

# Symfony

## Base de Données et Templating

**Maxime Puy**

29 avril 2024

## 1 Bases de données avec Doctrine

# Base de Données avec Doctrine

# Introduction à Doctrine

- ❑ ORM recommandé des les SGDBR dans Symfony.
- ❑ Accès de bas niveau pour exécuter des requêtes SQL brutes (similaire au PDO de PHP).
- ❑ Compatible avec MySQL, PostgreSQL, SQLite, MongoDB, etc

# Installation de Doctrine

Installation de Doctrine via `composer` :

```
1 composer require symfony/orm-pack  
2 composer require --dev symfony/maker-bundle
```

- `orm-pack` : Installation de Doctrine
- `maker-bundle` : Permet la génération de code

# Configuration de la Base de Données

- La connexion à la base de données est définie dans une variable d'environnement `DATABASE_URL` .
- Pour le développement, mettre dans `.env` :

```
1 // .env :  
2 # Exemple avec MySQL  
3 DATABASE_URL="mysql://user:passwd@127.0.0.1:3306/nom_base?serverVersion=5.7"  
4 # Exemple avec SQLite  
5 DATABASE_URL="sqlite:///kernel.project_dir%/var/app.db"
```

# Création d'une Entité

- Création d'une classe d'entité pour représenter les objets dans la BDD, par exemple, `Produit` :

```
1 $ symfony console make:entity
2
3 Nom de la classe :
4 > Produit
5
6 Nom de la propriété :
7 > nom
8
9 Type de champ :
10 > string
11
12 Longueur du champ :
13 > 255
```

# Classe Entité Générée

- Voici à quoi ressemble la classe `Produit` générée :

```
1 // src/Entity/Produit.php
2 namespace App\Entity;
3
4 use Doctrine\ORM\Mapping as ORM;
5
6 #[ORM\Entity]
7 class Produit
8 {
9     #[ORM\Id]
10    #[ORM\GeneratedValue]
11    #[ORM\Column(type: 'integer')]
12    private $id;
13
14    #[ORM\Column(type: 'string', length: 255)]
15    private $nom;
16    // ...
17 }
```

# Migrations : Créer les Tables de Base de Données

Utilisez le `DoctrineMigrationsBundle` pour créer les tables :

- 1 `symfony console make:migration`
- 2 `symfony console doctrine:migrations:migrate`

```

25 public function getDescription(): string
26 {
27     return '';
28 }
29
30 public function up(Schema $schema): void
31 {
32     // this up() migration is auto-generated, please modify it to your needs
33     $this->addSql('CREATE SEQUENCE vinyl_mix_id_seq INCREMENT BY 1 MINVALUE
34     $this->addSql('CREATE TABLE vinyl_mix (id INT NOT NULL, title VARCHAR(255) NOT NULL, created_at DATETIME NOT NULL)');
35     $this->addSql('COMMENT ON COLUMN vinyl_mix.created_at IS \'(DC2Type:datetime_immutable)\'');
36 }
37
38 public function down(Schema $schema): void
39 {
40     // this down() migration is auto-generated, please modify it to your needs
  
```

# Persiste des Objets dans la Base de Données

- Dans le code, créez un nouvel objet, définissez ses données et sauvegardez-le :

```
1 // src/Controller/ProduitController.php
2 // ...
3 $produit = new Produit();
4 $produit->setNom('Clavier');
5 // ...
6 $entityManager->persist($produit);
7 $entityManager->flush();
```

- Rappel : Injection de dépendances pour accéder à `$entityManager`
- Penser à appeler `flush` pour exécuter les requêtes

# Validation des Objets

Symfony réutilise les métadonnées Doctrine pour la validation :

```
1 // src/Controller/ProduitController.php
2 public function createProduct(ValidatorInterface $validator): Response {
3     $product = new Product();
4     // ...
5     $erreurs = $validator->validate($produit);
6     if (count($erreurs) > 0) {
7         // Gérer les erreurs
8         return new Response((string) $errors, 400);
9     }
10 }
```

# Récupération d'Objets de la Base de Données

- A la création de l'entité, création automatique d'un 'Repository'

```
1 // src/Repository/ProductRepository.php
2 namespace App\Repository;
3
4 use App\Entity\Product;
5 use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
6 use Doctrine\Persistence\ManagerRegistry;
7
8 class ProductRepository extends ServiceEntityRepository
9 {
10     public function __construct(ManagerRegistry $registry)
11     {
12         parent::__construct($registry, Product::class);
13     }
14 }
```

# Méthodes du Repository

Le repository offre plusieurs méthodes utiles :

- `find($id)` pour chercher par clé primaire.
- `findOneBy(['name' => 'Nom'])` pour chercher par critères.
- `findBy(['name' => 'Nom'], ['price' => 'ASC'])` pour plusieurs objets.
- `findAll()` pour tous les objets.

# Méthodes Personnalisées

Ajoutez des méthodes personnalisées pour des requêtes complexes dans votre `ProductRepository` .

```
1 // src/Repository/ProductRepository.php
2 class ProductRepository extends ServiceEntityRepository {
3     // ...
4     public function findAllGreaterThanPrice(int $price): array {
5         $entityManager = $this->getEntityManager();
6         $query = $entityManager->createQuery(
7             'SELECT p
8             FROM App\Entity\Product p
9             WHERE p.price > :price
10            ORDER BY p.price ASC'
11            // ATTENTION, ceci n'est pas du pure SQL !!
12        )->setParameter('price', $price);
13
14        // returns an array of Product objects
15        return $query->getResult();
16    }
17 }
```

# Utilisation du Repository dans le Controller

- Récupération manuelle du `Repository` via le `ManagerRegistry` :

```
1 // src/Controller/ProductController.php
2 public function show(ManagerRegistry $doctrine, int $id): Response {
3     // Récupération du repository de produit
4     $productRepo = $doctrine->getRepository(Product::class);
5
6     // Récupération du produit via le repository
7     $product = $productRepo->find($id);
8
9     // Utilisation des données
10    if (!$product) {
11        throw $this->createNotFoundException(
12            'No product found for id '.$id
13        );
14    }
15
16    return new Response('Check out this great product: '.$product->getName());
17 }
```

# Utilisation du ProductRepository

- Sinon injection automatique de `ProductRepository` via l'autowiring :

```
1 // src/Controller/ProductController.php
2 class ProductController extends AbstractController
3 {
4     public function show(ProductRepository $productRepository, int $id): Response
5     {
6         $product = $productRepository->find($id);
7         // ...
8     }
9 }
```

- Mais ne permet pas le `flush` !

# Résolution Automatique d'Entités

- Possible de rechercher automatiquement les paramètres
- Via le bundle SensioFrameworkExtraBundle
- Intégré dans Symfony à partir de la version 6.2 (ne plus utiliser le bundle dans les versions suivantes !)

```
1 // src/Controller/ProductController.php
2 /**
3  * @Route("/product/{id}", name="product_show")
4  */
5 public function show(Product $product): Response
6 {
7     // Utilisez le produit directement !
8     // ...
9 }
```

- Requête automatique via les paramètres (ici `id`), 404 si pas trouvé
- Documentation sur la conversion des paramètres

# Mise à Jour d'un Objet

```
1 // src/Controller/ProductController.php
2 public function update(ManagerRegistry $doctrine, int $id): Response {
3     $entityManager = $doctrine->getManager();
4     $product = $entityManager->getRepository(Product::class)->find($id);
5
6     if (!$product) {
7         // ...
8     }
9
10    $product->setName('New product name!');
11    $entityManager->flush();
12
13    // ...
14 }
```

# Suppression d'un Objet

Supprimer un objet en utilisant la méthode `remove()` :

```
1 $entityManager->remove($product);  
2 $entityManager->flush();
```

# Utilisation Directe de SQL

- Très similaire à PDO :

```
1 $conn = $entityManager->getConnection();
2 $sql = '
3     SELECT * FROM product p
4     WHERE p.price > :price
5     ORDER BY p.price ASC
6     ';
```

```
7 $stmt = $conn->prepare($sql);
8 $resultSet = $stmt->executeQuery(['price' => $price]);

9 // returns an array of arrays (i.e. a raw data set)
0 return $resultSet->fetchAllAssociative();
```

- Attention ! On ne récupère pas des objets

## Templating avec Twig

# Introduction au Langage de Templating Twig

- Twig : moteur de template pour PHP
- Va utiliser des variables PHP pour générer du contenu statique
- Peut-être utilisé dans des codes HTML, CSS, JS, etc
- Twig utilise trois constructions principales :
  - ▷ `{{ ... }}` pour afficher le contenu d'une variable ou le résultat d'une expression.
  - ▷ `{% ... %}` pour exécuter de la logique, comme une condition ou une boucle.
  - ▷ `{# ... #}` pour ajouter des commentaires (non inclus dans la page rendue).

# Exemple de Template Twig

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Welcome to Symfony!</title>
5   </head>
6   <body>
7     <h1>{{ page_title }}</h1>
8
9     {% if user.isLoggedIn %}
10      Hello {{ user.name }}!
11    {% endif %}
12  </body>
13 </html>
```

# Création de Templates

Créez un fichier dans le répertoire `templates/` :

```

1  {{{ templates/user/notifications.html.twig #}}
2  <h1>Hello {{ user_first_name }}!</h1>
3  <p>You have {{ notifications|length }} new notifications.</p>

```

Puis, créez un contrôleur pour rendre ce template :

```

1  // src/Controller/UserController.php
2  class UserController extends AbstractController
3  {
4      public function notifications(): Response
5      {
6          $userFirstName = '...';
7          $userNotifications = ['...', '...'];
8          return $this->render('user/notifications.html.twig', [
9              'user_first_name' => $userFirstName,
10             'notifications' => $userNotifications,
11         ]);
12     }
13 }

```

# Emplacement des Templates

- Les templates sont stockés par défaut dans le répertoire

```
<votre_projet>/templates/
```

- `render("product/index.html.twig")` dans le code  $\Rightarrow$

```
<votre_projet>/templates/products/index.html.twig
```

 sur le disque

- `/!\` Penser à l'extension `.twig` `/!\`

- Si `{{ foo.bar }}` dans le template, Twig va tenter de résoudre le symbole comme :

1 `$foo['bar']`

2 `$foo->bar`

3 `$foo->bar()`

4 `$foo->getBar()`

5 etc.

6 `null` or erreur suivant la configuration

## Lien vers les Pages / CS, JS, etc

- Utilisez la fonction `path()` pour générer des URL basées sur la configuration du routage.

```

1 /**
2  * @Route("/article/{slug}", name="blog_post")
3  */
4 public function show(string $slug): Response {}

```

```

1 <a href="{ path('blog_post', {slug: post.slug}) }" >{ post.title }</a>

```

- Utilisez la fonction `asset()` pour générer l'URL des assets statiques.

```

1 
2 

```

- `https://example.com`  $\Rightarrow$  `/images/logo.png`
- `https://example.com/my_app`  $\Rightarrow$  `/my_app/images/logo.png`

# Variable Globale App

- Symfony injecte automatiquement une variable `app` dans chaque template Twig, offrant un accès à des informations de l'application :

```
1 <p>Username: {{ app.user.username ?? 'Utilisateur anonyme' }}</p>
2 {% if app.debug %}
3     <p>Méthode de la requête : {{ app.request.method }}</p>
4     <p>Environnement de l'application : {{ app.environment }}</p>
5 {% endif %}
```

# Rendu de Templates dans les Services

- Injectez Twig dans vos propres services pour utiliser sa méthode `render()` :

```
1 // src/Service/SomeService.php
2 class SomeService
3 {
4     private $twig;
5
6     public function __construct(Environment $twig)
7     {
8         $this->twig = $twig;
9     }
10
11    public function someMethod()
12    {
13        $htmlContents = $this->twig->render('product/index.html.twig', [
14            // ...
15        ]);
16    }
17 }
```

- `renderView()` renvoie uniquement le contenu créé par le template.

# Rendu Direct de Templates depuis une Route

- Rendez des pages statiques directement depuis la définition de la route avec `TemplateController` :

```
1 # config/routes.yaml
2 acme_privacy:
3     path:          /privacy
4     controller:    Symfony\Bundle\FrameworkBundle\Controller\TemplateController
5     defaults:
6         # the path of the template to render
7         template:  'static/privacy.html.twig'
```

# Définition de Variables

- Vous pouvez attribuer des valeurs aux variables avec le tag `set` :

```
1 {% set foo = 'foo' %}  
2 {% set foo = [1, 2] %}  
3 {% set foo = {'foo': 'bar'} %}
```

# Filtres

- Les variables peuvent être modifiées par des filtres, séparés par un symbole pipe (|). Les filtres peuvent être chaînés :

```
1 {{ nom|striptags|title }}
```

- Pour appliquer un filtre à une section, utilisez le tag `apply` :

```
1 {% apply upper %}  
2     Ce texte devient en majuscules  
3 {% endapply %}
```

# Fonctions

- Les fonctions génèrent du contenu. Elles sont appelées par leur nom suivi de parenthèses :

```
1 {% for i in range(0, 3) %}  
2     {{ i }},  
3 {% endfor %}
```

- Les arguments nominaux sont supportés dans les fonctions et filtres :

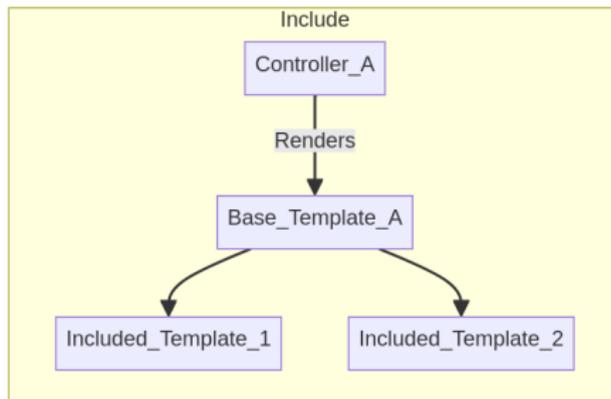
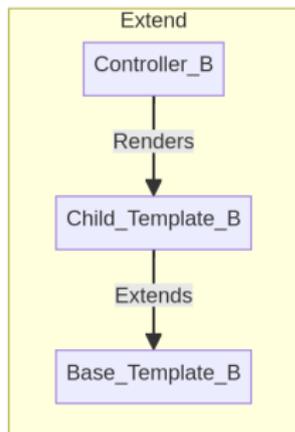
```
1 {% for i in range(low=1, high=10, step=2) %}  
2     {{ i }},  
3 {% endfor %}
```

# Structures de Contrôle

- Les structures de contrôle dirigent le flux du programme - conditionnels ( `if/elseif/else` ), boucles ( `for` ), etc.

```
1 {% if users|length > 0 %}  
2     <ul>  
3         {% for user in users %}  
4             <li>{{ user.username|e }}</li>  
5         {% endfor %}  
6     </ul>  
7 {% endif %}
```

# Inclusion vs. Heritage



# Inclusion d'Autres Templates

- La fonction `include` permet d'inclure un template :

```
1 {{ include('sidebar.html.twig') }}
```

- Les templates inclus ont le même contexte que le template appelant.

```
1 {% for box in boxes %}  
2     {{ include('render_box.html.twig') }}  
3 {% endfor %}
```

- Variable `box` différente à chaque inclusion de `render_box.html.twig`

# Héritage de Template

- Héritage : squelette de base et blocs définis dans les templates enfants

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     {% block head %}
5       <title>{% block title %}{% endblock %} - My Webpage</title>
6     {% endblock %}
7   </head>
8   ...
9 </html>
```

```
1 {% extends "base.html.twig" %}
2
3 {% block titre %}Index{% endblock %}
4 {% block head %}
5   {{ parent() }}
6   <style type="text/css">
7     .important { color: #336699; }
8   </style>
9 {% endblock %}
```

# Échappement HTML

- Twig peut échapper automatiquement le HTML.

```
1 {% autoescape %}  
2     Everything will be automatically escaped in this block (using the HTML strategy)  
3 {% endautoescape %}
```

- Sinon escpate manuel avec le filtre `escape` :

```
1 {{ utilisateur.nom|escape('html') }}
```

# Macros et Expressions

- Les macros sont des raccourcis sous forme de fonction :

```
1 {% macro input(name, value, type = "text") %}  
2     <input type="{{ type }}" name="{{ name }}" value="{{ value }}" />  
3 {% endmacro %}
```

```
1 <p>{{ forms.input('username') }}</p>  
2 <p>{{ forms.input('password', null, 'password') }}</p>
```

# Expressions dans Twig - Littéraux

- ❑ **Attention**, les opérations déportent la logique du contrôleur/modèle dans la vue !
  - ▷ Risque de se mélanger !
- ❑ Les formes les plus simples d'expressions sont les littéraux :
  - ▷ Valeurs en dur, à utiliser dans des expressions avec des variables
- ❑ Exemples :
  - ▷ Chaînes : `"Bonjour le monde"` , `'C'est bon'`
  - ▷ Nombres : `42` , `42.23`
  - ▷ Tableaux : `["foo", "bar"]`
  - ▷ Dictionnaires : `{"foo": "bar"}` , `{foo: 'foo', bar: 'bar'}`
  - ▷ Booléens : `true` , `false`
  - ▷ Null : `null`

# Expressions dans Twig - Mathématiques

- Twig autorise les opérations mathématiques dans les templates.

```
1  {# `+` : Addition #}  
2  {{ date + 7 }}  
  
3  {# `-` : Soustraction #}  
4  {{ heure - 2 }}  
  
5  {# `/` : Division #}  
6  {{ set moyenne = somme / longueur }}  
  
7  {# `%` : Modulo #}  
8  {{ (nombre % 2) == 0 }}  
  
9  {# `//` : Division entière #}  
0  {{ set nbGroups = (nbElevés / 14) + 1 }}  
  
1  {# `*` : Multiplication #}  
2  {{ quantité * prixUnitaire }}  
  
3  {# `**` : Puissance #}  
4  {{ set surface = longueur ** 2 }}
```

# Expressions dans Twig - Logique

- Les expressions peuvent être combinées avec des opérateurs logiques :
  - ▷ `and` : Retourne vrai si les deux opérandes sont vrais.
  - ▷ `or` : Retourne vrai si l'un des opérandes est vrai.
  - ▷ `not` : Négation d'une déclaration.
  - ▷ `(expr)` : Groupe une expression.
- Twig supporte les opérateurs de comparaison : `==` , `!=` , `<` , `>` , `>=` , `<=` .

# Expressions dans Twig - Autres Opérateurs

```
1 {% if 'Fabien' starts with 'F' %}{% endif %}  
2 {% if phone matches '/^[\\d\\.]+$/ ' %}{% endif %}  
3 {{ "Hello #{name}, how are you?" }}
```

## Opérateurs spéciaux :

- `1..10` : Crée une séquence.
- `"Hello " ~ "World!"` : Concatène des chaînes.
- `object.attribute` , `array[index]` : Accède à un attribut d'une variable.
- `condition ? then : else` : Opérateur ternaire.
- `age ?? 0` : Coalescence nulle (retourne la valeur si existe sinon).
- `post.status is "PUBLISHED"` : Egalité
- `groupeTP in 1..10` : Inclusion
- `not` : Inverse