

SYMFONY

framework PHP

Base

Anthony PARIS - Formateur

CRÉATION DE PREMIERE PAGE

Symfony fonctionne sur un système MVC, pour créer une page il faut créer un contrôleur.

Il est possible de le créer à la main ou en utilisant un package.

Dans votre terminal :

```
PS D:\wamp64\www\sf4m2icours> composer require symfony/maker-bundle --dev
Using version ^1.15 for symfony/maker-bundle
```

Cette dépendance est en «dev» car non utile en production.

Regardons ensemble :

<https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html>

CRÉATION DE PREMIERE PAGE

Créons notre premier contrôleur pour créer une page d'accueil.

```
PS D:\wamp64\www\sf4m2icours> php bin/console make:controller
```

```
Choose a name for your controller class (e.g. OrangeKangarooController):
```

```
> CmsController
```

```
created: src/Controller/CmsController.php
```

```
created: templates/cms/index.html.twig
```

Success!

Cette commande a créé deux fichiers, un contrôleur (Controller) et un fichier de template (View).

Allons regarder ces fichiers

CRÉATION DE PREMIERE PAGE

```
namespace App\Controller;

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

class CmsController extends AbstractController
{
    #[Route('/cms', name: 'app_cms')]
    public function index(): Response
    {
        return $this->render('cms/index.html.twig', [
            'controller_name' => 'CmsController',
        ]);
    }
}
```

CmsController est une classe étendue d'AbstractController regroupant beaucoup de méthodes internes de Symfony.

Pour définir l'url accessible on utilise les «attributs» natifs de PHP (#) On peut également utiliser les annotations (commentaires) en installant une dépendance
La méthode render renvoie une vue (celle créé par la commande) se trouvant dans le dossier templat

CRÉATION DE PREMIERE PAGE

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

{% block title %}Hello CmsController!{% endblock %}

{% block body %}
<style>
    .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
    .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
    <h1>Hello {{ controller_name }}! ✓</h1>

    This friendly message is coming from:
    <ul>
        <li>Your controller at <code><a href="{{ 'D:/wamp64/www/sf6m2icours/src/Controller/CmsController' }}</a></code>
        <li>Your template at <code><a href="{{ 'D:/wamp64/www/sf6m2icours/templates/cms/index.html.twig' }}</a></code>
    </ul>
</div>
{% endblock %}
```

Voici notre vue qui est lancée quand l'url atteind la page d'accueil.
Il est possible de générer cela à la main mais on remarque que l'utilisation d'une commande facilite grandement la tâche.

Vous pouvez lancer la commande : `php bin/console debug:router` pour afficher les routes dispos.

CRÉATION DE PREMIERE PAGE

Vous avez vu que la génération des routes(url) se fait via des commentaires (annotations ou attributs). Il est possible de générer ses routes d'une autre manière.

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class CmsController extends AbstractController
{
    public function index()
    {
        return $this->render('cms/index.html.twig', [
            'controller_name' => 'CmsController',
        ]);
    }
}
```

Ensuite dans config/routes.yaml

```
index:
    path: /
    controller: App\Controller\CMSController::index
```

Forme : (Nom de la route [unique] , chemin et contrôleur

IL faut respecter exactement 4 espaces et non une tabulation pour l'indentation.

CRÉATION DE PREMIERE PAGE

En passant par wamp vous ne pouvez bénéficier du «Profiler» de SF4, vous devez voir une erreur dans une barre noire.
Pour corriger cela, il faut passer par un virtual Host et télécharger un package.

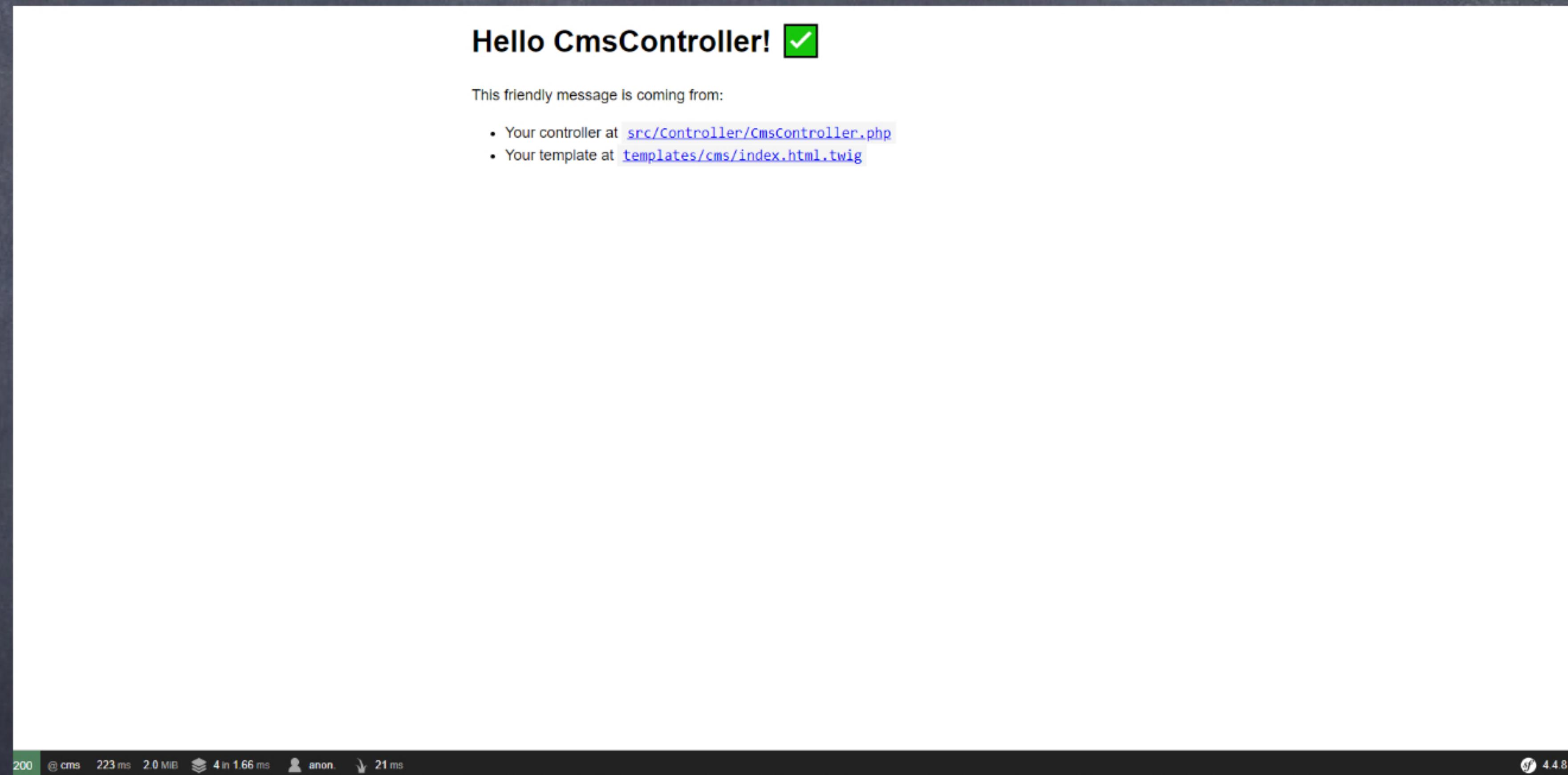
```
PS D:\wamp64\www\sf4m2icours> composer require symfony/apache-pack  
Using version ^1.0 for symfony/apache-pack
```

```
Do you want to execute this recipe?  
[y] Yes  
[n] No  
[a] Yes for all packages, only for the current installation session  
[p] Yes permanently, never ask again for this project  
(defaults to n): y
```

Pensez à indiquer «Y»

CRÉATION DE PREMIERE PAGE

Le profiler est un outil très important pour suivre pas à pas les informations transmises à votre projet. Attention ce n'est pas un console comme pour javascript
Vous pouvez cliquez sur l'élément pour avoir plus d'infos.
A noter si dans votre template, il n'y a pas de balise «body» le profiler ne s'affiche pas



SYSTEME DE CACHE

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

Pour supprimer le cache :

```
php bin/console cache:clear
```

Pour supprimer le cache en prod :

```
php bin/console cache:clear --env=prod
```

En cas d'erreur, relancer la commande ou supprimer le cache à la main dans var/cache/dev ou var/cache/prod en fonction de votre besoin

LE ROUTEUR

A partir de l'url il détermine quel contrôleur utiliser avec quels arguments + slug.
C'est comme L'url Rewriting (apache) mais cette fois ci côté «PHP»

Dans notre fichier CmsController créons une deuxième méthode pour tester les urls

```
#[Route(path: '/info/{id}/{slug}', name: 'test')]  
public function test($id, $slug) : Response  
{  
    return $this->render('cms/index.html.twig', [  
        'controller_name' => $id."/".$slug,  
    ]);  
}
```

Vous définissez des variables au sein de l'URL, que vous récupérez en tant qu'arguments de votre méthode et vous les renvoyez à votre vue.

Voyons le cheminement

LE ROUTEUR

On appelle l'URL /info/1/bonjour-a-tous

Le routeur essaie de faire correspondre cette URL avec le path de la première route. Ici, /info/1/bonjour-a-tous ne correspond pas du tout à / (ligne path de la première route).

Le routeur passe donc à la route suivante. Il essaie de faire correspondre /info/1/bonjour-a-tous avec /info/{id}/{slug}. Cette route correspond

Le routeur s'arrête donc, il a trouvé sa route.

Il demande à la route : « Quels sont tes paramètres de sortie ? », la route répond : « Mes paramètres sont 1/ le contrôleur CmsController::test, 2/ la valeur \$id = 1 et 3/ \$slug = bonjour-a-tous »

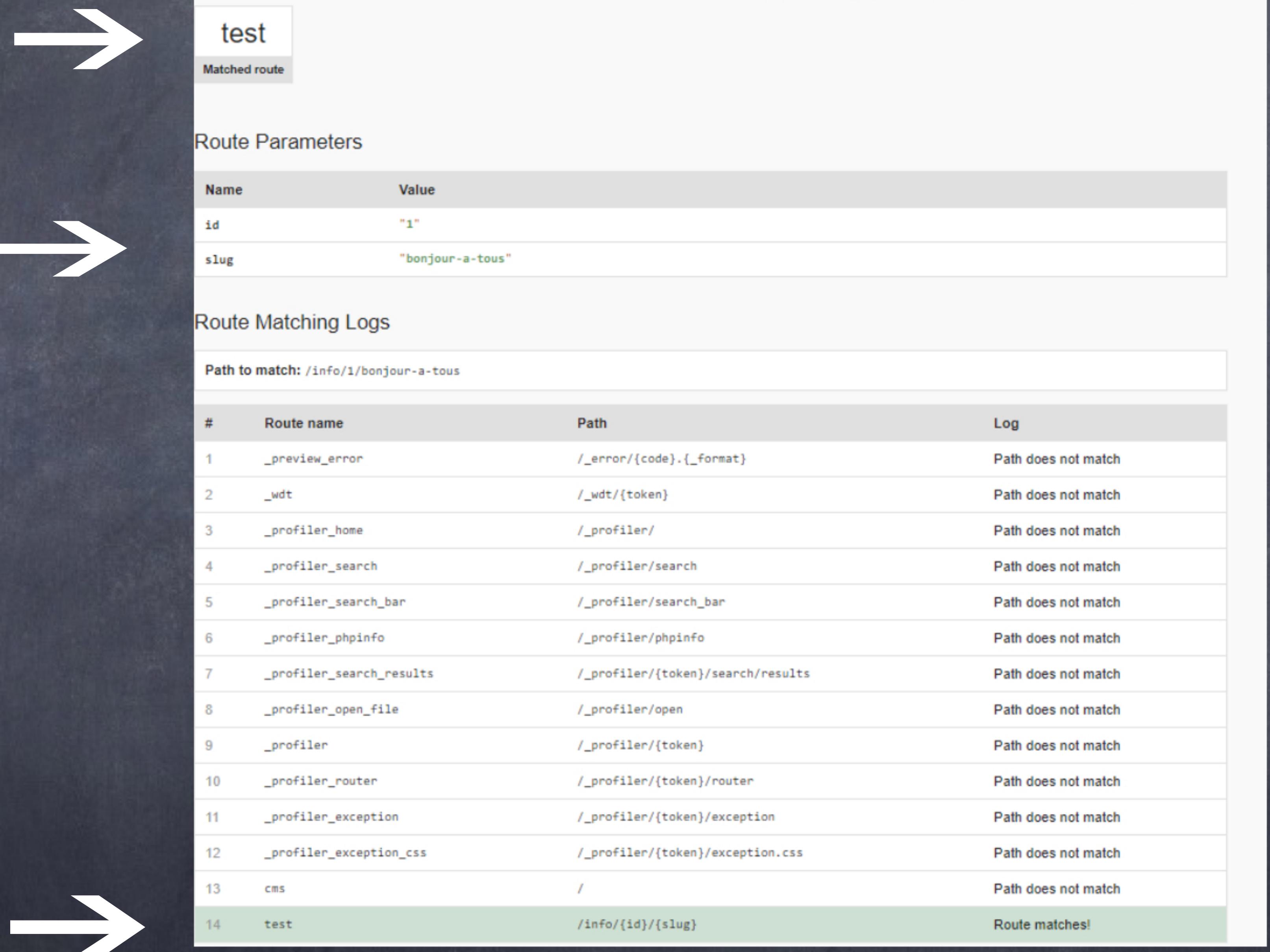
Le routeur renvoie donc ces informations au Kernel (le noyau de Symfony).

Le noyau exécute le bon contrôleur avec les bons paramètres !

Si il ne trouve rien il renvoie une erreur 404.

Pour visualiser cela direction profiler > routing

LE ROUTEUR



LE ROUTEUR COMPLEXE

Il est possible de définir des patterns dans le routing ainsi que des paramètres facultatifs

```
#Route(path: '/info/{id}/{slug}.{format}',  
name: 'test2',  
requirements: ['id' => '\d{2,4}', 'slug' => '.{2,}', 'format' => 'html|html|php|json'],  
defaults: ['format' => 'html'])  
public function test2($id, $slug, $format)  
{  
    return $this->render('cms/index.html.twig', [  
        'controller_name' => "/info/".$id."/". $slug,  
    ]);  
}
```

Le paramètre requirements nécessite l'utilisation de regex.

Le paramètre par défaut est toujours en fin d'url (dans ce cas son séparateur (.) n'est pas obligatoire).

PARAMETRE SYSTEME

PARAMETRE {_format}

Ajuste le header en fonction du format. (XML,JSON,HTML,...). Très utile

PARAMETRE {_locale}

Utile pour les sites Internet nécessitant une traduction, _locale se base par rapport à la constante Locale définie.

<https://symfony.com/doc/current/translation/locale.html>

Plus d'infos :

<https://symfony.com/doc/current/routing.html#special-parameters>

GENERER DES URLs

```
#[Route(path: '/info/{id}/{slug}.{!format}',  
name: 'test2',  
requirements: ['id' => '\d{2,4}', 'slug' => '.{2,}', 'format' => 'html|html|php|json'],  
defaults: ['format' => 'html'])]  
public function test2($id, $slug, $format)  
{  
    return $this->render('cms/index.html.twig', [  
        'controller_name' => "/info/".$id."/". $slug,  
    ]);  
}  
  
#[Route(path: '/url', name: 'url')]  
public function url()  
{  
    $url = $this->generateUrl('test2', ["id"=>55, "slug"=>"salut"], UrlGeneratorInterface::ABSOLUTE_URL);  
    return new Response($url);  
}
```

generateUrl : permet de générer des urls en fournissant les arguments.
Si vous souhaitez des urls absolues il faudra importer

use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

et utiliser par exemple :

\$this->generateUrl('sign_up', [], UrlGeneratorInterface::ABSOLUTE_URL);

Si vous avez un argument facultatif que vous souhaitez voir apparaître dans l'url il faudra ajouter un «!» devant l'argument.

A RETENIR : ROUTEUR

Une route possède mini 2 arguments :

- Le pattern correspondant et le controller à exécuter.

Quand une URL est lancée le routeur va essayer de trouver une correspondance au pattern.

On peut définir nos routes directement dans le controller ou dans un fichier de configuration

Une route peut contenir des arguments par défaut et des valeurs prédéfinies

On peut générer des URL (relatives ou absolues) au sein d'un controller, c'est pour cela que le nom de votre route doit être unique.

A QUOI SERT LE CONTRÔLEUR (CONTROLLER)?

Il contient le logique de votre application.

Il permet d'interagir avec votre base de données en lançant des méthodes créées dans des repositories, faire des appels à des services ou des modèles, renvoyer des réponses ou des vues.

Le rôle minimal d'un contrôleur est de retourner une «réponse»

L'OBJET RESPONSE

Le contrôleur le plus simple que l'on peut écrire (sans héritage ni injection) est celui-ci, nous avons ici une méthode renvoyant un objet de type «response». Lancez l'outil pour dev. > Network / headers et Response

```
namespace App\Controller;

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

class CmsController
{
    #[Route('/cms', name: 'app_cms')]
    public function index(): Response
    {
        return new Response("Hello word");
    }
}
```

L'OBJET RESPONSE

Voici l'exemple précédent permettant cette fois de rendre une vue en passant par response. On injecte la dépendance de twig en tant qu'argument de la méthode

```
namespace App\Controller;

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

class CmsController
{
    #[Route('/cms', name: 'app_cms')]
    public function index(Environment $twig): Response
    {
        return new Response($twig->render('cms/index.html.twig', []));
    }
}
```

Symfony utilise l'autowiring qui permet d'injecter automatiquement dans la méthode les classes dont nous avons besoin sans aucune configuration / instantiation
https://symfony.com/doc/current/service_container/autowiring.html
https://symfony.com/doc/current/service_container.html

AUTOWIRING

Autowiring de par un procédé interne à symfony permet d'utiliser toutes les classes dans src en tant qu'argument de services ou controller sans configurer ou instancier quoi que ce soit.

Pour les connaitre, lancez cette commande :
`php bin/console debug:autowiring`

https://symfony.com/doc/current/service_container/autowiring.html

L'OBJET RESPONSE

Voici l'équivalent de la version précédente mais cette fois-ci utilisant non pas l'injection de dépendance mais l'héritage. «`AbstractController`» possède des raccourcis de rendu comme `«$this->render»`.

Par simplicité nous utiliserons toujours l'héritage dans les contrôleurs.

```
namespace App\Controller;

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

class CmsController extends AbstractController
{
    #[Route('/cms', name: 'app_cms')]
    public function index(): Response
    {
        return $this->render('cms/index.html.twig', [
            'controller_name' => 'CmsController',
        ]);
    }
}
```

L'OBJET REQUEST

L'objet Request est important : il permet de manipuler les paramètres de la requête

Nous avons vu qu'il était possible de récupérer les paramètres de la route à travers le controller en le passant en tant qu'argument (section «Routeur»)

Il permet également de récupérer des paramètres hors routing

```
namespace App\Controller;

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

class CmsController extends AbstractController
{

    #[Route(path: '/info/{id}/{slug}', name: 'test')]
    public function test($id, $slug, Request $request) : Response
    {

        $cpm = $request->query->get('cpm');
        return new Response("{$cpm}");
    }
}
```

L'OBJET REQUEST

Voici l'ensemble des variables que peut récupérer request

Variables d'URL	\$request t->query	\$_GET	<code>\$request->query->get('tag')</code>
Variables de formulaire	\$request t->request t	\$_POST	<code>\$request->request->get('tag')</code>
Variables de cookie	\$request t->cookie s	\$_COOKIE	<code>\$request->cookies->get('tag')</code>
Variables de serveur	\$request t->server	\$_SERVER	<code>\$request->server->get('REQUEST_URI')</code>
Variables d'en-tête	\$request t->header s	\$_SERVER['HTTP_*']	<code>\$request->headers->get('USER_AGENT')</code>
Paramètres de route	\$request t->attributes	n/a	On utilise <code>\$id</code> dans les arguments de la méthode, mais vous pourriez également écrire <code>\$request->attributes->get('id')</code>

L'OBJET REQUEST

Cet objet associé à Response permet également de disposer de méthodes permettant de tester si les données sont de type post, si un requête ajax est lancée, le chemin du serveur.

<https://symfony.com/doc/current/controller.html#the-request-and-response-object>

https://symfony.com/doc/current/components/http_foundation.html#request
(Request object dans un projet from scratch)

https://symfony.com/doc/current/introduction/http_fundamentals.html#requests-and-responses-in-php

MANIPULATION COMPLEXE REponse

L'objet response permet en plus de fournir une réponse texte ou s'associer à une vue de réaliser par exemple des redirections

```
#[Route(path: '/redirection', name: 'redirection')]
public function redirection()
{
    $url = $this->generateUrl('test', ["id"=>55, "slug"=>"salut"]);
    // return new RedirectResponse($url);
    return $this->redirect($url);
}
```

Modifiez le fichier config/packages/web_profiler.yaml

```
when@dev:
    web_profiler:
        toolbar: true
        intercept_redirects: true

    framework:
        profiler:
            only_exceptions: false
            collect_serializer_data: true
```

Cela permet d'intercepter la redirection

MANIPULATION COMPLEXE REponse

Gérer une json response : (dans le cas d'une création d'API par exemple)

```
#[Route(path: '/json', name: 'json')]  
public function jsonReponse()  
{  
    $tab = ["id"=>3,"nom"=>"Dupond","prenom"=>"Louis"];  
    //return new JsonResponse($tab);  
    return $this->json($tab);  
}
```

MANIPULATION COMPLEXE REponse

Gérer une session :

```
#Route(path: '/session', name: 'session')
public function sessionUser(SessionInterface $session)
{
    $session->set('user', ["id"=>3, "nom"=>"Dupond", "prenom"=>"Louis"]);
    $user = $session->get('user');
    return new Response("<body>Bonjour ".$user['prenom']."</body>");
}
```

Visualisez le profiler pour voir l'ensemble de la session

MANIPULATION COMPLEXE REponse

Gérer un message flash :

Variable session qui ne dure que le temps d'un affichage après il est supprimé
Très utile pour la validation des formulaires ou des alertes informatives

```
#[Route(path: '/ajoutflash', name: 'flashbag')]
public function ajoutFlash()
{
    //CREATION DES MESSAGES FLASHBAG
    $this->addFlash('info', 'Formulaire validé');
    $this->addFlash('info', 'Vous pouvez maintenant attendre la validation modérateur');
    //REDIRECTION
    return $this->redirectToRoute('app_cms');
}
```

```
#[Route('/cms', name: 'app_cms')]
public function index(): Response
{
    return $this->render('cms/index.html.twig', [
        'controller_name' => 'CmsController',
    ]);
}
```

MANIPULATION COMPLEXE REponse

Pour les afficher il suffit de modifier la vue.

```
<div>
    {# On affiche tous les messages flash dont le nom est « info » #}
    {% for message in app.flashes('info') %}
        <p>Message flash : {{ message }}</p>
    {% endfor %}
</div>
```

Si vous actualisez la page, les messages disparaissent automatiquement.
Nous reviendrons sur les vues par la suite.

Sachez que «app» est une variable globale à twig pour symfony permettant de récupérer énormément d'éléments comme ici des variables internes

MANIPULATION COMPLEXE REponse

Gérer des pages erreurs 404 :

```
#[Route(path: '/erreur404', name: 'erreur404')]
public function erreur404()
{
    throw $this->createNotFoundException('Page inexistante.');
}
```

Gérer des pages erreurs 410 (pour google) :

```
#[Route(path: '/erreur410', name: 'erreur410')]
public function erreur410()
{
    // $response = new Response();
    // $response->headers->set('Content-Type', 'text/html');
    // $response->setStatusCode(410);
    // return $response;
    return new Response($this->renderView('cms/index.html.twig', [
        'controller_name' => 'CmsController',
    ]), Response::HTTP_GONE);
}
```

A RETENIR

Un contrôleur doit toujours retourner une «Response» soit à travers un objet ou une méthode renvoyant un objet «Response»

L'Objet «Request» permet de travailler sur les paramètres

Dans un contrôleur vous jouerez avec la BDD, les entités, les paramètres, les templates, les services afin de fournir une réponse

Vérifiez qu'il n'existe pas une méthode ou un objet «tout fait» avant de coder quoi que ce soit. Vous pourriez être surpris.

EXERCICE

Nous allons créer un blog.

Pour cela vous allez renommer votre contrôleur CMS en mettant un . devant
Vous travaillerez sur BlogController.php pour obtenir cette structure

```
homepage:
    path: /
    controller: App\Controller\BlogController::index

article_add:
    path: /add
    controller: App\Controller\BlogController::add

article_show:
    path: /show/{url}
    controller: App\Controller\BlogController::show

article_edit:
    path: /edit/{id}
    controller: App\Controller\BlogController::edit
    requirements:
        id: '\d+'

article_remove:
    path: /remove/{id}
    controller: App\Controller\BlogController::remove
    requirements:
        id: '\d+'
```

EXERCICE

```
// src/Controller/BlogController.php

namespace App\Controller;

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

class BlogController extends AbstractController
{
    #[Route(path: '/', name: 'homepage')]
    public function index()
    {
        return new Response('<h1>Page d\'accueil</h1>');
    }

    #[Route(path: '/add', name: 'article_add')]
    public function add()
    {
        return new Response('<h1>Ajouter un article</h1>');
    }

    #[Route(path: '/article/{url}-{id}', name: 'article_show', requirements: ['id' => '\d{1,8}', 'slug' => '.{1,}'])]
    public function show($url, $id)
    {
        return new Response("<h1>Lire l'article {$url} - {$id}</h1>");
    }
}
```

EXERCICE

```
#Route(path: '/edition/{id}', name: 'article_edit', requirements: ['id' => '\d{1,8}'])
public function edit($id)
{
    return new Response('<h1>Modifier l\'article ' . $id. '</h1>');
}

#Route(path: '/suppression/{id}', name: 'article_remove', requirements: ['id' => '\d{1,8}'])
public function remove($id)
{
    return new Response('<h1>Supprimer l\'article ' . $id. '</h1>');
}
```