UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Symfony 7

Sommaire

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Présentation de Symfony et de son écosystème

Présentation de Symfony et de son écosystème

Définition

- Symfony est un framework PHP très populaire pour la création d'applications web.
- Il a été publié pour la première fois en 2005 par Fabien Potencier et est actuellement l'un des frameworks les plus utilisés dans le monde du développement web PHP.
- Symfony est conçu pour permettre aux développeurs de construire des applications web robustes, évolutives et performantes, tout en favorisant une programmation rapide et une maintenance facile.

Présentation de Symfony et de son écosystème

Composants clés de Symfony

1. **Kernel** : Le cœur du framework, gère le cycle de vie des requêtes et des réponses.
2. **Routing** : Permet de définir des routes pour diriger les requêtes HTTP vers les contrôleurs appropriés.
3. **Controller** : Lieu où la logique de gestion des requêtes est définie.
4. **Twig** : Le moteur de template par défaut de Symfony, permettant de créer des vues séparées de la logique PHP.
5. **Doctrine** : Couche d'abstraction de base de données souvent utilisée avec Symfony pour faciliter la manipulation de bases de données.
6. **Forms** : Composant pour créer et gérer des formulaires.
7. **Validator** : Fournit des outils pour valider les données envoyées par les utilisateurs.
8. **Security** : Gère l'authentification, l'autorisation, et la sécurité des applications.
9. **HTTP Client** : Permet de réaliser des requêtes HTTP.
10. **Mailer** : Gère l'envoi d'e-mails.

Présentation de Symfony et de son écosystème

Écosystème

L'écosystème Symfony comprend divers outils et projets qui complètent ou étendent les fonctionnalités du framework :

- **Symfony Flex** : Un outil qui facilite la configuration et la gestion des dépendances pour les applications Symfony.
- **API Platform** : Un framework puissant pour construire des API hypermédias avancées et des applications React ou Vue.js avec un backend Symfony.
- **Symfony Bundles** : Des paquets réutilisables qui peuvent être intégrés dans n'importe quelle application Symfony pour ajouter des fonctionnalités.
- **Symfony Console** : Permet de créer des applications en ligne de commande.
- **Symfony Workflow** : Un composant qui aide à définir des processus complexes, séquentiels ou parallèles, connus sous le nom de workflows.

Présentation de Symfony et de son écosystème

Communauté et ressources

- Symfony bénéficie d'une grande communauté de développeurs.
- Il existe de nombreuses conférences (SymfonyCon, SymfonyLive), des meetups, et une documentation très complète accessible à tous.
- Le framework est aussi supporté par SensioLabs et une large communauté de contributeurs qui continuent à développer et à améliorer ses fonctionnalités.

Présentation de Symfony et de son écosystème

Environnement de travail

- Symfony propose un outil de ligne de commande très puissant nommé **Symfony CLI** qui est essentiel pour gérer efficacement le développement des applications Symfony.
- Cet outil offre une gamme complète de fonctionnalités qui facilitent la création, la gestion et l'exploitation des applications Symfony.

Présentation de Symfony et de son écosystème

Environnement de travail

1. **Serveur Web Local** : Symfony peut être utilisé avec divers serveurs web, comme Apache ou Nginx. Cependant, pour un développement rapide, Symfony CLI inclut un serveur web local qui peut être lancé très facilement pour tester les applications.
2. **Environnement PHP** : Par exemple, pour Symfony 7, il est nécessaire de disposer au minimum de PHP 8.2, bien que les versions ultérieures puissent être recommandées pour exploiter toutes les fonctionnalités et améliorations de performance.
3. **Composer** : Il s'agit du gestionnaire de dépendances pour PHP. Composer est utilisé pour gérer les bibliothèques dont dépend le projet Symfony.

Présentation de Symfony et de son écosystème

Environnement de travail

4. **Base de données** : Symfony supporte plusieurs systèmes de gestion de base de données comme MySQL, PostgreSQL, SQLite, etc., et utilise souvent Doctrine ORM pour interagir avec la base de données.
5. **Outils de développement** : Des outils comme Xdebug pour le débogage, PHPUnit pour les tests unitaires, et des intégrations avec des systèmes de versionnage comme Git.

Présentation de Symfony et de son écosystème

Symfony CLI

L'outil Symfony CLI est conçu pour augmenter la productivité des développeurs en fournissant une interface en ligne de commande pour effectuer de nombreuses tâches courantes de développement et de déploiement.

- **Démarrage du serveur local** : Vous pouvez démarrer un serveur web local en utilisant la commande `symfony server:start`. Ce serveur est configuré pour fonctionner avec les applications Symfony et fournit des performances optimales en développement.
- **Création de nouvelles applications** : Avec la commande `symfony new <nom-du-projet>`, vous pouvez créer une nouvelle application Symfony avec ou sans les dépendances full-stack.

Présentation de Symfony et de son écosystème

Symfony CLI

- **Gestion de la sécurité** : Le CLI peut vérifier les vulnérabilités de sécurité dans les bibliothèques utilisées par l'application.
- **Gestion des environnements** : Facilite la configuration des variables d'environnement nécessaires pour différents environnements de déploiement.
- **Debugging et logs** : Offre des outils pour visualiser les logs et faire du debugging de l'application.
- **Interactions avec les bases de données** : Permet de créer, gérer et migrer les bases de données directement depuis la ligne de commande.
- **Intégration avec SymfonyCloud** : Symfony CLI est parfaitement intégré avec SymfonyCloud, ce qui facilite le déploiement des applications Symfony dans le cloud.

Présentation de Symfony et de son écosystème

Comparaison

Fonctionnalité	Symfony 4	Symfony 5	Symfony 6	Symfony 7
Version PHP minimale requise	PHP 7.1.3	PHP 7.2.5	PHP 8.0	PHP 8.1 ou supérieur
Symfony Flex	Introduit	Amélioré	Amélioré	Amélioré
Composant Security	Basique	Refonte partielle	Refonte complète	Améliorations supplémentaires
Composant Mailer	Non disponible	Introduit	Amélioré	Amélioré
Support des microservices	Limité	Amélioré	Amélioré	Très amélioré

Présentation de Symfony et de son écosystème

Comparaison

Fonctionnalité	Symfony 4	Symfony 5	Symfony 6	Symfony 7
Version PHP minimale requise	PHP 7.1.3	PHP 7.2.5	PHP 8.0	PHP 8.1 ou supérieur
Intégration API Platform	Possible	Amélioré	Amélioré	Intégration plus profonde
Twig (moteur de templates)	Amélioré	Amélioré	Amélioré	Fonctionnalités supplémentaires
Performances	Bonnes	Améliorées	Très améliorées	Optimisées
Dépréciation et	Ongoing	Continu	Continu	Nettoyage final des

Présentation de Symfony et de son écosystème

Apport Symfony 7

Fonctionnalité	Améliorations Clés
Performance	Optimisations du compilateur et des composants.
Authentification	Système refondu, support pour OAuth2 et JWT.
Docker	Meilleure intégration pour le développement et déploiement.
Mailer	Nouvelles fonctionnalités pour la gestion des e-mails.
Workflow	Support pour les sous-processus, meilleure visualisation.
API Platform	Meilleure création d'APIs REST et GraphQL, plus d'outils.
Twig 4	Moteur de templates amélioré, plus expressif.
Debugging et Profiling	Interface utilisateur améliorée, plus de détails sur les performances.
Tests	Support pour les tests parallèles, meilleure intégration avec

Installation de Symfony et configuration de l'environnement de développement.

Installation de Symfony

1. Installation de Symfony

Symfony propose plusieurs méthodes d'installation, mais l'utilisation de Composer est la méthode recommandée.

a. Installation de Symfony via Composer

```
composer create-project symfony/website-skeleton my_project_name
```

Si vous prévoyez de construire une micro-application ou une API, vous pouvez choisir de démarrer avec la version minimale de Symfony

```
composer create-project symfony/skeleton my_project_name
```

Installation de Symfony et configuration de l'environnement de développement.

Installation de Symfony

1. Installation de Symfony

Symfony propose plusieurs méthodes d'installation, mais l'utilisation de Composer est la méthode recommandée.

b. Utilisation de Symfony CLI

Une autre option est d'utiliser Symfony CLI, un outil qui fournit des fonctionnalités supplémentaires comme la création de serveurs de développement locaux et la gestion de certificats SSL locaux. Vous pouvez télécharger Symfony CLI depuis le site web officiel de Symfony.

```
symfony new my_project_name --webapp
```

Installation de Symfony et configuration de l'environnement de développement.

Installation de Symfony

2. Configuration de l'Environnement de Développement

- Serveur Web Local: Symfony CLI fournit un serveur web local pratique pour tester vos applications Symfony.

```
cd my_project_name
symfony server:start
```

- Configuration Environnementale
- Symfony utilise des variables d'environnement stockées dans le fichier `.env` à la racine de votre projet pour gérer la configuration de l'application.
- Vous pouvez personnaliser les configurations de la base de données, les clés API, et d'autres paramètres sensibles en modifiant ce fichier.

Structure d'un projet Symfony et philosophie.

Structure d'un Projet Symfony

- **`bin/`**: Contient les exécutables, y compris la console Symfony pour exécuter des commandes.
- **`config/`**: Stocke tous les fichiers de configuration de votre projet. La configuration est divisée par packages et environnements, permettant une personnalisation fine.
- **`public/`**: Le dossier accessible publiquement. Il contient le fichier `index.php`, qui est le point d'entrée de toutes les requêtes dans votre application, ainsi que des ressources statiques comme les images et les fichiers JavaScript et CSS.
- **`src/`**: Cœur de votre application, contenant le code PHP, comme les contrôleurs, les entités (modèles), les formulaires, et plus. Il est structuré de manière à suivre les principes de l'architecture logicielle.
- **`templates/`**: Contient les templates Twig, le moteur de template de Symfony, pour la génération des vues.

Structure d'un projet Symfony et philosophie.

Structure d'un Projet Symfony

- `translations/`: Dossier pour les fichiers de traduction, permettant l'internationalisation de votre application.
- `var/`: Contient les fichiers générés par Symfony comme le cache et les logs, spécifiques à l'environnement de votre application (dev, test, prod).
- `vendor/`: Géré par Composer, ce dossier contient toutes les bibliothèques tierces et les composants de Symfony utilisés dans votre projet.
- `tests/`: Dossier pour les tests unitaires et fonctionnels de votre application, encouragés pour suivre la méthodologie TDD (Test-Driven Development).

Structure d'un projet Symfony et philosophie.

Philosophie de Symfony

- **Flexibilité**: Au cœur de Symfony se trouve le principe de la flexibilité. Le framework est conçu pour s'adapter à vos besoins, pas l'inverse. Que vous construisiez une petite API ou une application web complexe, Symfony peut être aussi léger ou complet que nécessaire.
- **Réutilisabilité**: Grâce à sa structure modulaire et à l'utilisation intensive de composants réutilisables (bundles), Symfony encourage le développement de solutions pouvant être facilement partagées et réutilisées dans différents projets.
- **Meilleures pratiques**: Symfony est conçu pour encourager les développeurs à suivre les meilleures pratiques de programmation et les principes de conception de logiciels, comme le modèle MVC (Modèle-Vue-Contrôleur) pour une séparation claire des préoccupations.
- **Communauté**: Une large communauté de développeurs soutient Symfony, contribuant à une riche écosystème de bundles, de plugins, et de documentation, facilitant ainsi l'apprentissage et l'adoption du framework.
- **Performance**: Bien que riche en fonctionnalités, Symfony est optimisé pour la performance, avec des outils comme le composant HttpCache pour réduire le temps de réponse des applications.
- **Interopérabilité**: Symfony respecte les standards PHP et encourage l'utilisation de composants et de bibliothèques qui suivent les normes de l'industrie, permettant une intégration fluide avec d'autres systèmes et frameworks.

Injection de dépendance et inversion de contrôle.

Inversion de Contrôle (IoC)

- L'Inversion de Contrôle est un principe de conception loicielle où le contrôle du flux d'exécution est inversé par rapport à la programmation traditionnelle.
- Dans la programmation traditionnelle, votre code (les appels de méthodes, les instanciations d'objets, etc.) contrôle le flux de l'application. Avec l'IoC, ce flux est externalisé à un conteneur ou un framework.
- L'IoC est souvent réalisé à travers différents mécanismes tels que l'injection de dépendance, les "templates method" et les "strategy pattern".
- Le but est de réduire le couplage entre les composants du logiciel, rendant le système plus flexible, plus facile à tester et à maintenir.

Injection de dépendance et inversion de contrôle.

Injection de Dépendance (DI)

- L'injection de dépendance est une forme spécifique de l'IoC où les dépendances (c'est-à-dire les instances de classes dont un objet a besoin pour fonctionner) sont "injectées" dans un objet au lieu que l'objet les crée lui-même.
- Il existe plusieurs façons d'injecter des dépendances, y compris par constructeur, par setter, ou directement dans les propriétés.
- Dans le contexte d'un framework comme Symfony, l'injection de dépendance est gérée par un composant spécialisé appelé "conteneur de services".
- Ce conteneur crée et gère les instances des objets et leurs dépendances, les injectant là où elles sont nécessaires selon la configuration spécifiée par le développeur.

Injection de dépendance et inversion de contrôle.

Avantages de l'IoC et de la DI

- **Couplage faible :** Les composants de votre application sont moins dépendants les uns des autres, ce qui rend votre système plus modulaire et plus flexible.
- **Facilité de test :** Il est plus facile de tester les composants en isolation en injectant des implémentations factices ou des mock objects pour les dépendances.
- **Gestion centralisée des dépendances :** Le conteneur de services de Symfony gère la création et l'injection des dépendances, ce qui simplifie la gestion des instances de classe à travers l'application.
- **Configuration externe :** Les dépendances peuvent être configurées en dehors du code source, par exemple dans des fichiers de configuration YAML ou XML de Symfony, facilitant les changements sans nécessiter de modifier le code.

Injection de dépendance et inversion de contrôle.

Utilisation dans Symfony

- Symfony utilise intensivement l'injection de dépendance et l'inversion de contrôle à travers son conteneur de services.
- Par exemple, lorsque vous définissez un service dans Symfony (une classe que vous souhaitez utiliser à travers l'application), vous pouvez spécifier ses dépendances dans le fichier de configuration du service.
- Symfony s'occupe ensuite de l'instanciation de ces services et de leurs dépendances lorsque le service est requis.

```
# config/services.yaml
services:
    App\Service\MyService:
        arguments:
            $dependency: '@another_service'
```

Création de routes et de contrôleurs : gestion des requêtes HTTP.

Création de Routes

- Les routes peuvent être configurées de plusieurs manières : annotations dans les contrôleurs, fichiers YAML, XML ou PHP.
- Par exemple, vous pouvez définir une route dans un contrôleur en utilisant des annotations PHP :

```php
use Symfony\Component\Routing\Annotation\Route;

class MonController {
    #[Route('/chemin', name: 'nom_route')]
    public function maMethode(): Response {
        // retour route
    }
}
```

Création de routes et de contrôleurs : gestion des requêtes HTTP.

Contrôleurs

- Un contrôleur dans Symfony est une classe qui traite les requêtes HTTP et retourne des réponses. - La classe doit étendre `AbstractController` ou implémenter une logique de retour de réponse appropriée.

```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ConferenceController extends AbstractController {
    #[Route('/conference', name: 'app_conference')]
    public function index(): Response {
        return $this->render('conference/index.html.twig', [
            'controller_name' => 'ConferenceController',
        ]);
    }
}
```

Ce contrôleur retourne une vue Twig, mais vous pouvez également retourner directement une réponse HTTP.

Création de routes et de contrôleurs : gestion des requêtes HTTP.

Gestion des Requêtes HTTP

- Les méthodes HTTP (GET, POST, PUT, DELETE, etc.) spécifient le type d'action que le client souhaite effectuer.
- Par défaut, une route accepte toutes les méthodes HTTP, mais vous pouvez restreindre ce comportement en spécifiant les méthodes autorisées via l'argument `methods` de l'annotation `Route` :

```
#[Route('/blog', name: 'article_list', methods: ['GET'])]
```

Introduction à Twig et création de templates.

Introduction à Twig

- Twig est conçu pour être à la fois convivial pour les développeurs et sécurisé.
- Il offre une syntaxe claire et concise pour l'insertion de données PHP et l'exécution de structures logiques simples directement dans les fichiers de template HTML.
- Twig compile automatiquement les templates en code PHP pur, ce qui lui permet d'être très performant.

Introduction à Twig et création de templates.

Introduction à Twig

Les principales caractéristiques de Twig incluent :

- **Syntaxe Expressive** : Twig utilise une syntaxe concise qui simplifie l'ajout de logique dans les vues, comme des boucles, des conditions, et des filtres pour manipuler les données.
- **Héritage de templates** : Twig supporte l'héritage, permettant aux développeurs de définir un "layout" de base et de le réutiliser à travers différents templates, facilitant ainsi la maintenance et la cohérence du design.
- **Filtrage et Fonctions** : Une large gamme de filtres et de fonctions est disponible pour formater les données, réaliser des calculs et exécuter d'autres opérations courantes directement dans les templates.
- **Sécurité** : Twig offre une échappement automatique des sorties pour protéger contre les attaques XSS (Cross-site scripting), une considération de sécurité importante dans le développement web.

Introduction à Twig et création de templates.

Introduction à Twig

Création de Templates avec Twig dans Symfony 7

Avec Symfony 6, l'utilisation de Twig est intégrée de manière transparente, facilitant la création de systèmes de templates puissants et flexibles. Pour commencer avec Twig dans Symfony, suivez ces étapes :

1. **Installation** : Twig est inclus par défaut dans les applications Symfony, mais si nécessaire, vous pouvez l'installer ou le mettre à jour via Composer avec `composer require symfony/twig-bundle`.
2. **Configuration** : Les configurations de Twig peuvent être ajustées dans le fichier `config/packages/twig.yaml` de votre application Symfony, permettant de personnaliser le comportement du moteur de template (comme le répertoire des templates, l'échappement automatique, etc.).
3. **Création de Templates** : Les templates Twig sont généralement stockés dans le répertoire `templates/` de votre application Symfony. Un fichier de template Twig utilise l'extension `.html.twig` et peut contenir à la fois du HTML standard et la syntaxe Twig pour insérer des données dynamiques.
4. **Rendu de Templates** : Dans vos contrôleurs Symfony, vous pouvez rendre un template Twig en utilisant la méthode `render()` du contrôleur, en passant le chemin du template et un tableau de données qui seront disponibles dans le template.

```php
return $this->render('mon_template.html.twig', ['maVariable' => $maVariable]);
```

Introduction à Twig et création de templates.

Introduction à Twig

```twig
{# templates/products_list.html.twig #}

<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Liste des Produits</title>
</head>
<body>
    <h1>Liste des Produits</h1>

    {% if products is not empty %}
        <ul>
            {% for product in products %}
                <li>
                    <strong>{{ product.name }}</strong><br>
                    Prix: {{ product.price|number_format(2, ',', ' ') }} €<br>
                    {% if product.inStock %}
                        <em>En stock</em>
                    {% else %}
                        <em>Rupture de stock</em>
                    {% endif %}
                </li>
            {% endfor %}
        </ul>
    {% else %}
        <p>Aucun produit disponible.</p>
    {% endif %}
</body>
</html>
```

Introduction à Twig et création de templates.

Introduction à Twig

- **Boucle For** : `{% for product in products %}` itère sur chaque élément du tableau `products`. Pour chaque itération, la variable `product` contient l'élément courant du tableau (un produit).
- **Filtre `number_format`** : `{{ product.price|number_format(2, ',', ' ') }}` formate le prix du produit pour avoir deux chiffres après la virgule, utilise la virgule comme séparateur décimal et un espace pour séparer les milliers. Cela rend le prix plus lisible.
- **Condition If** : `{% if product.inStock %}` vérifie si le produit est en stock. Selon le résultat, il affiche soit "En stock" soit "Rupture de stock".
- **Vérification de la vacuité** : `{% if products is not empty %}` vérifie si le tableau `products` n'est pas vide avant d'essayer d'afficher la liste des produits. Si le tableau est vide, il affiche un message indiquant qu'aucun produit n'est disponible.

Introduction à Twig et création de templates.

Introduction à Twig

Inclusion de Fichiers Statiques

- Dans Twig, pour inclure des fichiers statiques comme des fichiers CSS ou JavaScript, vous utilisez généralement la fonction `asset()` dans le cadre d'un projet Symfony.
- Cette fonction aide à générer des URL vers des fichiers statiques de manière flexible et sécurisée.
- L'usage de `asset()` est particulièrement utile pour gérer les chemins des ressources statiques dans des environnements de développement, de test, et de production, en s'adaptant automatiquement aux configurations spécifiques de chaque environnement.

Introduction à Twig et création de templates.

Introduction à Twig

Exemple :

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="{{ asset('css/style.css') }}">
</head>
<body>
    ...
    <script src="{{ asset('js/script.js') }}"></script>
</body>
</html>
```

Dans cet exemple, `{{ asset('css/style.css') }}` génère le chemin vers le fichier CSS statique, et `{{ asset('js/script.js') }}` fait de même pour un fichier JavaScript. Ces chemins tiendront compte de la configuration de votre projet Symfony, y compris l'emplacement du dossier public et d'éventuels paramètres de versionnement des fichiers.

Introduction à Twig et création de templates.

Introduction à Twig

Extends dans Twig

- L'héritage de templates (`extends`) dans Twig permet de construire une base de template qui peut être étendue ou surchargée par d'autres templates
- Cela facilite la réutilisation de portions communes de votre site web (comme l'en-tête, le pied de page, la barre latérale, etc.) sans duplication de code.

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Mon Site Web{% endblock %}</title>
    <link rel="stylesheet" href="{{ asset('css/style.css') }}">
</head>
<body>
    <header>
        {% block header %}Entête du site{% endblock %}
    </header>

    <div id="content">
        {% block content %}Contenu principal ici{% endblock %}
    </div>

    <footer>
        {% block footer %}Pied de page du site{% endblock %}
    </footer>
</body>
</html>
```

Introduction à Twig et création de templates.

Introduction à Twig

Template enfant :

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

{% block title %}Page d'Accueil{% endblock %}

{% block content %}
    <h1>Bienvenue sur notre site !</h1>
    <p>Voici le contenu spécifique de la page d'accueil.</p>
{% endblock %}
```

- Dans cet exemple, le template enfant `extends` le template de base `base.html.twig`.
- Il redéfinit les blocs `title` et `content` pour personnaliser le titre de la page et le contenu principal, respectivement.
- Les blocs `header` et `footer` ne sont pas redéfinis dans cet exemple, donc ils hériteront du contenu par défaut défini dans le template de base.

Services et conteneur de services

Services : Définition

- Un service dans Symfony est simplement un objet PHP qui effectue une certaine tâche.
- Cela peut être n'importe quoi, d'un objet qui envoie des e-mails, à un objet qui génère des formulaires, ou même un objet qui effectue des calculs complexes.
- L'idée est de centraliser une fonctionnalité dans un service pour pouvoir la réutiliser facilement dans différentes parties de l'application.
- Les services sont souvent utilisés pour encapsuler la logique métier ou les fonctionnalités qui sont utilisées à plusieurs endroits dans l'application, rendant le code plus modulaire, réutilisable et facile à tester.

Services et conteneur de services

Conteneur de Services : Définition

- Le conteneur de services, souvent simplement appelé "le conteneur", est un objet qui sait comment instancier et gérer les services.
- Il permet de centraliser la configuration des services de votre application, et s'occupe de l'injection de dépendances, ce qui signifie qu'il fournit automatiquement aux services tout ce dont ils ont besoin pour fonctionner.
- Lorsque vous demandez un service au conteneur, celui-ci s'assure que toutes les dépendances du service sont satisfaites, instancie le service si ce n'est pas déjà fait, et vous le retourne.
- Cela simplifie grandement la gestion des dépendances dans votre application.

Services et conteneur de services

Utilisation Basique dans les Contrôleurs

Injection de Dépendances

Exemple avec le constructeur :

```php
use App\Service\MonService; // Assurez-vous d'importer votre service

class MonController extends AbstractController
{
    private $monService;

    public function __construct(MonService $monService)
    {
        $this->monService = $monService;
    }

    public function index()
    {
        $resultat = $this->monService->faireQuelqueChose();

    }
}
```

Services et conteneur de services

Utilisation Basique dans les Contrôleurs

Exemple avec l'injection dans les méthodes d'action :

```php
use App\Service\MonService; // Assurez-vous d'importer votre service

class MonController extends AbstractController
{
    public function index(MonService $monService)
    {
        $resultat = $monService->faireQuelqueChose();

        // Utilisez $resultat pour quelque chose
    }
}
```

Doctrine ORM et Formulaires

Doctrine ORM

- **Doctrine ORM (Object-Relational Mapping)** est une bibliothèque pour gérer les bases de données dans les applications PHP.
- Elle permet aux développeurs de travailler avec des bases de données en utilisant des objets PHP, ce qui facilite l'interaction avec la base de données de manière plus intuitive et orientée objet.
- **Abstraction de la base de données :** Doctrine fournit une couche d'abstraction qui vous permet de travailler avec différents types de bases de données (comme MySQL, PostgreSQL, SQLite, etc.) de manière transparente, sans avoir à modifier votre code en fonction de la base de données spécifique que vous utilisez.
- **Travail avec des objets :** Au lieu d'écrire des requêtes SQL brutes, vous travaillez avec des objets PHP représentant vos tables de base de données. Ces objets, appelés entités, sont des classes PHP que vous définissez pour chaque table de votre base de données.

Doctrine ORM et Formulaires

Doctrine ORM

- **DQL (Doctrine Query Language) :** Doctrine dispose de son propre langage de requête, le DQL, qui est une abstraction au-dessus de SQL. Le DQL permet de réaliser des requêtes sur les entités et leurs relations d'une manière qui est cohérente avec la programmation orientée objet.
- **Migrations :** Doctrine offre un système de migrations, vous permettant de versionner et de partager la structure de votre base de données. Cela facilite les modifications de schéma de base de données au fil du temps et entre différents environnements de développement.

Doctrine ORM et Formulaires

Doctrine ORM

- **Data Mapping :** Doctrine vous permet de définir des mappings entre vos classes PHP et les tables de base de données. Ces mappings peuvent être définis avec des annotations, des fichiers XML, ou YAML.
- **Gestion des relations :** Doctrine ORM gère tous les types de relations entre les tables (comme One-To-One, One-To-Many, Many-To-One, et Many-To-Many) et permet de naviguer entre les objets associés de manière très simple.

```php
// src/Entity/Product.php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 * @ORM\Table(name="product")
 */
class Product
{
    /**
     * @ORM\Id
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue
     */
    private $id;

    /**
```

Doctrine ORM et Formulaires

Doctrine ORM

- **Configuration :** Configurez votre connexion à la base de données dans le fichier `.env` de Symfony.
- **Création d'entités :** Utilisez la commande `php bin/console make:entity` pour générer des entités basées sur vos tables de base de données.
- **Interactions avec la base de données :** Utilisez le gestionnaire d'entités pour créer, lire, mettre à jour et supprimer des données.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

AssetMapper

AssetMapper

Definition

- **AssetMapper** est une fonctionnalité introduite dans **Symfony 6.3** pour simplifier la gestion des assets front-end (CSS, JavaScript, images, etc.) dans vos applications Symfony.
- Elle permet de remplacer des outils comme Webpack Encore dans des cas simples où il n’est pas nécessaire d'utiliser un outil de build complet.
- Cela offre une approche plus légère et native pour gérer vos fichiers statiques.

AssetMapper

À quoi sert AssetMapper ?

1. **Gestion simplifiée des assets :**
   Permet de gérer les fichiers CSS, JS, images, et autres ressources statiques directement sans avoir recours à des outils de compilation complexes.
2. **Mapping des fichiers sources :**
   Les fichiers situés dans des répertoires comme `assets/` sont automatiquement copiés vers le répertoire `public/` en conservant une structure organisée.
3. **Versionnement et cache-busting :**
   Génère des URLs versionnées (avec des hash) pour forcer le navigateur à récupérer la dernière version d'un fichier lorsque celui-ci est modifié.
4. **Interopérabilité avec Twig et Symfony UX :**
   Permet d'intégrer facilement des assets dans vos vues Twig ou vos composants UX grâce à des fonctions intégrées.
5. **Optimisation des performances :**
   Gère automatiquement la minification des fichiers CSS et JS si nécessaire.

AssetMapper

Comment fonctionne AssetMapper ?

1. **Structure des fichiers :**
   - Les fichiers statiques (CSS, JS, images, polices) sont stockés dans un répertoire `assets/` ou un chemin configuré dans le fichier `config/packages/asset_mapper.yaml`.
2. **Configuration minimale :**
   Par défaut, AssetMapper est configuré pour mapper les fichiers de `assets/` vers `public/assets/`.

   Exemple de configuration dans `asset_mapper.yaml` :

```
asset_mapper:
    paths:
        'assets': ~ # Définit le dossier source des assets
```

AssetMapper

Comment fonctionne AssetMapper ?

3. **Utilisation dans Twig :**
   Grâce à la fonction `asset()` ou `asset_path()` en Twig, vous pouvez référencer vos assets dans vos templates.

   Exemple :

```
<link rel="stylesheet" href="{{ asset('app.css') }}">
<script src="{{ asset('app.js') }}"></script>
```

AssetMapper

Comment fonctionne AssetMapper ?

4. **Installation et synchronisation des assets :**
   Symfony met à disposition des commandes pour synchroniser vos fichiers statiques avec le répertoire public. Par exemple :

```
php bin/console asset:map
```

5. **Gestion des dépendances externes :**
   Vous pouvez inclure des bibliothèques ou des fichiers externes dans `asset_mapper.yaml` en ajoutant leurs chemins.

AssetMapper

Avantages d’AssetMapper :

1. **Simplicité :**
   Idéal pour des projets où les besoins front-end sont basiques.
2. **Remplace des outils lourds :**
   Supprime le besoin de Webpack Encore ou d'autres outils complexes pour des projets simples.
3. **Prêt à l'emploi :**
   Fonctionne immédiatement avec la configuration par défaut de Symfony.
4. **Intégration native :**
   Totalement intégré dans l'écosystème Symfony.

AssetMapper

Avantages d'AssetMapper :

1. **Pas adapté pour les projets complexes :**
   Si votre projet front-end nécessite une gestion avancée (comme des frameworks JavaScript modernes, ou un système de build complexe), AssetMapper sera limité.

2. **Pas de fonctionnalités avancées de build :**
   Il ne gère pas les fonctionnalités comme les transpilations (TypeScript ou SCSS), les optimisations poussées ou les dépendances NPM.

AssetMapper

Exemple complet : Utilisation d'AssetMapper

1. Structure du projet

```
/assets
    /css
        app.css
    /js
        app.js
/public
    /assets
        (auto-rempli par AssetMapper)
```

2. Configuration `asset_mapper.yaml`

```
asset_mapper:
    paths:
        'assets': ~
```

AssetMapper

Exemple complet : Utilisation d'AssetMapper

3. Utilisation dans Twig

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="{{ asset('css/app.css') }}">
    </head>
    <body>
        <script src="{{ asset('js/app.js') }}"></script>
    </body>
</html>
```

AssetMapper

Exemple complet : Utilisation d’AssetMapper

Les commandes d'assetMapper

Commande	Description
`asset-map:compile`	Prépare les fichiers finaux pour les servir (minifiés, optimisés en production).
`asset-map:dump`	Génère les fichiers sans appliquer d’optimisations (principalement pour le développement).
`asset-map:watch`	Surveille les modifications et génère les fichiers automatiquement en temps réel.
`asset-map:import`	Importe des actifs depuis des packages ou sources externes.
`asset-map:clean`	Supprime les fichiers inutilisés générés par Asset Mapper.

AssetMapper

Quand utiliser AssetMapper ?

1. **Pour des projets simples ou débutants :**
   Idéal si votre application n’a pas de gros besoins front-end.
2. **Pour des besoins légers en CSS/JS :**
   Parfait pour des applications backend où l'interface utilisateur utilise peu de fichiers front-end.
3. **Pour réduire la complexité :**
   Supprime le besoin de configurer et de maintenir des outils de build.

AssetMapper

Utilisez Asset Mapper si :
- Vous travaillez principalement sur des projets Symfony backend.
- Votre projet nécessite une gestion simple des assets (CSS, JS, images).
- Vous préférez une solution sans dépendances supplémentaires (comme Node.js).
- Vous voulez une configuration légère et rapide à mettre en place.

Utilisez Webpack/Vite si :
- Vous développez un projet frontend complexe (React, Vue, Angular).
- Vous avez besoin de transpiler du TypeScript ou ES6+ en JS compatible navigateur.
- Vous souhaitez optimiser les performances avec des outils avancés (tree-shaking, code-splitting, etc.).
- Vous avez besoin d'un écosystème riche de plugins pour des besoins spécifiques.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Symfony Form

Symfony Form

Qu’est-ce que Symfony Form ?

Symfony Form est un composant qui simplifie la création, la validation et le traitement des formulaires dans une application Symfony. Il permet de :

1. **Créer facilement des formulaires HTML dynamiques**.
2. **Valider les données des formulaires** avec le composant Validator.
3. **Mapper les données des formulaires à des objets ou tableaux**.
4. **Gérer des formulaires imbriqués ou complexes** (par exemple, relation entre entités).
5. **Étendre les fonctionnalités** pour répondre à des besoins spécifiques.

Symfony Form

Les concepts fondamentaux

a. Structure d'un formulaire Symfony

Un formulaire est composé de trois éléments principaux :

- **FormBuilder** : définit les champs et leurs types.
- **FormView** : génère le HTML final.
- **FormData** : contient les données mappées entre le formulaire et les objets.

Symfony Form

Les concepts fondamentaux

b. Types de champs

Symfony Form fournit des types de champs prêts à l'emploi :

- **Texte** : `TextType`, `TextareaType`, `EmailType`, `PasswordType`.
- **Sélection** : `ChoiceType`, `EntityType`.
- **Date et temps** : `DateType`, `DateTimeType`, `TimeType`.
- **Collections** : `CollectionType`.
- **Spécifiques** : `FileType`, `CheckboxType`, `RadioType`, etc.

Symfony Form

Les concepts fondamentaux

c. Méthodes courantes

- **Form::createView()** : Génére la vue du formulaire.
- **Form::isSubmitted()** : Vérifie si le formulaire a été soumis.
- **Form::isValid()** : Valide les données.
- **Form::getData()** : Récupère les données du formulaire.

Symfony Form

Exemple de base : Création d'un formulaire

Étape 1 : Créer un FormType

Un `FormType` est une classe qui définit la structure et les champs d'un formulaire.

```
// src/Form/CategoryType.php
namespace App\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;
use Symfony\Component\Form\Extension\Core\Type\TextType;

class CategoryType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('name', TextType::class, [
                'label' => 'Nom de la catégorie',
                'required' => true,
                'attr' => ['class' => 'form-control']
            ]);
    }
// suite
}
```

Symfony Form

Exemple de base : Création d'un formulaire

Étape 2 : Utiliser le formulaire dans un contrôleur

Dans un contrôleur, on crée, gère et affiche le formulaire.

```php
namespace App\Controller;

class CategoryController extends AbstractController
{
    public function new(Request $request): Response
    {
        $category = new Category();
        $form = $this->createForm(CategoryType::class, $category);

        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->persist($category);
            $entityManager->flush();

            return $this->redirectToRoute('category_list');
        }

        return $this->render('category/new.html.twig', [
            'form' => $form->createView(),
        ]);
```

Symfony Form

Exemple de base : Création d'un formulaire

Étape 3 : Afficher le formulaire dans Twig

Symfony fournit une fonction Twig pour rendre un formulaire.

```
{# templates/category/new.html.twig #}

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

{% block body %}
    <h1>Créer une catégorie</h1>

    {{ form_start(form) }}
        {{ form_row(form.name) }}
        <button class="btn btn-primary">Enregistrer</button>
    {{ form_end(form) }}
{% endblock %}
```

Symfony Form

Exemple de base : Création d'un formulaire

Étape 3 : Afficher le formulaire dans Twig

Symfony fournit une fonction Twig pour rendre un formulaire.

```
{# templates/category/new.html.twig #}

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

{% block body %}
    <h1>Créer une catégorie</h1>

    {{ form_start(form) }}
        {{ form_row(form.name) }}
        <button class="btn btn-primary">Enregistrer</button>
    {{ form_end(form) }}
{% endblock %}
```

Symfony Form

Options avancées

a. Validation des données

Symfony Form est étroitement lié au composant Validator. Ajoutez des contraintes directement dans les entités.

```
// src/Entity/Category.php
use Symfony\Component\Validator\Constraints as Assert;

class Category
{
    #[Assert\NotBlank]
    #[Assert\Length(min: 3, max: 50)]
    private $name;
}
```

Symfony Form

Options avancées

b. Champs liés à des entités

Le `EntityType` permet de créer un champ lié à une entité.

```
$builder->add('category', EntityType::class, [
    'class' => Category::class,
    'choice_label' => 'name',
]);
```

Symfony Form

Options avancées

c. Collections de champs

Le `CollectionType` permet de gérer des ensembles dynamiques (par exemple, une liste de tags).

```
use Symfony\Component\Form\Extension\Core\Type\CollectionType;

$builder->add('tags', CollectionType::class, [
    'entry_type' => TextType::class,
    'allow_add' => true,
    'allow_delete' => true,
]);
```

Symfony Form

Options avancées

d. Champs personnalisés

Vous pouvez créer vos propres types de champs.

```php
// src/Form/Type/CustomType.php
namespace App\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;

class CustomType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        // Logique pour construire le champ
    }
}
```

Symfony Form

Bonnes pratiques

1. **Isoler la logique dans les FormType :** Évitez de surcharger vos contrôleurs avec la logique des formulaires.
2. **Réutilisation :** Créez des FormTypes réutilisables dans plusieurs formulaires.
3. **Validation cohérente :** Placez vos règles de validation dans vos entités pour centraliser la logique.
4. **Personnalisation :** Utilisez des classes CSS ou des templates Twig pour personnaliser le rendu.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Symfony UX

Symfony UX

Introduction

- **Symfony UX** est une initiative de Symfony qui vise à rapprocher le développement **frontend** et **backend**, en facilitant l'intégration d'outils JavaScript modernes dans les applications Symfony.
- Il repose sur des principes de **progressive enhancement**, où le JavaScript améliore l'expérience utilisateur sans complexifier inutilement votre projet.
- Symfony UX a été introduit avec Symfony 5.3 et reste entièrement compatible avec Symfony 7.

Symfony UX

Les Objectifs de Symfony UX

Symfony UX facilite :

- **L'intégration d'outils modernes** comme Stimulus, Webpack Encore, et Turbo.
- **La création de composants interactifs** sans écrire de JavaScript complexe.
- **L'automatisation des workflows** pour inclure des librairies front-end (charts, éditeurs de texte riche, carrousels, etc.).
- **Une expérience utilisateur optimisée** grâce à l'enrichissement progressif des fonctionnalités.

Symfony UX

Les Composants Clés de Symfony UX

A. Stimulus

Stimulus est un framework JavaScript léger conçu pour travailler en harmonie avec votre HTML. Il permet de rendre vos applications interactives en associant des contrôleurs JavaScript à des éléments HTML via des attributs `data-*`.

Exemple :

```
<button data-controller="hello" data-action="click->hello#greet">
    Cliquez-moi !
</button>
```

```
// assets/controllers/hello_controller.js
import { Controller } from '@hotwired/stimulus';

export default class extends Controller {
    greet() {
        alert('Bonjour depuis Symfony UX avec Stimulus !');
    }
}
```

Symfony UX

Les Composants Clés de Symfony UX

B. Turbo

- Turbo (issu du projet Hotwire) remplace la navigation complète des pages par une navigation "turbo", où seules les parties pertinentes de la page sont mises à jour.
- Cela permet des **performances améliorées** et une **expérience utilisateur fluide**, sans nécessiter un framework JavaScript complet comme React ou Vue.js.

Symfony UX

Les Composants Clés de Symfony UX

C. Webpack Encore

Symfony UX repose sur **Webpack Encore** pour gérer les assets (JavaScript, CSS, images, etc.) de manière efficace. Cela inclut la compilation, la minification et la gestion des dépendances npm.

Commandes associées :

```
composer require symfony/webpack-encore-bundle
yarn install
```

Symfony UX

Les Composants Clés de Symfony UX

D. Packages Symfony UX

Symfony UX propose une série de packages préconfigurés pour des fonctionnalités courantes :

- **symfony/ux-chartjs** : Intégration de graphiques avec Chart.js.
- **symfony/ux-twig-component** : Composants Twig dynamiques.
- **symfony/ux-dropzone** : Upload de fichiers avec une interface glisser-déposer.
- **symfony/ux-swup** : Transitions fluides entre les pages.

Exemple d'installation d'un package UX :

```
composer require symfony/ux-chartjs
yarn install
yarn dev
```

Symfony UX

Focus : Turbo UX

- Turbo (anciennement partie de "Hotwire" par Basecamp) est un ensemble d'outils front-end conçu pour construire des applications web modernes avec des interactions rapides et fluides, sans avoir besoin de beaucoup de JavaScript personnalisé.
- Turbo est souvent utilisé dans les frameworks tels que Ruby on Rails ou Symfony pour améliorer l'expérience utilisateur (UX) et réduire la complexité côté client.

Symfony UX

Focus : Turbo UX

1. Introduction à Turbo UX

Turbo est composé de plusieurs modules qui facilitent la création d'interfaces utilisateurs interactives en tirant parti des fonctionnalités natives du navigateur et d'une architecture back-end riche.
Les principaux modules de Turbo sont :

- **Turbo Drive** : Remplace le comportement des liens et des formulaires pour réduire le rechargement complet de la page.
- **Turbo Frames** : Permet de mettre à jour des parties spécifiques d'une page sans rechargement complet.
- **Turbo Streams** : Facilite la mise à jour dynamique des contenus via WebSocket ou AJAX, basé sur des commandes spécifiques.
- **Turbo Native** : Intègre Turbo dans des applications mobiles pour partager le même code entre web et mobile.

Symfony UX

Focus : Turbo UX

2. Turbo Drive

Fonctionnalité :

- Turbo Drive intercepte les clics sur les liens et les soumissions de formulaire pour recharger uniquement la partie pertinente de la page.
- Remplace le comportement classique des requêtes HTTP avec des requêtes AJAX suivies d'un remplacement partiel du DOM.

Principaux avantages :

- **Navigation rapide** : Seules les parties nécessaires de la page sont rechargées.
- **Amélioration de l'UX** : Pas de flashs d'écran, l'état du DOM est préservé (exemple : champ de recherche non réinitialisé).
- **Gestion historique automatique** : Turbo gère l'historique du navigateur automatiquement.

Exemple :

```
<a href="/posts" data-turbo="true">View Posts</a>
```

- Lors du clic sur ce lien, Turbo Drive chargera la page `/posts` sans effectuer de rechargement complet.

Symfony UX

Focus : Turbo UX

3. Turbo Frames

Fonctionnalité :

- Turbo Frames permet de diviser une page en plusieurs "frames" indépendantes.
- Chaque frame peut être mis à jour individuellement sans recharger la page entière.

Principaux avantages :

- **Chargement asynchrone** : Charge les parties nécessaires de la page.
- **Réutilisation de composants** : Rend les sections indépendantes et réutilisables.
- **Simplicité** : Réduit la nécessité d'écrire du JavaScript personnalisé.

Exemple :

```
<turbo-frame id="post-form">
    <form method="post" action="/post/create">
        <input type="text" name="title" placeholder="Title">
        <textarea name="content" placeholder="Content"></textarea>
        <button type="submit">Submit</button>
    </form>
</turbo-frame>
```

- Lorsque vous soumettez ce formulaire, seule la frame `post-form` est mise à jour avec la réponse

Symfony UX

Focus : Turbo UX

4. Turbo Streams

Fonctionnalité :

- Turbo Streams permet de manipuler dynamiquement le DOM en réponse à des actions côté serveur.
- Utilisé avec des actions telles que `append`, `replace`, `remove`, ou `update` pour modifier des éléments spécifiques.

Principaux avantages :

- **Interopérabilité** : Peut être utilisé avec AJAX ou WebSocket.
- **Modifications ciblées** : Effectue des changements précis sur le DOM, améliorant la performance.
- **Optimisé pour les interactions en temps réel**.

Symfony UX

Focus : Turbo UX

4. Turbo Streams

Exemple :

Turbo Stream pour ajouter un nouveau post à une liste :

Côté serveur (Symfony) :

```
return $this->render('post/stream.html.twig', [
    'post' => $newPost,
], new Response('', 200, ['Content-Type' => 'text/vnd.turbo-stream.html']));
```

Fichier Twig (`stream.html.twig`) :

```
<turbo-stream action="append" target="posts">
    <template>
        <li>{{ post.title }}</li>
    </template>
</turbo-stream>
```

Symfony UX

Focus : Turbo UX

5. Turbo Native

Fonctionnalité :

- Permet d'utiliser Turbo dans des applications mobiles (iOS et Android).
- Les applications mobiles peuvent utiliser des vues HTML servies depuis un back-end partagé.

Principaux avantages :

- **Code partagé** : Réutilise les vues HTML du site web dans l'application mobile.
- **Performances natives** : Bénéficie de la rapidité des applications natives.
- **Facilité d'intégration** : Simple à intégrer dans des projets mobiles existants.

Symfony UX

Focus : Turbo UX

6. Comment Turbo améliore l'UX

6.1. Performance :

- Turbo minimise les rechargements complets de page, ce qui réduit le temps de chargement global et améliore la fluidité des interactions.

6.2. Réactivité :

- Avec Turbo Streams, les actions côté serveur (comme la création ou la suppression d'éléments) sont immédiatement reflétées côté client, sans avoir à écrire de JavaScript.

6.3. État et continuité :

- Turbo Drive préserve l'état des formulaires et de la page entre les chargements, évitant des comportements déroutants pour l'utilisateur.

6.4. Simplicité pour les développeurs :

- Turbo réduit la quantité de JavaScript personnalisé nécessaire pour des interactions complexes, permettant aux développeurs de se concentrer sur le back-end.

Symfony UX

Focus : Twig Component et Live Component

- Symfony UX propose des outils pour construire des interfaces utilisateur modernes, réactives et dynamiques tout en restant dans l'environnement PHP avec Twig.
- Les **Twig Components** et **Live Components** sont des éléments clés de cette approche, permettant respectivement de structurer et de dynamiser vos interfaces.

Symfony UX

Focus : Twig Component et Live Component

1. Twig Component

Les **Twig Components** offrent un moyen structuré de créer des éléments réutilisables avec leur propre logique et template.

Caractéristiques principales :

- **Modularité** : Les Twig Components encapsulent la logique et l'affichage dans des modules réutilisables.
- **Interopérabilité** : Ils fonctionnent exclusivement avec Twig, sans nécessiter de connaissances spécifiques en JavaScript.
- **Simplicité** : Parfait pour les interfaces statiques ou légèrement dynamiques.

Symfony UX

Focus : Twig Component et Live Component

Mise en place d'un Twig Component

Installation :

Ajoutez le package requis pour utiliser Twig Component :

```
composer require symfony/ux-twig-component
```

Création d'un composant :

Utilisez la commande pour générer un composant :

```
php bin/console make:twig-component Alert
```

Cela génère deux fichiers :

1. `src/Twig/Component/Alert.php` (logique métier du composant).
2. `templates/components/alert.html.twig` (template associé).

Symfony UX

Focus : Twig Component et Live Component

Exemple de composant simple :

Fichier PHP :

```php
namespace App\Twig\Component;

use Symfony\UX\TwigComponent\Attribute\AsTwigComponent;

#[AsTwigComponent('alert')]
class Alert
{
    public string $type = 'success'; // Type par défaut
    public string $message;          // Message à afficher
}
```

Fichier Twig :

```twig
<div class="alert alert-{{ type }}">
    {{ message }}
</div>
```

Utilisation dans un autre template :

```twig
{{ component('alert', { type: 'warning', message: 'Attention, Twig Components rocks!' }) }}
```

Symfony UX

Focus : Twig Component et Live Component

Syntaxe HTML enrichie avec `twig:`

Avec Symfony 7+, vous pouvez utiliser une syntaxe proche de celle des frameworks front-end :

```
<twig:alert type="info" message="This is a reusable component!" />
```

Symfony UX

Focus : Twig Component et Live Component

Slots dans Twig Components

Les **slots** permettent d'inclure du contenu dynamique ou conditionnel dans un composant.

Exemple avec un slot :

Template avec slot :

```
<div class="alert alert-{{ type }}">
    {{ message }}
    {% if slot('extra') %}
        <div class="extra-content">
            {{ slot('extra') }}
        </div>
    {% endif %}
</div>
```

Utilisation :

```
<twig:alert type="info" message="Here's the message">
    <slot name="extra">
        <p>Contenu additionnel avec un lien : <a href="#">cliquez ici</a></p>
    </slot>
</twig:alert>
```

Symfony UX

Focus : Twig Component et Live Component

2. Live Component

Les **Live Components** enrichissent les Twig Components en leur ajoutant des capacités interactives et dynamiques via AJAX, sans recharger la page. Ils permettent de construire des interfaces réactives en restant dans un environnement Symfony et PHP.

Caractéristiques principales :

- **Réactivité** : Dynamisez les interfaces utilisateur avec des mises à jour côté serveur sans rafraîchissement de page.
- **Interopérabilité JavaScript** : Basé sur Stimulus et Turbo Streams, mais sans écrire de JS complexe.
- **Automatisation** : Synchronisation automatique entre l'état côté serveur et l'interface utilisateur.

Symfony UX

Focus : Twig Component et Live Component

Mise en place d'un Live Component

Installation :

Ajoutez les packages nécessaires :

```
composer require symfony/ux-live-component
yarn add @symfony/ux-live-component
yarn dev
```

Création d'un composant :

```
php bin/console make:live-component Counter
```

Cela génère deux fichiers :

1. `src/Twig/Component/Counter.php` : Classe PHP du composant.
2. `templates/components/counter.html.twig` : Template Twig associé.

Symfony UX

Focus : Twig Component et Live Component

Exemple de Live Component simple :

Classe PHP :

```php
namespace App\Twig\Component;

use Symfony\UX\LiveComponent\Attribute\LiveProp;
use Symfony\UX\LiveComponent\DefaultActionTrait;
use Symfony\UX\LiveComponent\Attribute\AsLiveComponent;

#[AsLiveComponent('counter')]
class Counter
{
    use DefaultActionTrait;

    #[LiveProp]
    public int $count = 0;

    public function increment(): void
    {
        $this->count++;
    }
}
```

Symfony UX

Focus : Twig Component et Live Component

Utilisation dans un template :

```
{{ component('counter') }}
```

Dans cet exemple :

- `#[LiveProp]` permet de synchroniser la variable `$count` entre le front-end et le serveur.
- `wire:click="increment"` déclenche l'exécution de la méthode `increment()` côté serveur.

Symfony UX

Focus : Twig Component et Live Component

Fonctionnalités avancées avec Live Components

1. Props synchronisées avec des formulaires

Les Live Props peuvent être liées à des champs de formulaire pour des mises à jour en temps réel.

Classe PHP :

```
#[LiveProp]
public string $name = '';
```

Template Twig :

```
<input type="text" wire:model="name" />
<p>Bonjour, {{ name }} !</p>
```

Symfony UX

Focus : Twig Component et Live Component

2. Gestion des événements

Vous pouvez écouter ou déclencher des événements depuis un Live Component.

Classe PHP :

```php
public function resetCount(): void
{
    $this->count = 0;
}
```

Template Twig :

```twig
<button wire:click="resetCount">Réinitialiser</button>
```

Symfony UX

Focus : Twig Component et Live Component

Twig Components vs Live Components

Aspect	Twig Component	Live Component
Usage principal	Composants statiques ou légèrement dynamiques	Interfaces interactives et réactives
Interopérabilité JS	Pas nécessaire	Nécessite Stimulus et Turbo
Rafraîchissement DOM	Complet	Partiel grâce à Turbo Streams
Synchronisation	Non (données passées uniquement en paramètres)	Oui, entre le front-end et le serveur
Installation	Simplicité	Plus complexe

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Workflow

Workflow

Introduction

- Le composant **Workflow** de Symfony permet de modéliser des processus métiers complexes en définissant des **états**, des **transitions**, et des **logiques conditionnelles**.
- Il est particulièrement utile pour gérer des flux tels que des systèmes de validation, des pipelines de traitement ou des étapes de processus (ex. : commandes, publications d'articles, etc.).

Workflow

Introduction

1. Concepts Clés

1. **Sujet (Subject)** : L'objet sur lequel le workflow s'applique.
2. **Places** : Les différents **états** dans lesquels un sujet peut se trouver.
3. **Transitions** : Les **actions** ou **changements** qui permettent de passer d'un état à un autre.
4. **Marquage (Marking)** : Représentation de l'état actuel du sujet dans le workflow.
5. **Guards (Gardiens)** : Des conditions qui doivent être remplies pour qu'une transition soit possible.
6. **Événements** : Des hooks permettant d'exécuter des actions pendant une transition.

Workflow

Introduction

2. Installation et Configuration

Installation

Ajoutez le composant Workflow à votre projet :

```
composer require symfony/workflow
```

Workflow

Introduction

Configuration

Dans le fichier `config/packages/workflow.yaml`, configurez votre workflow. Voici un exemple de workflow pour un article de blog :

```yaml
framework:
    workflows:
        article_workflow: # Nom du workflow
            type: 'state_machine' # Peut être 'workflow' ou 'state_machine'
            marking_store:
                type: 'single_state' # Permet un seul état à la fois
            supports:
                - App\Entity\Article # Classe sur laquelle le workflow s'applique
            places: # États possibles
                - draft
                - review
                - published
            transitions: # Définition des transitions
                to_review:
                    from: draft
                    to: review
                publish:
                    from: review
                    to: published
```

Workflow

Introduction

3. Utilisation dans le Code

Définir une entité

Créez une entité qui représentera le sujet du workflow.

```php
namespace App\Entity;

class Article
{
    private string $currentState = 'draft'; // État initial

    public function getCurrentState(): string
    {
        return $this->currentState;
    }

    public function setCurrentState(string $state): void
    {
        $this->currentState = $state;
    }
}
```

Workflow

Introduction

3. Utilisation dans le Code

Injecter le service Workflow

```
class ArticleController extends AbstractController
{
    public function index(WorkflowInterface $articleWorkflow): Response
    {
        $article = new Article();

        // Vérifier les transitions possibles
        $transitions = $articleWorkflow->getEnabledTransitions($article);
        foreach ($transitions as $transition) {
            echo $transition->getName(); // Affiche "to_review"
        }

        // Appliquer une transition
        if ($articleWorkflow->can($article, 'to_review')) {
            $articleWorkflow->apply($article, 'to_review');
        }

        return new Response('Workflow exécuté');
    }
}
```

Workflow

Introduction

4. Types de Workflows

1. **Workflow** :
   - Permet à un sujet d'être dans plusieurs états simultanément.
   - Exemple : Un article peut être en cours de rédaction et en cours de révision.
2. **State Machine** :
   - Le sujet ne peut être que dans un seul état à la fois.
   - Exemple : Un système de validation d'une commande (créée, en attente de validation, expédiée).

Workflow

Introduction

5. Guards (Conditions)

Les guards permettent de définir des conditions pour qu'une transition soit possible.

Exemple avec des guards :

```
framework:
    workflows:
        article_workflow:
            transitions:
                to_review:
                    from: draft
                    to: review
                    guard: "is_granted('ROLE_EDITOR')"
```

Dans cet exemple, la transition `to_review` ne peut être effectuée que par un utilisateur ayant le rôle `ROLE_EDITOR`.

Workflow

Introduction

6. Événements

Exemple d'écouteur d'événements :

```php
class WorkflowSubscriber implements EventSubscriberInterface
{
    public static function getSubscribedEvents(): array
    {
        return [
            'workflow.article_workflow.guard.to_review' => 'onGuardToReview',
            'workflow.article_workflow.completed.to_review' => 'onCompletedToReview',
        ];
    }

    public function onGuardToReview(GuardEvent $event): void
    {
        // Bloquer la transition si une condition n'est pas remplie
        $subject = $event->getSubject(); // L'objet lié au workflow
        if ($subject instanceof Article && $subject->getCurrentState() !== 'draft') {
            $event->setBlocked(true);
        }
    }
```

Workflow

Introduction

7. Visualisation du Workflow

Symfony offre une intégration avec Graphviz pour visualiser votre workflow.

Installation de Graphviz :

```
sudo apt install graphviz # Linux
brew install graphviz     # macOS
choco install graphviz    # Windows
```

Générer un graphe :

Ajoutez ceci à un contrôleur ou une commande :

```
use Symfony\Component\Workflow\Dumper\GraphvizDumper;

$dumper = new GraphvizDumper();
echo $dumper->dump($workflow->getDefinition());
```

Ensuite, utilisez Graphviz pour convertir ce graphe en image :

```
php bin/console workflow:dump article_workflow | dot -Tpng -o workflow.png
```

Workflow

Introduction

8. Bonnes Pratiques

1. **Modularité** : Gardez vos workflows simples et spécifiques.
2. **Tests** : Testez les conditions et transitions pour éviter des bugs.
3. **Documentation** : Documentez vos workflows pour les rendre compréhensibles.
4. **Événements** : Utilisez les événements pour des tâches comme la journalisation ou les notifications.