symblog
creating a blog in Symfony2

[Partie 1] - Configuration de Symfony2 et utilisation des templates

> Je propose également des formations en petits groupes sur 2 à 3 jours, plus d'infos sur la page dédiée. N'hésitez pas à me contacter (06.62.28.01.87 ou clement [@] keiruaprod.fr) pour en discuter !

Introduction

Ce chapitre va couvrir les premières étapes de la création d'un site web avec Symfony2. Nous allons télécharger et configurer la distribution standard de Symfony2, créer un bundle pour le Blog et construire le template HTML principal. A la fin de ce chapitre, vous aurez configuré un site avec Symfony2 qui sera disponible à une adresse locale, par exemple `http://symblog.dev/`. Le site web va contenir la structure HTML principale du blog, ainsi que du contenu factice.

Les thèmes suivants vont être abordés au cours de ce chapitre:

1. Mise en place d'une application Symfony2
2. Configuration d'un domaine de développement
3. Les Bundles Symfony2
4. Les routes
5. Les contrôleurs
6. Les templates avec Twig

Téléchargement et installation

Comme établi ci-dessus, nous allons utiliser la distribution standard de Symfony2 (Symfony2 Standard Distribution). Cette distribution est livrée avec les librairies du moteur de Symfony2, ainsi qu'avec les principaux bundles nécessaires au développement d'un site web. Vous pouvez télécharger l'archive correspondante sur le site de Symfony2. Comme je ne souhaite pas répéter le contenu de l'excellente documentation proposée dans le livre sur Symfony2, vous pouvez consulter le chapitre Installation et configuration de Symfony2 pour des besoins spécifiques. Cela vous permettra de choisir quelle archive choisir, comment installer les 'vendors' requis, et enfin comment assigner les permissions aux répertoires qui le nécessitent.

> **Attention**
>
> Il est important d'accorder un soin particulier à la section Mise en place des permissions du chapitre d'installation. Cela détaille les diverses manières d'accorder des permissions aux répertoires `app/cache` et `app/logs` afin que les utilisateurs du serveur web aient les droits d'accès en écriture.

Création d'un domaine de développement

Pour ce tutorial, nous allons utiliser le domaine local `http://symblog.dev/`, néanmoins vous pouvez utiliser celui de votre choix. Ces instructions sont spécifiques à Apache et supposent qu'il est déjà installé et fonctionne sur votre machine. Si vous êtes déjà à l'aise avec la mise en place de domaines locaux ou bien si vous utilisez un serveur web différent tel que nginx vous pouvez sauter cette section.

> **Note**
>
> Ces étapes ont été réalisées sur la distribution Linux Fedora. Il est donc possible que les chemins d'accès puissent différer si vous utilisez un autre système d'exploitation.

Commençons par créer un hôte virtuel avec Apache. Dans le fichier de configuration d'Apache, ajoutez les paramètres suivants en prenant bien soin de changer les chemins de `DocumentRoot` et `Directory` correctement. Le chemin et le nom de votre fichier de configuration Apache peut fortement varier selon votre système d'exploitation. Avec Fedora, il est situé dans `/etc/httpd/conf/httpd.conf`. Vous devrez éditer ce fichier avec les privilèges `sudo` .

```
# /etc/httpd/conf/httpd.conf

NameVirtualHost 127.0.0.1

<VirtualHost 127.0.0.1>
  ServerName symblog.dev
  DocumentRoot "/var/www/html/symblog.dev/web"
  DirectoryIndex app.php
```

```
  <Directory "/var/www/html/symblog.dev/web">
    AllowOverride All
    Allow from All
  </Directory>
</VirtualHost>
```

Ensuite ajoutez un nouveau domaine à la fin du fichier hôte situé dans `/etc/hosts`. A nouveau, vous devrez éditer ce fichier avec les privilèges `sudo` .

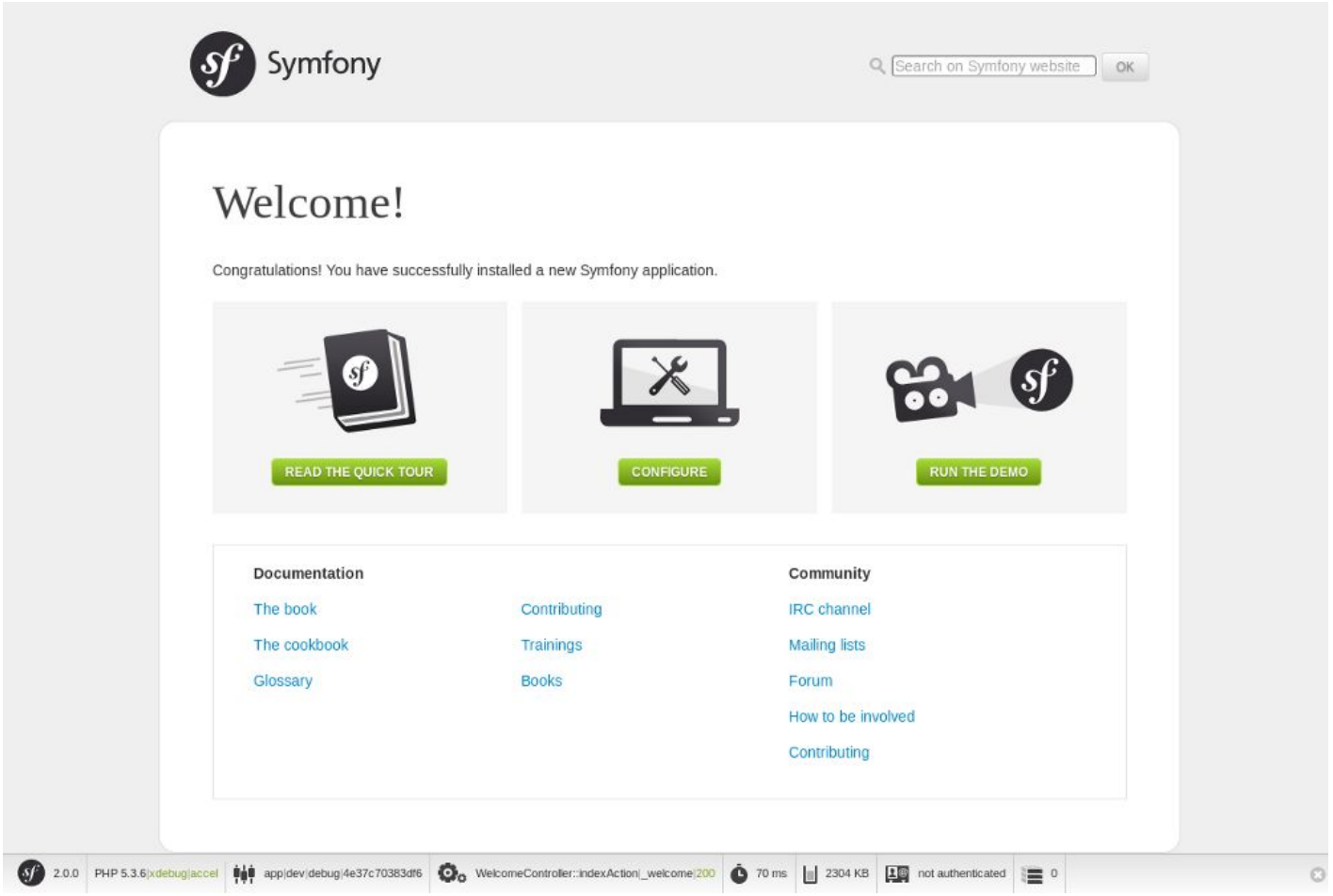
```
# /etc/hosts
127.0.0.1     symblog.dev
```

Finalement, n'oubliez pas de redémarrer le service Apache, afin de mettre à jour les paramètres de configuration avec les informations que nous venons de spécifier.

```
$ sudo service httpd restart
```

> **Tip**
>
> Si vous passez votre temps à créer des domaines virtuels, vous pouvez simplifier ce processus en utilisant des hôtes virtuels dynamiques.

Vous devriez désormais pouvoir visiter l'url `http://symblog.dev/app_dev.php/`.



S'il s'agit de votre première visite sur la page d'accueil de Symfony2, prenez un peu de temps pour regarder les pages de démonstration. Sur chacune d'entre elles vous pourrez trouver les bouts de code utilisés sous le capot.

> **Note**
>
> Vous pouvez également remarquer une barre d'outils au bas de l'écran d'accueil. Il s'agit de la barre d'outils pour les développeurs, qui fournit des informations précieuses sur l'état de l'application. Parmi ces informations, vous pourrez trouver le temps d'exécution de la page, l'utilisation mémoire, les requêtes effectuées dans la base de données, l'état d'authentification, et beaucoup plus. Par défaut, la barre d'outils est seulement visible dans l'environnement `dev`, car fournir la barre d'outils dans l'environnement de production serait une grosse faille de sécurité: celà exposerait beaucoup d'informations sur le fonctionnement interne de l'application. Des références à cette barre d'outils seront faites au cours de ce tutoriel afin de vous apprendre à l'utiliser.

Configurer Symfony2 : l'interface web

Symfony2 propose une interface web pour configurer divers aspects du site web tels que les paramètres de la base de données. Nous avons besoin d'une base de données pour ce projet, commençons par utiliser l'outil de configuration.

Rendez-vous à l'adresse `http://symblog.dev/app_dev.php/` et cliquez sur le bouton Configure. Entrez les détails pour paramétrer l'usage de votre base de données avec Symfony2 (ce tutorial suppose l'utilisation de MySQL, mais vous pouvez choisir n'importe quelle base de données à laquelle vous avez accès). Sur la page suivante, poursuivez par la génération d'une clé CSRF. Vous seront ensuite présentés les paramètres de Symfony2 que l'application a généré pour vous. Attention à la remarque figurant sur cette page, il y a des chances que votre fichier `app/paramaters.ini` ne soit pas accessible en écriture et que vous deviez copier/coller les paramètres dans ce fichier (Ces paramètres peuvent alors remplacer ceux déjà existants).

Les bundles : Les briques élémentaires de Symfony2

les bundles sont les blocs de construction élémentaires de n'importe quelle application Symfony2, en fait le framework Symfony2 est lui même un bundle. Les bundles permettent de séparer le code en briques fonctionnelles et réutilisables. Ils encapsulent le fonctionnement des diverses composantes telles que les contrôleurs, le modèle, les templates ainsi que les diverses ressources, aussi bien images que CSS. Nous allons créer un bundle pour notre site web dans l'espace de nom (namespace) Blogger. Si vous n'êtes pas familier avec les espaces de nom en PHP vous devriez passer un peu de temps sur le sujet, car ils sont très largement utilisés. Dans Symfony2, toute portion de code se trouve à l'intérieur d'un espace de nom. Regardez le chargement automatique des classes Symfony2 pour explorer en détail le chargement automatique des classes dans Symfony2.

> **Tip**
>
> Une bonne compréhension des espaces de nom peut éliminer les problèmes usuels auxquels vous pourriez faire face quand les structures de répertoires ne reflètent pas correctement les structures d'espaces de noms.

Création du bundle.

Pour encapsuler les fonctionnalités utilisées par le blog, nous allons créer le bundle du blog, qui va stocker tous les fichiers requis et pourrait alors être déposé directement dans n'importe quelle autre application Symfony2. Symfony2 propose un large éventail de tâches pour nous assister dans la réalisation d'opérations courantes. Une de ces opérations courantes est la création d'un bundle.

Pour démarrer le générateur de bundle, utilisez la commande suivante. Des informations permettant de configurer le bundle vous seront demandées. Sélectionner à chaque fois la proposition par défaut.

```
$ php app/console generate:bundle --namespace=Blogger/BlogBundle --format=yml
```

Une fois le travail du générateur achevé, Symfony2 aura agencé pour vous les divers éléments de base de la structure du bundle. Un certain nombre de changements sont à noter ici.

> **Tip**
>
> Vous n'êtes pas obliger d'utiliser les générateurs que Symfony2 propose, ils sont seulement là pour vous guider. Vous auriez pu créer manuellement la structure de répertoires et de fichiers du bundle. Bien qu'il n'est pas obligatoire d'utiliser les générateurs, ils ont pour avantage d'être rapide à utiliser et réalisent toutes les tâches requises pour qu'un bundle puisse fonctionner, par exemple l'enregistrement du bundle.

L'enregistrement du bundle

Notre nouveau bundle `BloggerBlogBundle` a été enregistré dans le noyau (Kernel) situé dans `app/AppKernel.php`. Symfony2 a besoin que nous enregistrions tous les bundles utilisés par l'application. Bous pourrez également remarquer que certains bundles sont seulement enregistrés dans les environnements `dev` ou `test`. Charger ces bundles dans l'environnement de production `prod` apporterait des calculs supplémentaires pour des fonctionnalités qui ne seraient pas nécessaires. Le code suivant montre comment notre `BloggerBlogBundle` a été enregistré

```
// app/AppKernel.php
class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
        // ..
            new Blogger\BlogBundle\BloggerBlogBundle(),
        );
        // ..

        return $bundles;
    }

    // ..
}
```

Les routes

Le routage du bundle a été importé dans le principal fichier de routage de l'application, situé dans `app/config/routing.yml`.

```
# app/config/routing.yml
BloggerBlogBundle:
    resource: "@BloggerBlogBundle/Resources/config/routing.yml"
    prefix:   /
```

L'option `prefix` nous permet d'associer le routage entier du `BloggerBlogBundle` avec un préfixe. Dans notre cas, nous avons opté pour utiliser le chemin par défaut, qui est `/`. Si par exemple vous vouliez que toutes les routes soient préfixées par `/blogger`, vous pouvez changer le préfixe pour `prefix: /blogger`.

Structure par défaut

L'architecture de répertoire par défaut du bundle a été créé dans le répertoirre `src`. Cela commence par le répertoire `Blogger` qui est associée à l'espace de nom `Blogger` dans lequel nous avons crée notre bundle. Dans ce répertoire se trouve le répertoire `BlogBundle` qui contient le bundle. Le contenu de ce dossier va être détaillé à mesure que nous avancerons dans ce tutoriel. Si vous êtes familier avec l'architecture MVC, certain des noms de répertoires doivent parler d'eux même.

Le contrôleur par défaut

Grâce au générateur de bundle, Symfony2 a créé pour nous un contrôleur par défaut. Nous pouvons utiliser ce contrôleur en allant à l'adresse `http://symblog.dev/app_dev.php/hello/symblog`. Vous devriez voir une page très simple. Essayez de remplacer le `symblog` à la fin de l'adresse par votre nom. Nous allons examiner comment cette page a été générée.

Routage

Le fichier de routage du `BloggerBlogBundle` situé dans `src/Blogger/BlogBundle/Resources/config/routing.yml` contient les règles de routage par défaut

```
# src/Blogger/BlogBundle/Resources/config/routing.yml
BloggerBlogBundle_homepage:
    pattern:  /hello/{name}
    defaults: { _controller: BloggerBlogBundle:Default:index }
```

Le routage est composé d'un motif et de paramètres par défaut. Le motif est comparé à l'URL, les paramètres désignent quel contrôleur exécuter lorsque la route est éligible. Dans le motif `/hello/{name}`, le substitut `{name}` va correspondre à n'importe quelle type de valeur car rien de spécifique n'a été précisé. Cette route ne précise également aucune culture, format ou méthode HTTP. Comme aucune méthode HTTP n'est précisée, les requêtes de type GET, POST, PUT ou autre sont éligibiles lors de la comparaison du motif.

Si une adresse valide tous les critères précisés par une route, alors elle sera exécutée par le contrôleur décrit dans l'option _controller. Cette option contient le nom logique du contrôleur qui permet à Symfony2 de l'associer à un fichier spécifique. L'exemple ci-dessus va conduire à l'exécution de l'action `index` du contrôleur `Default` situé dans le fichier `src/Blogger/BlogBundle/Controller/DefaultController.php`.

Le contrôleur

Le contrôleur dans cet exemple est très simple. La classe``DefaultController`` étend la classe `Controller` qui propose des méthodes utiles telles que la méthode `render` utilisée ci dessous. Comme notre route définit un substitut, il est passé comme argument à notre action sous le nom `$name`. L'action ne fait rien de plus qu'appeler la méthode `render` en lui précisant d'utiliser le fichier template `index.html.twig` situé dans le dossier de vues (Views/) du contrôleur Default de `BloggerBlogBundle` pour l'affichage. Le format du nom de template est `bundle:controlleur:template`. Dans notre cas il s'agit de `BloggerBlogBundle:Default:index.html.twig`, qui associe le template `index.html.twig`, dans le fichier de vues `Default` du `BloggerBlogBundle`, ou physiquement au fichier `src/Blogger/BlogBundle/Resources/views/Default/index.html.twig`. Un tel format de nommage pour les template permet de référer à des bundles depuis n'importe où dans l'application, ou même dans un autre bundle. Nous verrons cela plus tard dans le chapitre.

Nous passons également la variable `$name` au template via le paramètre `array` fourni à la méthode `render`.

```
<?php
// src/Blogger/BlogBundle/Controller/DefaultController.php

namespace Blogger\BlogBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class DefaultController extends Controller
{
    public function indexAction($name)
    {
        return $this->render('BloggerBlogBundle:Default:index.html.twig', array('name' => $name));
    }
}
```

Le template (la vue)

Comme vous pouvez le voir, le template est très simple. Il affiche Hello, suivi de l'argument `name` qui a été passé en paramètre par le contrôleur.

```
{# src/Blogger/BlogBundle/Resources/views/Default/index.html.twig #}
Hello {{ name }}!
```

Nettoyage

Comme nous n'avons pas besoin de certains des fichiers par défaut créés par le générateur, nous allons faire un peu de nettoyage.

Le fichier du contrôleur `src/Blogger/BlogBundle/Controller/DefaultController.php` peut être supprimé, ainsi que le répertoire pour la vue et son contenu `src/Blogger/BlogBundle/Resources/views/Default/`. Finalement, supprimez la route définie dans `src/Blogger/BlogBundle/Resources/config/routing.yml`

Template

Il y a 2 options par défaut pour les templates lorsque l'on utilise Symfony2; Twig et PHP. Vous pouvez bien sûr n'utiliser ni l'un ni l'autre et opter pour une autre librairie. C'est possible grâce au container d'injection de dépendances. Nous allons utiliser Twig comme moteur de template pour un certain nombre de raisons.

1. Twig est rapide - Les templates twig sont compilés en classes PHP, il y a donc très peu de surcharge lors de l'utilisation des templates Twig.
2. Twig est concis - Twig nous permet de réaliser les fonctionnalités liées au templates en très peu de code. C'est à comparer avec le PHP, qui peut parfois s'avérer très verbeux.
3. Twig supportes l'héritage de template - Il s'agit d'une de mes préférées. Les templates ont la capacité d'étendre et surcharger d'autres templates, ce qui permet aux templates enfants de remplacer ce qui été proposé par défaut par les parents.
4. Twig est sûr - Twig échappe par défaut ce qu'il affiche, et propose même un environnement de type `bac à sable` pour les templates importés.
5. Twig est extensible - Twig propose de base un certain nombre de fonctionnalités récurrentes que vous êtes en droit d'attendre d'un moteur de template, mais pour les situations ou vous pourriez avoir des besoins spécifiques, il est facile d'étendre Twig.

Il s'agit là de seulement quelques uns des bénéfices de Twig. Pour trouver plus de raisons pour lesquelles vous devriez utiliser Twig, rendez vous sur le site officiel de Twig.

Structure de présentation

Comme Twig supporte l'héritage de template, nous allons mettre en place l'approche d'héritage à 3 niveaux. Cette approche nous permet de modifier la vue a 3 niveaux distincts à l'intérieur de l'application, ce qui permet pas mal de personnalisation.

Template principal - Niveau 1

Commençons par créer le bloc de base du template pour Symblog. Nous avons pour cela besoin de 2 fichiers, le template et son fichier CSS associé. Comme Symfony2 supporte l'HTML5 nous allons également nous en servir.

```
<!-- app/Resources/views/base.html.twig -->
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html"; charset=utf-8" />
        <title>{% block title %}symblog{% endblock %} - symblog</title>
        <!--[if lt IE 9]>
            <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
        <![endif]-->
        {% block stylesheets %}
            <link href='http://fonts.googleapis.com/css?family=Irish+Grover' rel='stylesheet' type='text/css'>
            <link href='http://fonts.googleapis.com/css?family=La+Belle+Aurore' rel='stylesheet' type='text/css'>
            <link href="{{ asset('css/screen.css') }}" type="text/css" rel="stylesheet" />
        {% endblock %}
        <link rel="shortcut icon" href="{{ asset('favicon.ico') }}" />
    </head>
    <body>

        <section id="wrapper">
            <header id="header">
                <div class="top">
                    {% block navigation %}
                        <nav>
                            <ul class="navigation">
                                <li><a href="#">Home</a></li>
                                <li><a href="#">About</a></li>
                                <li><a href="#">Contact</a></li>
                            </ul>
                        </nav>
                    {% endblock %}
                </div>

                <hgroup>
                    <h2>{% block blog_title %}<a href="#">symblog</a>{% endblock %}</h2>
                    <h3>{% block blog_tagline %}<a href="#">creating a blog in Symfony2</a>{% endblock %}</h3>
                </hgroup>
            </header>

            <section class="main-col">
                {% block body %}{% endblock %}
```

```
            </section>
            <aside class="sidebar">
                {% block sidebar %}{% endblock %}
            </aside>

            <div id="footer">
                {% block footer %}
                    Symfony2 blog tutorial - created by <a href="https://github.com/dsyph3r">dsyph3r</a>
                {% endblock %}
            </div>
        </section>

        {% block javascripts %}{% endblock %}
    </body>
</html>
```

> **Note**
>
> Il y a 3 fichiers externes utilisés par le template, 1 fichier JavaScript et 2 fichiers CSS. Le fichier JavaScript résoud le problème du défaut de support d'HTML5 dans les navigateurs IE antérieurs à la version 9. Les 2 fichiers CSS importent des polices de caractères web Google.

Ce template met en place la structure principale de notre site de blogging. La plupart du fichier est composé de HTML, avec d'étranges commandes Twig. C'est à ces directives Twig que nous allons nous intéresser maintenant.

Commençons pour nous intéresser à la partie HEAD du document. Regardons la balise title:

```
<title>{% block title %}symblog{% endblock %} - symblog</title>
```

La première chose que vous aller remarquer est l'étrange tag `{%`. Ce ni du HTML, ni du PHP non plus. Il s'agit d'un des 3 tags de Twig. Ce tag signifie `Fais quelque chose`. Il est utilisé pour executer des blocs de code tels que les structures de contrôle, et pour définir des blocs. Une liste complète des structures de contrôle est disponible dans la documentation de Twig. Le bloc Twig que nous avons défini dans le titre fait 2 choses; Il crée un identificateur de block nommé `title`, et lui fournit une valeur de contenu par défaut entre les directives block et endblock. En définissant un bloc, nous pouvons nous servir du modèle d'héritage de Twig. Par exemple, sur une page qui sert à afficher un article, on peut souhaiter que le titre de la page reflète le titre de l'article. Cela peut être réalisé en étendant le template et en surchargeant le bloc title.

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

{% block title %}The blog title goes here{% endblock %}
```

Dans l'exemple ci dessus, nous avons étendu le template de base de l'application qui définissait initialement le bloc `title`. Vous pourrez remarquer que le format de template utilisé dans la directive `extends` ne contient ni la partie `Bundles` ni la partie `Controlleur` que nous avons évoqués précédemment: souvenez vous du format `bundle:controller:template`. En ne précisant ni le `Bundle` ni le `Controlleur`, on spécifie l'usage des templates au niveau de l'application, c'est à dire ceux situés dans `app/Resources/views/`.

Ensuite nous avons défini un autre bloc de titre et avons mis dedans du contenu, dans le cas présent le titre du blog. Comme le template parent a déjà défini un block `title`, il est remplacé par le nouveau. Le titre serait désormais 'The blog title goes here - symblog'. Cette fonctionnalité proposée par Twig va être largement utilisée lors de la création de templates.

Dans le bloc de la feuille de style, nous avons introduit le tag Twig suivant, `{{`, qui signifie `Dis quelque chose`.

```
<link href="{{ asset('css/screen.css') }}" type="text/css" rel="stylesheet" />
```

Ce tag est utilisé pour afficher la valeur d'une variable ou d'une expression. Dans l'exemple ci-dessus, il affiche la valeur de la fonction `asset`, qui nous fournit une manière portable de faire le lien avec les fichiers manipulés par l'application, tels que les fichiers CSS, JavaScript et les images.

Le tag `{{` peut également être combiné avec des filtres pour manipuler la sortie avant son affichage.

```
{{ blog.created|date("d-m-Y") }}
```

Pour une liste complète des filtres, se référer à la documentation de Twig.

Le dernier tag de Twig, que nous n'avons pas vu dans les templates, est le tag de commentaires `{#`. Son usage est le suivant :

```
{# The quick brown fox jumps over the lazy dog #}
```

Il n'y a pas d'autre concept introduit dans ce template. Il fournit la structure principale, prête à être personnalisée selon nos besoins.

Ensuite, il est temps d'ajouter du style. Créez une feuille de style dans `web/css/screen.css` et ajoutez le contenu suivant. Cela va ajouter du style pour le template principal.

```
html,body,div,span,applet,object,iframe,h1,h2,h3,h4,h5,h6,p,blockquote,pre,a,abbr,acronym,address,big,cite,code,del,dfn,em,img,ins,kbd,q,s,samp,small,s

body { line-height: 1;font-family: Arial, Helvetica, sans-serif;font-size: 12px; width: 100%; height: 100%; color: #000; font-size: 14px; }
.clear { clear: both; }
```

```
#wrapper { margin: 10px auto; width: 1000px; }
#wrapper a { text-decoration: none; color: #F48A00; }
#wrapper span.highlight { color: #F48A00; }

#header { border-bottom: 1px solid #ccc; margin-bottom: 20px; }
#header .top { border-bottom: 1px solid #ccc; margin-bottom: 10px; }
#header ul.navigation { list-style: none; text-align: right; }
#header .navigation li { display: inline }
#header .navigation li a { display: inline-block; padding: 10px 15px; border-left: 1px solid #ccc; }
#header h2 { font-family: 'Irish Grover', cursive; font-size: 92px; text-align: center; line-height: 110px; }
#header h2 a { color: #000; }
#header h3 { text-align: center; font-family: 'La Belle Aurore', cursive; font-size: 24px; margin-bottom: 20px; font-weight: normal; }

.main-col { width: 700px; display: inline-block; float: left; border-right: 1px solid #ccc; padding: 20px; margin-bottom: 20px; }
.sidebar { width: 239px; padding: 10px; display: inline-block; }

.main-col a { color: #F48A00; }
.main-col h1,
.main-col h2
    { line-height: 1.2em; font-size: 32px; margin-bottom: 10px; font-weight: normal; color: #F48A00; }
.main-col p { line-height: 1.5em; margin-bottom: 20px; }

#footer { border-top: 1px solid #ccc; clear: both; text-align: center; padding: 10px; color: #aaa; }
```

Template du Bundle - Niveau 2

Nous allons maintenant avancer vers la création de la présentation pour le bundle Blog. Créez un fichier dans `src/Blogger/BlogBundle/Resources/views/layout.html.twig` et ajoutez-y le contenu suivant :

```
{# src/Blogger/BlogBundle/Resources/views/layout.html.twig #}
{% extends '::base.html.twig' %}

{% block sidebar %}
    Sidebar content
{% endblock %}
```

A première vue, ce template peut sembler un peu simple, mais sa simplicité est sa force. Tout d'abord, il étend le template de base de l'application que nous avons créé précédemment. Ensuite, il remplace le bloc de la barre latérale avec un contenu factice. Comme la barre latérale va être présente dans toutes les pages de notre blog, il est logique de réaliser la personnalisation à ce niveau là. Vous pourriez demander pourquoi nous ne faisons pas la personnalisation dans le fichier de base de l'application, car il est également présent dans toutes les pages. C'est simple, l'application ne connait rien à propos d'un bundle et cele ne devrait jamais être le cas. Le bundle devrait contenir toutes ses fonctionnalités et afficher la barre latérale fait partie d'une de ces fonctionnalitées. Ok, dans ce cas pourquoi ne plaçons nous pas la barre latérale dans chacune des pages de template ? C'est à nouveau très simple, car il faudrait dupliquer la barre latérale à chaque fois que nous voudrions ajouter une page. Plus loin ce template du second étage va nous donner de la flexibilité pour ajouter de la personnalisation pour des besoins futurs et tous les templates enfants en hériteront. Par exemple, nous pourrions vouloir afficher le pied de page sur toutes les pages, et ce serait l'endroit idéal pour faire ceci.

Template de page - Niveau 3

Nous sommes enfin prêt pour la disposition du contrôleur. Ces agencements vont régulièrement être liés à des actions du contrôleur, par exemple l'action `show` va avoir un template `show`.

Commençons par créer le contrôleur pour la page d'accueil et son template. Comme c'est la première page que nous allons créer, nous allons avoir besoin de créer le contrôleur. Créez le contrôleur dans `src/Blogger/BlogBundle/Controller/PageController.php` et ajoutez-y le code suivant:

```
<?php
// src/Blogger/BlogBundle/Controller/PageController.php

namespace Blogger\BlogBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class PageController extends Controller
{
    public function indexAction()
    {
        return $this->render('BloggerBlogBundle:Page:index.html.twig');
    }
}
```

Maintenant nous allons créer le template pour cette action. Comme vous pouvez le voir dans ce contrôleur `Page`, nous allons afficher le template `index`. Créez le template dans `src/Blogger/BlogBundle/Resources/views/Page/index.html.twig`

```
{# src/Blogger/BlogBundle/Resources/views/Page/index.html.twig #}
{% extends 'BloggerBlogBundle::layout.html.twig' %}

{% block body %}
```

```
    Blog homepage
{% endblock %}
```

Cela présente le format de template final que nous pouvons spécifier. Dans cet exemple, le template `BloggerBlogBundle::layout.html.twig` est étendu là où la partie `Controlleur` du nom du template est omise. En excluant la partie `Controlleur`, nous précisons l'utilisation au niveau du bundle du template créé dans `src/Blogger/BlogBundle/Resources/views/layout.html.twig`.

Maintenant ajoutons une route pour notre page d'accueil. Mettez à jour la configuration de routage du bundle situé dans `src/Blogger/BlogBundle/Resources/config/routing.yml`.

```
# src/Blogger/BlogBundle/Resources/config/routing.yml
BloggerBlogBundle_homepage:
    pattern:  /
    defaults: { _controller: BloggerBlogBundle:Page:index }
    requirements:
        _method:  GET
```

Finalement, nous devons supprimer la route par défaut pour l'écran d'accueil de Symfony2. Supprimez la route `_welcome` en haut du fichier de routage de `dev`, situé dans le fichier `app/config/routing_dev.yml`.

Nous sommes désormais prêt à voir notre template pour le blog. Rendez vous avec votre navigateur à l'adresse `http://symblog.dev/app_dev.php/`.

Vous devriez désormais voir l'agencement de base du blog, avec le contenu principal du blog, et la barre latérale qui reflète lec blocs que nous avons surchargés dans les templates adéquats.

La page `A propos`

Notre dernière mission dans cette partie du tutorial est de créer une page statique pour la page `A propos`. Cela va montrer comment créer un lien entre plusieurs pages, et renforcer un peu plus l'approche à 3 niveaux que nous avons adopté.

La route

Lors de la création d'une nouvelle page, une des premières tâches devrait être la création d'une route. Ouvrez le fichier de route du `BloggerBlogBundle` situé dans `src/Blogger/BlogBundle/Resources/config/routing.yml` et ajoutez la règle de routage suivante :

```
# src/Blogger/BlogBundle/Resources/config/routing.yml
BloggerBlogBundle_about:
    pattern:  /about
    defaults: { _controller: BloggerBlogBundle:Page:about }
```

```
    requirements:
        _method:  GET
```

Le contrôleur

Ouvrez ensuite le contrôleur `Page` situé dans `src/Blogger/BlogBundle/Controller/PageController.php` et ajoutez l'action pour gérer la page `A propos`.

```
// src/Blogger/BlogBundle/Controller/PageController.php
class PageController extends Controller
{
    //  ..

    public function aboutAction()
    {
        return $this->render('BloggerBlogBundle:Page:about.html.twig');
    }
}
```

La vue

Concernant la vue, créez un nouveau fichier situé dans `src/Blogger/BlogBundle/Resources/views/Page/about.html.twig` et copiez-y le contenu suivant.

```
{# src/Blogger/BlogBundle/Resources/views/Page/about.html.twig #}
{% extends 'BloggerBlogBundle::layout.html.twig' %}

{% block title %}About{% endblock%}

{% block body %}
    <header>
        <h1>About symblog</h1>
    </header>
    <article>
        <p>Donec imperdiet ante sed diam consequat et dictum erat faucibus. Aliquam sit
        amet vehicula leo. Morbi urna dui, tempor ac posuere et, rutrum at dui.
        Curabitur neque quam, ultricies ut imperdiet id, ornare varius arcu. Ut congue
        urna sit amet tellus malesuada nec elementum risus molestie. Donec gravida
        tellus sed tortor adipiscing fringilla. Donec nulla mauris, mollis egestas
        condimentum laoreet, lacinia vel lorem. Morbi vitae justo sit amet felis
        vehicula commodo a placerat lacus. Mauris at est elit, nec vehicula urna. Duis a
        lacus nisl. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices
        posuere cubilia Curae.</p>
    </article>
{% endblock %}
```

Cette page `A propos` ne contient rien de spectaculaire. Son seul rôle est d'afficher un fichier de template qui possède un contenu factice. cela nous amène néanmoins à notre prochaine tâche.

Lier les pages

Nous avons désormais la page `A propos` en état de fonctionnement: rendez vous à l'adresse `http://symblog.dev/app_dev.php/about` pour vous en assurer. Par contre, il n'y a actuellement aucun autre moyen pour un utilisateur de voir la page `A propos`, que de taper l'adresse complète comme nous venons de le faire. Comme vous pouvez vous en douter, Symfony2 va nous permettre de créer un lien. Il permet de de faire correspondre des adresses comme nous l'avons déjà vu, et peut également générer des adresses pour ces routes. Vous devriez toujours utiliser les fonctions de routage proposées par Symfony2. Ne soyez jamais tenté, dans vos application, de faire la chose suivante :

```
<a href="/contact">Contact</a>

<?php $this->redirect("/contact"); ?>
```

Vous vous demandez sûrement ce qui ne va pas avec cette approche, c'est peut-être la manière que vous avez toujours utilisée pour faire des liens entre les pages. Néanmoins, il y a un certain nombre de problèmes avec cette approche :

1. Cela utilise un lien codé en dur et ignore tout du système de routage de Symfony2. Si vous vouliez changer l'adresse de la page de contact à n'importe quel moment, vous devriez trouver toutes les références au lien en dur et les changer.
2. Cela ignore le contrôleur d'environnement. Les environnements sont quelque chose que nous n'avons pas vraiment expliqué jusqu'à présent, mais vous vous en êtes servis. Le contrôleur de façade `app_dev.php` nous donne accès à notre application dans l'environnement `dev`. Si nous devions remplacer `app_dev.php` par `app.php`, alors nous ferions tourner l'application dans l'environnement de production. La signification de ces environnements sera expliquée plus loin dans le tutoriel, mais pour le moment, il est important de noter que les liens en dur ne maintiennent pas l'environnement dans lequel nous sommes car le contrôleur de façade n'est pas préfixé dans l'URL.

La manière correcte de faire des liens entre les pages est d'utiliser les fonctions `path` et `url` proposées par Twig. Elles sont toutes les deux très proches, sauf que la fonction `url` renvoit une URL absolue. Mettons à jour le template principal de notre application situé dans `app/Resources/views/base.html.twig` pour faire le lien entre la page d'accueil et la page d'à propos.

```
<!-- app/Resources/views/base.html.twig -->
{% block navigation %}
```

```
    <nav>
        <ul class="navigation">
            <li><a href="{{ path('BloggerBlogBundle_homepage') }}">Home</a></li>
            <li><a href="{{ path('BloggerBlogBundle_about') }}">About</a></li>
            <li><a href="#">Contact</a></li>
        </ul>
    </nav>
{% endblock %}
```

Maintenant mettez à jour votre navigateur pour voir les liens vers la page d'accueil (Home) et vers la page `A propos` (About) fonctionner comme attendu. Si vous regardez le code source de ces pages, vous pourrez remarquer que les liens ont été préfixés par `/app_dev.php/`. Il s'agit du contrôleur de façade expliqué plus tôt, et vous pouvez voir que l'utilisation de `path` maintient sa présence.

Finalement, mettons à jour le lien sur le logo pour vous rediriger vers la page d'accueil. Pour cela, mettez à jour le template situé dans `app/Resources/views/base.html.twig`.

```
<!-- app/Resources/views/base.html.twig -->
<hgroup>
    <h2>{% block blog_title %}<a href="{{ path('BloggerBlogBundle_homepage') }}">symblog</a>{% endblock %}</h2>
    <h3>{% block blog_tagline %}<a href="{{ path('BloggerBlogBundle_homepage') }}">creating a blog in Symfony2</a>{% endblock %}</h3>
</hgroup>
```

Conclusion

Nous avons couvert les domaines de base d'une application Symfony2, à commencer par la configuration et la mise en place de l'application. Nous avons commencé à explorer les concepts fondamentaux derrière une application Symfony2, en particulier le routage et le moteur de template Twig.

Nous verrons par la suite la création d'une page de contact. Cette page est légèrement plus compliquée que la page `A propos` car elle permet aux utilisateurs d'intéragir avec un formulaire pour envoyer des requêtes. Le chapitre suivant va présenter les concepts de validateurs et de formulaires.

diabgate • 5 years ago

Merci beaucoup Monsieur moi je suis nouveau dans l'apprentissage de symfony
j'utilise netbeans
Mais quand j'ai fini le niveau 1, 2 et trois j'ai une erreur