> Notes :

- https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony/le-routeur-de-symfony

-

-

> Sommaire :

Partie 1 - Vue d'ensemble de Symfony

1. Symfony, un framework PHP

2. Vous avez dit Symfony ?

3. Utilisons la console pour créer un bundle

4. Aux origines de Symfony (interview)

Quiz : Quiz 1

Partie 2 - Les bases de Symfony

1. Mon premier « Hello World ! » avec Symfony

2. Le routeur de Symfony

3. Les contrôleurs avec Symfony

4. Le moteur de templates Twig

5. Installer un bundle grâce à Composer

6. Les services, théorie et création

Quiz : Quiz 2

Activité : Créez votre CoreBundle pour les gouverner tous

Partie 3 - Gérer la base de données avec Doctrine2

1. La couche métier : les entités

2. Manipuler ses entités avec Doctrine2

3. Les relations entre entités avec Doctrine2

4. Récupérer ses entités avec Doctrine2

5. Les évènements et extensions Doctrine

6. TP : Consolidation de notre code

Quiz : Quiz 3

Activité : Créer un système pour nettoyer vos entités

Partie 4 - Allons plus loin avec Symfony

1. Créer des formulaires avec Symfony

2. Validez vos données

3. Sécurité et gestion des utilisateurs

4. Les services, utilisation poussée

5. Le gestionnaire d'évènements de Symfony

6. Traduire son site

Quiz : Quiz 4

Partie 5 - Préparer la mise en ligne

1. Convertir les paramètres de requêtes

2. Personnaliser les pages d'erreur

3. Utiliser Assetic pour gérer les codes CSS et JS de votre site

4. Utiliser la console directement depuis le navigateur

5. Déployer son site Symfony en production

Quiz : Quiz 5

Symfony, un framework PHP

Le fil rouge de notre cours : une plateforme d'échange

Dans ce cours, je vous propose de monter de toute pièce une plateforme d'échange.

Notre site proposera de poster des annonces de missions pour développeurs, designers, etc.

On pourra consulter ces annonces, les commenter, les chercher.

Tous les codes que vous trouverez dans ce cours s'articuleront donc autour de ce concept de plateforme d'échange,

pensez-y pour avoir une vision globale de ce qu'on construit !

Créons notre bundle

> php bin/console generate:bundle

- suivre les instructions

Test du bundle

- ajout du corps à src/OC/PlatformBundle/Resources/views/Default/index.html.twig

- Que s'est il passé ?

- Symfony a généré la structure du bundle

- Symfony a enregistré notre bundle auprès du Kernel

- Symfony a enregistré nos routes auprès du Routeur

À retenir

Ce qu'il faut retenir de tout cela, c'est que pour qu'un bundle soit opérationnel, il faut :

Son code source, situé dans src/Application/Bundle, et dont le seul fichier obligatoire est la classe à la racine OCPlatformBundle.php ;

Enregistrer le bundle dans le noyau pour qu'il soit chargé, en modifiant le fichier app/AppKernel.php ;

Enregistrer les routes (si le bundle en contient) dans le Routeur pour qu'elles soient chargées, en modifiant le fichier app/config/routing.yml.

Ces trois points sont bien sûr effectués automatiquement lorsqu'on utilise le générateur. Mais vous pouvez tout à fait créer un bundle sans l'utiliser, et il faudra alors remplir cette petite checklist manuellement.

Par la suite, tout notre code source sera situé dans des bundles. Un moyen très propre de bien structurer son application.

En résumé

Les commandes Symfony disponibles en ligne de commande ont pour objectif de nous faciliter la vie en automatisant certaines tâches.

Les commandes sont faites, comme tout Symfony, en PHP uniquement. La console n'est qu'un moyen différent du navigateur pour exécuter du code PHP.

La commande pour générer un nouveau bundle est php bin/console generate:bundle.

Le code du cours tel qu'il doit être à ce stade est disponible sur la branche iteration-1 du dépot Github.

Mon premier « Hello World ! » avec Symfony

Créons notre route

> éditer Resources/config/routing.yml dans le répertoire du bundle

# src/OC/PlatformBundle/Resources/config/routing.yml

hello\_the\_world:

path: /hello-world

defaults: { \_controller: OCPlatformBundle:Advert:index }

Créons notre contrôleur

> créer le fichier src/OC/PlatformBundle/Controller/AdvertController.php

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Component\HttpFoundation\Response;

class AdvertController

{

public function indexAction()

{

return new Response("Notre propre Hello World !");

}

}

Créons notre template Twig

> éditer le fichier src/OC/PlatformBundle/Resources/views/Advert/index.html.twig

{# src/OC/PlatformBundle/Resources/views/Advert/index.html.twig #}

<!DOCTYPE html>

<html>

<head>

<title>Bienvenue sur ma première page avec OpenClassrooms !</title>

</head>

<body>

<h1>Hello {{ nom }} !</h1>

<p>

Le Hello World est un grand classique en programmation.

Il signifie énormément, car cela veut dire que vous avez

réussi à exécuter le programme pour accomplir une tâche simple :

afficher ce hello world !

</p>

</body>

</html>

> modifier le controller

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

// N'oubliez pas ce use :

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

public function indexAction()

{

$content = $this->get('templating')->render('OCPlatformBundle:Advert:index.html.twig', array('nom' => 'winzou'));

return new Response($content);

}

}

Notre objectif : créer une plateforme d'annonces

Le fil conducteur

Comme je vous l'ai déjà annoncé, nous construirons une plateforme d'annonces tout au long de ce cours.

Cela me permet d'utiliser des exemples cohérents entre eux et de vous montrer comment construire un tel site de toutes pièces.

Bien sûr, libre à vous d'adapter les exemples au projet que vous souhaitez mener, je vous y encourage, même !

Le choix de ce site permet que l'on se comprenne bien : vous savez déjà ce qu'est une plateforme d'annonces (leboncoin, eBay, etc.),

vous comprendrez donc tous les exemples.

Notre plateforme

La plateforme que nous allons créer est très simple. En voici les grandes lignes :

Nous aurons des annonces (advert en anglais) de mission : développement d'un site internet, création d'une maquette, intégration HTML, etc. ;

Nous pourrons consulter, créer, modifier et rechercher des annonces ;

À chaque annonce, nous pourrons lier une image d'illustration ;

À chaque annonce, nous pourrons lier plusieurs candidatures (application en anglais) ;

Nous aurons plusieurs catégories (Développement, Graphisme, etc.) qui seront liées aux annonces. Nous pourrons créer, modifier et supprimer ces catégories ;

À chaque annonce, nous pourrons enfin lier des niveaux de compétence requis (Expert en PHP, maîtrise de Photoshop, etc.).

Au début, nous n'aurons pas de système de gestion des utilisateurs : nous devrons saisir notre nom lorsque nous rédigerons une annonce.

Puis, nous rajouterons la couche utilisateur.

Un peu de nettoyage

Avec tous les éléments générés par Symfony lors de la création du bundle et les nôtres,

il y a un peu de redondance. Vous pouvez donc supprimer joyeusement :

Le contrôleur Controller/DefaultController.php ;

Le répertoire de vues Resources/views/Default ;

La route oc\_platform\_homepage dans Resources/config/routing.yml.

Supprimez également tout ce qui concerne le bundle AppBundle, un bundle de démonstration intégré dans la

distribution standard de Symfony2 et dont nous ne nous servirons pas :

Le répertoire src/AppBundle ;

La lignes 19 du fichier app/AppKernel.php, celle qui active le bundle : new AppBundle\AppBundle() ;

Les lignes 7 à 9 du fichier app/config/routing.yml, celles qui importent le fichier de route

du bundle AppBundle (app: resource: "@AppBundle/Controller/" type: annotation" ).

Schéma de développement sous Symfony

Si vous rafraîchissez la page pour vérifier que tout est bon, il est possible que vous obteniez une erreur ! En effet, il faut prendre dès maintenant un réflexe Symfony : vider le cache. Car Symfony, pour nous offrir autant de fonctionnalités et être si rapide, utilise beaucoup son cache.

Le cache est constitué de fichiers PHP prêts à être exécutés, contenant tout le nécessaire pour faire tourner Symfony sous une forme plus rapide. Pensez par exemple à la configuration dans les fichiers YAML : quand Symfony génère une page, il va compiler cette configuration dans un tableau PHP (un array ), ce qui sera bien plus rapide à charger la fois suivante.

Or après certaines modifications, le cache peut ne plus être à jour, ce qui peut entrainer des erreurs. Deux cas de figure :

En mode « prod », c'est simple, Symfony ne regénère jamais le cache. Cela lui permet de ne faire aucune vérification sur la validité du cache (ce qui prend du temps), et de servir les pages très rapidement à vos visiteurs. La solution : vider le cache à la main à chaque fois que vous faites des changements. Cela se fait grâce à la commande php bin/console cache:clear --env=prod.

En mode « dev », c'est plus simple. Lorsque vous modifiez votre code, Symfony reconstruit une bonne partie du cache à la prochaine page que vous chargez. Donc pas forcément besoin de vider le cache. Seulement, comme il ne reconstruit pas tout, il peut parfois apparaître des bugs un peu étranges. Dans ce cas, un petit php bin/console cache:clear résout le problème en trois secondes !

Parfois, il se peut que la commande cache:clear génère des erreurs lors de son exécution. Dans ce cas, essayez de relancer la commande, parfois une deuxième passe peut résoudre les problèmes. Dans le cas contraire, supprimez le cache à la main en supprimant simplement le répertoire var/cache/dev (ou var/cache/prod suivant l’environnement).

Typiquement, un schéma classique de développement est le suivant :

Je fais des changements, je teste ;

Je fais des changements, je teste ;

Je fais des changements, je teste : ça ne marche pas, je vide le cache : ça marche ;

Je fais des changements, je teste ;

Je fais des changements, je teste ;

Je fais des changements, je teste : ça ne marche pas, je vide le cache : ça marche ;

...

En fin de journée, j'envoie tout sur le serveur de production, je vide obligatoirement le cache pour le mode prod, je teste : ça marche.

Évidemment, quand je dis « je teste : ça ne marche pas », j'entends « ça devrait marcher et l'erreur rencontrée est étrange ». Si vous faites une erreur dans votre propre code, ce n'est pas un cache:clear qui va la résoudre ! :p

Pour conclure

Et voilà, nous avons créé une page de A à Z ! Voici plusieurs remarques sur ce chapitre.

D'abord, ne vous affolez pas si vous n'avez pas tout compris.

Le but de ce chapitre était de vous donner une vision globale d'une page Symfony.

Vous avez des notions de bundles, de routes, de contrôleurs et de templates : vous savez presque tout !

Il ne reste plus qu'à approfondir chacune de ces notions, ce que nous ferons dès le prochain chapitre.

Ensuite, sachez que tout n'est pas à refaire lorsque vous créez une deuxième page.

Je vous invite là, maintenant, à créer une page /byebye-world et voyez si vous y arrivez.

Dans le cas contraire, relisez ce chapitre, puis si vous ne trouvez pas votre erreur, n'hésitez pas à poser

votre question sur le forum PHP, d'autres personnes qui sont passés par là seront ravis de vous aider. :)

Sur le forum, pensez à mettre le tag [Symfony] dans le titre de votre sujet, afin de s'y retrouver.

Allez, préparez-vous pour la suite, les choses sérieuses commencent !

En résumé

Le rôle du routeur est de déterminer quel route utiliser pour la requête courante.

Le rôle d'une route est d'associer une URL à une action du contrôleur.

Le rôle du contrôleur est de retourner au noyau un objet Response, qui contient la réponse HTTP à envoyer à l'internaute (page HTML ou redirection).

Le rôle des vues est de mettre en forme les données que le contrôleur lui donne, afin de construire une page HTML, un flux RSS, un e-mail, etc.

Le code du cours tel qu'il doit être à ce stade est disponible sur la branche iteration-2 du dépot Github.

Le routeur de Symfony

Le fonctionnement

> éditer votreBundle/Resources/config/routing.yml

# src/OC/PlatformBundle/Resources/config/routing.yml

oc\_platform\_home:

path: /platform

defaults: { \_controller: OCPlatformBundle:Advert:index }

oc\_platform\_view:

path: /platform/advert/{id}

defaults: { \_controller: OCPlatformBundle:Advert:view }

oc\_platform\_add:

path: /platform/add

defaults: { \_controller: OCPlatformBundle:Advert:add }

Les routes avancées

> dans le router du bundle

# src/OC/PlatformBundle/Resources/config/routing.yml

oc\_platform\_view\_slug:

path: /platform/{year}/{slug}.{format}

defaults:

\_controller: OCPlatformBundle:Advert:viewSlug

format: html

requirements:

year: \d{4}

format: html|xml

> dans le controller

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

// On récupère tous les paramètres en arguments de la méthode

public function viewSlugAction($slug, $year, $format)

{

return new Response(

"On pourrait afficher l'annonce correspondant au

slug '".$slug."', créée en ".$year." et au format ".$format."."

);

}

}

- Ajouter un préfixe lors de l'import de nos routes

> éditer le fichierapp/config/routing.yml

# app/config/routing.yml

oc\_platform:

resource: "@OCPlatformBundle/Resources/config/routing.yml"

prefix: /platform

Générer des URL

Depuis le contrôleur

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

public function indexAction()

{

// On veut avoir l'URL de l'annonce d'id 5.

$url = $this->get('router')->generate(

'oc\_platform\_view', // 1er argument : le nom de la route

array('id' => 5) // 2e argument : les valeurs des paramètres

);

// $url vaut « /platform/advert/5 »

return new Response("L'URL de l'annonce d'id 5 est : ".$url);

}

}

- deux méthode :

<?php

// Depuis un contrôleur

// Méthode longue

$url = $this->get('router')->generate('oc\_platform\_home');

// Méthode courte

$url = $this->generateUrl('oc\_platform\_home');

Depuis une vue Twig (notre moteur de template)

{# Dans une vue Twig, en considérant bien sûr

que la variable advert\_id est disponible #}

<a href="{{ path('oc\_platform\_view', { 'id': advert\_id }) }}">

Lien vers l'annonce d'id {{ advert\_id }}

</a>

Application : les routes de notre plateforme

# src/OC/PlatformBundle/Resources/config/routing.yml

oc\_platform\_home:

path: /{page}

defaults:

\_controller: OCPlatformBundle:Advert:index

page: 1

requirements:

page: \d\*

oc\_platform\_view:

path: /advert/{id}

defaults:

\_controller: OCPlatformBundle:Advert:view

requirements:

id: \d+

oc\_platform\_add:

path: /add

defaults:

\_controller: OCPlatformBundle:Advert:add

oc\_platform\_edit:

path: /edit/{id}

defaults:

\_controller: OCPlatformBundle:Advert:edit

requirements:

id: \d+

oc\_platform\_delete:

path: /delete/{id}

defaults:

\_controller: OCPlatformBundle:Advert:delete

requirements:

id: \d+

# app/config/routing.yml

oc\_platform:

resource: "@OCPlatformBundle/Resources/config/routing.yml"

prefix: /platform

Pour conclure

Ce chapitre est terminé, et vous savez maintenant tout ce qu'il faut savoir sur le routeur et les routes.

Retenez que ce système de routes vous permet premièrement d'avoir des belles URL, et deuxièmement de découpler le nom de vos URL du nom de vos contrôleurs.

Ajoutez à cela la génération d'URL, et vous avez un système extrêmement flexible et maintenable. Le tout sans trop d'efforts !

Pour plus d'informations sur le système de routes, n'hésitez pas à lire la documentation officielle.

En résumé

Une route est composée au minimum de deux éléments : l'URL à faire correspondre (sonpath), et le contrôleur à exécuter (paramètre \_controller).

Le routeur essaie de faire correspondre chaque route à l'URL appelée par l'internaute, et ce dans l'ordre d'apparition des routes : la première route qui correspond est sélectionnée.

Une route peut contenir des paramètres, facultatifs ou non, représentés par les accolades {paramètre}, et dont la valeur peut être soumise à des contraintes via la sectionrequirements.

Le routeur est également capable de générer des URL à partir du nom d'une route, et de ses paramètres éventuels.

Le code du cours tel qu'il doit être à ce stade est disponible sur la branche iteration-3 du dépot Github.

Les contrôleurs avec Symfony

Le rôle du contrôleur

Retourner une réponse

Il existe dans Symfony les classesRequest et Response.

Retourner une réponse signifie donc tout simplement : instancier un objetResponse, disons$response, et faire unreturn $response.

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

public function indexAction()

{

return new Response("Hello World !");

}

}

Manipuler l'objet Request

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

// …

// On injecte la requête dans les arguments de la méthode

public function viewAction($id, Request $request)

{

// On récupère notre paramètre tag

$tag = $request->query->get('tag');

return new Response(

"Affichage de l'annonce d'id : ".$id.", avec le tag : ".$tag

);

}

}

Récupérer la méthode de la requête HTTP

<?php

if ($request->isMethod('POST'))

{

// Un formulaire a été envoyé, on peut le traiter ici

}

Savoir si la requête est une requête AJAX

<?php

if ($request->isXmlHttpRequest())

{

// C'est une requête AJAX, retournons du JSON, par exemple

}

Manipuler l'objet Response

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

// On modifie viewAction, car elle existe déjà

public function viewAction($id)

{

// On crée la réponse sans lui donner de contenu pour le moment

$response = new Response();

// On définit le contenu

$response->setContent("Ceci est une page d'erreur 404");

// On définit le code HTTP à « Not Found » (erreur 404)

$response->setStatusCode(Response::HTTP\_NOT\_FOUND);

// On retourne la réponse

return $response;

}

}

Réponses et vues

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

public function viewAction($id, Request $request)

{

// On récupère notre paramètre tag

$tag = $request->query->get('tag');

// On utilise le raccourci : il crée un objet Response

// Et lui donne comme contenu le contenu du template

return $this->get('templating')->renderResponse(

'OCPlatformBundle:Advert:view.html.twig',

array('id' => $id, 'tag' => $tag)

);

}

}

<?php

public function viewAction($id, Request $request)

{

// On récupère notre paramètre tag

$tag = $request->query->get('tag');

return $this->render('OCPlatformBundle:Advert:view.html.twig', array(

'id' => $id,

'tag' => $tag,

));

}

{# src/OC/PlatformBundle/Resources/view/Advert/view.html.twig #}

<!DOCTYPE html>

<html>

<head>

<title>Affichage de l'annonce {{ id }}</title>

</head>

<body>

<h1>Hello Annonce n°{{ id }} !</h1>

<p>Tag éventuel : {{ tag }}</p>

</body>

</html>

Réponse et redirection

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\RedirectResponse; // N'oubliez pas ce use

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

public function viewAction($id)

{

$url = $this->get('router')->generate('oc\_platform\_home');

return new RedirectResponse($url);

}

}

<?php

public function viewAction($id)

{

$url = $this->get('router')->generate('oc\_platform\_home');

return $this->redirect($url);

}

<?php

public function viewAction($id)

{

return $this->redirectToRoute('oc\_platform\_home');

}

Changer leContent-typede la réponse

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller

{

public function viewAction($id)

{

// Créons nous-mêmes la réponse en JSON, grâce à la fonction json\_encode()

$response = new Response(json\_encode(array('id' => $id)));

// Ici, nous définissons le Content-type pour dire au navigateur

// que l'on renvoie du JSON et non du HTML

$response->headers->set('Content-Type', 'application/json');

return $response;

}

<?php

use Symfony\Component\HttpFoundation\JsonResponse;

// ...

public function viewAction($id)

{

return new JsonResponse(array('id' => $id));

}

Manipuler la session

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class AdvertController extends Controller

{

public function viewAction($id, Request $request)

{

// Récupération de la session

$session = $request->getSession();

// On récupère le contenu de la variable user\_id

$userId = $session->get('user\_id');

// On définit une nouvelle valeur pour cette variable user\_id

$session->set('user\_id', 91);

// On n'oublie pas de renvoyer une réponse

return new Response("<body>Je suis une page de test, je n'ai rien à dire</body>");

}

}

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class AdvertController extends Controller

{

public function viewAction($id)

{

return $this->render('OCPlatformBundle:Advert:view.html.twig', array(

'id' => $id

));

}

// Ajoutez cette méthode :

public function addAction(Request $request)

{

$session = $request->getSession();

// Bien sûr, cette méthode devra réellement ajouter l'annonce

// Mais faisons comme si c'était le cas

$session->getFlashBag()->add('info', 'Annonce bien enregistrée');

// Le « flashBag » est ce qui contient les messages flash dans la session

// Il peut bien sûr contenir plusieurs messages :

$session->getFlashBag()->add('info', 'Oui oui, elle est bien enregistrée !');

// Puis on redirige vers la page de visualisation de cette annonce

return $this->redirectToRoute('oc\_platform\_view', array('id' => 5));

}

}

{# src/OC/PlatformBundle/Resources/view/Advert/view.html.twig #}

<!DOCTYPE html>

<html>

<head>

<title>Affichage de l'annonce {{ id }}</title>

</head>

<body>

<h1>Affichage de l'annonce n°{{ id }} !</h1>

<div>

{# On affiche tous les messages flash dont le nom est « info » #}

{% for message in app.session.flashbag.get('info') %}

<p>Message flash : {{ message }}</p>

{% endfor %}

</div>

<p>

Ici nous pourrons lire l'annonce ayant comme id : {{ id }}<br />

Mais pour l'instant, nous ne savons pas encore le faire, cela viendra !

</p>

</body>

</html>

Application : le contrôleur de notre plateforme

Construction du contrôleur

# src/OC/PlatformBundle/Resources/config/routing.yml

oc\_platform\_home:

path: /{page}

defaults:

\_controller: OCPlatformBundle:Advert:index

page: 1

requirements:

page: \d\*

oc\_platform\_view:

path: /advert/{id}

defaults:

\_controller: OCPlatformBundle:Advert:view

requirements:

id: \d+

oc\_platform\_add:

path: /add

defaults:

\_controller: OCPlatformBundle:Advert:add

oc\_platform\_edit:

path: /edit/{id}

defaults:

\_controller: OCPlatformBundle:Advert:edit

requirements:

id: \d+

oc\_platform\_delete:

path: /delete/{id}

defaults:

\_controller: OCPlatformBundle:Advert:delete

requirements:

id: \d+

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;

class AdvertController extends Controller

{

public function indexAction($page)

{

// On ne sait pas combien de pages il y a

// Mais on sait qu'une page doit être supérieure ou égale à 1

if ($page < 1) {

// On déclenche une exception NotFoundHttpException, cela va afficher

// une page d'erreur 404 (qu'on pourra personnaliser plus tard d'ailleurs)

throw new NotFoundHttpException('Page "'.$page.'" inexistante.');

}

// Ici, on récupérera la liste des annonces, puis on la passera au template

// Mais pour l'instant, on ne fait qu'appeler le template

return $this->render('OCPlatformBundle:Advert:index.html.twig');

}

public function viewAction($id)

{

// Ici, on récupérera l'annonce correspondante à l'id $id

return $this->render('OCPlatformBundle:Advert:view.html.twig', array(

'id' => $id

));

}

public function addAction(Request $request)

{

// La gestion d'un formulaire est particulière, mais l'idée est la suivante :

// Si la requête est en POST, c'est que le visiteur a soumis le formulaire

if ($request->isMethod('POST')) {

// Ici, on s'occupera de la création et de la gestion du formulaire

$request->getSession()->getFlashBag()->add('notice', 'Annonce bien enregistrée.');

// Puis on redirige vers la page de visualisation de cettte annonce

return $this->redirectToRoute('oc\_platform\_view', array('id' => 5));

}

// Si on n'est pas en POST, alors on affiche le formulaire

return $this->render('OCPlatformBundle:Advert:add.html.twig');

}

public function editAction($id, Request $request)

{

// Ici, on récupérera l'annonce correspondante à $id

// Même mécanisme que pour l'ajout

if ($request->isMethod('POST')) {

$request->getSession()->getFlashBag()->add('notice', 'Annonce bien modifiée.');

return $this->redirectToRoute('oc\_platform\_view', array('id' => 5));

}

return $this->render('OCPlatformBundle:Advert:edit.html.twig');

}

public function deleteAction($id)

{

// Ici, on récupérera l'annonce correspondant à $id

// Ici, on gérera la suppression de l'annonce en question

return $this->render('OCPlatformBundle:Advert:delete.html.twig');

}

}

Pour conclure

Créer un contrôleur à ce stade du cours n'est pas évident, car vous ne connaissez et ne maîtrisez pas encore tous les services nécessaires.

Seulement, vous avez pu comprendre son rôle et voir un exemple concret.

Rassurez-vous, dans la partie 4 du cours, on apprendra tout le nécessaire pour construire l'intérieur de nos contrôleurs. ;)

En attendant, rendez-vous au prochain chapitre pour en apprendre plus sur les templates.

Pour plus d'informations concernant les contrôleurs, n'hésitez pas à lire la documentation officielle.

En résumé

Le rôle du contrôleur est de retourner un objetResponse: ceci est obligatoire !

Le contrôleur construit la réponse en fonction des données qu'il a en entrée : paramètre de route et objetRequest.

Le contrôleur se sert de tout ce dont il a besoin pour construire la réponse : la base de données, les vues, les différents services, etc.

Le code du cours tel qu'il doit être à ce stade est disponible sur la branche iteration-4 du dépot Github.

Le moteur de templates Twig

Les templates Twig

Afficher des variables

https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony/le-moteur-de-templates-twig-1

enregistrer nos propres variables globales

# app/config/parameters.yml

parameters:

# …

app\_webmaster: moi-même

# app/config/config.yml

twig:

globals:

webmaster: %app\_webmaster%

Structures de contrôle et expressions

Condition : {% if %}

{% if membre.age < 12 %}

Il faut avoir au moins 12 ans pour ce film.

{% elseif membre.age < 18 %}

OK bon film.

{% else %}

Un peu vieux pour voir ce film non ?

{% endif %}

Boucle : {% for %}

<ul>

{% for membre in liste\_membres %}

<li>{{ membre.pseudo }}</li>

{% else %}

<li>Pas d'utilisateur trouvé.</li>

{% endfor %}

</ul>

Définition : {% set %}

{% set foo = 'bar' %}

Les tests utiles

Defined

{% if var is defined %} … {% endif %}

Even / Odd

{% for valeur in liste %}

<span class="{% if loop.index is even %}pair{% else %}

impair{% endif %}">

{{ valeur }}

</span>

{% endfor %}

Hériter et inclure des templates

Voici à quoi peut ressembler un template père (appelé plus communément layout).

Mettons-le dans src/OC/PlatformBundle/Resources/views/layout.html.twig

{# src/OC/PlatformBundle/Resources/views/layout.html.twig #}

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

<title>{% block title %}OC Plateforme{% endblock %}</title>

</head>

<body>

{% block body %}

{% endblock %}

</body>

</html>

Et voici un de nos templates fils.

Mettons-le dans src/OC/PlatformBundle/Resources/views/Advert/index.html.twig

{# src/OC/PlatformBundle/Resources/views/Advert/index.html.twig #}

{% extends "OCPlatformBundle::layout.html.twig" %}

{% block title %}{{ parent() }} - Index{% endblock %}

{% block body %}

Notre plateforme est un peu vide pour le moment, mais cela viendra !

{% endblock %}

L'inclusion de templates

{# src/OC/PlatformBundle/Resources/views/Advert/add.html.twig #}

{% extends "OCPlatformBundle::layout.html.twig" %}

{% block body %}

<h2>Ajouter une annonce</h2>

{{ include("OCPlatformBundle:Advert:form.html.twig") }}

<p>

Attention : cette annonce sera ajoutée directement

sur la page d'accueil après validation du formulaire.

</p>

{% endblock %}

{# src/OC/PlatformBundle/Resources/views/Advert/form.html.twig #}

{# Cette vue n'hérite de personne, elle sera incluse par d'autres vues qui,

elles, hériteront probablement du layout. Je dis « probablement » car,

ici pour cette vue, on n'en sait rien et c'est une info qui ne nous concerne pas. #}

<h3>Formulaire d'annonce</h3>

{# On laisse vide la vue pour l'instant, on la comblera plus tard

lorsqu'on saura afficher un formulaire. #}

<div class="well">

Ici se trouvera le formulaire.

</div>

L'inclusion de contrôleurs

{# src/OC/PlatformBundle/Resources/views/layout.html.twig #}

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

<title>{% block title %}OC Plateforme{% endblock %}</title>

</head>

<body>

<div id="menu">

{{ render(controller("OCPlatformBundle:Advert:menu")) }}

</div>

{% block body %}

{% endblock %}

</body>

</html>

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;

class AdvertController extends Controller

{

// ...

public function menuAction()

{

// On fixe en dur une liste ici, bien entendu par la suite

// on la récupérera depuis la BDD !

$listAdverts = array(

array('id' => 2, 'title' => 'Recherche développeur Symfony'),

array('id' => 5, 'title' => 'Mission de webmaster'),

array('id' => 9, 'title' => 'Offre de stage webdesigner')

);

return $this->render('OCPlatformBundle:Advert:menu.html.twig', array(

// Tout l'intérêt est ici : le contrôleur passe

// les variables nécessaires au template !

'listAdverts' => $listAdverts

));

}

}

{# src/OC/PlatformBundle/Resources/views/Advert/menu.html.twig #}

{# Ce template n'hérite de personne,

tout comme le template inclus avec {{ include() }}. #}

<ul class="nav nav-pills nav-stacked">

{% for advert in listAdverts %}

<li>

<a href="{{ path('oc\_platform\_view', {'id': advert.id}) }}">

{{ advert.title }}

</a>

</li>

{% endfor %}

</ul>

Application : les templates de notre plateforme

Layout général

{# app/Resources/views/layout.html.twig #}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>{% block title %}OC Plateforme{% endblock %}</title>

{% block stylesheets %}

{# On charge le CSS de bootstrap depuis le site directement #}

<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">

{% endblock %}

</head>

<body>

<div class="container">

<div id="header" class="jumbotron">

<h1>Ma plateforme d'annonces</h1>

<p>

Ce projet est propulsé par Symfony,

et construit grâce au MOOC OpenClassrooms et SensioLabs.

</p>

<p>

<a class="btn btn-primary btn-lg" href="https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2">

Participer au MOOC »

</a>

</p>

</div>

<div class="row">

<div id="menu" class="col-md-3">

<h3>Les annonces</h3>

<ul class="nav nav-pills nav-stacked">

<li><a href="{{ path('oc\_platform\_home') }}">Accueil</a></li>

<li><a href="{{ path('oc\_platform\_add') }}">Ajouter une annonce</a></li>

</ul>

<h4>Dernières annonces</h4>

{{ render(controller("OCPlatformBundle:Advert:menu", {'limit': 3})) }}

</div>

<div id="content" class="col-md-9">

{% block body %}

{% endblock %}

</div>

</div>

<hr>

<footer>

<p>The sky's the limit © {{ 'now'|date('Y') }} and beyond.</p>

</footer>

</div>

{% block javascripts %}

{# Ajoutez ces lignes JavaScript si vous comptez vous servir des fonctionnalités du bootstrap Twitter #}

<script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>

<script src="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"></script>

{% endblock %}

</body>

</html>

Layout du bundle

{# src/OC/PlatformBundle/Resources/views/layout.html.twig #}

{% extends "::layout.html.twig" %}

{% block title %}

Annonces - {{ parent() }}

{% endblock %}

{% block body %}

{# On définit un sous-titre commun à toutes les pages du bundle, par exemple #}

<h1>Annonces</h1>

<hr>

{# On définit un nouveau bloc, que les vues du bundle pourront remplir #}

{% block ocplatform\_body %}

{% endblock %}

{% endblock %}

Les templates finaux

{# src/OC/PlatformBundle/Resources/views/Advert/index.html.twig #}

{% extends "OCPlatformBundle::layout.html.twig" %}

{% block title %}

Accueil - {{ parent() }}

{% endblock %}

{% block ocplatform\_body %}

<h2>Liste des annonces</h2>

<ul>

{% for advert in listAdverts %}

<li>

<a href="{{ path('oc\_platform\_view', {'id': advert.id}) }}">

{{ advert.title }}

</a>

par {{ advert.author }},

le {{ advert.date|date('d/m/Y') }}

</li>

{% else %}

<li>Pas (encore !) d'annonces</li>

{% endfor %}

</ul>

{% endblock %}

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

// Dans l'action indexAction() :

return $this->render('OCPlatformBundle:Advert:index.html.twig', array(

'listAdverts' => array()

));

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

public function menuAction($limit)

{

// On fixe en dur une liste ici, bien entendu par la suite

// on la récupérera depuis la BDD !

$listAdverts = array(

array('id' => 2, 'title' => 'Recherche développeur Symfony'),

array('id' => 5, 'title' => 'Mission de webmaster'),

array('id' => 9, 'title' => 'Offre de stage webdesigner')

);

return $this->render('OCPlatformBundle:Advert:menu.html.twig', array(

// Tout l'intérêt est ici : le contrôleur passe

// les variables nécessaires au template !

'listAdverts' => $listAdverts

));

}

{# src/OC/PlatformBundle/Resources/views/Advert/menu.html.twig #}

<ul class="nav nav-pills nav-stacked">

{% for advert in listAdverts %}

<li>

<a href="{{ path('oc\_platform\_view', {'id': advert.id}) }}">

{{ advert.title }}

</a>

</li>

{% endfor %}

</ul>

Voici un tableau d'annonces à ajouter temporairement dans la méthode indexAction(),

que vous pouvez passer en paramètre à la méthode render(). C'est un tableau pour l'exemple,

par la suite il faudra bien sûr récupérer les annonces depuis la base de données !

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

// …

public function indexAction($page)

{

// ...

// Notre liste d'annonce en dur

$listAdverts = array(

array(

'title' => 'Recherche développpeur Symfony',

'id' => 1,

'author' => 'Alexandre',

'content' => 'Nous recherchons un développeur Symfony débutant sur Lyon. Blabla…',

'date' => new \Datetime()),

array(

'title' => 'Mission de webmaster',

'id' => 2,

'author' => 'Hugo',

'content' => 'Nous recherchons un webmaster capable de maintenir notre site internet. Blabla…',

'date' => new \Datetime()),

array(

'title' => 'Offre de stage webdesigner',

'id' => 3,

'author' => 'Mathieu',

'content' => 'Nous proposons un poste pour webdesigner. Blabla…',

'date' => new \Datetime())

);

// Et modifiez le 2nd argument pour injecter notre liste

return $this->render('OCPlatformBundle:Advert:index.html.twig', array(

'listAdverts' => $listAdverts

));

}

Advert/view.html.twig

{# src/OC/PlatformBundle/Resources/view/Advert/view.html.twig #}

{% extends "OCPlatformBundle::layout.html.twig" %}

{% block title %}

Lecture d'une annonce - {{ parent() }}

{% endblock %}

{% block ocplatform\_body %}

<h2>{{ advert.title }}</h2>

<i>Par {{ advert.author }}, le {{ advert.date|date('d/m/Y') }}</i>

<div class="well">

{{ advert.content }}

</div>

<p>

<a href="{{ path('oc\_platform\_home') }}" class="btn btn-default">

<i class="glyphicon glyphicon-chevron-left"></i>

Retour à la liste

</a>

<a href="{{ path('oc\_platform\_edit', {'id': advert.id}) }}" class="btn btn-default">

<i class="glyphicon glyphicon-edit"></i>

Modifier l'annonce

</a>

<a href="{{ path('oc\_platform\_delete', {'id': advert.id}) }}" class="btn btn-danger">

<i class="glyphicon glyphicon-trash"></i>

Supprimer l'annonce

</a>

</p>

{% endblock %}

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

// …

public function viewAction($id)

{

$advert = array(

'title' => 'Recherche développpeur Symfony2',

'id' => $id,

'author' => 'Alexandre',

'content' => 'Nous recherchons un développeur Symfony2 débutant sur Lyon. Blabla…',

'date' => new \Datetime()

);

return $this->render('OCPlatformBundle:Advert:view.html.twig', array(

'advert' => $advert

));

}

Advert/edit.html.twig et add.html.twig

{# src/OC/PlatformBundle/Resources/views/Advert/edit.html.twig #}

{% extends "OCPlatformBundle::layout.html.twig" %}

{% block title %}

Modifier une annonce - {{ parent() }}

{% endblock %}

{% block ocplatform\_body %}

<h2>Modifier une annonce</h2>

{{ include("OCPlatformBundle:Advert:form.html.twig") }}

<p>

Vous éditez une annonce déjà existante, merci de ne pas changer

l'esprit général de l'annonce déjà publiée.

</p>

<p>

<a href="{{ path('oc\_platform\_view', {'id': advert.id}) }}" class="btn btn-default">

<i class="glyphicon glyphicon-chevron-left"></i>

Retour à l'annonce

</a>

</p>

{% endblock %}

{# src/OC/PlatformBundle/Resources/views/Advert/form.html.twig #}

<h3>Formulaire d'annonce</h3>

{# On laisse vide la vue pour l'instant, on la comblera plus tard

lorsqu'on saura afficher un formulaire. #}

<div class="well">

Ici se trouvera le formulaire.

</div>

<?php

// src/OC/PlatformBundle/Controller/AdvertController.php

public function editAction($id, Request $request)

{

// ...

$advert = array(

'title' => 'Recherche développpeur Symfony',

'id' => $id,

'author' => 'Alexandre',

'content' => 'Nous recherchons un développeur Symfony débutant sur Lyon. Blabla…',

'date' => new \Datetime()

);

return $this->render('OCPlatformBundle:Advert:edit.html.twig', array(

'advert' => $advert

));

}

Installer un bundle grâce à Composer