INFORMATIQUE
TECHNIQUE

Expert

Symfony 4 (LTS)

Développez des sites web PHP structurés et performants

En téléchargement

code source

+ QUIZ

Version en ligne
OFFERTE !
pendant 1 an

Bilal AMARNI
Etienne LANGLET

eni

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence ENI de l'ouvrage **EI4SYM** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Chapitre 1
Introduction

Chapitre 2
Mise en place d'un projet Symfony

Chapitre 3
Architecture du framework

Chapitre 4
Routage et contrôleur

Chapitre 5
L'injection de dépendances

Chapitre 6
Les templates avec Twig

Chapitre 7

Accéder aux bases de données avec Doctrine

Chapitre 8
La gestion des événements applicatifs

Chapitre 9
Les formulaires

Chapitre 10
La sécurité dans une application Symfony

Chapitre 11
Tester son application Symfony

Chapitre 12

Journalisation et surveillance avec Symfony

Chapitre 13
Amélioration des performances

Chapitre 14
Internationalisation des applications Symfony

Chapitre 15
Développer une API REST avec Symfony

Annexes

Chapitre 6
Les templates avec Twig

1. Présentation et concepts

1.1 Le concept de Templating

La notion de template (ou modèle) fait référence au principe d'élaboration de modèles statiques de pages HTML dans lesquels viendront s'insérer des données dynamiques résultant de l'exécution du code PHP.

Le templating est l'approche générale de découpage entre ces données statiques et dynamiques, toujours dans l'optique de séparer les responsabilités : les traitements applicatifs d'un côté, la présentation des données issues de ces traitements de l'autre.

1.2 Templating et modèle MVC

Dans la mise en œuvre du modèle MVC (cf. Architecture du framework - Le modèle de conception MVC), les templates endossent la responsabilité de l'affichage des données, c'est-à-dire : la vue. Ou plus précisément « les vues » dans la mesure où chaque action implémentée dans les contrôleurs a la responsabilité de prévoir un affichage spécifique en invoquant le template associé.

Chaque template est donc responsable d'une génération de contenu HTML à destination du navigateur web. Ce contenu HTML est produit en combinant le code statique, utilisé pour la mise en forme générale des pages, avec le code dynamique permettant d'intégrer les données, bien que ce dernier soit simplement limité à l'expression de variables contenant ces données. En effet, il est hors de question de trouver du code PHP exécutant une requête SQL dans une vue !

2. Twig

2.1 Présentation

Twig est un moteur de templates, une librairie permettant de gérer la couche « présentation », pour les applications utilisant le modèle de conception MVC.

Globalement, le principe est d'extraire tout ce qui est relatif à la vue dans des fichiers dédiés, appelés « *templates* », qui représentent pour la plupart du HTML.

Nous insistons ici sur le terme « représenter ». En effet, les templates ne contiennent pas forcément le code HTML tel qu'il sera retourné par l'application, ils contiennent bien souvent du code permettant de générer ce code HTML. Ce code utilise un langage spécifique au moteur de template et ce langage est communément appelé, par extension, du « Twig ».

Twig est un projet qui voit le jour en 2008. Son auteur Armin Ronacher s'inspire très largement du framework Jinja, un moteur de template pour Python. Aujourd'hui, le projet Twig est maintenu par les équipes de développement de Symfony dont il est le moteur de template par défaut. Le site officiel de Twig est accessible à l'adresse https://twig.symfony.com.

2.2 Pourquoi un nouveau langage ?

Une première question, tout à fait légitime, peut vous venir à l'esprit suite à la description que nous venons d'effectuer : pourquoi utiliser Twig et non PHP ?

On pourrait très bien imaginer une utilisation correcte du modèle de conception MVC, avec extraction de la couche Vue au sein de fichiers dédiés (*templates*) et utilisant le langage PHP.

Sachez que Symfony permet cela (même si c'est Twig qui est configuré par défaut). Néanmoins, nous n'aborderons pas cette solution au cours de ce chapitre ; nous nous concentrerons uniquement sur Twig, et ce pour plusieurs raisons :

– Twig est rapide. Bien qu'il utilise son propre langage, et par conséquent son propre moteur de parsing (car, au final, c'est du code PHP qui doit être généré). Le code PHP généré par chaque template est mis en cache. Ce processus n'a donc pas lieu à chaque requête. L'impact sur les performances, par rapport à du PHP pur, est donc *minime*.
– Twig est sécurisé. Les données affichées sont immunisées par défaut contre les éventuelles attaques malveillantes, comme le cross-site scripting (XSS) par exemple.
– Twig est adapté aux templates. Ce langage a été spécialement conçu dans cette optique et nous verrons tout au long de ce chapitre qu'il nous offre une multitude de raccourcis (qui sont impossibles en PHP) nous permettant de garder nos templates lisibles et simples. Si vous avez déjà eu l'occasion d'utiliser le framework Django (pour le langage Python), vous ne serez pas dépaysé : la syntaxe de Twig est très semblable à celle utilisée dans les templates sous Django.
– Twig est largement utilisé car il est le moteur de template par défaut de Symfony. La plupart des projets l'utilisent et beaucoup de bundles open source supportent uniquement Twig, ce qui est également le cas des tutoriels ou articles qu'on peut trouver sur Internet. Ils sont très souvent orientés sur ce moteur de template et non PHP.

2.3 Mise en pratique

Voyons comment créer concrètement cette couche Vue au sein de notre application. Nous connaissons pour l'instant le processus entre une requête et une réponse (cf. Architecture du framework et Routage et contrôleur). Nous savons que le contrôleur est chargé de retourner une réponse HTTP, sous la forme d'un objet de type **Symfony\Component\HttpFoundation\Response** :

```
namespace App\Controller;

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

class DefaultController extends AbstractController
{
   /**
    * @Route("/")
    */
   public function hello()
   {
       return new Response('Hello world!');
   }
}
```

Ici, l'application retourne une réponse ayant comme contenu **Hello world!** (la vue), qui est directement générée depuis le contrôleur. Dans ce cas précis, cela n'est pas particulièrement choquant, mais imaginez une page web, avec ses nombreuses balises... le code du contrôleur deviendrait rapidement illisible.

Déléguons la génération du contenu de la réponse à Twig.

Contrôleur

```
namespace App\Controller;

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

class DefaultController extends AbstractController
{
```

```
    /**
     * @Route("/")
     */
    public function hello()
    {
        return $this->render(
            'default/hello.html.twig'
        );
    }
}
```

Template

```
{# templates/default/hello.html.twig #}
Hello world!
```

Remarque

{# templates/default/hello.html.twig #} *n'est pas obligatoire car pour Twig, c'est un commentaire. Ici, nous l'utilisons uniquement dans le but d'aider le lecteur, en indiquant le chemin vers le template.*

2.4 Remarques sur l'utilisation

Nous avons vu au cours des chapitres précédents (Architecture du framework et L'injection de dépendances) que, pour Symfony, tous les outils/fonctionnalités étaient regroupés dans des « services ». Twig n'échappe pas à cette règle.

Effectivement, nous invoquons la méthode **render** de la classe **Symfony\Bundle\FrameworkBundle\Controller\AbstractController**. En allant voir son contenu, on remarque qu'elle fait bel et bien appel à un service (dont l'identifiant est « templating »).

Cela rejoint nos explications précédentes (cf. Architecture du framework - Architecture de Symfony, se référer notamment au schéma descriptif) ; la **vue** est tout simplement un service, sur lequel nous pouvons nous appuyer pour générer des réponses, au même titre que n'importe quel autre service. Ce n'est pas un service spécial ou particulier.

2.5 La notation des templates

De la même manière que le composant Router utilise une notation particulière pour référencer une action, chaque action utilisant un template doit le référencer selon une certaine convention.

Les templates sont enregistrés dans le répertoire **templates/** de l'application. Ce répertoire contient généralement un sous-répertoire pour chaque contrôleur contenant les templates spécifiques à ce contrôleur. L'emplacement de ce répertoire est défini dans la configuration de Twig qui se trouve dans le fichier **config/packages/twig.yaml** ; en voici le contenu par défaut :

```
twig:
    default_path: '%kernel.project_dir%/templates'
    debug: '%kernel.debug%'
    strict_variables: '%kernel.debug%'
    exception_controller: null
```

Nous pouvons notamment y constater la définition de la variable **default_path** qui pointe vers ce répertoire **templates/**.

Remarque

%kernel.project_dir% *est une variable représentant le répertoire racine du projet. Elle est utilisée dans de nombreux fichiers de configuration Symfony.*

La notation pour les templates est la suivante : **chemin/vers/template** ou le chemin est exprimé de manière relative à partir du répertoire **templates/**. Se présentent alors deux possibilités de localisation des templates :

– Les templates directement localisés dans le répertoire **templates/**, qui seront des templates généraux pour l'application ou bien des gabarits de page (cette notion est évoquée un peu plus loin dans ce chapitre). En utilisant par exemple **base.html.twig** dans un contrôleur, on référence le fichier **templates/base.html.twig**.

– Les templates associés à une action de contrôleur qui sont, eux, rangés dans un dossier portant le nom du contrôleur. Ainsi, conventionnellement, l'action **afficher()** du contrôleur **AdminController** sera associée au template présent dans **templates/admin/afficher.html.twig**, son appel dans le contrôleur se faisant avec le nom **admin/afficher.html.twig**.

2.6 Extension du système de templates

En plus du répertoire conventionnel **templates/**, il est possible d'ajouter d'autres dossiers contenant des templates afin de mieux organiser son code. Pour cela il faut intervenir dans la configuration de Twig stockée dans le fichier **config/packages/twig.yaml**. La propriété **paths** de ce fichier permet de référencer plusieurs répertoires supplémentaires et de les associer à un espace de noms afin d'éviter tout conflit de nommage. Prenons l'exemple de la configuration suivante ajoutée au fichier **config/packages/twig.yaml** :

```
twig:
   # ...
   paths:
       'admin/templates': 'admin'
       'backend/templates': 'back'
```

Les répertoires **admin/templates** et **backend/templates**, tous deux situés à la racine du projet, sont ajoutés au système de localisation des templates par Symfony et possèdent respectivement les espaces de noms **admin** et **back**. Les fichiers de ces répertoires seront identifiés par la syntaxe **@EspaceDeNom/nom_du_fichier.html.twig**.

Avec cet ajout, pour référencer le fichier **index.html.twig** situé dans **admin/templates**, on utilisera **@admin/index.html.twig**.

2.7 L'annotation @Template

Le bundle SensioFrameworkExtraBundle intégré par défaut, offre un raccourci intéressant pour l'affichage de vos templates. Plutôt que d'invoquer la méthode **render()** du contrôleur comme nous venons de le voir, vous pouvez tout simplement apposer une annotation sur votre action :

```
namespace App\Controller;

use
Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Template;
```