3 minutes

Validation et paramètres supplémentaires

Validation des paramètres

Avec le `Router` de `Symfony`, la première route qui `match` l'emporte.

Si vous avez par exemple `/blog/{id}` et `/blog/{name}` :

```
<?php

namespace App\Controller;

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

class DefaultController
{
  #[Route(path: '/', name: 'index', methods: ['GET'], schemes: ['https'])]
  public function index(Request $request)
  {
    dd($request);
    return new Response('<h1>Hello World !</h1>');
  }

  #[Route(path: '/blog/{id}', name: 'blogId')]
  public function blogById(int $id)
  {
    return new Response('<h1>Blog ID</h1>' . $id);
  }

  #[Route(path: '/blog/{name}', name: 'blogName')]
  public function blogByName(string $name = 'all')
  {
    return new Response('<h1>Blog NAME</h1>' . $name);
  }
}
```

Copier

Essayez de vous rendre maintenant sur `https://127.0.0.1:8000/blog/123` et sur `https://127.0.0.1:8000/blog/jesuis`.

La premier route matchera tout le temps quel que soit le paramètre entré après `blog`.

Dans le cas d'une chaîne de caractères vous aurez une erreur car nous avons typé les identifiants comme étant des entiers (`int`).

Pour cette raison, `Symfony` permet d'utiliser une **propriétés `requirements` qui va permettre de restreindre une route avec l'utilisation d'expressions régulières**.

Si vous ne connaissez pas les expressions régulières, vous pouvez aller en apprendre plus dans le chapitre `PHP` sur les chaînes de caractères.

```
<?php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Request;
```

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

class DefaultController
{
  #[Route(path: '/', name: 'index', methods: ['GET'], schemes: ['https'])]
  public function index(Request $request)
  {
    dd($request);
    return new Response('<h1>Hello World !</h1>');
  }

  #[Route(path: '/blog/{id}', name: 'blogId', requirements: ['id' => '\d+'])]
  public function blogById(int $id)
  {
    return new Response('<h1>Blog ID</h1>' . $id);
  }

  #[Route(path: '/blog/{name}', name: 'blogName')]
  public function blogByName(string $name = 'all')
  {
    return new Response('<h1>Blog NAME</h1>' . $name);
  }
}
```

Copier

Ici nous indiquons que le paramètre `id` ne peut être qu'un nombre, en supposant que les identifiants sont des nombres dans notre application.

Si ce n'est pas un nombre, la route `blogId` ne matchera pas et ce sera la route `blogName` qui matchera.

Essayez de vous rendre maintenant sur `https://127.0.0.1:8000/blog/123` et sur `https://127.0.0.1:8000/blog/jesuis`.

Paramètres supplémentaires

Il est possible de passer des paramètres par défaut qui ne sont pas inclus dans la configuration de la route. Cela peut être utile pour passer certains arguments en fonction des routes.

Pour ce faire, il suffit d'utiliser la propriété `defaults` et de lui passer un tableau associatif contenant les paramètres avec leur valeur par défaut :

```
<?php

namespace App\Controller;

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

class DefaultController
{
  #[Route(path: '/', name: 'index', methods: ['GET'], schemes: ['https'])]
  public function index(Request $request)
  {
    dd($request);
    return new Response('<h1>Hello World !</h1>');
  }

  #[Route(path: '/blog/{name}', name: 'blogName', defaults: ['title' => 'Dyma'])]
  public function blogByName(string $name = 'all', string $title)
  {
    return new Response('<h1>Blog NAME</h1>' . $name);
```

```
    }
}
```

Copier