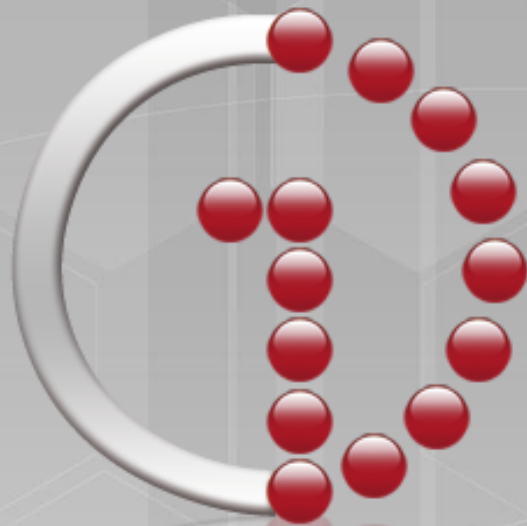


Software PHP

Framework Symfony 3

B3

PARTIE 2 – Les bases de Symfony



CampusID

Sophia Antipolis

Version 1.22

Crédits et références bibliographiques

- [Site web de Symfony](#)
- [Site web de Sensiolabs](#)
- [Livre de référence : Développez votre site web avec le framework Symfony3](#)

- PARTIE 2 – Les bases de Symfony
 - Un premier exemple : Hello World
 - Le routeur
 - Le contrôleur
 - Le moteur de templates Twig
 - Installer un bundle avec Composer
 - Les services

PARTIE 2 – Les bases de Symfony

Un premier exemple : Hello World

Hello World

- L'objectif est de créer un premier exemple sur l'URL :
http://localhost/Symfony/web/app_dev.php/hello-world
- C'est-à-dire ajouter une nouvelle route à notre bundle
OCPlatformBundle

Hello World !

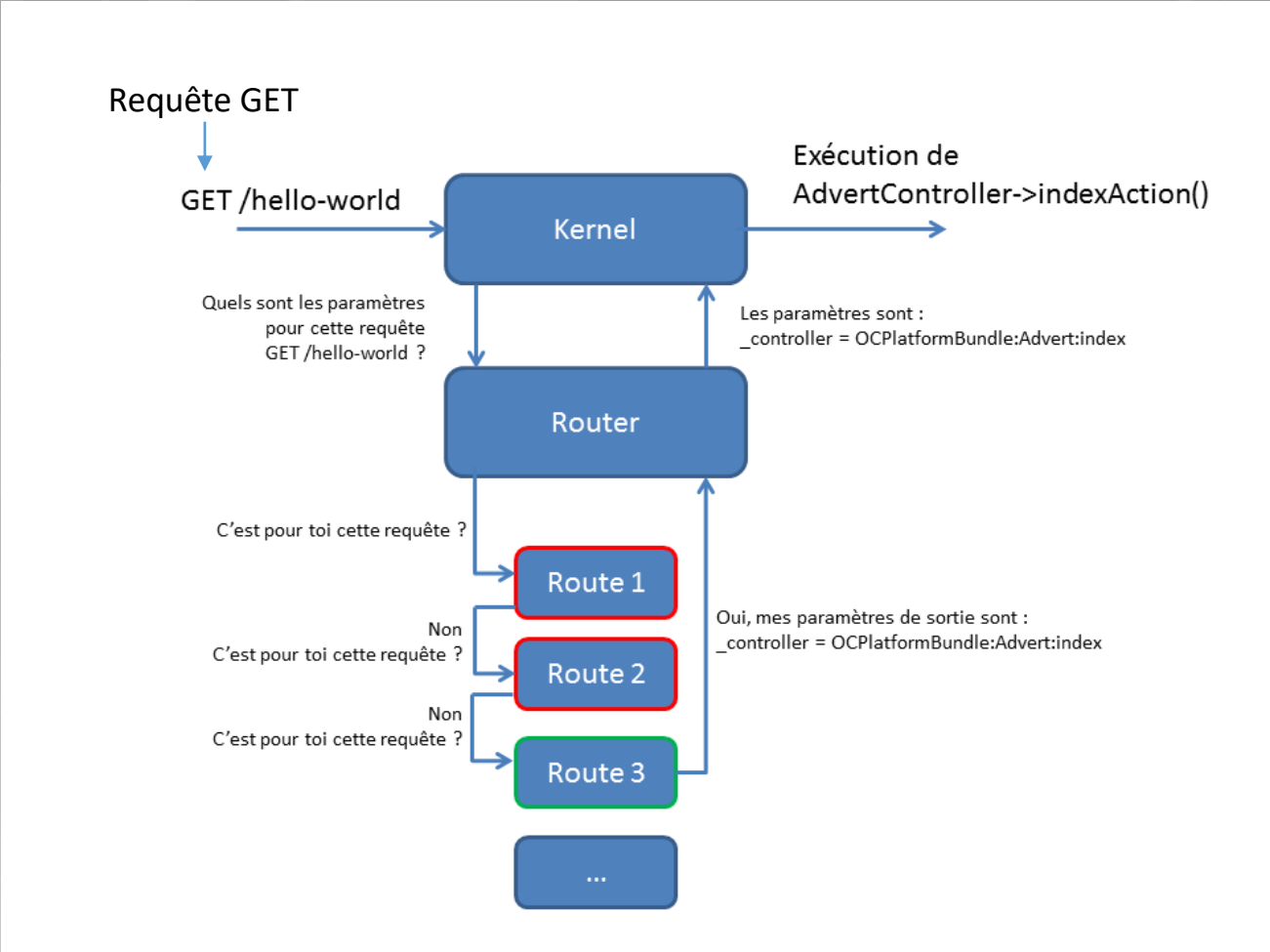
Le Hello World est un grand classique en programmation. Il signifie énormément, car cela veut dire que vous avez réussi à exécuter le programme pour accomplir une tâche simple : afficher ce hello world !

200



- L'objectif du routeur est de faire la correspondance entre une URL et des paramètres, c'est-à-dire d'identifier la route :

Par exemple, nous pourrions avoir une route qui dit que lorsque l'URL appelée est /hello-world, le contrôleur à exécuter est Advert

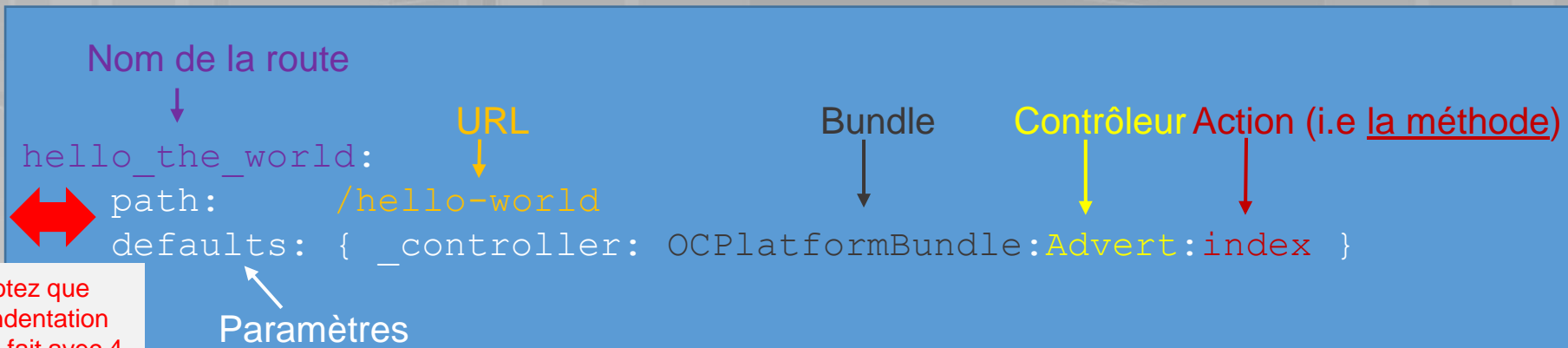


Fichier des routes

- Les routes se définissent dans un simple fichier texte, que Symfony a déjà généré automatiquement pour notre bundle :

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

- Ouvrez le fichier, et ajoutez cette route à la suite de celle qui existe déjà :



Notez que l'indentation se fait avec 4 espaces par niveau, et non avec des tabulations

Fichier contrôleur associé :

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


Fichier des routes

- On a vu également que le fichier de routes du bundle est fourni au routeur grâce au fichier `app/config/routing.yml`

```
oc_platform:
  resource: "@OCPlatformBundle/Resources/config/routing.yml"
  prefix:   /

app:
  resource: "@AppBundle/Controller/"
  type:     annotation
```

Création du contrôleur

- Dans un bundle, les contrôleurs se trouvent dans le répertoire `Controller`
- Le nom des fichiers des contrôleurs doit respecter une convention très simple : il doit commencer par le nom du contrôleur (ici `Advert`), suivi du suffixe `Controller`

- On va donc créer le fichier :

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

- Autre convention : Lorsque, dans la route, on parle de l'action `index`, dans le contrôleur on devra définir la méthode `indexAction()` – Cette convention permet de distinguer les méthodes qui vont être appelées par le noyau (les `xxxAction()`)

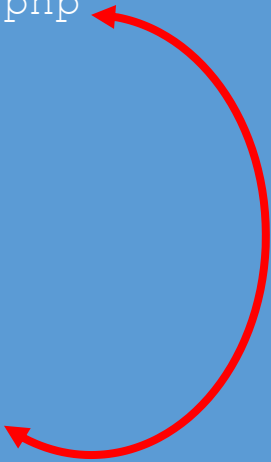
Création du contrôleur

```
<?php
// src/OC/PlatformBundle/Controller/AdvertController.php

// On se place dans l'espace de nom du contrôleur
namespace OC\PlatformBundle\Controller;

// On utilise la classe Response
use Symfony\Component\HttpFoundation\Response;

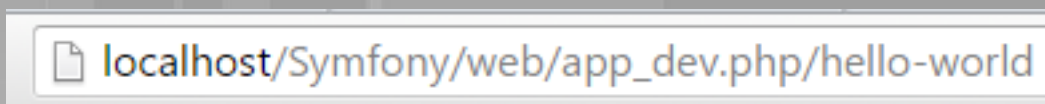
// La classe porte le nom du fichier pour l'autoload
class AdvertController
{
    // Ne pas oublier de mettre le suffixe Action derrière
    // le nom de la méthode
    public function indexAction()
    {
        //L'argument de l'objet Response est le contenu de la page
        // que vous envoyez au visiteur
        return new Response("Notre propre Hello World !");
    }
}
```



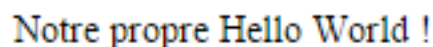
Test du contrôleur

- On peut tester le contrôleur :

http://localhost/Symfony/web/app_dev.php/hello-world



localhost/Symfony/web/app_dev.php/hello-world



Notre propre Hello World !

- Nous avons écrit le contenu de la page dans le contrôleur : ceci ne respecte pas l'architecture MVC : il faut donc utiliser une vue.
- Pour la vue, on va utiliser le moteur de *templates* TWIG qui permet de créer un modèle de page HTML sans utiliser PHP

Création du fichier template

- Le répertoire des templates (ou vues) d'un bundle est le dossier `Resources/views`
- Créez un répertoire `Advert` et ajoutez-y le fichier template `index.html.twig` (et encodez le en UTF-8)

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

```
<!DOCTYPE html>
<html>
  <head>
    <title>Bienvenue sur ma première page avec Symfony !</title>
  </head>
  <body>
    <h1>Hello World !</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>
```

Utilisation du fichier template

- On modifie le contrôleur afin qu'il utilise le template

```
<?php

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

namespace OC\PlatformBundle\Controller;

// On utilise les classes Response et Controller
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Response;

class AdvertController extends Controller
{
    public function indexAction()
    {
        $content = $this->get('templating')->render('OCPlatformBundle:Advert:index.html.twig');
        return new Response($content);
    }
}
```

Va permettre d'utiliser la classe Controller de Symfony

Pour accéder aux méthodes de gestion des templates, nous devons faire hériter notre contrôleur du contrôleur de base de Symfony

Méthode pour récupérer le contenu d'un template.
A noter : *templating* est un service Symfony

Test du fichier template

- On teste :

http://localhost/Symfony/web/app_dev.php/hello-world

Hello World !

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 !

200



Un peu de nettoyage

- Avec tous les éléments générés par Symfony lors de la création du bundle, il y a un peu de nettoyage à faire. On peut supprimer dans notre bundle :
 - 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 Symfony et dont nous ne nous servons pas :
 - Le répertoire `src/AppBundle`
 - La lignes du fichier `app/AppKernel.php`, celle qui active le bundle :
`new AppBundle\AppBundle()`
 - Les lignes des fichiers `app/config/routing.yml` , `app/config/services.yml` et `composer.json` faisant référence au bundle `AppBundle`
- Videz le cache Symfony : `php bin/console cache:clear`

Exercice

- Ajoutez une page (i.e route, action et template) au sein du même contrôleur

http://localhost/Symfony/web/app_dev.php/byebye-world

Bye bye World !

A une prochaine visite ..

200



A retenir

- Le rôle du **routeur** est de **déterminer la route à 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 routeur

Rôle et fonctionnement du routeur

- On a vu précédemment que le rôle du routeur est, à partir d'une URL, de **déterminer quel contrôleur appeler et avec quels arguments**
- On a vu également que le routeur utilise le fichier, généralement situé dans `votreBundle/Resources/config/routing.yml` contenant la définition des routes
- Chaque route fait la correspondance entre une URL et un jeu de paramètres. Le paramètre qui nous intéressera le plus est `_controller`, qui correspond au contrôleur à exécuter
- Dans le cas où le routeur ne trouve aucune route correspondante, le noyau de Symfony déclenche une erreur 404

Les routes de base

- On a déjà vu que l'on peut créer une route en donnant le path (URL) ainsi que le nom du contrôleur à appeler

```
oc_platform_home:  
  path:          /platform  
  defaults:  
    _controller: OCPlatformBundle:Advert:index
```

- On peut aussi donner des paramètres

```
oc_platform_view:  
  path:          /platform/advert/{id}  
  defaults:  
    _controller: OCPlatformBundle:Advert:view
```

- Grâce au paramètre {id} dans le path de notre route, toutes les URL du type /platform/advert/* seront gérées par cette route (A noter : l'URL /platform/advert ne sera pas interceptée, car le paramètre {id} n'est pas renseigné : on aura une erreur)

Exemple de fichiers de routes

3 blocs: chacun correspond à une route



Pour trouver la bonne route, le routeur va les parcourir une par une, dans l'ordre du fichier, et s'arrêter à la première route qui fonctionne

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

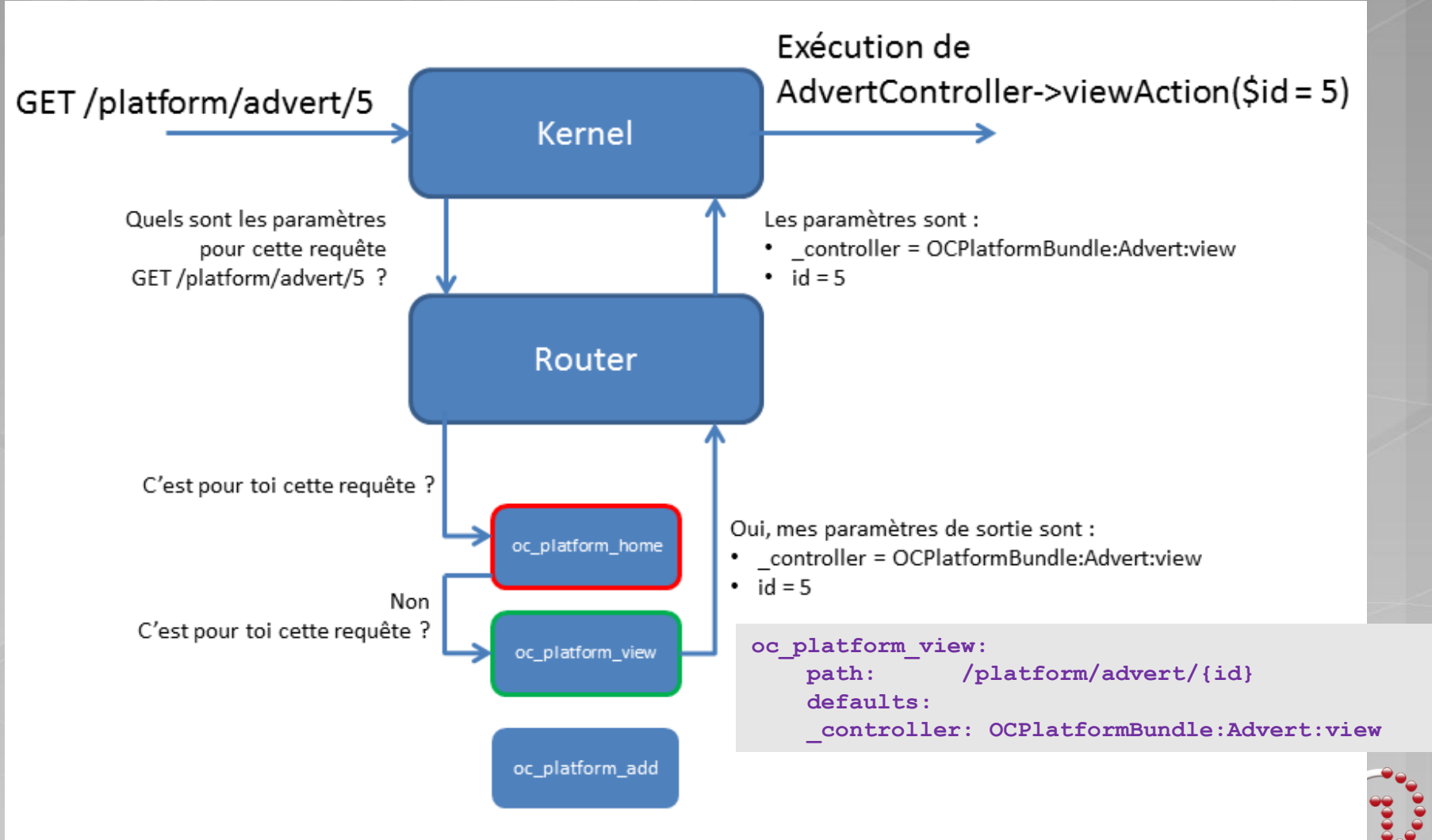
oc_platform_home:
  path:      /platform
  defaults:
    _controller: OCPlatformBundle:Advert:index

oc_platform_view:
  path:      /platform/advert/{id}
  defaults:
    _controller: OCPlatformBundle:Advert:view

oc_platform_add:
  path:      /platform/add
  defaults:
    _controller: OCPlatformBundle:Advert:add
```

Parcours d'une requête dans le routeur

- Par exemple une requête `/platform/advert/5` :

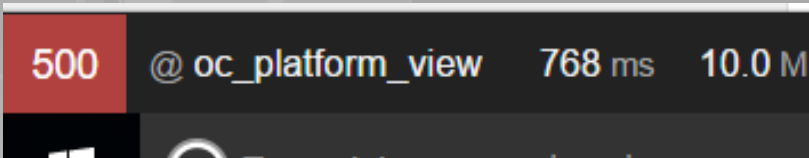


- En cliquant dans la toolbar on peut accéder au profiler qui indique la route et les paramètres passés :

http://localhost/Symfony/web/app_dev.php/platform/advert/5



Erreur 500 : **Internal Server Error** (Erreur interne du serveur)



En cliquant sur oc_plaform_view



Request Attributes	
Key	Value
_controller	"OC\PlatformBundle\Controller\AdvertController::viewAction"
_route	"oc_platform_view"
_route_params	[" id " => " 5 "]
id	" 5 "

- Dans le cas d'un fonctionnement sans erreur (lorsque le contrôleur sera renseigné) on aura :

Request / Response

Performance

Forms

Exception

Logs

Events

Routing

Security

Twig

Doctrine

E-Mails

Routing

oc_platform_view

14

Matched route

Tested routes before match

Route Parameters

Name	Value
id	5

Route Matching Logs

Path to match: /platform/advert/5

#	Route name	Path	Log
1	_wdt	/_wdt/{token}	Path does not match
2	_profiler_home	/_profiler/	Path does not match
3	_profiler_search	/_profiler/search	Path does not match
...
12	_twig_error_test	/_error/{code}.{_format}	Path does not match
13	oc_platform_home	/platform	Path does not match
14	oc_platform_view	/platform/advert/{id}	Route matches!

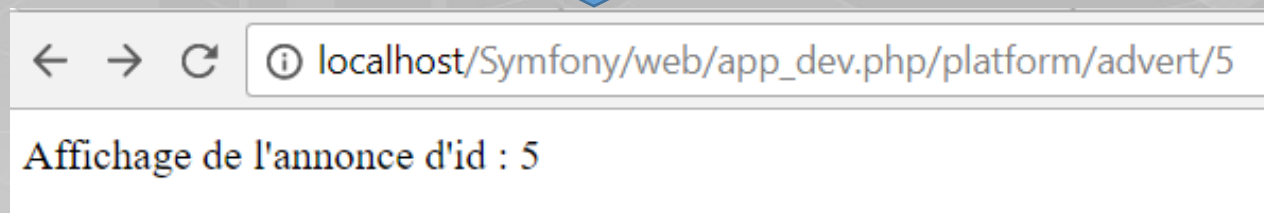
Note: These matching logs are based on the current route configuration, which might differ from the configuration used when profiling this request.

Passage de paramètres au contrôleur

- Les paramètres passés via une route sont transmis au contrôleur
 - Exemple : avec l'URL `/platform/advert/5` le contrôleur disposera de la variable `$id` (du nom du paramètre) valant 5 en argument de la méthode

```
class AdvertController extends Controller
{
    public function viewAction($id)
    {
        return new Response("Affichage de l'annonce d'id : ".$id);
    }
}
```

http://localhost/Symfony/web/app_dev.php/platform/advert/5



Passage de plusieurs paramètres

- Il est également possible de passer plusieurs paramètres

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

oc_platform_view_slug:
  path:      /platform/{year}/{slug}.{format}
  defaults:
    _controller: OCPlatformBundle:Advert:viewSlug
```

- Cette route intercepterait par exemple les URL suivantes :
/platform/2015/webmaster-expert.html et
/platform/2016/symfony.xml, etc
- La classe correspondante :

```
class AdvertController extends Controller
{
    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 . "'."
        );
    }
}
```

L'ordre des paramètres n'a pas d'importance

Contraintes sur paramètres

- Il est également possible de fournir des contraintes pour les paramètres, en les contrôlant avec une expression régulière

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

oc_platform_view_slug:
  path:      /platform/{year}/{slug}.{format}
  defaults:
    _controller: OCPlatformBundle:Advert:viewSlug
  requirements:
    year:      '\d{4}'
    format:    html|xml
```

Ici `year` doit être une suite de 4 digits et `format` doit être `html` ou `xml`

Notes :

- `\d` peut aussi s'écrire `[0-9]` : donc la contrainte `year: '[0-9]{4}'`
- Autre raccourci intéressant : `\w` à la place de `[A-Za-z0-9_]`

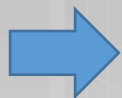
Utiliser des paramètres facultatifs

- On peut placer certains paramètres par défaut pour les rendre facultatifs :

```
# 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
```

L'URL `/platform/2014/webmaster` sera bien interceptée et le paramètre `format` sera mis à sa valeur par défaut, à savoir "html"



On pourrait afficher l'annonce correspondant au slug 'webmaster', créée en 2014 et au format html.

Remarque sur le paramètre `_controlleur`

- Le contrôleur est également un paramètre de route, c'est pour cela qu'il est dans les paramètres par défaut
- On pourrait donc écrire (**mais très mauvais**) :

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

oc_platform_view_slug:
  path:      /platform/{year}/{slug}.{format}/{_controlleur}
  defaults:
    format:    html
  requirements:
    year:      '\d{4}'
    format:    html|xml
```

- Et appeler une URL du genre :

`/platform/2014/webmaster/OCPlatformBundle:Advert:viewSlug`

Utiliser des paramètres systèmes

- Les paramètres systèmes s'utilisent exactement comme des paramètres classiques, mais le Kernel de Symfony va effectuer automatiquement des actions supplémentaires lorsqu'il les détecte :
 - Le paramètre `{_format}` : Lorsqu'il est utilisé (comme le paramètre `{format}` mais avec un underscore) un header, avec le Content-type correspondant, est ajouté à la réponse retournée (ex: `Content-type: application/xml`).
 - Le paramètre `{_locale}` : Lorsqu'il est utilisé, définit la langue dans laquelle l'utilisateur souhaite obtenir la page. Peut-être limité grâce au paramètre `requirements`.
 - Le paramètre `{_controller}` : c'est également un paramètre de route (comme on a vu précédemment : à ne jamais mettre dans le `path`)

Utiliser des paramètres systèmes

- Exemple de réponse xml (l'erreur provient du fait que dans notre exemple on ne renvoie pas du xml correctement formaté)

The screenshot shows a web browser window with the address bar displaying `localhost/Symfony/web/app_dev.php/platform/2065/webmaster.xml`. The browser's developer tools are open, showing a red error message on the left and the network request details on the right.

Error Message:

This page contains the following errors:

error on line 1 at column 1: Document is empty

Below is a rendering of the page up to the first error.

Network Details:

- Name:** webmaster.xml
- General:**
 - Request URL:** `http://localhost/Symfony/web/app_dev.php/platform/2065/webmaster.xml`
 - Request Method:** GET
 - Status Code:** 200 OK
 - Remote Address:** `[:,1]:80`
- Response Headers:**
 - Cache-Control:** no-cache, private
 - Connection:** Keep-Alive
 - Content-Length:** 112
 - Content-Type:** `text/xml; charset=UTF-8` (highlighted with a red box)
 - Date:** Thu, 26 Jan 2017 10:14:34 GMT
 - Keep-Alive:** timeout=5, max=100
 - Server:** Apache/2.4.23 (win64) PHP/5.6.25
 - X-Debug-Token:** 2016hf

Préfixe de route

- Pour éviter les répétitions au début de chaque `path`, il est possible d'ajouter un préfixe dans le fichier `app/config/routing.yml` qui sera ajouté automatiquement en début de chaque route

```
# app/config/routing.yml

oc_platform:
    resource: "@OCPlatformBundle/Resources/config/routing.yml"
    prefix:   /platform
```

- Autre intérêt : si l'on décide de modifier le préfixe des routes cela se fera à un seul endroit dans le fichier

Préfixe de route

- Et il faut modifier en conséquence le fichier
src\OC\PlatformBundle\Resources\config\routing.yml

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

oc_platform_home:
  path:      /
  defaults:
    _controller: OCPlatformBundle:Advert:index

oc_platform_view:
  path:      /advert/{id}
  defaults:
    _controller: OCPlatformBundle:Advert:view

oc_platform_add:
  path:      /add
  defaults:
    _controller: OCPlatformBundle:Advert:add

oc_platform_view_slug:
  path:      /{year}/{slug}.{format}
  defaults:
    _controller: OCPlatformBundle:Advert:viewSlug
    format:      html
  requirements:
    year:      '\d{4}'
    format:    html|xml
```

Générer des URL

- Le routeur a toutes les routes à sa disposition, il est capable d'associer une route à une certaine URL, mais également de reconstruire l'URL correspondant à une certaine route, pour pouvoir, par exemple, modifier l'arborescence du site, stocker l'URL dans une variable ou simplement l'afficher
- Pour générer une URL, vous devez le demander au routeur en lui donnant deux arguments : le nom de la route ainsi que les éventuels paramètres de cette route
- 2 approches :
 - Depuis le contrôleur
 - Depuis une vue

Générer des URL depuis le contrôleur

- Depuis un contrôleur, il faut appeler la méthode

`$this->get('router')->generate()`

```
class AdvertController extends Controller
{
    public function viewAction($id)
    {
        // On veut avoir l'URL de l'annonce d'id $id.
        $url = $this->get('router')->generate(
            'oc_platform_view', // 1er argument : le nom de la route
            array('id' => $id)  // 2e argument : les valeurs des paramètres
                                // (optionnel si pas de paramètre)
        );

        // $url vaut « /platform/advert/$id »
        return new Response("L'URL de l'annonce d'id $id est : ".$url);
    }
}
```

Il existe aussi un raccourci :
`$this->generateURL(...`

http://localhost/Symfony/web/app_dev.php/platform/advert/5

L'URL de l'annonce d'id 5 est : /Symfony/web/app_dev.php/platform/advert/5

URL absolue

- Il est possible de générer une URL absolue (utile par exemple pour envoyer dans un email)

```
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

class AdvertController extends Controller
{
    public function viewAction($id)
    {
        // On veut avoir l'URL de l'annonce d'id $id.
        $url = $this->get('router')->generate(
            'oc_platform_view', // 1er argument : le nom de la route
            array('id' => $id), // 2e argument : les valeurs des paramètres
            UrlGeneratorInterface::ABSOLUTE_URL
        );
        // $url vaut « http://localhost/Symfony/web/app_dev.php/platform/advert/$id »

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

http://localhost/Symfony/web/app_dev.php/platform/advert/5

L'URL de l'annonce d'id 5 est : http://localhost/Symfony/web/app_dev.php/platform/advert/5

Générer des URL depuis une vue

- Depuis une vue (c'est-à-dire depuis un template Twig) il faut utiliser la fonction `path`

```
{# 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>
```

- pour générer une URL absolue depuis Twig, pas de troisième argument, mais on utilise la fonction `url()` au lieu de `path()`

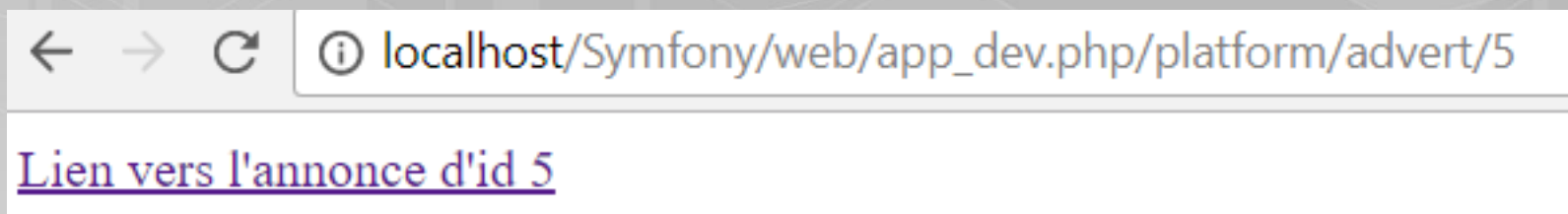
```
{# Dans une vue Twig, en considérant bien sûr que la variable advert_id  
est disponible #}  
  
<a href="{{ url('oc_platform_view', { 'id': advert_id }) }}">  
    Lien vers l'annonce d'id {{ advert_id }}  
</a>
```

Exercice

- Testez le cas avec une URL relative :
 - Créez le template twig comme indiqué page 50
 - Modifiez le contrôleur associé à la route pour utiliser le template
 - Vous pouvez passer le tableau associatif `array('advert_id' => $id)` au template via la méthode `render`

Testez avec

http://localhost/Symfony/web/app_dev.php/platform/advert/5



Exemple applicatif : plateforme d'annonces

- Comme plateforme d'annonces on pourrait avoir :
 - Des annonces (*advert* en anglais) de mission : création d'une maquette, intégration HTML..
 - Nous pourrions consulter, créer, modifier et rechercher des annonces
 - À chaque annonce, nous pourrions lier une image d'illustration
 - À chaque annonce, nous pourrions lier plusieurs candidatures (*application* en anglais) ;
 - Nous aurions plusieurs catégories (Développement, Graphisme, etc.) qui seront liées aux annonces. Nous pourrions créer, modifier et supprimer ces catégories ;
 - À chaque annonce, nous pourrions lier des niveaux de compétence requis (Expert en PHP, maîtrise de Photoshop, etc.).
 - .../...

Exercice : routes page accueil et affichage d'annonce

- Fichier `src/OC/PlatformBundle/Resources/config/routing.yml`
- Donnez les routes pour les pages :
 - d'accueil : On souhaite avoir une URL très simple pour la page d'accueil : `/platform`. On fixera `/platform` comme préfixe (fichier `app/config/routing.yml`) lors du chargement des routes de notre bundle, le path de la route sera `/`. Cette page va lister les dernières annonces. Mais on veut aussi pouvoir parcourir les annonces plus anciennes, donc il nous faut une notion de page (par défaut valant 1). En ajoutant le paramètre facultatif `{page}`
 - de visualisation d'une annonce : Pour la page de visualisation d'une annonce, la route est très simple. Il suffit juste de bien mettre un paramètre `{id}` qui nous servira à récupérer la bonne annonce côté contrôleur.

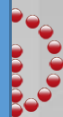
Exemple applicatif : plateforme d'annonces

- Sur le même principe : Ajout, modification et suppression

```
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+'
```



A retenir

- Une **route** est composée au **minimum** de **deux éléments** : l'**URL** à faire correspondre (son **path**), 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 section `requirements`
- Le **routeur** est également capable de **générer des URL** à partir du nom d'une route, et de ses paramètres éventuels

Le contrôleur

Requête et réponse HTTP

- Symfony s'est inspiré des concepts du protocole HTTP. Il existe dans Symfony les classes Request et Response
- Visualisation sous Chrome de la requête et sa réponse (onglet Network)

The screenshot shows the Chrome DevTools Network tab. The address bar displays `localhost/Symfony/web/app_dev.php/platform`. The page content shows a message: "L'URL de l'annonce d'id 5 est : /Symfony/web/app_dev.php/platform/advert/5". The Network tab is open, showing a single request to `platform` with a status of 200 OK. The request headers are visible, including `Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8` and `User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36`. The response headers are also visible, including `Cache-Control: no-cache` and `Server: Apache/2.4.9 (Win32) PHP/5.5.12`.

Annotations on the screenshot:

- A red vertical line points to the status bar text "la réponse".
- A green vertical line points to the "Request Headers" section, labeled "la requête".
- An orange vertical line points to the "Request Headers" section, labeled "les entêtes de la requête".
- A red vertical line points to the "Response Headers" section, labeled "les entêtes de la réponse".

Rôle du contrôleur

- Le rôle du contrôleur va être d'utiliser des services, les modèles et appeler la vue.
- Concrètement, on l'a vu précédemment, il va avoir en charge de retourner une instance de la classe `Response`
 - Exemple de contrôleur simple :


```
<?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

- Il y a deux types de paramètres dans une requête :
 - Les **paramètres contenus dans les routes** : Ces paramètres seront récupérés par la route et transmis au contrôleur sous la forme d'arguments
 - Exemple requête du type `/platform/advert/5`
 - Les **paramètres hors route**
 - Par exemple sur une requête de la forme `/platform/advert/5?tag=developer`, il faut un moyen pour récupérer ce paramètre `tag` : C'est ici qu'intervient l'objet `Request`

Paramètres contenus dans une route

- Reprenons la route `oc_platform_view`

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

oc_platform_view:
  path:      /advert/{id}
  defaults:
    _controller: OCPlatformBundle:Advert:view
  requirements:
    id: \d+
```

- Ici, le paramètre `{id}` de la requête est récupéré par la route, qui le transforme en argument `$id` pour le contrôleur

```
class AdvertController extends Controller
{
    // ...

    public function viewAction($id)
    {
        return new Response("Affichage de l'annonce d'id : ".$id);
    }
}
```


Paramètres hors route

- Il faut dans ce cas ajouter un `use` pour accéder à la classe `Request` et l'ajouter à la méthode du contrôleur comme argument supplémentaire :

```
<?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)
    {
        // Vous avez accès à la requête HTTP via $request
    }
}
```



- Après avoir demandé au routeur quel contrôleur exécuter, et avant de l'exécuter effectivement, le Kernel regarde si l'un des arguments de la méthode est typé avec `Request` . Si c'est le cas, il ajoute la requête aux arguments avant d'exécuter le contrôleur

Paramètres hors route

- Récupération des paramètres contenus dans l'URL :

```
<?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 le paramètre tag
        $tag = $request->query->get('tag');

        return new Response(
            "Affichage de l'annonce d'id : ".$id.", avec le tag : ".$tag
        );
    }
}
```

/platform/advert/5?tag=developper

Affichage de l'annonce d'id : 5, avec le tag : developper

Types de paramètres récupérables

Type de paramètres	Méthode Symfony	Méthode traditionnelle	Exemple
Variables d'URL	<code>\$request->query</code>	<code>\$_GET</code>	<code>\$request->query->get('tag')</code>
Variables de formulaire	<code>\$request->request</code>	<code>\$_POST</code>	<code>\$request->request->get('tag')</code>
Variables de cookie	<code>\$request->cookies</code>	<code>\$_COOKIE</code>	<code>\$request->cookies->get('tag')</code>
Variables de serveur	<code>\$request->server</code>	<code>\$_SERVER</code>	<code>\$request->server->get('REQUEST_URI')</code>
Variables d'entête	<code>\$request->headers</code>	<code>\$_SERVER['HTTP_*']</code>	<code>\$request->headers->get('USER_AGENT')</code>
Paramètres de route	<code>\$request->attributes</code>	n/a	On utilise <code>\$id</code> dans les arguments de la méthode, mais vous pourriez également écrire : <code>\$request->attributes->get('id')</code>

En cas de paramètre non défini dans l'URL, `get()` retournera une chaîne vide, et non une erreur

Autres méthodes de l'objet Request

- L'objet `Request` ne se limite pas à la récupération de paramètres. Il permet de savoir plusieurs choses intéressantes à propos de la requête en cours :
 - Récupérer la méthode de la requête HTTP : Pour savoir si la page a été récupérée via `GET` (clic sur un lien) ou via `POST` (envoi d'un formulaire), il existe la méthode `$request->isMethod()`
 - Savoir si la requête est une requête AJAX :
`$request->isXmlHttpRequest()`
- Liste exhaustive :
<http://api.symfony.com/3.0/Symfony/Component/HttpFoundation/Request.html>

Manipuler l'objet Response

- L'architecture MVC préconise que la réponse soit contenue dans une vue
 - Le contrôleur dispose de la méthode `render()`
 - Elle prend en paramètres le nom du template et des variables dans un tableau associatif, puis s'occupe de tout

```
<?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,
    ));
}
```

Manipuler l'objet Response

- Exemple de template Twig associé :

```
{# 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>
```

http://localhost/Symfony/web/app_dev.php/platform/advert/5?tag=developpeur

Hello Annonce n°5 !

Tag éventuel : developpeur

Réponse et redirection

- Il est possible de créer une redirection avec la méthode `redirectToRoute` qui prend directement en argument la route vers laquelle rediriger

```
<?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)
    {
        return $this->redirectToRoute('oc_platform_home');
    }
}
```

En allant à l'adresse </platform/advert/5> vous serez redirigés vers l'accueil

Intercepter les redirections

- On peut demander à Symfony d'intercepter les redirections afin d'afficher des informations sur son déroulement. Pour cela modifier la ligne suivante dans le fichier `\app\config\config_dev.yml`

```
web_profiler:  
    toolbar: true  
    intercept_redirects: false
```



```
web_profiler:  
    toolbar: true  
    intercept_redirects: true
```



This request redirects to /Symfony/web/app_dev.php/platform.

The redirect was intercepted by the web debug toolbar to help debugging. For more information, see the

Content type

- Lorsque vous retournez autre chose que du HTML, il faut que vous changiez l'en-tête `Content-type` de la réponse.
- Exemple : vous recevez une requête AJAX et souhaitez retourner un tableau en JSON

```
<?php

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

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\JsonResponse;

class AdvertController extends Controller
{
    public function viewAction($id)
    {
        return new JsonResponse(array('id' => $id));
    }
}
```

- Dans Symfony, il existe un objet `Session` qui permet de gérer la session (méthodes `get()` et `set()`)

```
<?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)
    {
        // On récupère notre paramètre tag
        $tag = $request->query->get('tag');

        // 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);

        return $this->render('OCPlatformBundle:Advert:view.html.twig', array(
            'id' => $id,
            'tag' => $tag,
        ));
    }
}
```



- On peut voir les variables de sessions dans le *profiler*

Profiler token [9e470d](#)

Kernel name app

Environment dev

Debug enabled


PHP version 5.6.25 [View phpinfo\(\)](#)

PHP Extensions xdebug accel

PHP SAPI apache2handler

Resources [Read Symfony 3.2.2 Docs](#)

Help [Symfony Support Channels](#)

 3.2.2



RequestResponseSessionFlashes

Session Metadata

Key	Value
Created	Fri, 09 Sep 16 17:27:05 +0200
Last used	Fri, 09 Sep 16 17:36:21 +0200
Lifetime	0

Session Attributes

Attribute	Value
user_id	91

Cliquez sur "sf" puis sur "Profiler token"

Session – Messages flash

- L'objet Session permet de créer ce que l'on appelle « messages flash » c'est à dire une variable de session qui ne dure que le temps d'une seule page
- Exemple :
 - La page qui traite un formulaire définit un message flash (« Annonce bien enregistrée » par exemple) puis redirige vers la page de visualisation de l'annonce nouvellement créée
 - Sur cette page, le message flash s'affiche, et est détruit de la session. Alors si l'on change de page ou qu'on l'actualise, le message flash ne sera plus présent

Session – Messages flash

- Dans l'action on ajoute des messages ...

.../...

```
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));
}
```

Session – Messages flash

- ... que l'on affiche ensuite avec la vue

```
{# 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>

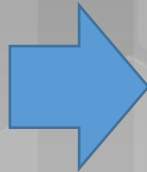
    <p>
      Ici nous pourrions lire l'annonce ayant comme id : {{ id }}<br />
      Mais pour l'instant, nous ne savons pas encore le faire, cela viendra !
    </p>
  </body>
</html>
```

La variable Twig `app` est une variable globale, disponible partout dans vos vues

Session – Messages flash

http://localhost/Symfony/web/app_dev.php/platform/advert/5

http://localhost/Symfony/web/app_dev.php/platform/add



Affichage de l'annonce n°5 !

Message flash : Annonce bien enregistrée

Message flash : Oui oui, elle est bien enregistrée !

Ici nous pourrions lire l'annonce ayant comme id : 5
Mais pour l'instant, nous ne savons pas encore le faire, cela viendra !



F5
(reload de page)

Affichage de l'annonce n°5 !

Ici nous pourrions lire l'annonce ayant comme id : 5
Mais pour l'instant, nous ne savons pas encore le faire, cela viendra !

Application : le contrôleur de notre plateforme

```
<?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.');
```


Application : le contrôleur de notre plateforme

```
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 cette 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');
```

}

Application : le contrôleur de notre plateforme

```
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');
}
}
```

Commentaires

- L'erreur 404 : Dès qu'une erreur de ce type arrive le noyau l'attrape et génère une page d'erreur 404
- Les méthodes vont être appelées par le noyau : elles doivent donc respecter le nom et les arguments que nous avons définis dans nos routes et se trouver dans le scope `public`.
- Pour les méthodes internes vous ne devez pas les suffixer avec « Action » (afin de ne pas confondre)

A retenir

- Le rôle du **contrôleur** est de **retourner un objet Response** : ceci est obligatoire
- Le contrôleur **construit la réponse en fonction des données** qu'il a en entrée : paramètre de route et objet Request.
- 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 moteur de templates Twig

Pourquoi un template ?

- Les *templates* sont **utilisés** par les vues
- Leur objectif est de **séparer le code PHP du code HTML**
 - Lorsque vous faites du PHP, vous n'avez pas des balises HTML qui gênent la lecture de votre code PHP
 - De même, lorsque un designer fait du HTML, il ne sera pas gêné par le code PHP
- **Twig :**
 - Offre une syntaxe plus concise et plus claire
 - Sécurise les variables automatiquement
 - Permet de faire de l'héritage de templates
 - Mise en cache pour améliorer le passage vers HTML

Retourner une réponse HTTP

- On l'a vu, le contrôleur peut retourner une réponse HTTP toute faite, dont le contenu est celui d'un certain template auquel on peut passer des arguments :

```
<?php
// Depuis un contrôleur

return $this->render('OCPlatformBundle:Advert:index.html.twig', array(
    'var1' => $var1,
    'var2' => $var2
));
```

Récupérer le contenu d'un template

- Il peut également récupérer le contenu d'un template en texte

```
<?php
// Depuis un contrôleur

$contentu = $this->renderView('OCPlatformBundle:Advert:email.txt.twig',
array('pseudo' => $var1));

// Puis on envoie l'e-mail, par exemple :
mail('moi@campusid.com', 'Inscription OK', $contentu);
```

- Template :

```
{# src/OC/PlatformBundle/Resources/views/Advert/email.txt.twig #}

Bonjour {{ pseudo }},

Toute l'équipe du site se joint à moi pour vous souhaiter
la bienvenue sur notre site !

Revenez nous voir souvent !
```


Afficher des variables

- Afficher une variable se fait avec les doubles accolades `{{ ... }}`

Description	Exemple Twig	Équivalent PHP
Afficher une variable	Pseudo : <code>{{ pseudo }}</code>	Pseudo : <code><?php echo \$pseudo; ?></code>
Afficher l'index d'un tableau	Identifiant : <code>{{ user['id'] }}</code>	Identifiant : <code><?php echo \$user['id']; ?></code>
Afficher l'attribut d'un objet, dont le getter respecte la convention \$objet->getAttribut()	Identifiant : <code>{{ user.id }}</code>	Identifiant : <code><?php echo \$user->getId(); ?></code>
Afficher une variable en lui appliquant un filtre. Ici, « upper » met tout en majuscules :	Pseudo en majuscules : <code>{{ pseudo upper }}</code>	Pseudo en lettre majuscules : <code><?php echo strtoupper(\$pseudo); ?></code>
Afficher une variable en combinant les filtres. « striptags » supprime les balises HTML. « title » met la première lettre de chaque mot en majuscule. Notez l'ordre d'application des filtres : striptags est appliqué, puis title.	Message : <code>{{ news.texte striptags title }}</code>	Message : <code><?php echo ucwords(strip_tags(\$news->getTexte())); ?></code>
Utiliser un filtre avec des arguments. Attention, il faut que date soit un objet de type Datetime ici.	Date : <code>{{ date date('d/m/Y') }}</code>	Date : <code><?php echo \$date->format('d/m/Y'); ?></code>
Concaténer	Identité : <code>{{ nom ~ " " ~ prenom }}</code>	Identité : <code><?php echo \$nom.' '.\$prenom; ?></code>

Précisions sur la syntaxe `{{ objet.attribut }}`

- Le fonctionnement de la syntaxe `{{ objet.attribut }}` est le suivant :
 - Elle vérifie si `objet` est un tableau, et si `attribut` en est un index valide. Si c'est le cas, elle affiche `objet['attribut']`
 - Sinon, et si `objet` est un objet, elle vérifie si `attribut` en est un attribut valide (public donc). Si c'est le cas, elle affiche `objet->attribut`
 - Sinon, et si `objet` est un objet, elle vérifie si `attribut()` en est une méthode valide (publique donc). Si c'est le cas, elle affiche `objet->attribut()`
 - Sinon, et si `objet` est un objet, elle vérifie si `getAttribut()` en est une méthode valide. Si c'est le cas, elle affiche `objet->getAttribut()`
 - Sinon, et si `objet` est un objet, elle vérifie si `isAttribut()` en est une méthode valide. Si c'est le cas, elle affiche `objet->isAttribut()`
 - Sinon, elle n'affiche rien et retourne `null`

Exercice

- Modifiez la méthode `viewAction` pour qu'elle passe un tableau (`$annonce`) comme argument à la méthode `render`
- Modifiez la vue associée
- Testez l'écriture `annonce['id']` et `annonce.id`. Vérifiez que les deux fonctionnent à l'identique

- Il y a quelques filtres disponibles nativement avec Twig :

Filtre	Description	Exemple Twig
upper	Met toutes les lettres en majuscules.	<code>{{ var upper }}</code>
striptags	Supprime toutes les balises XML.	<code>{{ var striptags }}</code>
date	Formate la date selon le format donné en argument. La variable en entrée doit être une instance de Datetime.	<code>{{ date date('d/m/Y') }}</code> Date d'aujourd'hui : <code>{{ "now" date('d/m/Y') }}</code>
format	Insère des variables dans un texte, équivalent à printf	<code>{{ "Il y a %s pommes et %s poires" format(153, nb_poires) }}</code>
length	Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne.	Longueur de la variable : <code>{{ texte length }}</code> Nombre d'éléments du tableau : <code>{{ tableau length }}</code>

<http://twig.sensiolabs.org/doc/filters/index.html>

- Dans tous les exemples précédents, les variables ont déjà été protégées par **Twig** qui **applique par défaut un filtre sur toutes les variables à afficher**, afin de les protéger de balises HTML malencontreuses. Ainsi, si le pseudo d'un des membres contient un « < » par exemple, lorsque vous écrivez `{{ pseudo }}` celui-ci est échappé, et le texte généré est en réalité « `mon<pseudo` » au lieu de « `mon<pseudo` », ce qui poserait problème dans une structure HTML. Et donc à savoir : **inutile de protéger vos variables en amont, Twig s'occupe de tout en fin de chaîne !**
- Et dans le cas où vous voulez afficher volontairement une variable qui contient du HTML (JavaScript, etc.), et que vous ne voulez pas que Twig l'échappe, il vous faut utiliser le filtre `raw` comme ceci : `{{ ma_variable_html|raw }}`. Avec ce filtre, Twig désactive localement la protection HTML, et affiche la variable en brut, quel que soit ce qu'elle contient

Variables globales

- Symfony enregistre **par défaut une variable globale** `{{ app }}` dans **Twig** pour nous faciliter la vie.
- Voici la liste de ses attributs, qui sont donc disponibles dans tous les templates :

Variable	Description
<code>{{ app.request }}</code>	La requête « request »
<code>{{ app.session }}</code>	Le service « session »
<code>{{ app.environment }}</code>	L'environnement courant : « dev » ou « prod »
<code>{{ app.debug }}</code>	True si le mode debug est activé, False sinon.
<code>{{ app.user }}</code>	L'utilisateur courant

Variables globales utilisateur

- On peut également créer ses propres variables globales en éditant le fichier de configuration et les paramètres de l'application

```
# app/config/config.yml

twig:
  globals:
    webmaster: %app_webmaster%
```

```
# app/config/parameters.yml

parameters:
  # ...
  app_webmaster: LB
```

- Ainsi, la variable `{{ webmaster }}` sera injectée dans toutes nos vues, et donc utilisable comme ceci :

```
<footer>Responsable du site : {{ webmaster }}.</footer>
```

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 %}
```

Equivalent au code PHP suivant :

```
<?php if($membre->getAge() < 12) { ?>  
    Il faut avoir au moins 12 ans pour ce film.  
<?php } elseif($membre->getAge() < 18) { ?>  
    OK bon film.  
<?php } else { ?>  
    Un peu vieux pour voir ce film non ?  
<?php } ?>
```


Structures de contrôle et expressions

■ Boucle : { % for % }

```
<ul>
  {% for membre in liste_membres %}
    <li>{{ membre.pseudo }}</li>
  {% else %}
    <li>Pas d'utilisateur trouvé.</li>
  {% endfor %}
</ul>
```

Equivalent au code PHP suivant :

```
<ul>
<?php if(count($liste_membres) > 0) {
  foreach($liste_membres as $membre) {
    echo '<li>'.$membre->getPseudo().'</li>';
  }
} else { ?>
  <li>Pas d'utilisateur trouvé.</li>
<?php } ?>
</ul>
```

Structures de contrôle et expressions

- Boucle : `{% for %}`

```
<select>
  {% for valeur, option in liste_options %}
    <option value="{{ valeur }}">{{ option }}</option>
  {% endfor %}
</select>
```

Equivalent au code PHP suivant :

```
<?php
foreach($liste_options as $valeur => $option) {
    // ...
}
```

Structures de contrôle et expressions

- La structure `{% for %}` définit une variable `{{ loop }}` au sein de la boucle, qui contient les attributs suivants :

Variable	Description
<code>{{ loop.index }}</code>	Le numéro de l'itération courante (en commençant par 1).
<code>{{ loop.index0 }}</code>	Le numéro de l'itération courante (en commençant par 0).
<code>{{ loop.revindex }}</code>	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 1).
<code>{{ loop.revindex0 }}</code>	Le nombre d'itérations restantes avant la fin de la boucle (en finissant par 0).
<code>{{ loop.first }}</code>	true si c'est la première itération, false sinon.
<code>{{ loop.last }}</code>	true si c'est la dernière itération, false sinon.
<code>{{ loop.length }}</code>	Le nombre total d'itérations dans la boucle.

Structures de contrôle et expressions

- Définition : { % set % }

```
{% set foo = 'bar' %}
```

Equivalent au code PHP suivant :

```
<?php $foo = 'bar'; ?>
```

Héritage de templates

- Le principe est simple :
 - Le **template père** (communément appelé *layout*) contient le **design du site ainsi que quelques trous** (appelés *blocks*)
 - Les **templates fils** vont **remplir ces trous**.
- Les fils vont donc venir hériter du père en remplaçant certains éléments par leur propre contenu.
- L'avantage est que les templates fils peuvent modifier plusieurs blocs du template père

Exemple d'héritage de templates

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

Template père

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>{% block title %}OC Plateforme{% endblock %}</title>
  </head>
  <body>

    {% block body %}
    {% endblock %}

  </body>
</html>
```

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

Template fils

```
{% 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 %}
```

Nom du template père

- On place le template père directement dans `src/.../Resources/views/` du bundle ou bien `app/Resources/views` de l'application (i.e du site internet) et non dans un sous répertoire car il est inutile de mettre dans un sous-répertoire les templates qui ne concernent pas un contrôleur particulier et qui peuvent être réutilisés par plusieurs contrôleurs
- Attention à la notation pour accéder à ce template : ce n'est plus `OCPlatformBundle:MonController:layout.html.twig`, mais `OCPlatformBundle::layout.html.twig` (on enlève juste la partie qui correspond au répertoire `MonController`, on garde les `::`)

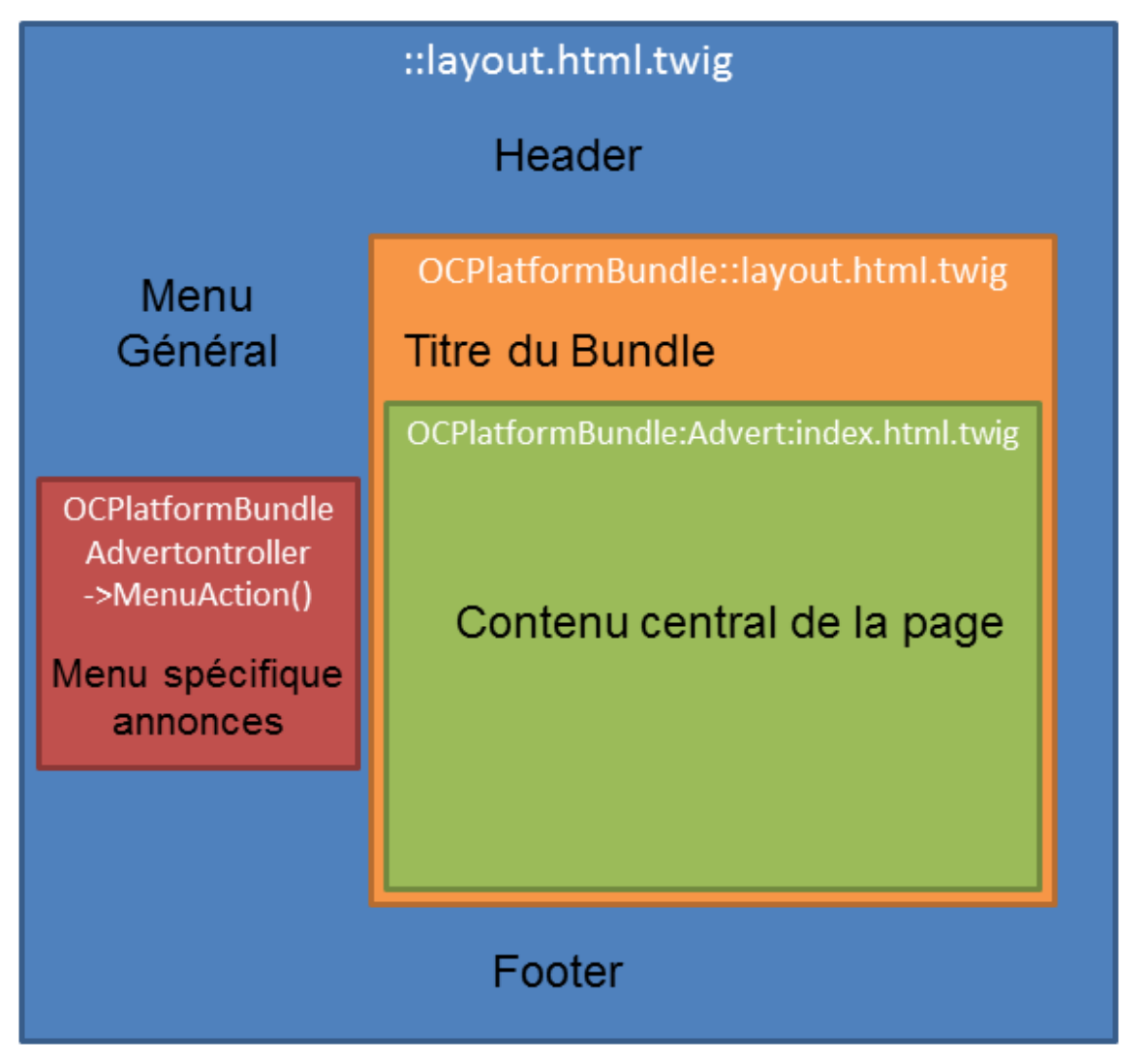
La balise `{% block %}` côté père

- Pour définir un « trou » (bloc) dans le template père, nous avons utilisé la balise `{% block %}`. **Un bloc doit avoir un nom afin que le template fils puisse modifier tel ou tel bloc de façon nominative.** La base, c'est juste de faire `{% block nom_du_block %}{% endblock %}` et c'est ce que nous avons fait pour le body. Mais vous pouvez insérer un texte par défaut dans les blocs, comme on l'a fait pour le titre. C'est utile pour deux cas de figure :
 - Lorsque le template fils ne redéfinit pas ce bloc. Plutôt que de n'avoir rien d'écrit, vous aurez cette valeur par défaut.
 - Lorsque les templates fils veulent réutiliser une valeur commune. Par exemple, si vous souhaitez que le titre de toutes les pages de votre site commence par « OC Plateforme », alors depuis les templates fils, vous pouvez utiliser `{{ parent() }}` qui permet d'utiliser le contenu par défaut du bloc côté père. Regardez, nous l'avons fait pour le titre dans le template fils.

La balise `{% block %}` côté fils et héritage en cascade

- Elle se définit exactement comme dans le template père, sauf que cette fois-ci on y met notre contenu. Mais étant donné que les blocs se définissent et se remplissent de la même façon on peut hériter en cascade
- Pour bien organiser ses templates, une bonne pratique consiste à faire de **l'héritage de templates sur trois niveaux**, chacun des niveaux remplissant un rôle particulier :
 - **Layout général** : c'est le **design de votre site**, indépendamment de vos bundles. Il contient l'en-tête, le pied de page, etc. La structure de votre site donc (c'est notre template père). On nomme ce fichier par exemple `app/Resources/views/layout.html.twig`
 - **Layout du bundle** : il hérite du layout général et **contient les parties communes à toutes les pages d'un même bundle**. Par exemple répertoire `src/OC/PlatformBundle/Resources/views/`
 - **Template de page** : il hérite du layout du bundle et **contient le contenu central de votre page**. Par exemple répertoire `src/OC/PlatformBundle/Resources/views/Advert/`

Bonne pratique : le triple héritage



Inclusion de templates

- Il est également possible de faire de l'inclusion de templates, par exemple pour utiliser un même morceau de code (formulaire...)

```
{# 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 %}
```

- A l'intérieur du template inclus, on aura toutes les variables qui sont disponibles dans le template qui fait l'inclusion (ex : add.html.twig)

Inclusion de contrôleur

- Dans certains cas, depuis le template qui fait l'inclusion, vous voudrez inclure un autre template, mais n'aurez pas les variables nécessaires pour lui (exemple : le menu spécifique annonce)
- C'est depuis le layout général que l'on va inclure non pas un template (nous n'aurions pas les variables à lui donner) mais une action d'un contrôleur : Le contrôleur va créer les variables dont il a besoin, et les donner à son template, pour ensuite être inclus là où on le veut
- La mise en œuvre est simple, il suffit d'utiliser la fonction `{{ render() }}` à la place de la fonction `{{ include() }}`

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

- Attention néanmoins : impact sur les performances

Inclusion de contrôleur

■ Exemple de layout du bundle

```
{# 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>
```

Inclusion de contrôleur

- Côté contrôleur, on ajoute la méthode `menuAction()` qui retourne une réponse avec le template menu comme contenu :

```
// ...

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
        ));
    }
}
```

Inclusion de contrôleur

- Exemple de ce que pourrait être le template menu.html.twig

```
{# 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"> ← Note : classes Bootstrap
  {% for advert in listAdverts %}
    <li>
      <a href="{{ path('oc_platform_view', {'id': advert.id}) }}">
        {{ advert.title }}
      </a>
    </li>
  {% endfor %}
</ul>
```

- Modifiez le fichier `view.html.twig` pour qu'il hérite du layout du bundle et puisse afficher le menu au dessus de l'annonce :

- [Recherche développeur Symfony](#)
- [Mission de webmaster](#)
- [Offre de stage webdesigner](#)

Affichage de l'annonce n°5 !

Ici nous pourrions lire l'annonce ayant comme id : 5

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

Templates de notre plateforme d'annonces

- Étant donné que l'on n'a pas encore accès à la base de données, on va travailler avec des variables vides : cela va se remplir par la suite, mais le fait d'employer des variables vides va nous permettre dès maintenant de construire le template.
- Pour la mise en place CSS on va utiliser Bootstrap (framework CSS : <http://getbootstrap.com/>)

Templates de notre plateforme d'annonces

- Layout général de l'application (c.ad. du site)

```
{# 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>
```

Templates de notre plateforme d'annonces

```
<body>
  <div class="container">
    <div id="header" class="jumbotron">
      <h1>Ma plateforme d'annonces</h1>
      <p>
        Ce projet est propulsé par Symfony.
      </p>
      <p>
        <a class="btn btn-primary btn-lg" href="http://www.campus-
id.com">
          Ecole supérieure d'informatique»
        </a>
      </p>
    </div>
```

Templates de notre plateforme d'annonces

```
<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>
```

Templates de notre plateforme d'annonces

```
<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>
```

Templates de notre plateforme d'annonces

- Layout du bundle
 - Comme on l'a dit, ce template va hériter du layout général, ajouter la petite touche personnelle au bundle Advert, puis être hérité à son tour par les templates finaux.
 - Une bonne pratique consiste à préfixer le nom des blocs par le nom du bundle courant.

Templates de notre plateforme d'annonces

■ 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
    #}
    <h2>Annonces</h2>

    <hr>

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

{% endblock %}
```

Templates de notre plateforme d'annonces

■ Template de la page d'accueil

```
{# 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 %}
```


Contrôleur de notre plateforme d'annonces

- L'action menu doit également être ajoutée :

```
<?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
    ));
}
```

Vue menu de notre plateforme d'annonces

- il nous faut modifier l'action `indexAction()` du contrôleur pour passer une variable `{{ listAdverts }}` à la vue (annonces comme si elles venaient de la BDD ...)

```
<?php
// src/OC/PlatformBundle/Controller/AdvertController.php

// ...

public function indexAction($page)
{
    // ...

    // Notre liste d'annonce en dur
    $listAdverts = array(
        array(
            'title'    => 'Recherche développeur Symfony',
            'id'       => 1,
            'author'   => 'Alexandre',
            'content'  => 'Nous recherchons un développeur Symfony débutant sur
Lyon. Blabla...',
            'date'     => new \DateTime()),
```

Vue menu de notre plateforme d'annonces

- ... la suite :

```
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
));
}
```

Vue menu de notre plateforme d'annonces

- Et la vue associée :

```
{# 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>
```

Vue des annonces

- Le fichier de la vue :

```
{# 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>
```

Vue des annonces

```
<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 %}
```

Vue des annonces

■ L'adaptation du contrôleur :

```
<?php
// src/OC/PlatformBundle/Controller/AdvertController.php

// ...

public function viewAction($id)
{
    $advert = array(
        'title'    => 'Recherche développeur 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
    ));
}
```

La vue pour l'édition d'annonces

- Voici le fichier

```
{# 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") }}

{% endblock %}
```


La vue pour l'édition d'annonces

```
<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 %}
```

Le contrôleur pour l'édition d'annonces

- Voici le fichier modifié

```
<?php
// src/OC/PlatformBundle/Controller/AdvertController.php

public function editAction($id, Request $request)
{
    // ...

    $advert = array(
        'title'    => 'Recherche développeur 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
    ));
}
```

Le formulaire commun à l'ajout/édition d'annonces

- Voici le fichier

```
{# 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>
```

La vue pour l'ajout d'annonces

- Voici le fichier

```
{# 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 %}
```

Le contrôleur pour l'ajout d'annonces

- Le fichier avec un exemple d'ajout d'annonce

```
public function addAction(Request $request)
{
    // Si la requête est en POST, c'est que le visiteur a soumis le
    formulaire
    if ($request->isMethod('POST')) {
        $request->getSession()->getFlashBag()->add('notice', 'Annonce bien
        enregistrée.');
```

// Puis on redirige vers la page de visualisation de cette
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');
```

La vue de suppression d'une annonce

- Fichier exemple simple

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

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

{% block body %}

    <h2>Supprimer </h2>

    <p>
        Cette annonce a été supprimée
    </p>

{% endblock %}
```

Test de notre plateforme d'annonces

- Rendez vous à l'adresse :

http://localhost/Symfony/web/app_dev.php/platform

Ma plateforme d'annonces

Ce projet est propulsé par Symfony.

Ecole supérieure d'informatique

Les annonces

[Accueil](#)

[Ajouter une annonce](#)

Dernières annonces

[Recherche développeur Symfony](#)

[Mission de webmaster](#)

[Offre de stage webdesigner](#)

Annonces

Liste des annonces

- [Recherche développeur Symfony](#) par Alexandre, le 31/05/2017
- [Mission de webmaster](#) par Hugo, le 31/05/2017
- [Offre de stage webdesigner](#) par Mathieu, le 31/05/2017

The sky's the limit © 2017 and beyond.

Test de notre plateforme d'annonces

- Visualisation d'une annonce

Les annonces

[Accueil](#)

[Ajouter une annonce](#)

Dernières annonces

[Recherche développeur Symfony](#)

[Mission de webmaster](#)

[Offre de stage webdesigner](#)

Annonces

Recherche développpeur Symfony2

Par Alexandre, le 27/01/2017

Nous recherchons un développeur Symfony2 débutant sur Lyon. Blabla...

[◀ Retour à la liste](#)[✎ Modifier l'annonce](#)[🗑 Supprimer l'annonce](#)

Test de notre plateforme d'annonces

- Modification d'une annonce

Annonces

Modifier une annonce

Formulaire d'annonce

Ici se trouvera le formulaire.

Vous éditez une annonce déjà existante, merci de ne pas changer l'esprit général de l'annonce déjà publiée.

[◀ Retour à l'annonce](#)

- Un moteur de templates tel que **Twig** permet de bien **séparer le code PHP du code HTML**, dans le cadre de l'architecture MVC ;
- La syntaxe `{{ var }}` affiche la variable `var` ;
- La syntaxe `{% if %}` exécute quelque chose, ici une condition ;
- Twig offre un **système d'héritage** (via `{% extends %}`) **et d'inclusion** (via `{{ include() }}` et `{{ render() }}`) très intéressant pour bien organiser les templates ;
- Le **modèle triple héritage** est très utilisé pour des projets avec **Symfony**.

Installer un bundle
avec composer

Composer, qu'est-ce que c'est ?

- Composer est un outil permettant d'installer des bundle (Symfony est également un bundle)
- Permet de gérer les dépendances en PHP (toutes les bibliothèques dont votre projet dépend pour fonctionner)
- Composer utilise le site <https://packagist.org/> pour connaître les dépendances entre bibliothèques.
- Les bibliothèques sont dans un dépôt GIT
- Composer s'utilise de la manière suivante :
 - On définit dans un fichier la liste des bibliothèques dont le projet dépend, ainsi que leur version
 - On exécute une commande pour installer ou mettre à jour ces bibliothèques (et leurs propres dépendances)
 - On inclut alors le fichier d'autoload généré par Composer dans notre projet

Installer Composer

- Il faut tout d'abord récupérer l'archive .phar : se placer dans le répertoire `\wamp\www` et entrer la commande suivante :

```
php -r "eval('?'>'.file_get_contents('http://getcomposer.org/installer'));"
```

- Ensuite tester l'installation :

```
php composer.phar --version
```

```
C:\wamp\www>php -r "eval('?'>'.file_get_contents('http://getcomposer.org/installer'));"  
All settings correct for using Composer  
Downloading...
```

```
Composer (version 1.3.2) successfully installed to: C:\wamp\www\composer.phar  
Use it: php composer.phar
```

```
C:\wamp\www>php composer.phar --version  
Composer version 1.3.2 2017-01-27 18:23:41
```

Mise à jour de Composer

- De temps à autre il peut être nécessaire de mettre à jour Composer, cela est possible avec la commande suivante :

```
php composer.phar self-update
```

```
C:\wamp\www>php composer.phar self-update  
Updating to version 1.4.2 (stable channel).  
Downloading: 100%
```

- Pour revenir en arrière :

```
php composer.phar composer self-update --rollback
```

```
C:\wamp\www>php composer.phar self-update --rollback  
Rolling back to version 2017-01-27_18-23-41-1.3.2.
```

Installer GIT

- Composer nécessite d'avoir **GIT** (logiciel de gestion de versions) installé
- Sous Windows, il faut utiliser **msysgit** : <https://git-for-windows.github.io/>
- Puis ajouter au Path Windows les chemins des binaires :

```
C:\Program Files\Git\mingw64\bin
```

```
C:\Program Files\Git\bin
```

- Sous Linux : `sudo apt-get install git-core`

Règles de versions

- Composer va gérer les versions selon les règles suivantes

Valeur	Exemple	Description
Un numéro de version exact	"2.0.17"	Composer téléchargera cette version exacte.
Une plage de versions	">=2.0,<2.6"	Composer téléchargera la version la plus à jour, à partir de la version 2.0 et en s'arrêtant avant la version 2.6. Par exemple, si les dernières versions sont 2.4, 2.5 et 2.6, Composer téléchargera la version 2.5.
Une plage de versions sémantique	"~2.1"	Composer téléchargera la version la plus à jour, à partir de la version 2.1 et en s'arrêtant avant la version 3.0. C'est une façon plus simple d'écrire ">=2.1,<3.0" avec la syntaxe précédente. C'est la façon la plus utilisée pour définir la version des dépendances.
Un numéro de version avec joker « * »	"2.0.*"	Composer téléchargera la version la plus à jour qui commence par 2.0. Par exemple, il téléchargerait la version 2.0.17, mais pas la version 2.1.1.
Un nom de branche "dev-XXX"		C'est un cas un peu particulier, où Composer ira chercher la dernière modification d'une branche Git en particulier. N'utilisez cette syntaxe que pour les bibliothèques dont il n'existe pas de vraie version. Vous verrez assez souvent "dev-master", où "master" correspond à la branche principale d'un dépôt Git.

Un exemple de règle

- Pour illustrer le tableau précédent créons un repertoire `\test` contenant le fichier `composer.json` avec le code suivant :

```
{  
  "require": {  
    "twig/extensions": "~1.0"  
  }  
}
```

Dans notre cas la dernière version de la branche 1.*

<https://packagist.org/packages/twig/extensions>

- Pour mettre à jour toutes les dépendances, "twig/extensions" dans notre cas, il faut exécuter la commande update de Composer, comme ceci :

```
php ../composer.phar update
```

```
C:\wamp\www\test>php ../composer.phar update  
Loading composer repositories with package information  
Updating dependencies (including require-dev)  
Package operations: 3 installs, 0 updates, 0 removals  
 - Installing symfony/polyfill-mbstring (v1.3.0): Downloading (100%)  
 - Installing twig/twig (v2.3.2): Downloading (100%)  
 - Installing twig/extensions (v1.5.0): Downloading (100%)  
twig/extensions suggests installing symfony/translation (Allow the time_diff output to be translated)  
Writing lock file  
Generating autoload files
```

Un exemple de règle

- Le répertoire `\test\vendor` contient à présent :
 - la dépendance `"twig/extensions"` que l'on a définie, dans `vendor/twig/extensions`
 - la dépendance `"twig/twig"` de notre dépendance à nous, dans `vendor/twig/twig`
 - les fichiers nécessaires pour l'autoload, allez vérifier le fichier `vendor/composer/autoload_namespaces.php`

Mettre à jour Symfony

- Symfony étant un bundle, il peut être mis à jour grâce à Composer de la même façon que notre exemple
- Avant de mettre à jour et voir si cela en vaut la peine, on peut vérifier s'il y a des failles connues sur la version installée

```
php bin/console security:check
```

```
C:\wamp\www\Symfony>php bin/console security:check

Symfony Security Check Report
=====

// Checked file: C:\wamp\www\Symfony\composer.lock

[OK] No packages have known vulnerabilities.
```

- Pour mettre à jour Symfony :

```
php ../composer.phar update
```

Faire fonctionner Symfony avec une ancienne version de PHP

- On a vu que Symfony fonctionne à présent avec PHP 7, il est possible néanmoins de le faire fonctionner avec une version antérieure, par exemple la version 5.6.25 :
 - Pour cela il faut éditer le fichier `composer.json`
 - Et ajouter les lignes suivantes

```
.../  
"config": {  
    "preferred-install": "dist",  
    "platform": {  
        "php": "5.6.25"  
    },  
    ...  
}
```

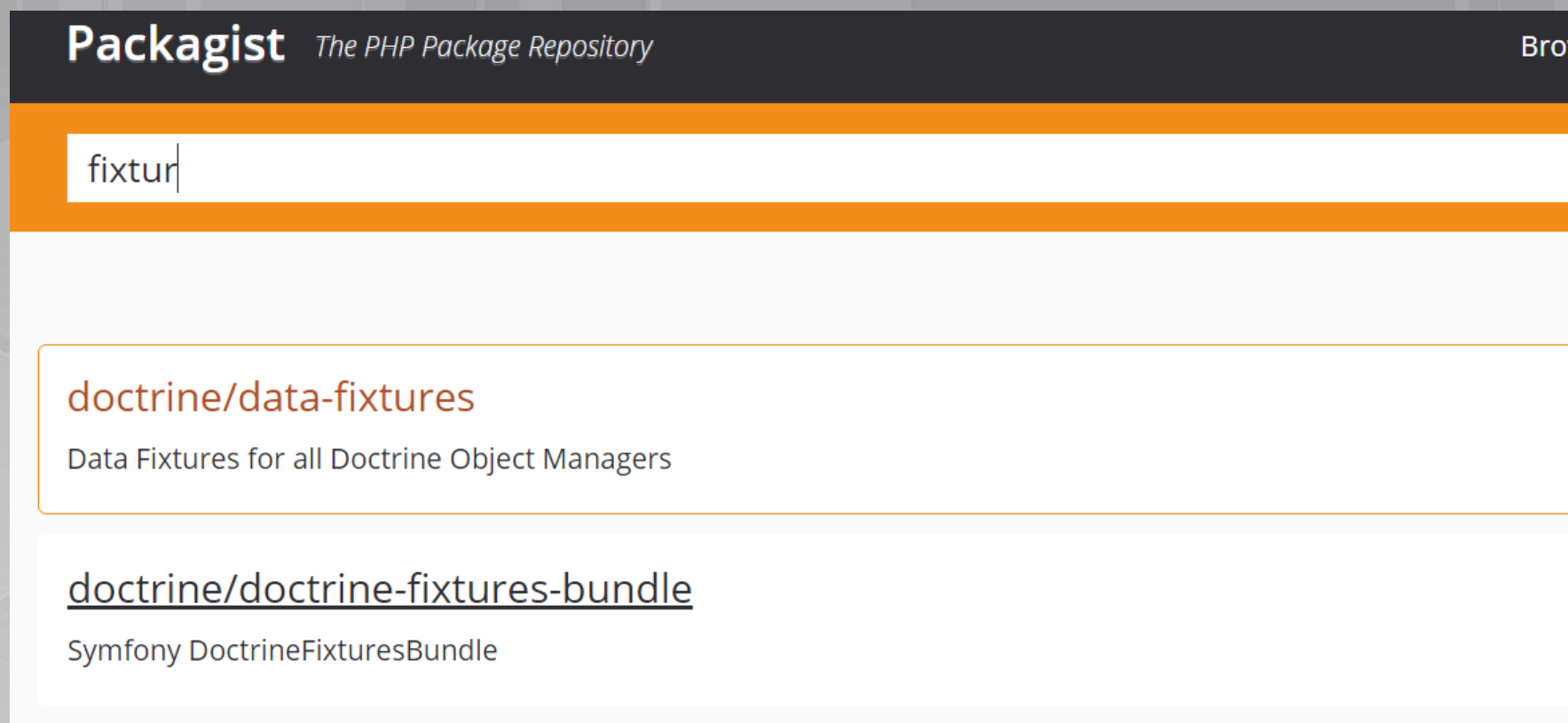
- Cela aura pour effet de respecter la compatibilité avec les bundles lors de mise à jour avec l'outil Composer
- Note : Dans WAMP activer la version PHP correspondante

Installation d'un bundle

- Nous allons installer le bundle DoctrineFixtureBundle, qui permet de préremplir la base de données avec des données, afin de faciliter les tests d'applications
- Etapes :
 - Trouver le nom du bundle
 - Déterminer la version à installer
 - Déclarer le bundle à Composer
 - Mettre à jour les dépendances
 - Enregistrer le bundle dans le Kernel

Trouver le nom du bundle

- Se rendre sur <http://packagist.org/> et effectuer une recherche : dans notre cas, recherchez « fixture », et cliquez sur le bundle de Doctrine, « doctrine/doctrine-fixtures-bundle »



The screenshot shows the Packagist website interface. At the top, the header reads 'Packagist The PHP Package Repository'. Below the header is a search bar with the text 'fixtur' entered. The search results are displayed in a list. The first result is 'doctrine/data-fixtures' with the description 'Data Fixtures for all Doctrine Object Managers'. The second result is 'doctrine/doctrine-fixtures-bundle' with the description 'Symfony DoctrineFixturesBundle'. The second result is underlined.

Packagist *The PHP Package Repository* Bro

fixtur

doctrine/data-fixtures
Data Fixtures for all Doctrine Object Managers

doctrine/doctrine-fixtures-bundle
Symfony DoctrineFixturesBundle

Déterminer la version du bundle

- Soit il y a une version bien déterminée, soit dev-master. Dans ce cas ne pas hésiter à lire la documentation pour d'éventuelles incompatibilités avec votre projet
- Dans notre cas la version est bien indiquée – Les prérequis aussi :

2.3.0

requires

- php: >=5.3.2
- **symfony/doctrine-bridge**: ~2.3|~3.0
- **doctrine/doctrine-bundle**: ~1.0
- **doctrine/data-fixtures**: ~1.0

provides

None

requires (dev)

None

conflicts

None

- On choisit alors la branche 2.* à partir de la version 2.3 grâce à la contrainte "~2.3"

Déclarer le bundle à Composer

- On va éditer le fichier `composer.json` de la façon suivante :

```
// composer.json

// ...

"require": {
    "php": ">=5.5.9",
    // ...
    "incenteev/composer-parameter-handler": "~2.0",
    "doctrine/doctrine-fixtures-bundle": "~2.3"
},

// ...
```


Mettre à jour les dépendances

- Puis on met à jour Symfony :

```
php ../composer.phar update
```

```
C:\wamp\www\Symfony>php ../composer.phar update
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 2 installs, 6 updates, 0 removals
  - Updating twig/twig (v1.31.0 => v2.1.0) Loading from cache
  - Updating doctrine/annotations (v1.2.7 => v1.3.1) Loading from cache
  - Updating doctrine/collections (v1.3.0 => v1.4.0) Loading from cache
  - Updating doctrine/common (v2.6.2 => v2.7.2) Loading from cache
  - Updating doctrine/dbal (v2.5.6 => v2.5.10) Loading from cache
  - Installing doctrine/data-fixtures (v1.2.2) Downloading: 100%
  - Updating doctrine/doctrine-bundle (1.6.6 => 1.6.7) Loading from cache
  - Installing doctrine/doctrine-fixtures-bundle (2.3.0) Downloading: 100%
doctrine/data-fixtures suggests installing doctrine/mongodb-odm (For loading MongoDB ODM fixtures)
doctrine/data-fixtures suggests installing doctrine/phpcr-odm (For loading PHPCR ODM fixtures)
Writing lock file
Generating autoload files
> Incenteev\ParameterHandler\ScriptHandler::buildParameters
```

Déclarer le bundle à Composer

- Enfin, il faut déclarer le bundle dans le Kernel de Symfony. Allez dans `app/AppKernel.php` et ajoutez le bundle à la liste :

```
<?php
// app/AppKernel.php

// ...

if (in_array($this->getEnvironment(), array('dev', 'test'))) {
    // ...
    $bundles[] = new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle();
}
```

- L'autoload est géré automatiquement par Composer qui a modifié les fichiers correspondant pour que les espaces de noms de Doctrine soit connus par Symfony
- Si vous avez besoin de gérer l'autoload d'une lib non référencée sur Packagist, il faudra alors ajouter les informations à la section "autoload" de `composer.json` (`"VotreNamespace": "chemin/vers/la/bibliotheque"`)

- **Composer** est un outil pour gérer les **dépendances d'un projet en PHP**, qu'il soit sous Symfony ou non.
- Le fichier `composer.json` permet de **lister les dépendances** que doit inclure Composer dans votre projet.
- **Composer** détermine la meilleure version possible pour vos dépendances, les télécharge, et **configure leur autoload tout seul**.
- Composer trouve toutes les **bibliothèques** sur le site <http://www.packagist.org>, sur lequel vous pouvez envoyer votre propre bibliothèque si vous le souhaitez.
- La très grande majorité des bundles Symfony sont installables avec Composer, ce qui simplifie énormément leur utilisation dans un projet.

Les services

Introduction

- Un **objet** remplit une **fonction donnée** (envoi de mail, interaction avec la base de données..)
- Parfois, un objet a **besoin d'un ou plusieurs autres objets** pour réaliser sa fonction
- Une **application** fait travailler tous ces **objets ensemble**
- Chaque **objet** est défini en tant que **service**, et le **conteneur de services** permet d'instancier, d'organiser et de récupérer les services de votre application.
- Un service (objet PHP) a pour vocation d'être **accessible depuis n'importe où dans votre code**
- La **configuration** d'un service est le moyen de l'enregistrer dans le conteneur de services

Exemple : envoi de mail

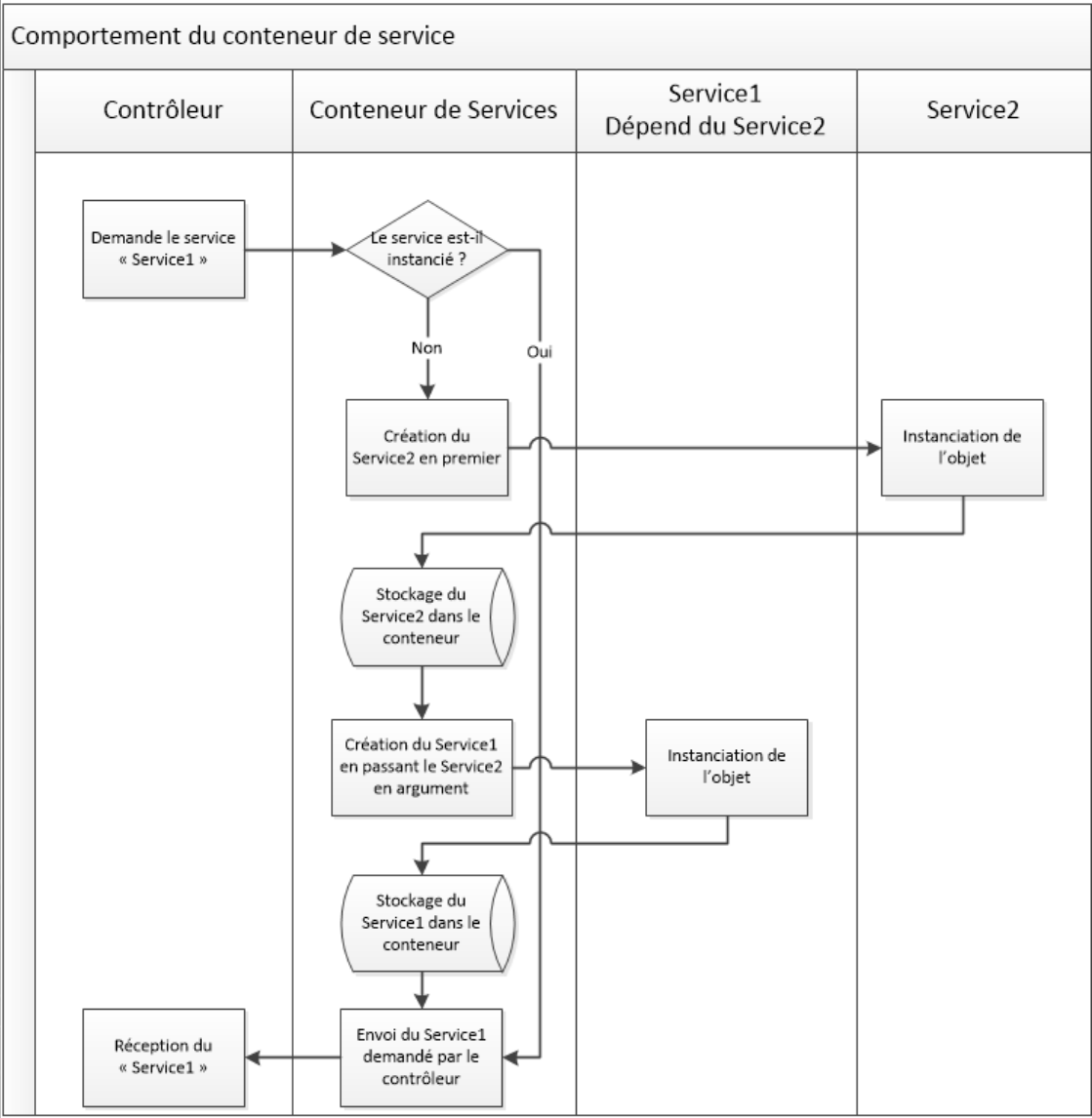
- Dans Symfony, il existe le **composant SwiftMailer** qui permet de gérer les e-mails.
- Ce composant contient une **classe** nommée **Swift_Mailer** qui envoie effectivement les e-mails.
- Symfony, qui intègre le composant SwiftMailer, définit déjà cette classe en tant que **service mailer** grâce à un peu de configuration.
- Le **conteneur de service** de Symfony peut donc accéder à la classe **Swift_Mailer** grâce au service mailer
- On parle d'**architecture orientée services**

Le conteneur de services

- L'intérêt des services réside dans leur association avec le conteneur de services.
- Le **conteneur de services** (services container en anglais) est une sorte de super-objet qui **gère tous les services**. Ainsi, pour accéder à un service, il faut passer par le conteneur
- Le **rôle du conteneur** est d'organiser et d'instancier vos services très facilement : on simplifie ainsi la récupération des services depuis le code (depuis le contrôleur ou autre).
- Le conteneur de services est **régénéré à chaque changement de configuration**. Il se trouve donc dans le cache :
`var/cache/dev/appDevDebugProjectContainer.php`

Comportement du conteneur de services

- L'exemple ci contre est constitué de deux services, sachant que le *Service1* nécessite le *Service2* pour fonctionner



Comment définir les dépendances entre services ?

- Comment dire au conteneur que le *Service2* doit être instancié avant le *Service1* ? Cela se fait grâce à la configuration dans Symfony.
- L'idée est de **définir pour chaque service** :
 - Son **nom**, qui permettra de l'identifier au sein du conteneur ;
 - Sa **classe**, qui permettra au conteneur d'instancier le service ;
 - Les **arguments** dont il a besoin. Un argument peut être un autre service, mais aussi un paramètre (défini dans le fichier `parameters.yml` par exemple).
- Dans Symfony, **chaque service est « partagé »**. Cela signifie simplement que la classe du service est instanciée une seule fois (à la première récupération du service) par le conteneur. Si, plus tard dans l'exécution de la page, vous voulez récupérer le même service, c'est cette même instance que le conteneur retournera.

Lister les services disponibles

- On peut connaître la liste des services disponibles avec la commande `php bin/console debug:container`

```
c:\wamp\www\Symfony3>php bin/console debug:container
```

```
Symfony Container Public Services
=====
```

Service ID	Class name
annotation_reader	Doctrine\Common\Annotations\CachedReader
assets.context	Symfony\Component\Asset\Context\RequestStackContext
assets.packages	Symfony\Component\Asset\Packages
cache.app	Symfony\Component\Cache\Adapter\FilesystemAdapter
cache.app_clearer	alias for "cache.default_clearer"
cache.default_clearer	Symfony\Component\HttpKernel\CacheClearer\Psr6CacheClearer
cache.system	Symfony\Component\Cache\Adapter\AdapterInterface
cache_clearer	Symfony\Component\HttpKernel\CacheClearer\ChainCacheClearer
cache_warmer	Symfony\Component\HttpKernel\CacheWarmer\CacheWarmerAggregate
config_cache_factory	Symfony\Component\Config\ResourceCheckerConfigCacheFactory
console.command.sensiolabs_security_command_securitycheckercommand	alias for "sensio_distribution.security_checker.command"
data_collector.dump	Symfony\Component\HttpKernel\DataCollector\DumpDataCollector
data_collector.form	Symfony\Component\Form\Extension\DataCollector\FormDataCollector
data_collector.form.extractor	Symfony\Component\Form\Extension\DataCollector\FormDataExtractor
data_collector.request	Symfony\Bundle\FrameworkBundle\DataCollector\RequestDataCollector

Utiliser un service en pratique

- Par exemple le service Swiftmailer s'utiliserait de la manière suivante (ce service étant déjà créé, et sa configuration faite, il ne reste plus qu'à l'utiliser) :

```
<?php
// src/OC/PlatformBundle/Controller/AdvertController.php

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class AdvertController extends Controller
{
    public function indexAction()
    {
        // On a donc accès au conteneur :
        $mailer = $this->container->get('mailer');

        // On peut envoyer des e-mails, etc.
    }
}
```

- Note : dans un contrôleur, `$this->get('...')` est strictement équivalent à `$this->container->get('...')`

Créer un service simple

- Un service n'est qu'une classe, il suffit de créer un fichier n'importe où et de créer une classe dedans
- Il faut néanmoins respecter la norme PSR-4 pour l'autoload (exemple : la classe `OC\PlatformBundle\Antispam\OCAntispam` doit se trouver dans `src/OC/PlatformBundle/Antispam/OCAntispam.php`)
- Pour continuer la plateforme d'annonce, on pourrait par exemple créer un système anti-spam afin de détecter les spams à partir d'un simple texte. Comme on aurait besoin d'elle à plusieurs endroits (pour les annonces et pour les futurs commentaires) le mieux est d'en faire un service.
- On nommera ce service au choix, par exemple `OCAntispam`

Service d'antispam – création du fichier de classe

- On crée le fichier :
`src/OC/PlatformBundle/Antispam/OCAntispam.php`

```
<?php
// src/OC/PlatformBundle/Antispam/OCAntispam.php

namespace OC\PlatformBundle\Antispam;

class OCAntispam
{
    .... / ....
}
```

Service d'antispam – création de la classe

- Pour notre exemple considérons un cas simple : on considèrera comme spam tout texte de longueur inférieure à 50 caractères :

```
<?php
// src/OC/PlatformBundle/Antispam/OCAntispam.php

namespace OC\PlatformBundle\Antispam;

class OCAntispam
{
    /**
     * Vérifie si le texte est un spam ou non
     *
     * @param string $text
     * @return bool
     */
    public function isSpam($text)
    {
        return strlen($text) < 50; // Retourne TRUE si moins
                                   // de 50 caractères
    }
}
```

Service d'antispam – Création de la configuration du service

- Maintenant que nous avons créé la classe, il faut la signaler au conteneur de services.
- Un service se définit par sa classe ainsi que sa configuration. Pour cela, nous pouvons utiliser le fichier

src/OC/PlatformBundle/Resources/config/services.yml

```
# src/OC/PlatformBundle/Resources/config/services.yml

services:
    oc_platform.antispam:
        class: OC\PlatformBundle\Antispam\OCAntispam
```

- `oc_platform.antispam` est le nom de notre service. De cette manière, le service sera accessible via

```
$container->get('oc_platform.antispam');
```

(Préfixez le nom des services celui du bundle « `oc_platform` ».)

- `class` est un attribut obligatoire de la configuration, il définit le namespace de la classe du service : indique au conteneur de services quelle classe instancier lorsqu'on lui demandera le service.

Service d'antispam – Utilisation du service

- Voici un exemple simple d'utilisation

```
<?php

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

namespace OC\PlatformBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;

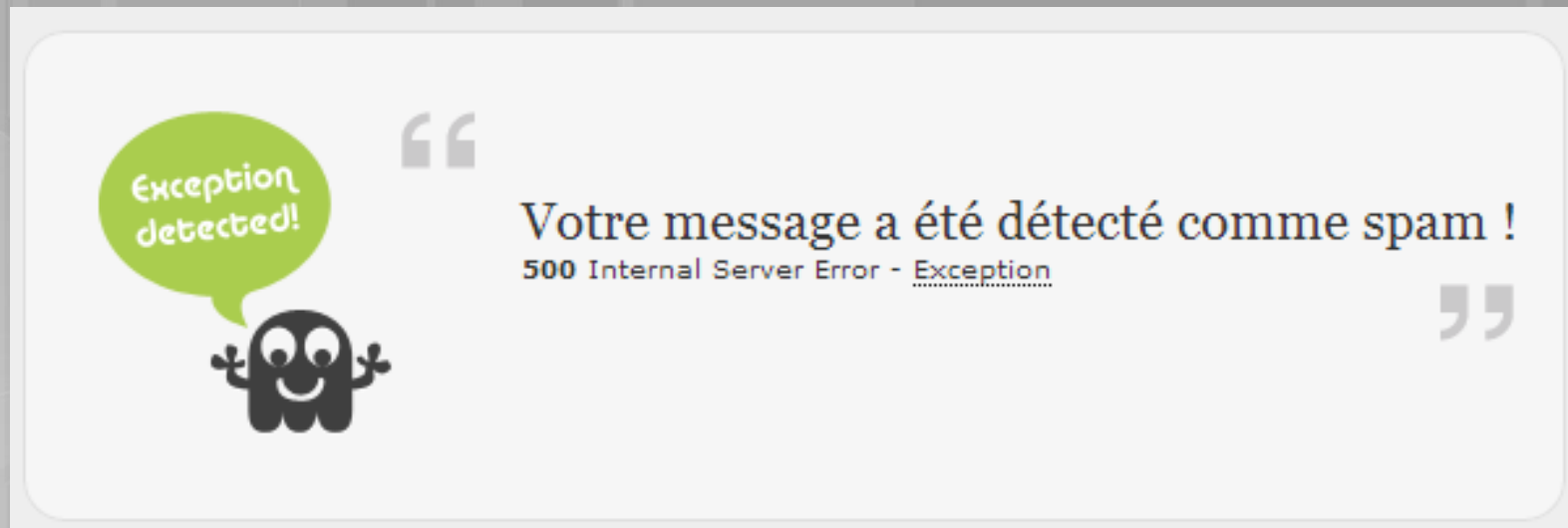
class AdvertController extends Controller
{
    public function addAction(Request $request)
    {
        // On récupère le service
        $antispam = $this->container->get('oc_platform.antispam');

        // Je pars du principe que $text contient le texte d'un message quelconque
        $text = '...';
        if ($antispam->isSpam($text)) {
            throw new \Exception('Votre message a été détecté comme spam !');
        }

        // Ici le message n'est pas un spam
    }
}
```


Service d'antispam – Utilisation du service

- Si vous définissez la variable `$text` avec moins de 50 caractères, vous aurez droit au message d'erreur de la figure suivante.



Créer un service avec arguments

- La plupart du temps les services ne fonctionnent pas seuls, et nécessitent l'utilisation d'autres services, de paramètres ou de variables. Pour passer des arguments à votre service, il faut utiliser sa configuration :

```
# src/OC/PlatformBundle/Resources/config/services.yml

services:
    oc_platform.antispam:
        class: OC\PlatformBundle\Antispam\OCAntispam
        arguments: # Tableau d'arguments
```

- Les arguments peuvent être :
 - Des services : l'identifiant du service est précédé d'une arobase :
@nomDuService
 - Des paramètres (définis dans `parameters.yml` par exemple) : l'identifiant du paramètre est encadré de signes « % » : %nomDuParametre%
 - Des valeurs normales en YAML (des booléens, des chaînes de caractères, des nombres, etc.)

Service d'antispam - arguments

- Dans notre service d'antispam nous allons injecter 3 paramètres :

```
# src/OC/PlatformBundle/Resources/config/services.yml

services:
    oc_platform.antispam:
        class: OC\PlatformBundle\Antispam\OCAntispam
        arguments:
            - "@mailer"
            - "%locale%"
            - 50
```

- Où :
 - @mailer : le service Mailer (pour envoyer des e-mails) ;
 - %locale% : le paramètre local (pour récupérer la langue, définie dans le fichier app/config/parameters.yml) ;
 - 50 : le nombre décimal 50.

Récupérer les arguments

- Une fois vos arguments définis dans la configuration, il vous suffit de les récupérer avec le constructeur du service.

```
<?php
// src/OC/PlatformBundle/Antispam/OCAntispam.php

namespace OC\PlatformBundle\Antispam;

class OCAntispam
{
    private $mailer;
    private $locale;
    private $minLength;

    public function __construct(\Swift_Mailer $mailer, $locale, $minLength)
    {
        $this->mailer      = $mailer;
        $this->locale      = $locale;
        $this->minLength = (int) $minLength;
    }

    .../...

}
```

Le \ indique que l'on utilise l'espace de noms racine

Injection de dépendances

- Nous venons de réaliser ce qu'on appelle une injection de dépendances, à savoir injecter un service dans un autre (ici le service Mailer dans le service antispam)
- L'instanciation du service injecté est réalisé de façon automatique par le conteneur de services (ici c'est lui qui prend va prendre soin d'instancier Mailer)
- Vous pouvez bien entendu utiliser votre nouveau service antispam dans un prochain service (au même titre que vous avez mis `@mailer` en argument, vous pourrez mettre `@oc_platform.antispam`)

Code généré du conteneur de services

- Le fichier `var/cache/dev/appDevDebugProjectContainer.php` contient :

```
.../...

protected function getOcPlatform_AntispamService()    {
    return $this->services['oc_platform.antispam'] = new
\OC\PlatformBundle\Antispam\OCAntispam(
    $this->get('swiftmailer.mailer.default'),
    'en',
    50);
}

.../...
```

- Notes :
 - `mailer` est un alias de `swiftmailer.mailer.default`
 - `%locale%` a été transformé en `en` (valeur qui se trouve dans `parameters.yml`), équivalent à `$this->getParameter('locale')`

Résumé

- Un **service** est une **simple classe** associée à une certaine configuration.
- Le **conteneur de services** organise et **instancie** tous vos **services**, grâce à leur configuration.
- Les services sont la **base de Symfony**, et sont très utilisés par le cœur même du framework.
- L'**injection de dépendances** est assurée **par le conteneur**, qui connaît les arguments dont a besoin un service pour fonctionner, et les lui donne donc à sa création.