

Le routing

Table des matières

I. Contexte	3
II. Le routing avec les annotations	3
A. Configuration routing avec annotations	3
B. Paramètres Route.....	4
C. Exercice : Quiz	7
III. Le routing avec YAML et débogage	8
A. Introduction.....	8
B. Le routing avec YAML	8
C. Outils de débogage	9
D. Exercice : Quiz.....	11
IV. Essentiel	11
V. Auto-évaluation	12
A. Exercice	12
B. Test.....	12
Solutions des exercices	13

I. Contexte

Durée : 1 h

Environnement de travail :

PC ou Mac avec 8 Go de RAM minimum

IDE : VSCode ou autre

Prérequis :

Notion de PHP

Notion de POO

Contexte

Dans une application moderne qui suit le MVC (Modèle-Vue-Contrôleur), il est nécessaire de mettre en place côté back-end un système qui permet une navigation cohérente entre les pages. Ce système s'appelle le *routing*.

Le routing, ou routage, permet de définir comment les URL d'une application doivent être mappées. Quant au mapping des URL, il s'agit du processus consistant à associer une URL à une action spécifique.

En effet, à chaque requête HTTP, l'application doit savoir non seulement quelle route est active mais aussi quelle méthode de contrôleur y est associée. Une route peut également contenir des paramètres qui permettront aux contrôleurs de fournir des informations précises. Ainsi, le routing est important pour améliorer l'expérience utilisateur, car il permet de fournir des URL claires et indexées par les moteurs de recherches.

Le framework Symfony fournit bien évidemment les outils adaptés pour maîtriser le routing. Mais comment fonctionne-t-il ? Quelles sont les erreurs à éviter ? C'est ce que nous aborderons à travers ce cours sur le routing dans Symfony 5.4.

II. Le routing avec les annotations

A. Configuration routing avec annotations

Selon les bonnes pratiques, il est recommandé d'écrire son système de routes *via* les PHP attributes. Nous verrons ensuite le routing écrit en annotations (qui deviendra *deprecated* dans la prochaine version majeure) ou encore au format YAML dans le routing avec YAML et débogage de ce cours. C'est aussi une syntaxe commune que l'on peut retrouver dans certains bundles ou anciens projets Symfony.

Les annotations sont utilisées dans la plupart des langages informatiques. Il s'agissait du nom des métadonnées qu'utilisaient PHP avant sa version 8, et par extension Symfony.

Ainsi depuis PHP 8, bien qu'il soit aussi possible d'utiliser les annotations, les attributs ont été introduits pour remplacer ceux-ci. Les attributs ont l'avantage d'être maintenant pris nativement en charge par PHP, contrairement aux annotations qui nécessitent un logiciel tiers pour les compiler.

Simplement dit, ne soyez pas étonné si on parle parfois d'annotations dans le routing alors qu'il s'agit d'attributs.

Exemple

Annotations :

```
1 /**
2  *@Route("/tasks", name="task_list")
3  * public function list()
4  * {
5  *     // ...
6  * }
```

Attributs :

```
1 #[Route("/tasks", name:"task_list", methods: ['GET'])]
2 public function list()
3 {
4     // ...
5 }
```

Méthode

Pour que le système d'annotations fonctionne, il faut installer le bundle suivant à l'aide du terminal de commande :

```
1 $ composer require doctrine/annotations
```

Dans le dossier routes, créez ensuite le fichier annotations.yaml et inscrivez-y le code suivant :

```
1 #config/routes/annotations.yaml
2 controllers:
3     resource: ../../src/Controller
4     type: annotation
5
6 kernel:
7     resource: ../../src/kernel.php
8     type: annotation
```

Ces instructions vont permettre au kernel, c'est-à-dire le noyau ou le cœur de notre application, de comprendre qu'il doit lire les annotations du contrôleur pour connaître la route attribuée. Pour chaque requête HTTP, le kernel va parser et lexer chaque annotation, puis va associer la route au contrôleur ainsi que ses paramètres.

Voyons comment fonctionne maintenant notre application.

Méthode

Importer un namespace

Il est nécessaire à présent d'importer un namespace en utilisant cette commande dans chaque classe de contrôleur :

```
1 use Symfony\Component\Routing\Annotation\Route
```

Nous avons ainsi configuré tous les éléments nécessaires pour que les annotations des contrôleurs fonctionnent.

L'énorme avantage d'utiliser les annotations ou les PHP attributes pour le routing plutôt que d'utiliser des fichiers dédiés, c'est qu'il n'y a pas besoin de naviguer entre les fichiers contrôleurs et routes, puisque vous pouvez accéder immédiatement à l'information associant la route à un contrôleur.

B. Paramètres Route

Les paramètres de la fonction d'annotation « *Route* » définissent les propriétés d'une route. Cette fonction permet de mapper une URL à une action ou une méthode dans le contrôleur de l'application. En voici d'ailleurs quelques-unes.

Methods

Par défaut, une route fonctionne avec toutes les méthodes HTTP. Cependant il est possible de restreindre une route en précisant un ou plusieurs verbes HTTP. Si vous ne maîtrisez pas les méthodes de requête http, n'hésitez pas à vous référer à un site comme [rfc-editor.org/rfc/rfc2616](https://www.rfc-editor.org/rfc/rfc2616)¹, qui est les standards officiels du web, ou

1 <https://www.rfc-editor.org/rfc/rfc2616>

developer.mozilla.org/fr/docs/Web/HTTP/Methods¹. Toutefois, les formulaires HTML ne prennent en charge que les méthodes GET et POST. Il faudra donc tricher avec Symfony pour soumettre des formulaires avec d'autres méthodes comme DELETE ou PUT.

Exemple

```
1 #[Route("/tasks", name:"task_list", methods: ['GET', 'HEAD'])]
2 public function list()
3 {
4     // ...
5 }
```

Schemes

Schemes permet de spécifier si nous voulons qu'une page soit accessible en HTTP ou seulement en HTTPS.

Exemple

```
1 #[Route("/tasks", name:"task_list", methods: ['GET', 'HEAD'], schemes:[HTTP] )]
2 public function list()
3 {
4     // ...
5 }
```

Path et son paramètre

Le paramètre « **path** » dans le routage désigne une partie variable d'une URL qui peut être utilisée pour fournir des informations supplémentaires.

Son paramètre de chemin est défini dans une portion variable entourée de crochets. Il peut être dynamique.

Méthode Paramètre dynamique dans la route

Par exemple, dans une liste d'utilisateur, le paramètre dynamique de path va permettre de récupérer un utilisateur précis. Mais bien sûr il ne serait pas raisonnable de créer une route par utilisateur, imaginez qu'il y en ait des milliers ! On a donc besoin d'un paramètre dynamique.

Symfony apporte des solutions. Il faut pour ajouter un paramètre dynamique ajouter à la route des accolades avec entre elles, le nom de la propriété que l'on utilise pour identifier de façon unique un élément de la liste. Id est souvent utilisé mais on peut la nommer autrement.

```
1 #[Route("/user/{name} " )]
```

Méthode Paramètre defaults

Il est possible d'ajouter le paramètre « **defaults** » à la route avec une valeur par défaut. Ce paramètre fait référence au paramètre de path.

```
1 #[Route("/user/{name} " , defaults: [ 'name' => 'john' ])]
```

On peut aussi mettre cette valeur par défaut en ajoutant un point d'interrogation après le paramètre et la valeur par défaut. C'est un choix que certains préfèrent, tandis que d'autres le trouvent moins lisible.

```
1 #[Route("/user/{name?john} " )]
```

¹ <https://developer.mozilla.org/fr/docs/Web/HTTP/Methods>

Méthode Récupération du paramètre dans le contrôleur

Comment récupérer le paramètre dans le contrôleur ? Par exemple avec `attributes-get()` de `request`.

```
1 #[Route("/user/{name} " , defaults: [ 'name' => 'john'])]
2 public function show(Request $request)
3 {
4     $title= $request->attributes->get('name');
5
6     // ...
7 }
```

Cependant, Symfony, qui utilise Doctrine (son ORM), est très efficace. Vous pouvez simplement préciser la propriété dans les paramètres de la fonction et il sera capable de comprendre seul ce qu'il doit chercher, grâce au typage du paramètre. C'est ce que l'on appelle un `ParamConverter`.

Exemple

```
1 #[Route("/user/{name} " , defaults: [ 'name' => 'john'])]
2 public function show(Request $request, $name)
3 {
4     // ...
5 }
```

Propriété requirements

Ce paramètre peut servir de validation de paramètres de route. Il permet de pouvoir s'assurer que le paramètre de path correspond au bon format. Souvent la valeur sera décrite en **regex**, un langage que vous ne maîtrisez peut-être pas, mais l'on va rester sur des notions simples.

- Dans le premier exemple, les caractères acceptés sont des lettres alphabétiques de a à z en minuscule, et en majuscule A à Z
- Dans le 2^e exemple, les caractères acceptés sont entre 0 à 9
- Dans le 3^e exemple, le requirement disparaît, il est précisé entre des chevrons, c'est une autre façon de faire, similaire à `defaults`

```
1 #[Route("/user/{name} " , requirements: [ 'name' => '[a-zA-Z]+' ])
2 public function showUserWithName(Request $request, $name)
3 {
4     // ...
5 }
6
7 #[Route("/user/{id} " , requirements: [ 'id' => '\d+' ])
8 public function showUserWithId(Request $request, $name)
9 {
10    // ...
11 }
12 #[Route("/user/{id<\d+>} ")
13 public function showUser(Request $request, $name)
14 {
15    // ...
16 }
```

En revanche, ce paramètre peut également servir à ajouter des paramètres spéciaux.

Pour cela, il suffit de les ajouter dans le tableau associatif de requirements :

- `_controller` est utilisé pour déterminer quel contrôleur et quelle action sont exécutés lorsque la route est appelée,
- `_format` est utilisé pour définir le format de la requête de l'objet Request,
- `_fragment` est utilisé pour définir l'identifiant de fragment qui est la dernière partie facultative d'une URL qui commence par #,
- `_locale` est utilisé pour définir les paramètres régionaux de la demande.

C. Exercice : Quiz

[solution n°1 p.15]

Question 1

Avec un projet qui tourne sous PHP 8 ou une version ultérieure et après avoir installé doctrine/annotations et appelé par le namespace annotation\Route, quelle affirmation est vraie ?

- ☐ Seulement les annotations permettront de déclarer une route
- ☐ Seulement les attributs permettront de déclarer une route
- ☐ Les annotations et les attributs peuvent être utilisés pour déclarer une route

Question 2

Il est possible de mettre un paramètre fixe dans une route.

- ☐ Vrai
- ☐ Faux

Question 3

Si on ne définit pas la propriété `methods` dans la route :

- ☐ Le verbe de la méthode sera GET
- ☐ Le verbe de la méthode sera POST
- ☐ Tous les verbes seront acceptés

Question 4

Que signifie cette définition de route : `#[Route("/category/{name?news}")] ?`.

- ☐ Que `news` est le paramètre de la route `category`
- ☐ Que `news` est la valeur par défaut du paramètre de la route `category`
- ☐ Que `category` n'a pas de `name`

Question 5

Quelle route est écrite sans erreur ?

- ☐ `#[Route("/user/{username}" , requirements: ['_locale' => 'fr' , 'username' => '[a-zA-Z]+']`
- ☐ `#[Route("/task/{id = 0}" , requirements: ['name' => '[a-zA-Z]+']`
- ☐ `#[Route("/comment/{date}" , requirements: ['date' => '[a-zA-Z]+']`

III. Le routing avec YAML et débogage

A. Introduction

Il existe plusieurs façons de configurer les routes dans un projet Symfony. Il est possible de les définir dans un fichier distinct YAML, XML, ou PHP. Aujourd'hui cette méthode, qui consiste à mettre les routes dans un fichier annexe, est de plus en plus rare, surtout en XML et PHP. Il faut savoir qu'ils nécessiteront, contrairement au YAML, de mettre à jour le fichier Kernel.php.

B. Le routing avec YAML

Avant de voir comment on configure un fichier YAML, sachez qu'il est possible, même si cela n'est pas un avantage, de choisir dans un même contrôleur des méthodes différentes de configuration de routing. Par exemple, une méthode de contrôleur aura une route configurée avec php attributes et une autre méthode de ce même contrôleur peut être configurée à partir du fichier routes.yaml

Méthode

Voyons à présent comment créer des routes en YAML. Dans le fichier routes.yaml, qui se trouve dans le dossier config à la racine du projet, indiquez le nom de la méthode du contrôleur que vous souhaitez associer à la route.

En dessous, avec une indentation dans **path:**, indiquez le chemin de la route avec ses éventuels paramètres puis encore en dessous dans **controller:**, indiquez le chemin d'accès jusqu'au contrôleur.

Si vous souhaitez ajouter d'autres paramètres de routes, vous pouvez les ajouter une à une en respectant bien l'indentation (utilisez des espaces et non des tabulations). En effet, la syntaxe YAML y est très sensible.

Exemple

```
1 list:
2   path: /tasks
3   controller: App\Controller\TaskController::list
4
5
6 task_show:
7   path: /task/{id<\d+>?0}
8   controller: App\Controller\TaskController::task_show
9   methods: [GET, POST]
10  schemes: [https]
```

Dans l'exemple ci-dessus, vous remarquez certainement que le principe reste le même qu'avec le système d'annotation.

Par exemple, dans la méthode task_show, il y a dans path un paramètre dynamique suivi entre chevrons d'un requiemment ainsi que d'une valeur par défaut. « Requirement » et « defaults » auraient aussi pu être énumérés en dessous.

Remarque

Si vous choisissez de définir l'ensemble de vos routes en YAML alors il est préférable, pour une meilleure organisation, de créer dans le dossier routes (qui se trouvent dans le dossier config à la racine du projet), des fichiers qui seront classés dans des dossiers, suivant l'arborescence que vous désirerez, contenant chacun sa route.

Ainsi vous aurez :

```
config\
  routes\
```



```
users\  
  tasks\  
    index.yaml  
    task_show.yaml
```

Imaginez maintenant que pour une quelconque raison, une route de votre application n'existe plus. Symfony propose un contrôleur de redirection afin que les utilisateurs soient redirigés vers la page que vous avez remplacée.

Exemple

```
1 redirection_vers_nouvelle_page:  
2   path: /ancienne_page  
3   controller: Symfony\Bundle\FrameworkBundle\Controller\RedirectController  
4   defaults:  
5     route: 'nom_nouvelle_route'  
6     permanent: true  
7     keepQueryParams: true
```

Si l'on reprend ce code YAML, ligne par ligne, voici ce que l'on obtient :

- Dans la première ligne, `redirection_vers_nouvelle_page` est le nom que vous choisirez pour la redirection.
- Dans la deuxième, `path` contient le nom de l'ancienne adresse.
- Dans la troisième, `controller` précise le contrôleur que nous offre Symfony pour la redirection.
- Dans la quatrième, `defaults` contient les nouveaux éléments.
- Dans la cinquième, `route` donne le nom de la nouvelle route
- Dans la sixième, `permanent` permet de préciser si la redirection est permanente
- Enfin, dans la dernière, `keepQueryParams` permet de récupérer les paramètres de routes.

C. Outils de débogage

Les outils de débogage pour Symfony sont des fonctionnalités intégrées dans le framework qui permettent aux développeurs de diagnostiquer et de résoudre les erreurs et les problèmes dans leur application Symfony. Dans cette partie du cours, nous allons nous en imprégner en commençant par présenter brièvement la commande `debug:router`.

Définition

La commande `debug:router` permet de connaître toutes les routes qui sont dans l'application.

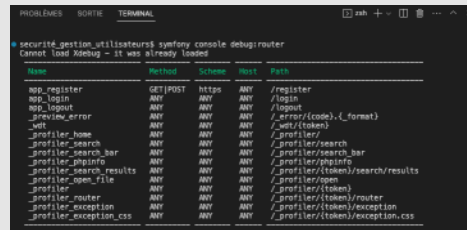
Méthode

```
1 $ symfony console debug:router
```

Grâce à `debug:router` nous pouvons visualiser toutes les routes de l'application dans un tableau.

Ici, il y a 3 routes dans le cadre d'un environnement en développement. Nous verrons plus de routes, notamment celles liées au débogage (commençant par un underscore). Celles-ci sont `app_register`, `app_login`, `app_logout`. À la droite de chaque route, vous avez les propriétés de route les plus souvent utilisées, à savoir `Method`, `Scheme`, `Host` ainsi que le `Path`.

Cet outil vous permet de visualiser en un clin d'œil les informations nécessaires sur les routes.



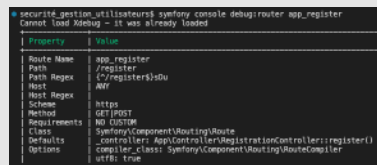
Name	Method	Scheme	Host	Path
app_register	GET POST	https	ANY	/register
app_login	ANY	ANY	ANY	/login
app_logout	ANY	ANY	ANY	/logout
previous_error	ANY	ANY	ANY	/error/{code}/{format}
url	ANY	ANY	ANY	/url/{token}
_profiler_home	ANY	ANY	ANY	/_profiler/
_profiler_search	ANY	ANY	ANY	/_profiler/search
_profiler_search_bar	ANY	ANY	ANY	/_profiler/search_bar
_profiler_phpinfo	ANY	ANY	ANY	/_profiler/phpinfo
_profiler_search_results	ANY	ANY	ANY	/_profiler/token/search/results
_profiler_open_file	ANY	ANY	ANY	/_profiler/open
_profiler	ANY	ANY	ANY	/_profiler/token
_profiler_router	ANY	ANY	ANY	/_profiler/token/router
_profiler_exception	ANY	ANY	ANY	/_profiler/token/exception
_profiler_exception_css	ANY	ANY	ANY	/_profiler/token/exception.css

Résultat debug:router dans le terminal

Méthode

Cependant, vous pouvez aussi avoir un peu plus d'informations sur une route en particulier en tapant la même ligne de commande suivie du nom de la route sur laquelle vous souhaitez être renseigné, comme suit :

- 1 `$ symfony console debug:router app_register`
- 2 `$ symfony console router:match /register`



Property	Value
Route Name	app_register
Path	/register
Path Regex	{/register}\$du
Host	ANY
Host Regex	
Scheme	https
Method	GET POST
Requirements	NO_CUSTOM
Class	Symfony\Component\Routing\Route
Defaults	_controller: App\Controller\RegistrationController::register()
Options	compiler_class: Symfony\Component\Routing\RouteCompiler
utf8	true

Résultat d'un route dans debug:router

Vous trouverez ici toutes les propriétés possibles et les informations nécessaires sur la route app_register.

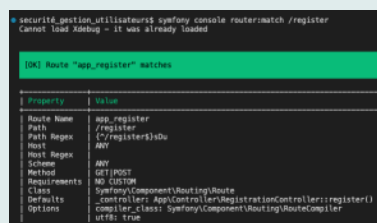
Définition

La ligne de commande router:match permet de vérifier si la route matche, c'est-à-dire fonctionne correctement.

- 1 `$ symfony console router:match /register`

Dans cette commande, nous avons ajouté le chemin de la route et non le nom de la route.

Dans l'exemple ci-dessous, ça a matché et tout fonctionne correctement. Cependant, essayez d'ajouter la propriété schemes et de mettre en HTTPS si vous êtes en local ou HTTP sur un serveur configuré en HTTPS, vous aurez alors un tout autre résultat.



```

[OK] Route "app_register" matches

```

Property	Value
Route Name	app_register
Path	/register
Path Regex	{/register}\$du
Host	ANY
Host Regex	
Scheme	ANY
Method	GET POST
Requirements	NO_CUSTOM
Class	Symfony\Component\Routing\Route
Defaults	_controller: App\Controller\RegistrationController::register()
Options	compiler_class: Symfony\Component\Routing\RouteCompiler
utf8	true

Résultat router:match dans le terminal

D. Exercice : Quiz

[solution n°2 p.16]

Question 1

On peut créer exactement les mêmes routes avec les mêmes paramètres en YAML qu'avec le système d'annotations.

- ☐ Vrai
- ☐ Faux

Question 2

Quel est l'intérêt de `debug:router` ?

- ☐ De pouvoir visualiser quelle route n'est pas active
- ☐ De pouvoir modifier une route
- ☐ De pouvoir visualiser toutes les routes de l'application

Question 3

Il est possible de créer autant de fichiers que l'on souhaite dans le dossier routes du dossier config.

- ☐ Vrai
- ☐ Faux

Question 4

La commande `router:match` permet d'avoir aussi des informations sur :

- ☐ Les contraintes de paramètres
- ☐ Les id
- ☐ Tous les paramètres de la route

Question 5

On peut mettre plusieurs routes dans un même fichier YAML.

- ☐ Vrai
- ☐ Faux

IV. Essentiel

Le routing ou routage est un élément clé des applications web modernes. Il permet de définir comment les URL de votre application doivent être mappées aux actions. Symfony permet de configurer simplement le routing à l'aide d'un système d'annotations ou de PHP attributes, d'un fichier tiers en YAML ou plus rarement en XML voire en PHP.

De plus, nous avons vu comment relier le routage à nos contrôleurs. Nous avons également vu qu'il existe plusieurs paramètres de route et nous avons étudié les plus utilisés : `name`, `path` et son paramètre, `methods`, `requirements`, `schemes`, `defaults`.

Même s'il est possible de faire cohabiter plusieurs méthodes de routing, il est de rigueur pour vous et pour ceux qui reprendront éventuellement le code, de choisir une seule méthode, à savoir un système d'annotations qui permet de visualiser immédiatement le routing avec la méthode du contrôleur.

Cependant, les fichiers YAML sont une bonne alternative, même s'ils nécessitent plus de manipulation dans votre code. Pensez à bien organiser vos fichiers si vous travaillez ou optez pour cette solution de routing, ainsi qu'à utiliser les outils de débogage qui vous feront gagner un temps précieux.

V. Auto-évaluation

A. Exercice

Vous êtes un développeur Junior dans une agence web. On vous confie une application web Symfony 5.4 dans laquelle de nombreuses modifications ont été faites.

Question 1

[solution n°3 p.17]

On vous demande dans cette application de mettre en place les routes pour les méthodes de UserController, à savoir listUsers(), createUser(), editUser(), en précisant les méthodes, le nom, le chemin (path) sachant qu'editUser() a pour paramètre de path :id et qu'il est nécessaire de mettre une contrainte de validation ne permettant d'ajouter que des caractères de 0 à 9. Utilisez les annotations pour répondre à cette demande.

Question 2

[solution n°4 p.17]

Votre chef vient vers vous et vous dit que finalement les routes doivent être mises dans un fichier YAML. Mettez toutes ces routes dans un fichier nommé user.yaml.

B. Test

Exercice 1 : Quiz

[solution n°5 p.18]

Question 1

À combien de verbes est limité methods ?

- ☐ 2
- ☐ 3
- ☐ Autant qu'il existe de verbes

Question 2

Le kernel ou noyau de l'application lit le routing après :

- ☐ Une requête
- ☐ Une réponse
- ☐ Un formulaire

Question 3

Il existe des paramètres dits « *spéciaux* ».

- ☐ Vrai
- ☐ Faux

Question 4

Quelle ligne de commande va vous permettre de déboguer une route ?

- ☐ \$ symfony console debug:router:match
- ☐ \$ symfony console debug:router:match /path_route
- ☐ \$ symfony debug:router:match /nom_route

Question 5


Il est possible de mettre une valeur par défaut directement dans le paramètre de path.

- ☐ Vrai
- ☐ Faux

Solutions des exercices


Exercice p. 7 Solution n°1**Question 1**

Avec un projet qui tourne sous PHP 8 ou une version ultérieure et après avoir installé doctrine/annotations et appelé par le namespace annotation\Route, quelle affirmation est vraie ?

- ☐ Seulement les annotations permettront de déclarer une route
- ☐ Seulement les attributs permettront de déclarer une route
- ☒ Les annotations et les attributs peuvent être utilisés pour déclarer une route
-  Les annotations, particulièrement pour les routes, constituent un terme générique qui englobe les attributs.


Question 2

Il est possible de mettre un paramètre fixe dans une route.

- ☒ Vrai
- ☐ Faux
-  Vrai. Néanmoins, il est bien plus avantageux de mettre un paramètre dynamique, ce qui évite de multiplier les routes.


Question 3

Si on ne définit pas la propriété methods dans la route :

- ☐ Le verbe de la méthode sera GET
- ☐ Le verbe de la méthode sera POST
- ☒ Tous les verbes seront acceptés
-  Methods permet de restreindre à certaines formes de requête HTTP.


Question 4

Que signifie cette définition de route : `#[Route ("/category/{name?news} ")] ?`.

- ☐ Que news est le paramètre de la route category
- ☒ Que news est la valeur par défaut du paramètre de la route category
- ☐ Que category n'a pas de name
-  Name est le paramètre dynamique de la route category et news est la valeur par défaut.

Question 5


Quelle route est écrite sans erreur ?

- ☒ `#[Route("/user/{username}" , requirements: ['_locale' => 'fr', 'username' => '[a-zA-Z]+'])`
- ☐ `#[Route("/task/{id = 0}" , requirements: ['name' => '[a-zA-Z]+'])`
- ☐ `#[Route("/comment/{date}" , requirements: ['date' => '[a-zA-Z]+'])`
-  Requirements peut contenir des contraintes de paramètre et aussi des paramètres spéciaux.

Exercice p. 11 Solution n°2


Question 1

On peut créer exactement les mêmes routes avec les mêmes paramètres en YAML qu'avec le système d'annotations.

- ☒ Vrai
- ☐ Faux
-  Vrai. C'est tout simplement une autre façon de coder. Les applications modernes ont tendance à préférer les annotations car cela permet une lecture plus facile et donne moins de manipulation de fichiers. Mais évitez de cumuler les 2 méthodes, cela alourdirait fortement la maintenance du projet dans le temps.


Question 2

Quel est l'intérêt de `debug:router` ?

- ☐ De pouvoir visualiser quelle route n'est pas active
- ☐ De pouvoir modifier une route
- ☒ De pouvoir visualiser toutes les routes de l'application
-  `debug:router` permet de visualiser rapidement toutes les routes de l'application ainsi que leurs paramètres principaux.


Question 3

Il est possible de créer autant de fichiers que l'on souhaite dans le dossier routes du dossier config.

- ☒ Vrai
- ☐ Faux
-  On peut aussi créer des sous-dossiers avec une arborescence pour une meilleure organisation des routes si on code celles-ci en YAML.

Question 4

La commande `router:match` permet d'avoir aussi des informations sur :


- ☒ Les contraintes de paramètres
- ☐ Les id
- ☒ Tous les paramètres de la route
-  Non seulement on peut voir si la route matche mais aussi avoir toutes les informations nécessaires sur celle-ci.

Question 5

On peut mettre plusieurs routes dans un même fichier YAML.

☒ Vrai

☐ Faux

 Ce qui est important c'est de bien respecter la syntaxe YAML avec ses indentations, toutes les routes déclarées seront ensuite lues par le kernel de Symfony.

p. 12 Solution n°3

Il vous faut commencer par importer dans les namespaces : « *use Symfony\Component\Routing\Annotation\Route;* ».

Puis voici les routes que vous devez mettre en place :

```

1  #[Route("/users", name:"user_list", methods:['GET'])]
2  public function listUsers( UserRepository $userRepo)
3  {
4      // ...
5  }
6
7  #[Route('/users/create', name: 'user_create', methods:['GET', 'POST'])]
8  public function createUsers(Request $request, EntityManagerInterface $em,
9  UserPasswordHasherInterface $encoder )
10 {
11     // ...
12 }
13
14 #[Route("/users/{id<\d+>}/edit", name:"user_edit", methods: ['GET', 'POST'])]
15 public function editAction(User $user, Request $request, EntityManagerInterface $em)
16 {
17     // ...
18 }
```

p. 12 Solution n°4


```

1 #config/routes/user.yaml
2
3 user_list:
4     path: /users
5     controller: App\Controller\UserController::list
6     methods: [GET]
7
8 user_create:
9     path: /users/create
10    controller: App\Controller\UserController::createUser
11    methods: [GET, POST]
12
13 user_edit:
14     path: /users/{id<\d+>}
15     controller: App\Controller\TaskController::editUser
16     methods: [GET, POST]
```

Exercice p. 12 Solution n°5


Question 1

À combien de verbes est limité methods ?

- ☐ 2
- ☐ 3
- ☒ Autant qu'il existe de verbes
-  Effectivement la limite n'est que dans le nombre de verbes mais si on les ajoute tous, on enlève l'intérêt de mettre des restrictions de verbes.


Question 2

Le kernel ou noyau de l'application lit le routing après :

- ☒ Une requête
- ☐ Une réponse
- ☐ Un formulaire
-  Le kernel va après chaque requête chercher la route et l'associer au contrôleur avec ses paramètres.


Question 3

Il existe des paramètres dits « spéciaux ».

- ☒ Vrai
- ☐ Faux
-  Il en existe 4 : _locale, _controller, _fragment, _format.


Question 4

Quelle ligne de commande va vous permettre de débayer une route ?

- ☐ \$ symfony console debug:router:match
- ☒ \$ symfony console debug:router:match /path_route
- ☐ \$ symfony debug:router:match /nom_route
-  Grâce à cette ligne de commande, vous saurez si la route fonctionne bien ou matche, mais c'est le chemin ou path qu'il faut entrer.

Question 5

Il est possible de mettre une valeur par défaut directement dans le paramètre de path.

- ☒ Vrai
- ☐ Faux
-  Il suffit dans l'accolade d'ajouter un « ? » suivi de la valeur par défaut.