TEK-UP Ecole Supérieure Privée Technologie & Ingénierie

Développement Web (PHP Symfony 5)

Wissem ELJAOUED
wissem.eljaoued@ensi-uma.tn

A.U. 2021-2022

Plan du cours

I. Introduction Symfony	TP1: Création d'un projet
II. Routing (Routeur de Symfony)	TP2: Router + Controller
III. Les contrôleurs	
IV. Les Twigs	TP3: Twig
V. ORM Doctrine	TP4: Doctrine / TP5: DQL
VI. Les Formulaires	TP6: Forms

Ch. II

Routing (Routeur de Symfony)

Ch. II: Routing

II.1. Routeur de Symfony

II.2. Routes de base

II.3. Routes avancées

II.4. Génération des URL

III.1. Routeur de Symfony

- Symfony intègre un système de routing permettant d'interpréter une URL et en déduire l'action et donc la page à afficher

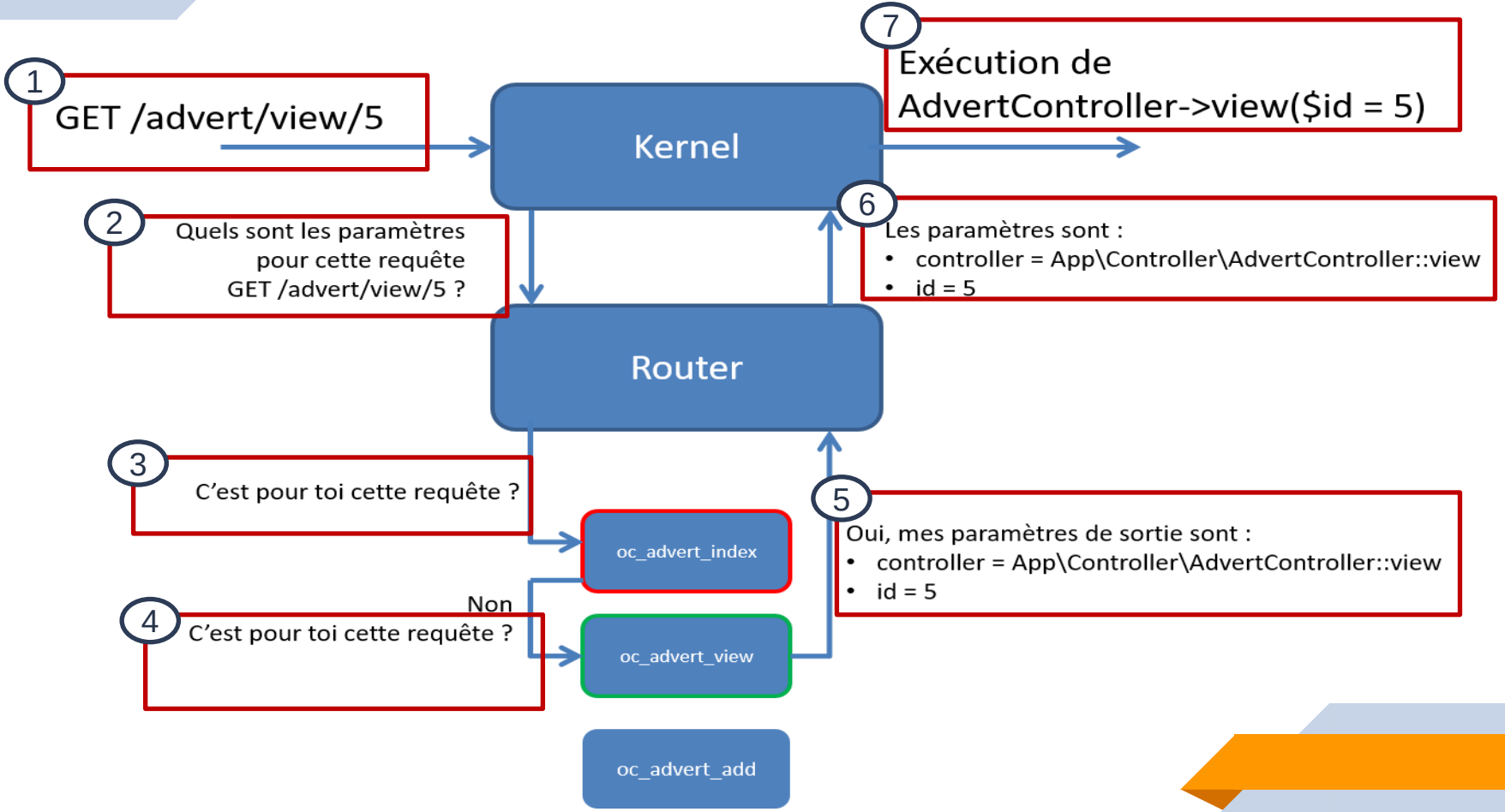
- C'est un système très puissant qui nous permettra de gérer tous les liens internes du projet

- Le routing nous permet aussi de gérer facilement des URLs plus parlantes et de savoir à quelle action et donnée elle correspond. (URL rewriting etc.)
  - Exemple: l'URL d'une page wiki peut être

```
http://example.com/w/index.php?title=Page_title
```

Mais peut être réécrit comme

```
http://example.com/wiki/Page_title
```


1
GET /advert/view/5
Kernel
7
Exécution de AdvertController->view($id = 5)
2
Quels sont les paramètres pour cette requête GET /advert/view/5 ?
6
Les paramètres sont :
• controller = App\Controller\AdvertController::view
• id = 5
Router
3
C'est pour toi cette requête ?
oc_advert_index
5
Oui, mes paramètres de sortie sont :
• controller = App\Controller\AdvertController::view
• id = 5
Non
4
C'est pour toi cette requête ?
oc_advert_view
oc_advert_add

1. On appelle l'URL /advert/view/5.
2. Le routeur passe donc à la route suivante. Il essaie de faire correspondre /advert/view/5 avec /advert/view/{id}. Nous le verrons plus loin, mais {id} est un paramètre, une sorte de joker « je prends tout ». Cette route correspond, car nous avons bien :
    - /advert/view (URL) =/advert/view (route) ;
    - 5 (URL) = {id} (route)
3. Le routeur s'arrête donc, il a trouvé sa route.
4. Le routeur essaie de faire correspondre cette URL avec le path de la première route. Ici,/advert/view/5 ne correspond pas du tout à /advert (ligne path de la première route).
5. Il demande à la route : « Quels sont tes paramètres de sortie ? », la route répond : « Mes paramètres sont 1/ le contrôleur App/Controller/AdvertController::view, et 2/ la valeur $id = 5. »
6. Le routeur renvoie donc ces informations au Kernel (le noyau de Symfony).
7. Le noyau exécute le bon contrôleur avec les bons paramètres !

Le but du routeur est donc, à partir d'une URL, de trouver la route correspondante et de retourner les paramètres de sortie que définit cette route, dont le contrôleur.
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

❖ 1. Configuration avec YAML (Yet Another Markup Language)

Chaque route prend :

- Une entrée (ligne path) : c'est l'URL à capturer ;
- Une sortie (ligne controller) : c'est un des paramètres de la route, celui qui indique quel contrôleur à exécuter

```yaml
# config/routes.yaml
blog_list:
    path: /blog
    # the controller value has the format 'controller_class::method_name'
    controller: App\Controller\BlogController::list

    # if the action is implemented as the __invoke() method of the
    # controller class, you can skip the '::method_name' part:
    # controller: App\Controller\BlogController
```

Les routes se définissent dans un fichier qui se trouve sous config/routes.yml.

❖ 2. Configuration avec Annotations

Définir les routes directement depuis les contrôleurs

```php
// src/Controller/BlogController.php
namespace App\Controller;

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

class BlogController extends AbstractController
{
    /**
     * @Route("/blog", name="blog_list")
     */
    public function list()
    {
        // ...
    }
}
```

Juste avant la méthode qui sera exécuté par le controller

Ch. II: Routing

II.1. Routeur de Symfony

II.2. Routes de base

II.3. Routes avancées

II.4. Génération des URL

Route sans paramètres

❖ ***Route sans paramètres***

```yaml
# config/routes.yaml
blog_list:
    path: /blog
    # the controller value has the format 'controller_class::method_name'
    controller: App\Controller\BlogController::list
```

Une route est composée au minimum de deux éléments:

- Une entrée (ligne path) : c'est l'URL à capturer ;
- Une sortie (ligne controller) : c'est un des paramètres de la route, celui qui indique quel contrôleur à exécuter

❖ *Route avec paramètres*

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

namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AdvertController
{
  // ...

  /**
   * @Route("/advert/view/{id}", name="oc_advert_view")
   */
  public function view($id)
  {
    // $id vaut 5 si l'URL appelée est /advert/view/5

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

Le path contient un paramètre supplémentaire **{id}**

Route **avec** paramètres

- Grâce au paramètre {id} dans le *path* de notre route, toutes les URL du type /advert/view/* seront gérées par cette route, par exemple :
  - /advert/view/5
  - ou /advert/view/654

- Par contre, l'URL /advert/view ne sera pas interceptée, car le paramètre {id} n'est pas renseigné. En effet, les paramètres sont par défaut obligatoires, nous verrons quand et comment les rendre facultatifs plus loin dans ce chapitre.

- Ce paramètre {id} est accessible depuis votre contrôleur. Si vous appelez l'URL /advert/view/5, alors depuis votre contrôleur vous aurez la variable $id (du nom du paramètre) en argument de la méthode, variable qui aura pour valeur « 5 » !

Route avec paramètres

- Vous pouvez bien sûr multiplier les paramètres au sein d'une même route:

```php
<?php
# src/Controller/AdvertController.php

/**
 * @Route("/advert/view/{year}/{slug}.{format}", name="oc_advert_view_slug")
 */
public function viewSlug($slug, $year, $format)
{
  return new Response(
    "On pourrait afficher l'annonce correspondant au
    slug '".$slug."', créée en ".$year." et au format ".$format."."
  );
}
```

- Cette route intercepte par exemple les URL suivantes:
  - /advert/view/2011/webmaster-aguerri.html ,
  - et /advert/view/2012/symfony.xml,
  - etc.

- Notez que l'ordre des arguments dans la méthode viewSlug() n'a pas d'importance. La route fait la correspondance à partir du nom des variables utilisées, non à partir de leur ordre.

Route **avec** paramètres

- Revenez à notre route et notez également le point entre les paramètres {slug} et {format} : vous pouvez en effet séparer vos paramètres soit avec le slash (« / »), soit avec le point (« . »).

➡ Donc, ne pas utiliser de point dans le contenu de vos paramètres

- Par exemple:

Pour notre paramètre {slug}, une URL /advert/view/2011/webmaster.aguerri.html ne va pas fonctionner comme on l'attend, car :

  - {year} = 2011 ;
  - {slug} = webmaster ;
  - {format} = aguerri.html (il prend tout le reste de l'URL)

Ch. II: Routing

II.1. Routeur de Symfony

II.2. Routes de base

II.3. Routes avancées

II.4. Génération des URL

Route avec paramètres + Contraintes

- Continuant dans le même exemple de la section précédente :

- Si quelqu'un essaie d'atteindre l'URL /advert/view/sdf/azerty.bouh !!!
  Et pourtant, « sdf » n'est pas tellement une année valide ! **La solution ?**

  ➡ ***Les contraintes sur les paramètres***

- Nous voulons ne récupérer que les bonnes URL où l'année vaut « 2010 » et non « sdf », par exemple. Cette dernière devrait retourner une erreur 404 (page introuvable).

  ➡ Pour cela, il nous suffit qu'aucune route ne l'intercepte

Route avec paramètres + Contraintes

- Comment faire pour que notre paramètre {year} n'intercepte pas « sdf » ?

➡ Ajouter des contraintes sur nos paramètres :

```
/**
 * @Route("/advert/view/{year}/{slug}.{format}", name="oc_advert_view_slug", requirements={
 *   "year"   = "\d{4}",
 *   "format" = "html|xml"
 * })
 */
```

- Nous avons ajouté la section ***requirements***
- On utilise les expressions régulières pour déterminer les contraintes que doivent respecter les paramètres. Ici :
  - \d{4} veut dire « quatre chiffres à la suite ». L'URL /advert/view/sdf/webmaster.html ne sera donc pas interceptée car "sdf" n'est pas une suite de 4 chiffres.
  - Html | xml signifie « soit HTML, soit XML ». L'URL /platform/2011/webmaster.rss ne sera donc pas interceptée.

Paramètres facultatifs

- Nous avons ajouté un quatrième argument à l'annotation, le tableau ***defaults*** , contenant un seul élément : format = html.

- Ainsi, l'URL /advert/view/2014/webmaster sera bien interceptée et le paramètre format prendra sa valeur par défaut, à savoir « html ».

- Au niveau du contrôleur, rien ne change : vous gardez l'argument $format comme avant et celui-ci vaudra « html », la valeur par défaut.

```
/**
 * @Route("/advert/view/{year}/{slug}.{format}", name="oc_advert_view_slug", requirements={
 *    "year"   = "\d{4}",
 *    "format" = "html|xml"
 * }, defaults={"format" = "html"})
 */
```

Préfixe de route

- Dans l'exemple: nous avons mis /advert au début du *path* de chacune de nos routes au lieu de le répéter dans chaque route. Symfony vous propose de rajouter un préfixe pour toutes les routes d'un même fichier ou contrôleur

- Pour un contrôleur, ajoutez donc cette annotation sur la classe et non sur une méthode, comme suit :

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

/**
 * @Route("/advert")
 */
class AdvertController
{
  // ...
}
```

- Grâce à cette annotation sur la classe tout entière, vous pouvez enlever la partie /advert de chacune de vos routes définies dans cette classe.

Ch. II: Routing

II.1. Routeur de Symfony

II.2. Routes de base

II.3. Routes avancées

II.4. Génération des URL

Pourquoi générer des URL ?

- Vu que 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 correspondante à une certaine route

- Par exemple, nous avons une route nommée « oc_advert_view » qui écoute l'URL /advert/view/{id}. Vous décidez un jour de raccourcir vos URL et vous aimeriez bien que vos annonces soient disponibles depuis /advert/v/{id}. Si vous aviez écrit toutes vos URL à la main dans vos templates, vous auriez du toutes les changer à la main, une par une. Grâce à la génération d'URL, vous ne modifiez que la route : ainsi, toutes les URL générées seront mises à jour

❖ **1. Depuis le contrôleur, avec injection**

- C'est la méthode generate qui nous permet de générer une URL à partir du nom d'une route ainsi que de ses paramètres

- La méthode generate a besoin de 2 argument :

  1. Le premier est le nom de la route ;
  2. Le deuxième est un tableau contenant les valeurs des paramètres pour la génération.
     En effet, l'objectif du routeur n'est pas de générer /advert/view/{id} qui n'aurait pas de sens, mais de générer une URL prête à être utilisée, dans notre cas de l'annonce d'id 5 : /advert/view/5 . Ce deuxième argument est bien sûr facultatif si votre route n'utilise pas de paramètre.

- **1. Depuis le contrôleur, avec injection**

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

namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Routing\RouterInterface;

/**
 * @Route("/advert")
 */
class AdvertController
{
  /**
   * @Route("/", name="oc_advert_index")
   */
  public function index(RouterInterface $router)
  {
    $url = $router->generate(
        'oc_advert_view', // 1er argument : le nom de la route
        ['id' => 5]       // 2e argument : les paramètres
    );
    // $url vaut « /advert/view/5 »

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

1. Depuis le contrôleur, avec injection

Pour générer une URL absolue, comprenant le nom de domaine, lorsque vous l'envoyez par e-mail par exemple, il faut définir le troisième argument de la méthode generate à UrlGeneratorInterface::ABSOLUTE_URL

```php
<?php

use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

$url = $router->generate('oc_advert_list', [], UrlGeneratorInterface::ABSOLUTE_URL);
```

Ainsi, $url vaut http://monsite.com/advert et pas uniquement /advert

2. Depuis le contrôleur, avec héritage

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

namespace App\Controller;

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

/**
 * @Route("/advert")
 */
class AdvertController extends AbstractController // Notez l'héritage
{
  /**
   * @Route("/", name="oc_advert_list")
   */
  public function index()
  {
    $url = $this->generateUrl(
        'oc_advert_view', // 1er argument : le nom de la route
        ['id' => 5]       // 2e argument : les paramètres
    );
    // $url vaut « /advert/view/5 »

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

Dans la méthode précédente, nous avons utilisé directement l'objet du routeur.

Mais, il existe une méthode plus simple:

- I faut faire hériter notre contrôleur d'une classe de Symfony
- Ligne 13 : notre contrôleur hérite de AbstractController , une classe de Symfony ;
- Ligne 18 : inutile d'injecter l'objet Router ;
- Ligne 20 : nous utilisons directement la méthode $this->generateUrl() plutôt que la méthode $router->generate()

❖ 3. Depuis un template Twig

Pour ce faire, c'est la fonction path qu'il faut utiliser depuis un template Twig

```twig
{# Dans une vue Twig, en considérant bien sûr
   que la variable advertId est disponible #}

<a href="{{ path('oc_advert_view', { 'id': advertId }) }}">
    Lien vers l'annonce d'id {{ advertId }}
</a>
```

Et pour générer une URL absolue depuis Twig, pas de troisième argument, mais on utilise la fonction url() au lieu de path(). Elle s'utilise exactement de la même manière, seul le nom change.