test
- Développer des contrôleurs et des vues
- Créer des méthodes personnalisées dans les repositories

Prérequis

- PHP 8.2 ou supérieur
- Composer installé
- Symfony CLI installé
- Un serveur de base de données (MySQL/MariaDB recommandé pour ce TP)

Partie 1 : Création du projet et structure

Étape 1 : Création du projet Symfony

```
symfony new samkhaser --webapp
cd samkhaser
```

🔍 **Explication du flag `--webapp`:** Le flag `--webapp` installe automatiquement les bundles essentiels pour une application web complète :

- `TwigBundle` : pour les templates
- `DoctrineBundle` : pour la base de données
- `SecurityBundle` : pour l'authentification
- `WebProfilerBundle` : pour le débogage
- Et bien d'autres...

Sans ce flag, vous auriez une installation minimale et devriez installer ces bundles manuellement.

Étape 2 : Exploration de la structure du projet

Votre projet Symfony a maintenant cette structure :

```
samkhaser/
├── bin/                    # Exécutables (console)
├── config/                 # Configuration de l'application
│   ├── packages/          # Configuration des bundles
│   └── routes/           # Configuration des routes
├── public/                 # Point d'entrée web (index.php)
├── src/                    # Code source de votre application
│   ├── Controller/       # Contrôleurs MVC
│   ├── Entity/          # Entités Doctrine (modèles)
│   └── Repository/      # Classes pour les requêtes BDD
├── templates/              # Templates Twig (vues)
├── var/                    # Cache et logs
└── vendor/                # Dépendances Composer
```

Points clés :

- `src/` : Tout votre code PHP personnalisé
- `templates/` : Vos fichiers Twig pour l'affichage
- `public/` : Le seul dossier accessible depuis le web
- `config/` : Toute la configuration de Symfony

Partie 2 : Configuration de la base de données

Étape 3 : Configuration de la connexion

Modifiez le fichier `.env` pour configurer votre base de données :

```
# Pour MySQL/MariaDB
DATABASE_URL="mysql://utilisateur:motdepasse@127.0.0.1:3306/samkhaser?
serverVersion=8.0.32&charset=utf8mb4"
```

Créez la base de données :

```
php bin/console doctrine:database:create
```

Comprendre la commande :

- `php bin/console` : Point d'entrée pour toutes les commandes Symfony
- `doctrine:database:create` : Commande qui crée la base de données définie dans DATABASE_URL

Partie 3 : Création de l'entité Vélo

Étape 4 : Génération de l'entité avec Maker Bundle

```
php bin/console make:entity Velo
```

Le Maker Bundle va vous poser une série de questions. Répondez ainsi :

```
New property name: type
Field type: string
Field length: 50
Can this field be null: no

New property name: taille
Field type: string
Field length: 10
Can this field be null: no

New property name: genre
Field type: string
Field length: 20
Can this field be null: no

New property name: marque
Field type: string
Field length: 100
Can this field be null: no

New property name: modele
Field type: string
Field length: 100
Can this field be null: no

New property name: prix
Field type: decimal
Precision: 10
Scale: 2
Can this field be null: no

New property name: stock
Field type: integer
Can this field be null: no

New property name: couleur
Field type: string
Field length: 50
Can this field be null: yes

New property name: description
Field type: text
Can this field be null: yes

New property name: imageUrl
Field type: string
Field length: 255
```

```
Can this field be null: yes

New property name: estEnPromotion
Field type: boolean
Can this field be null: no

New property name: prixPromotion
Field type: decimal
Precision: 10
Scale: 2
Can this field be null: yes

New property name: dateAjout
Field type: datetime
Can this field be null: no

# Appuyez sur Entrée pour terminer
```

Étape 5 : Analyse du code généré

Ouvrez le fichier `src/Entity/Velo.php` généré :

🔍 Analyse détaillée du code :

```php
<?php

namespace App\Entity;  // ← Namespace : organise le code, ici dans App\Entity

use App\Repository\VeloRepository;  // ← Import de la classe Repository
use Doctrine\DBAL\Types\Types;      // ← Import des types Doctrine
use Doctrine\ORM\Mapping as ORM;    // ← Import des annotations ORM

#[ORM\Entity(repositoryClass: VeloRepository::class)]  // ← Attribut PHP 8 : lie
l'entité à son repository
class Velo
{
    #[ORM\Id]                    // ← Clé primaire
    #[ORM\GeneratedValue]        // ← Auto-incrémentation
    #[ORM\Column]                // ← Colonne en base
    private ?int $id = null;     // ← Propriété privée, nullable

    #[ORM\Column(length: 50)]    // ← Colonne VARCHAR(50)
    private ?string $type = null;

    // ... autres propriétés

    // Getters et setters générés automatiquement
    public function getId(): ?int
    {
        return $this->id;
    }
```

```
    public function getType(): ?string
    {
        return $this->type;
    }

    public function setType(string $type): static  // ← Retourne $this pour le
chaînage
    {
        $this->type = $type;
        return $this;
    }
}
```

Concepts importants :

1. **Namespace** : `App\Entity` organise votre code et évite les conflits
2. **Attributs PHP** : `#[ORM\...]` remplacent les annotations depuis PHP 8
3. **Encapsulation** : Propriétés privées + getters/setters publics
4. **Nullable types** : `?int`, `?string` permettent les valeurs nulles
5. **Fluent interface** : `return $this` permet le chaînage des méthodes

Étape 6 : Création de la migration

```
php bin/console make:migration
```

Cette commande génère un fichier dans `migrations/` qui contient les instructions SQL pour créer votre table.

Comprendre les migrations :

- Les migrations sont des scripts qui modifient la structure de la BDD
- Elles permettent de versionner l'évolution de votre schéma
- Chaque migration a un timestamp pour l'ordre d'exécution

Appliquez la migration :

```
php bin/console doctrine:migrations:migrate
```

Partie 4 : Les Fixtures - Données de test

Étape 7 : Installation du bundle Fixtures

```
composer require --dev doctrine/doctrine-fixtures-bundle
```

Étape 8 : Création des fixtures

```
php bin/console make:fixtures AppFixtures
```

Modifiez le fichier `src/DataFixtures/AppFixtures.php` :

```
<?php

namespace App\DataFixtures;

use App\Entity\Velo;                    // ← Import de notre entité
use Doctrine\Bundle\FixturesBundle\Fixture;  // ← Classe de base pour les fixtures
use Doctrine\Persistence\ObjectManager;      // ← Pour persister les données c'est
a dire les sauvegarder en BDD

class AppFixtures extends Fixture
{
    public function load(ObjectManager $manager): void  // ← Méthode principale
des fixtures
    {
        // Données d'exemple sous forme de tableau
        $velos = [
            [
                'type' => 'VTT',
                'taille' => 'XL',
                'genre' => 'Homme',
                'marque' => 'Trek',
                'modele' => 'X-Caliber 8',
                'prix' => '899.99',
                'stock' => 5,
                'couleur' => 'Noir et rouge',
                'description' => 'Excellent VTT pour débuter en montagne',
                'imageUrl' => 'https://example.com/vtt1.jpg',
                'estEnPromotion' => false,
                'prixPromotion' => null
            ],
            [
                'type' => 'Vélo de route',
                'taille' => 'M',
                'genre' => 'Femme',
                'marque' => 'Giant',
                'modele' => 'Contend 3',
                'prix' => '649.99',
                'stock' => 8,
                'couleur' => 'Blanc et bleu',
                'description' => 'Vélo de route léger et performant',
                'imageUrl' => 'https://example.com/route1.jpg',
                'estEnPromotion' => true,
                'prixPromotion' => '549.99'
            ],
```

```
            // Ajoutez d'autres vélos...
        ];

        // Boucle pour créer chaque vélo
        foreach ($velos as $veloData) {
            $velo = new Velo();  // ← Création d'une nouvelle instance

            // Utilisation des setters pour définir les propriétés
            $velo->setType($veloData['type']);
            $velo->setTaille($veloData['taille']);
            $velo->setGenre($veloData['genre']);
            $velo->setMarque($veloData['marque']);
            $velo->setModele($veloData['modele']);
            $velo->setPrix($veloData['prix']);
            $velo->setStock($veloData['stock']);
            $velo->setCouleur($veloData['couleur']);
            $velo->setDescription($veloData['description']);
            $velo->setImageUrl($veloData['imageUrl']);
            $velo->setEstEnPromotion($veloData['estEnPromotion']);
            $velo->setPrixPromotion($veloData['prixPromotion']);
            $velo->setDateAjout(new \DateTime());  // ← Date actuelle

            $manager->persist($velo);  // ← Prépare l'objet pour la sauvegarde
        }

        $manager->flush();  // ← Exécute toutes les insertions en BDD
    }
}
```

🔍 **Concepts des fixtures :**

1. **ObjectManager** : Interface pour persister les objets en BDD
2. **persist()** : Met l'objet en attente de sauvegarde
3. **flush()** : Exécute réellement les requêtes SQL
4. **new \DateTime()** : Crée un objet DateTime avec la date/heure actuelle

Étape 9 : Chargement des fixtures

```
php bin/console doctrine:fixtures:load
```

Cette commande vide la base et y insère vos données de test.

Partie 5 : Le Repository - Requêtes personnalisées

Étape 10 : Comprendre le Repository généré

Ouvrez `src/Repository/VeloRepository.php` :

```php
<?php

namespace App\Repository;

use App\Entity\Velo;  // ← Import de l'entité associée
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @extends ServiceEntityRepository<Velo>  ← Indique que ce repository gère les entités Velo
 */
class VeloRepository extends ServiceEntityRepository
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, Velo::class);  // ← Lie le repository à l'entité Velo
    }

    // Méthodes héritées disponibles :
    // find($id)           - Trouve par ID
    // findAll()           - Trouve tous les enregistrements
    // findBy($criteria)   - Trouve selon des critères
    // findOneBy($criteria)- Trouve un seul selon des critères
}
```

Étape 11 : Ajout de méthodes personnalisées

Ajoutez ces méthodes dans votre repository :

```php
/**
 * Récupère tous les vélos
 * @return Velo[] Returns an array of Velo objects
 */
public function findAllVelo(): array
{
    return $this->findAll();  // ← Utilise la méthode héritée
}

/**
 * Récupère le premier vélo (pour "Mon vélo")
 */
public function findFirstVelo(): ?Velo
{
    return $this->findOneBy([], ['id' => 'ASC']);  // ← Tri par ID croissant
}

/**
 * Récupère les vélos en promotion
 * @return Velo[] Returns an array of Velo objects
```

```
     */
    public function findVelosEnPromotion(): array
    {
        return $this->createQueryBuilder('v')  // ← 'v' est l'alias pour Velo
            ->andWhere('v.estEnPromotion = :promotion')  // ← Condition WHERE
            ->setParameter('promotion', true)             // ← Paramètre sécurisé
            ->orderBy('v.dateAjout', 'DESC')             // ← Tri par date décroissant
            ->getQuery()                                  // ← Construit la requête
            ->getResult();                               // ← Exécute et retourne les
    résultats
    }
```

🔍 **Analyse du QueryBuilder :**

1. **createQueryBuilder('v')** : Crée un constructeur de requête avec l'alias 'v'
2. **andWhere()** : Ajoute une condition WHERE
3. **setParameter()** : Définit un paramètre sécurisé (évite les injections SQL)
4. **orderBy()** : Définit le tri
5. **getQuery()->getResult()** : Exécute et retourne un tableau d'objets

---

Partie 6 : Le Contrôleur - Logique métier

Étape 12 : Création du contrôleur

```
php bin/console make:controller VeloController
```

Étape 13 : Développement du contrôleur

Modifiez `src/Controller/VeloController.php` :

```
<?php

namespace App\Controller;

use App\Repository\VeloRepository;  // ← Import du repository
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

final class VeloController extends AbstractController
{
    /**
     * Page d'accueil avec tous les vélos
     */
    #[Route('/velo', name: 'app_velo')]  // ← Route : URL '/velo' → nom 'app_velo'
    public function index(VeloRepository $veloRepository): Response  // ←
    Injection de dépendance
```

```
    {
        // Récupération des données via le repository
        $velos = $veloRepository->findAllVelo();

        // Rendu du template avec transmission des données
        return $this->render('velo/index.html.twig', [
            'controller_name' => 'VeloController',
            'velos' => $velos  // ← Variable accessible dans Twig
        ]);
    }

    /**
     * Page "Mon vélo" - affiche le premier vélo
     */
    #[Route('/mybike', name: 'app_mybike')]
    public function showMyBike(VeloRepository $veloRepository): Response
    {
        $myBike = $veloRepository->findOneBy([], ['id' => 'ASC']);

        return $this->render('velo/mybike.html.twig', [
            'myBike' => $myBike,
        ]);
    }

    /**
     * Page détail d'un vélo
     */
    #[Route('/velo/{id}', name: 'app_velo_show')]  // ← {id} est un paramètre
d'URL
    public function show(int $id, VeloRepository $veloRepository): Response
    {
        $velo = $veloRepository->find($id);  // ← Recherche par ID

        if (!$velo) {
            throw $this->createNotFoundException('Vélo non trouvé');  // ← Erreur
404
        }

        return $this->render('velo/show.html.twig', [
            'velo' => $velo,
        ]);
    }
}
```

Concepts du contrôleur :

1. **#[Route()]** : Attribut qui définit l'URL et le nom de la route
2. **Injection de dépendance** : Symfony injecte automatiquement le VeloRepository
3. **render()** : Méthode qui génère la réponse HTML via Twig
4. **Paramètres d'URL** : `{id}` dans la route devient `$id` en paramètre
5. **createNotFoundException()** : Génère une erreur 404

Partie 7 : Les Templates Twig - Interface utilisateur

Étape 14 : Template de liste (index)

Créez `templates/velo/index.html.twig` :

```
{% extends 'base.html.twig' %}  {# ← Héritage du template de base #}

{% block title %}Nos Vélos{% endblock %}  {# ← Définition du titre #}

{% block body %}
<div class="container mt-4">
    <h1>Catalogue des vélos</h1>

    <div class="row">
        {% for velo in velos %}  {# ← Boucle sur la variable velos du contrôleur
#}
            <div class="col-md-4 mb-4">
                <div class="card">
                    {% if velo.imageUrl %}  {# ← Condition Twig #}
                        <img src="{{ velo.imageUrl }}" class="card-img-top" alt="
{{ velo.marque }} {{ velo.modele }}">
                    {% endif %}

                    <div class="card-body">
                        <h5 class="card-title">{{ velo.marque }} {{ velo.modele }}
</h5>  {# ← Affichage des propriétés #}
                        <p class="card-text">{{ velo.description }}</p>

                        <div class="mb-2">
                            <span class="badge bg-primary">{{ velo.type }}</span>
                            <span class="badge bg-secondary">{{ velo.genre }}
</span>
                        </div>

                        {% if velo.estEnPromotion %}  {# ← Condition pour la
promotion #}
                            <p class="card-text">
                                <span class="text-decoration-line-through">{{
velo.prix }}€</span>
                                <span class="text-danger fw-bold">{{
velo.prixPromotion }}€</span>
                                <span class="badge bg-danger">PROMO</span>
                            </p>
                        {% else %}
                            <p class="card-text">
                                <span class="text-success fw-bold">{{ velo.prix }}
€</span>
                            </p>
                        {% endif %}

                        <a href="{{ path('app_velo_show', {id: velo.id}) }}"
```

```
class="btn btn-primary">
                            Voir détails  {# ← Génération d'URL avec paramètres #}
                        </a>
                    </div>
                </div>
            </div>
        {% else %}  {# ← Clause else de la boucle for #}
            <div class="col-12">
                <p>Aucun vélo trouvé.</p>
            </div>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

Étape 15 : Template de détail

Créez `templates/velo/show.html.twig` :

```
{% extends 'base.html.twig' %}

{% block title %}{{ velo.marque }} {{ velo.modele }}{% endblock %}

{% block body %}
<div class="container mt-4">
    <nav aria-label="breadcrumb">
        <ol class="breadcrumb">
            <li class="breadcrumb-item">
                <a href="{{ path('app_velo') }}">Vélos</a>  {# ← Lien vers la
liste #}
            </li>
            <li class="breadcrumb-item active">{{ velo.marque }} {{ velo.modele }}
</li>
        </ol>
    </nav>

    <div class="row">
        <div class="col-md-6">
            {% if velo.imageUrl %}
                <img src="{{ velo.imageUrl }}" class="img-fluid" alt="{{
velo.marque }} {{ velo.modele }}">
            {% endif %}
        </div>

        <div class="col-md-6">
            <h1>{{ velo.marque }} {{ velo.modele }}</h1>

            <ul class="list-unstyled">
                <li><strong>Type:</strong> {{ velo.type }}</li>
                <li><strong>Taille:</strong> {{ velo.taille }}</li>
                <li><strong>Genre:</strong> {{ velo.genre }}</li>
```

```
                <li><strong>Couleur:</strong> {{ velo.couleur }}</li>
                <li><strong>Stock:</strong> {{ velo.stock }} unités</li>
            </ul>

            {% if velo.description %}
                <p class="lead">{{ velo.description }}</p>
            {% endif %}

            <p class="text-muted">
                Ajouté le {{ velo.dateAjout|date('d/m/Y') }}  {# ← Filtre Twig
pour formater la date #}
            </p>
        </div>
    </div>
</div>
{% endblock %}
```

Étape 16 : Template "Mon vélo"

Créez `templates/velo/mybike.html.twig` :

```
{% extends 'base.html.twig' %}

{% block title %}Mon Vélo{% endblock %}

{% block body %}
<div class="container mt-4">
    {% if myBike %}  {# ← Vérification que la variable existe #}
        <h1>Mon Vélo : {{ myBike.marque }} {{ myBike.modele }}</h1>

        <div class="row">
            <div class="col-md-6">
                {% if myBike.imageUrl %}
                    <img src="{{ myBike.imageUrl }}" class="img-fluid" alt="{{
myBike.marque }} {{ myBike.modele }}">
                {% endif %}
            </div>
            <div class="col-md-6">
                {# Même structure que show.html.twig mais adapté pour "mon vélo"
#}
                <ul class="list-unstyled">
                    <li><strong>Marque:</strong> {{ myBike.marque }}</li>
                    <li><strong>Modèle:</strong> {{ myBike.modele }}</li>
                    <li><strong>Type:</strong> {{ myBike.type }}</li>
                    <li><strong>Taille:</strong> {{ myBike.taille }}</li>
                </ul>

                <a href="{{ path('app_velo') }}" class="btn btn-secondary">
                    Voir tous les vélos
                </a>
            </div>
```

```
        </div>
    {% else %}
        <div class="alert alert-warning">
            <h4>Aucun vélo trouvé</h4>
            <p>Exécutez les fixtures : <code>php bin/console
doctrine:fixtures:load</code></p>
        </div>
    {% endif %}
</div>
{% endblock %}
```

🔍 **Concepts Twig :**

1. **extends** : Héritage de template
2. **block** : Zones redéfinissables dans les templates enfants
3. **{{ }}** : Affichage de variables
4. **{% %}** : Instructions de contrôle (for, if, etc.)
5. **path()** : Génération d'URLs à partir du nom de route
6. **Filtres** : `|date()` transforme l'affichage des données

Partie 8 : Navigation et menu

Étape 17 : Amélioration du template de base

Modifiez `templates/base.html.twig` pour ajouter un menu :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}SamKhaser - Boutique Vélo{% endblock %}</title>

        <!-- Bootstrap CSS -->
        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">

        {% block stylesheets %}{% endblock %}
    </head>
    <body>
        <!-- Navigation -->
        <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
            <div class="container">
                <a class="navbar-brand" href="{{ path('app_velo') }}">🚴
SamKhaser</a>

                <div class="navbar-nav">
                    <a class="nav-link" href="{{ path('app_velo') }}">Tous les
vélos</a>
                    <a class="nav-link" href="{{ path('app_mybike') }}">Mon
```

```
vélo</a>
                </div>
            </div>
        </nav>

        <!-- Contenu principal -->
        {% block body %}{% endblock %}

        <!-- Bootstrap JS -->
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
></script>
        {% block javascripts %}{% endblock %}
    </body>
</html>
```

Partie 9 : Test de l'application

Étape 18 : Lancement du serveur

```
symfony server:start
```

Testez vos pages :

- `http://localhost:8000/velo` : Liste des vélos
- `http://localhost:8000/mybike` : Mon vélo
- `http://localhost:8000/velo/1` : Détail du vélo ID 1

Partie 10 : Analyse du projet réalisé

Architecture MVC mise en place

Modèle (Model) :

- `Entity/Velo.php` : Représentation des données
- `Repository/VeloRepository.php` : Accès aux données

Vue (View) :

- `templates/velo/*.html.twig` : Templates pour l'affichage

Contrôleur (Controller) :

- `Controller/VeloController.php` : Logique métier et coordination

Flux de données

1. **Requête HTTP** → Route → Contrôleur

2. **Contrôleur** → Repository → Base de données
3. **Données** → Template → Réponse HTML

Avantages de cette architecture

- **Séparation des responsabilités**
- **Code réutilisable**
- **Facilité de maintenance**
- **Tests possibles** sur chaque couche

Exercices complémentaires

Exercice 1 : Nouvelles méthodes Repository

Ajoutez ces méthodes dans `VeloRepository` :

1. **findByType($type)** : Trouve les vélos par type

```
public function findByType(string $type): array
{
    return $this->findBy(['type' => $type]);
}
```

2. **findExpensiveVelos($price)** : Vélos au-dessus d'un prix

```
public function findExpensiveVelos(float $price): array
{
    return $this->createQueryBuilder('v')
        ->where('v.prix > :price')
        ->setParameter('price', $price)
        ->orderBy('v.prix', 'DESC')
        ->getQuery()
        ->getResult();
}
```

Exercice 2 : Nouvelles routes

Ajoutez dans le contrôleur :

1. **Route pour les promotions** :

```
#[Route('/promotions', name: 'app_promotions')]
public function promotions(VeloRepository $veloRepository): Response
{
    $velos = $veloRepository->findVelosEnPromotion();
```

```
        return $this->render('velo/promotions.html.twig', [
            'velos' => $velos
        ]);
    }
```

2. **Route par type** :

```
#[Route('/velo/type/{type}', name: 'app_velo_by_type')]
public function byType(string $type, VeloRepository $veloRepository): Response
{
    $velos = $veloRepository->findByType($type);

    return $this->render('velo/by_type.html.twig', [
        'velos' => $velos,
        'type' => $type
    ]);
}
```

Exercice 3 : Nouvelles propriétés

Ajoutez un champ `poids` à l'entité :

1. **Ajout de la propriété** :

```
php bin/console make:entity Velo
# Ajoutez : poids, decimal, precision 5, scale 2, nullable
```

2. **Migration** :

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

3. **Mise à jour des fixtures** avec le nouveau champ

Ressources et commandes utiles

Commandes Doctrine importantes

```
# Base de données
php bin/console doctrine:database:create
php bin/console doctrine:database:drop --force

# Migrations
php bin/console make:migration
```

```
php bin/console doctrine:migrations:migrate
php bin/console doctrine:migrations:status

# Fixtures
php bin/console doctrine:fixtures:load

# Schema
php bin/console doctrine:schema:validate
```

Commandes Maker Bundle

```
php bin/console make:entity        # Création d'entité
php bin/console make:controller    # Création de contrôleur
php bin/console make:form          # Création de formulaire
php bin/console make:fixtures      # Création de fixtures
```

Debug et informations

```
php bin/console debug:router       # Liste des routes
php bin/console debug:container    # Services disponibles
php bin/console doctrine:mapping:info  # Infos entités
```

Points clés à retenir

1. **Symfony suit le pattern MVC** : séparation claire des responsabilités
2. **Doctrine ORM** : mapping objet-relationnel automatique
3. **Maker Bundle** : génération automatique de code
4. **Twig** : moteur de template puissant et sécurisé
5. **Injection de dépendances** : Symfony fournit automatiquement les services
6. **Routes** : liaison URL ↔ méthodes de contrôleur
7. **Repository** : couche d'accès aux données avec requêtes personnalisées

---

Conclusion

Vous avez maintenant créé une application Symfony complète avec :

- Gestion d'entités et base de données
- Contrôleurs avec logique métier
- Templates pour l'affichage
- Navigation entre les pages
- Données de test via les fixtures

Cette base vous permet de développer des applications web robustes et maintenables avec Symfony !