Note :

-

> Tuto

// - http://keiruaprod.fr/symblog-fr/index.html# //

// - http://symfony.com/doc/master/setup.html#creating-symfony-applications-without-the-installer //

> Install installeur

- sudo mkdir -p /usr/local/bin

- sudo curl -LsS http://symfony.com/installer -o /usr/local/bin/symfony

- sudo chmod a+x /usr/local/symfony

> Install Symfony

- cd /home/nathan/public\_html/

- symfony new symblog [version]

> Test

- cd public\_html/symblog/

- php bin/console server:run

- ctrl+c pour quitter

> Création du Virtual Host

- cd /etc/apache2/sites-available

- sudo cp dev.conf symblog.dev.conf

- sudo mv symblog.dev symblog.dev.conf

- sudo gedit symblog.dev.conf

<VirtualHost \*:80>

# The ServerName directive sets the request scheme, hostname and port that

# the server uses to identify itself. This is used when creating

# redirection URLs. In the context of virtual hosts, the ServerName

# specifies what hostname must appear in the request's Host: header to

# match this virtual host. For the default virtual host (this file) this

# value is not decisive as it is used as a last resort host regardless.

# However, you must set it for any further virtual host explicitly.

#ServerName www.example.com

ServerAdmin webmaster@localhost

ServerName symblog.dev

DocumentRoot /home/nathan/public\_html/symfony/web

DirectoryIndex app.php

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,

# error, crit, alert, emerg.

# It is also possible to configure the loglevel for particular

# modules, e.g.

#LogLevel info ssl:warn

<Directory /home/nathan/public\_html/symfony/web>

# Active les options :

# FollowSymLinks permet de suivre les liens symboliques.

# Indexes autorise le listage de fichiers d'un répertoire qui ne contient pas d'index.

Options Indexes FollowSymLinks MultiViews

# AllowOverride permet de surcharger certaines options en utilisant des fichiers .htaccess dans le répertoire du site.

AllowOverride All

# Droits par defaut

Order allow,deny

allow from all

# permet de rediriger les requêtes vers un fichier d'index précis

# RedirectMatch ^/$ /index.php

</Directory>

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

# For most configuration files from conf-available/, which are

# enabled or disabled at a global level, it is possible to

# include a line for only one particular virtual host. For example the

# following line enables the CGI configuration for this host only

# after it has been globally disabled with "a2disconf".

#Include conf-available/serve-cgi-bin.conf

</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

- sudo gedit /etc/hosts

127.0.0.1 symblog.dev

- a2ensite symblog.dev.conf

- service apache2 reload

- a2dissite symblog.dev.confmv

> Si page blanche

- http://symfony.com/doc/current/setup/file\_permissions.html avec ACL

- https://help.ubuntu.com/community/FilePermissionsACLs

- Comme nous avons activé le UserDir spécifier son chemin en plus /var au lieu de var dans les deux dernières commandes

- HTTPDUSER=`ps axo user,comm | grep -E '[a]pache|[h]ttpd|[\_]www|[w]ww-data|[n]ginx' | grep -v root | head -1 | cut -d\ -f1`

- sudo setfacl -R -m u:"$HTTPDUSER":rwX -m u:`whoami`:rwX /home/nathan/public\_html

- sudo setfacl -dR -m u:"$HTTPDUSER":rwX -m u:`whoami`:rwX /home/nathan/public\_html

> Version antérieur

- sudo apt-get install composer

- sudo composer create-project symfony/framework-standard-edition /home/nathan/public\_html/symblog 2.2.11

- Code lors de la configuration : e4cabab69f38fe60245960d99969a7f030c8cf25

> Génération du Bundle BloggerBlogBundle

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

// ---------------------------------------------------------------

> Livre Symfony 2 + Site OpenClassroom

>> Développement

- Avant

- MOA / MOE

- Scrum

- UML

-

- Environnement

- Objet et Structure

- Commande console

- Configuration et Développement

- Socle Applicatif

- Objet et Structure

/app | socle applicatif

-- AppCache.php |

-- AppKernel.php |

-- autoload.php |

-- bootstrap.php.cache |

-- /cache | swap du kernel et de bundles - affecter droits w

-- check.php |

-- /config |

-- config\_dev.yml |

-- config\_prod.yml |

-- config\_test.yml |

-- config.yml |

-- parameters.yml |

-- parameters.yml.dist |

-- routing-dev.yml |

-- rooting.yml |

-- security.yml |

-- services.yml |

-- console\* |

-- .htaccess |

-- /log | logs de dev et de prod - affecter droits w

-- phpunit.xml.dist |

-- /Ressources |

-- /views |

-- SymfonyRequirements.php|

/bin |

-- doctrine |

-- doctrine-dbal |

-- doctrine.php |

-- security-checker.php |

composer.json |

composer.lock |

/src | contients notre code en bundles

-- /AppBundle |

-- /Controller |

-- /Tests |

-- AppBundle.php |

/vendor | contient toutes les bibliothèques externes à notre application

-- /composer |

-- /doctrine |

-- /incenteev |

-- /jdorn |

-- /kriswallsmith |

-- /monolog |

-- /paragonie |

-- /psr |

-- /sensio |

-- /sensiolabs |

-- /swiftmailer |

-- /symfony |

-- /twig |

-- autoload.php |

/web | contient tous les fichiers destinés à vos visiteurs : images, fichiers CSS et JavaScript, etc

-- /bundles | contient des css et images du socle

-- .htaccess |

-- app.php |

-- app\_dev.php |

-- config.php |

- Deux environnements de travail

L'environnement de développement, appelé « dev », accessible en utilisant le contrôleur frontal app\_dev.php. C'est l'environnement que l'on utilisera toujours pour développer.

L'environnement de production, appelé « prod », accessible en utilisant le contrôleur frontal app.php.

- Commande console

- Installation

- sudo composer create-project symfony/framework-standard-edition eyrolles/ "2.6.\*"

- Test

- en terminal : php app/check.php

- en nav : http://localhost/eyrolles/code/web/config.php

- Configurer les permissions d’écriture

- Utiliser chmod +a

$ rm -rf app/cache/\*

$ rm -rf app/logs/\*

$ HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[\_]www|[w]ww-data|[n]ginx' | grep -v root | head -1 | cut -d\ -f1`

$ sudo chmod +a "$HTTPDUSER allow delete,write,append,file\_inherit,directory\_inherit" app/cache app/logs

$ sudo chmod +a "`whoami` allow delete,write,append,file\_inherit,directory\_inherit" app/cache app/logs

- Utiliser les ACL

$ HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[\_]www|[w]ww-data|[n]ginx' | grep -v root | head -1 | cut -d\ -f1`

$ sudo setfacl -R -m u:"$HTTPDUSER":rwX -m u:`whoami`:rwX app/cache app/logs

$ sudo setfacl -dR -m u:"$HTTPDUSER":rwX -m u:`whoami`:rwX app/cache app/logs

- Page de démonstration

- http://localhost/eyrolles/code/web/app\_dev.php/

- Mettre à jour Symfony

- Mettre à jour une version patch du dossier du projet

- sudo composer update symfony/symfony

- sudo composer update

- Mettre à jour vers une version mineure

- modifier le fichier composer.json

{

"require": {

"symfony/symfony": "2.6.\*",

},

}

- sudo composer update symfony/symfony

- sudo composer update

- Présentation du console

- php app/console

- Configuration et Développement

De manière itérative pour chaque bundle, on va :

- créer\* le bundle (\*géré par Symfony, si on utilise sa console)

- créer\* et développer le App/AppKernel.php de Symfony (\*géré par Symfony, si on utilise sa console)

- créer\* et développer le App/config/routing.yml de Symfony (\*géré par Symfony, si on utilise sa console)

- créer\* et développer\*\* le (path\_bundle)Resources/config/routing.yml (\*géré par Symfony, si on utilise sa console)

- créer\* et développer\*\* le(s) contrôler(s) (\*géré par Symfony, si on utilise sa console)

- créer\* et développer\*\* le(s) Modèle(s) (\*géré par Symfony, si on utilise sa console)

- créer\* et développer\*\* le(s) vue(s) (\*géré par Symfony, si on utilise sa console)

\*\*développement itérative

- Création d'un bundle

- Objet et Structure

Découper votre application en une multitude de bundles (http://knpbundles.com > FOSUserBundle | FOSCommentBundle | GravatarBundle), représentant toutes les fonctionnalités que vous proposez dans votre application.

/Controller | Contient vos contrôleurs

/DependencyInjection | Contient des informations sur votre bundle (chargement automatique de la configuration par exemple)

/Entity | Contient vos modèles

/Form | Contient vos éventuels formulaires

/Resources

-- /config | Contient les fichiers de configuration de votre bundle (nous placerons les routes ici, par exemple)

-- /public | Contient les fichiers publics de votre bundle : fichiers CSS et JavaScript, images, etc.

-- /views | Contient les vues de notre bundle, les templates Twig

- Commande console

- php app/console generate:bundle

1. Choisir le namespace

<NamespaceRacine>/<NomduBundle><Bundle> ex. : OC/PlatformBundle

2. Choisir le nom du Bundle

Valider ce qui est proposé : OCPlatformBundle

3. Choisir la destination

Valider ce qui est proposé : Symfony/src

4. Choisir le format de configuration (yml, xml, php, or annotation)

yml car bien adapté pour un buldle

5. Choisir quelle structure générer

choisir non pour le début

6. Confirmez, et c'est joué !

- Configuration et Développement

- Barre de Développement

Ajouter la balise <body> dans la vue (src/OC/PlatformBundle/Resources/views/Default/index.html.twig) du bundle pour avoir la barre de développement :

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

<html>

<body>

Hello {{ name }}!

</body>

</html>

- Processus de création du bundle par Symfony2

1. Symfony2 a généré la structure du bundle dans src/

2. Symfony2 a enregistré notre bundle auprès du Kernel

dans app/AppKernel.php

3. Symfony2 a enregistré nos routes auprès du Routeur app/config/routing.yml

le path src/OC/PlatformBundle/Resources/config/routing.yml

- Processus de développement du bundle

- formatage du route

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

oc\_platform\_view\_slug: (nom unique de la route)

path: /platform/{year}/{slug}.{format} (URL)

defaults:

\_controller: OCPlatformBundle:Advert:viewSlug (paramètres dont le controleur définit le bundle:le controler:l'action)

format: html

requirements:

year: \d{4}

format: html|xml

Autre exemple

# 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+

Avec transfert de préfixe dans app/config/routing.yml :

# app/config/routing.yml

oc\_platform:

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

prefix: /platform

- formatage du contrôleur

Création du fichier src/OC/PlatformBundle/Controller/AdvertController.php

<?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."."

);

}

}

- formatage du modèle

- formatage de la vue

Créer 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\_var }} !</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>

Nettoyage des fichiers exemples :

supprimer :

Le contrôleur Controller/DefaultController.php

Le répertoire de vues Resources/views/Default

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

Le répertoire src/AppBundle

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

Les lignes 5 à 7 du fichier app/config/routing.yml, celles qui importent le fichier de route du bundle AppBundle (app: resource: "@AppBundle/Controller/" type: annotation" )

Notes :

près 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, Symfony2 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 app/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 étrange. Dans ce cas, un petit php app/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 app/cache/dev (ou app/cache/prod suivant l’environnement).

Générer des URL

1. 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 méthodes :

<?php

// Depuis un contrôleur

// Méthode longue

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

// Méthode courte

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

2. 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>

Manipuler l'objet Request

Passe l'objet Request de Symfony en paramètre de la méthode du controller

<?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

);

}

}

Type de paramètres Méthode Symfony Méthode traditionnelle Exemple

Variables d'URL $request->query $\_GET $request->query->get('tag')

Variables de formulaire $request->request $\_POST $request->request->get('tag')

Variables de cookie $request->cookies $\_COOKIE $request->cookies->get('tag')

Variables de serveur $request->server $\_SERVER $request->server->get('REQUEST\_URI')

Variables d'entête $request->headers $\_SERVER['HTTP\_\*'] $request->headers->get('USER\_AGENT')

Paramètres de route $request->attributes n/a On utilise $id dans les arguments de la méthode, mais vous pourriez également faire $request->attributes->get('id')

Autres méthodes de l'objet Request > http://api.symfony.com/2.7/Symfony/Component/HttpFoundation/Request.html

<?php

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

{

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

}

<?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)

{

// 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)

);

}

}

<?php

public function viewAction($id)

{

return $this->render(

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

array('id' => $id)

);

}

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)

{

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

}

Pour bloquer la redirection modifier la valeur du paramètre intercept\_redirects à true dans le fichier app/config/config\_dev.yml.

Changer le Content-type de 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("Je suis une page de test, je n'ai rien à dire");

}

}

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

</head>

<body>

<h1>Affichage d'une annonce</h1>

<p>

{# 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 %}

</p>

<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>

Encore du code :

# 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');

}

}

Gestion des 404

> <https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/personnaliser-les-pages-d-erreur>

Créer la page twig app/Resources/TwigBundle/views/Exception/error404.html.twig

- Ajout d'un bundle

- Objet et Structure

- Commande console

- Configuration et Développement

-

-

- Tests

-

- Recette (Tests env client)

-

- Pré-Production

-

- Livraison (Mise en Production)

-

- TMA

-