Symfony

m2iformation.fr

Sommaire

- Fondamentaux de Symfony, Injection de Dépendance et Inversion de Contrôle
  - Présentation de Symfony et de son écosystème.
  - Installation de Symfony et configuration de l'environnement de développement.
  - Structure d'un projet Symfony et philosophie.
  - Injection de dépendance et inversion de contrôle.
  - Création de routes et de contrôleurs : gestion des requêtes HTTP.
  - Introduction à Twig et création de templates.
  - Services et conteneur de services : définition et utilisation basique dans les contrôleurs.
  - Exercices pratiques.

Présentation de Symfony et de son écosystème

- Symfony est un framework PHP pour le développement d'applications web.
- Symfony vise à accélérer la création et la maintenance de vos applications web en remplaçant les tâches répétitives de codage par des solutions rapides, fiables et adaptées.

1. Composants de Symfony : Symfony est construit autour d'une série de composants PHP réutilisables, qui peuvent être utilisés dans n'importe quelle application PHP, pas seulement celles construites avec Symfony.
2. Bundle: Les bundles sont similaires aux plugins dans d'autres logiciels. Ils fournissent une façon de regrouper des fonctionnalités liées, qui peuvent ensuite être réutilisées dans différents projets.
3. Architecture: Symfony suit le modèle architectural MVC (Modèle-Vue-Contrôleur), facilitant ainsi l'organisation du code de votre application.
4. Flexibilité: Symfony est extrêmement flexible. Grâce à son système de bundles et à sa conception modulaire, vous pouvez utiliser autant ou aussi peu de fonctionnalités Symfony que vous le souhaitez.

Installation de Symfony et configuration de l'environnement de développement

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
```

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 --full
```

Installation de Symfony et configuration de l'environnement de développement

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

- `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.
- `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).

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.

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 (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.

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.

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.

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

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
    }
}
```

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. Voici un exemple simple d'un contrôleur répondant à une route `/conference` :

```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.

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

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.

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.

Création de Templates avec Twig dans Symfony 6

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]);
```

Exemple de Template Twig

Imaginons que vous avez un tableau de produits (`products`) que vous souhaitez afficher dans une page web. Chaque produit est représenté par un tableau associatif contenant le nom du produit (`name`), son prix (`price`), et une indication de disponibilité en stock (`inStock`).

```
{# 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>
```

Explications :

- **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.

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.

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.

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.

Template de base (`base.html.twig`) :

```
<!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>
```

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 : 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.

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.

Utilisation Basique dans les Contrôleurs

Dans Symfony, les contrôleurs sont souvent l'endroit où vous interagissez avec différents services pour traiter les requêtes HTTP.

Injection de Dépendances

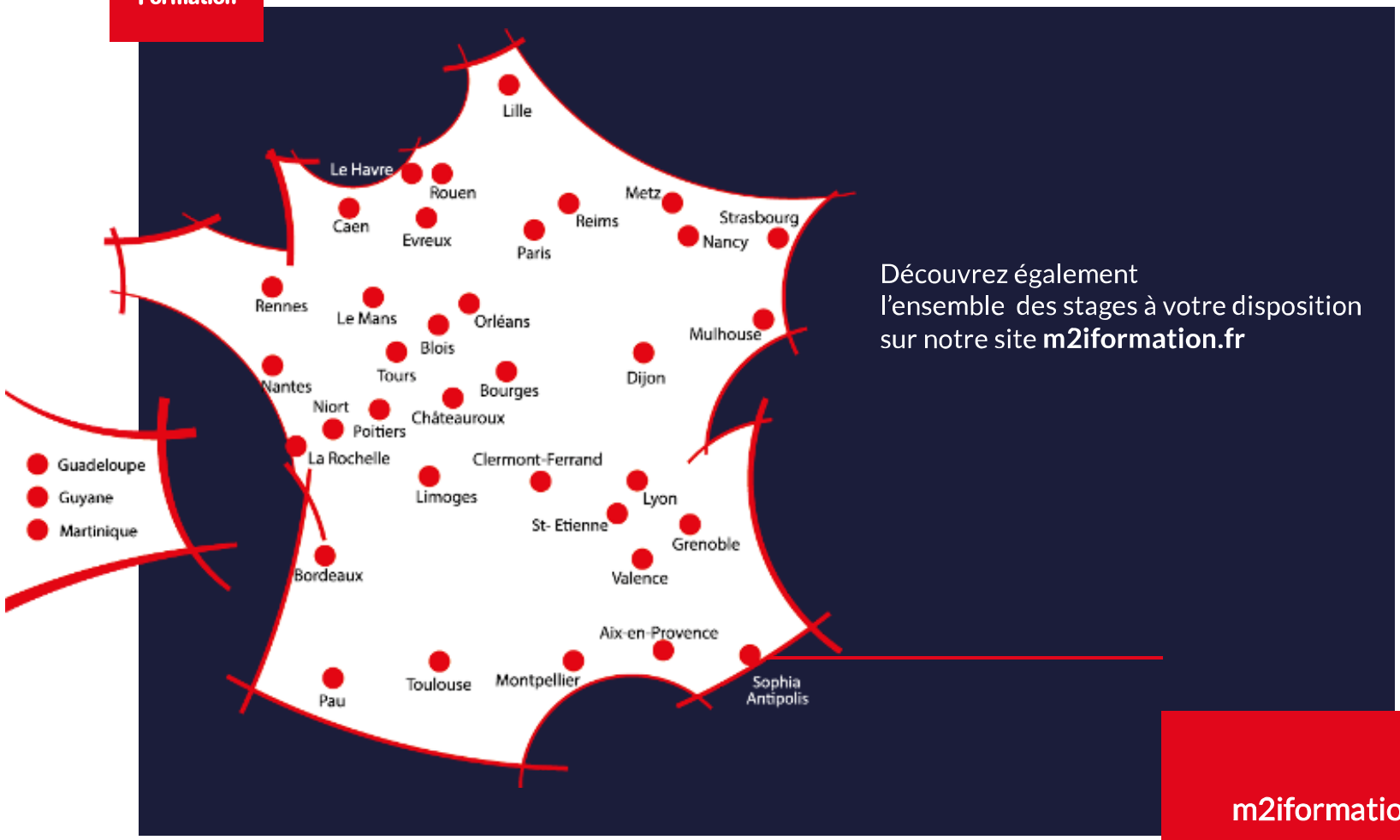
La manière recommandée d'utiliser un service dans un contrôleur est par l'injection de dépendances, soit via le constructeur, soit via l'injection dans les méthodes d'action.

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();

        // Utilisez $resultat pour quelque chose, par exemple le renvoyer dans une vue
    }
}
```

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
    }
}
```



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr