API REST

Premiers pas avec Symfony

Pré-requis

Télécharger le binaire symfony : https://symfony.com/download

Vérifier les pré-requis :

```
symfony check:requirements
```

PHP 8.1, extensions xml, etc.

Installation

Installation d’un « micro-service » Symfony

Installation d’une application « micro-service »

```
symfony new myapp
```

alternative avec

```
composer create-project symfony/skeleton:"7.1.*" my_project_directory
cd my_project_directory
```

Lancement du serveur web

```
symfony serve –no-tls -d
```

Versionnement
La commande symfony a créé un premier commit

Structure MVC

Rappels des concepts et explications sur le routage des requêtes.

Premier controleur

Installation du maker bundle

Pour gagner du temps, on peut utiliser le maker bundle

```
composer require --dev symfony/maker-bundle
```

Pré-requis : les annotations

```
composer require doctrine/annotations
```

bonne pratique : mettre à jour git

Premier controleur

Commande de création :

```
php bin/console make:controller CoolStuffController
```

Pour voir les fichiers créés :

```
git status
```

Analyse du code produit :

1. Classe qui hérite de AbstractController

2. L'annotation de route :

```
#[Route('/cool/stuff', name: 'app_cool_stuff')]
```

NB : use Symfony\Component\Routing\Annotation\Route;

3. Méthode de classe qui retourne du json

Tester l'url

⇒ On constate que le format de retour est en json.

Routing dans une application Symfony

Définition des routes dans un fichier de configuration (YAML, XML, PHP)

Exemple :

```
# config/routes.yaml
blog_list:
  path: /blog
  controller: App\Controller\CoolStuffController::blog
```

```
# src/Controller/CoolStuffController.php
    public function blog(): JsonResponse
    {
        return $this->json("Page de blog");
    }
```

Attention, si plusieurs routes sont définies pour la même url, c'est la première qui sera prise en compte.

Paramètres

Syntaxe élémentaire :

Exemple : pagination des posts d'un blog

```
    #[Route('/blog/{page}')]
    public function paginate(int $page): JsonResponse
    {
        return $this->json('page ' . $page);
    }
```

NB : Symfony ignore les query_strings lors de la phase de routing

Plusieurs paramètres

```
#[Route('/blog/articles-de-la-categorie-{category}/page/{number}')]
```

Paramètres requis et typés

Exemple : pagination des posts d'un blog

```
    #[Route('/blog/{page}', requirements: ['page' => '\d+'])]
    public function paginate(int $page): JsonResponse
    {
        return $this->json('page ' . $page);
```

```
}
```

Exemple : affichage d'un article de blog

```
    #[Route('/blog/{slug}', requirements: ['slug' => '[a-z]+.*'])]
    public function show(string $slug): JsonResponse
    {
        return $this->json('slug ' . $slug);
}
```

La syntaxe est basée sur les expressions régulières.

Syntaxe alternative :

```
#[Route('/blog/{page<\d+>}')]
```

Valeur par défaut :
Dans le cas d'une pagination de blog, on souhaite que la route /blog/1 et la route /blog affiche la même chose.

*** Si on a défini la route par l'attribut « Route », la valeur par défaut peut être renseigné dans les paramètres du controleur :

```
    #[Route('/blog/{page}', requirements: ['page' => '\d+'])]
    public function paginate(int $page = 1): JsonResponse
    {
        return $this->json('page ' . $page);
    }
```

Ou bien dans la définition de la route
Syntaxe 1 :

```
#[Route('/blog/{page}', defaults: ['page' => 1])]
```

Syntaxe 2 :

```
#[Route('/blog/{page<\d+>?1}')]
```

Si on a définit la route dans un fichier yaml, xml ou php, la valeur par défaut est renseigné dans l'option « default »

Exercice

Etant donné cette définition de route :

```
#[Route('/blog/{slug}', requirements: ['slug' => '[a-z]+'], name: 'post')]
public function show(string $slug): JsonResponse
{
    return $this->json('slug ' . $slug);
}
```

La méthode show sera-t-elle appelée sur les requêtes suivantes ?

blog/article……oui

blog/article42 ……non

blog/article/edit ……non

blog/performace-web ……non

Méthodes HTTP

Par défaut, toutes les méthodes sont acceptées. On peut spécifier les méthodes :

```
    #[Route('/api/posts/{id}', methods: ['GET'])]
    public function show(int $id): Response
    {
        // ... retourne le post au format json
    }

    #[Route('/api/posts/{id}', methods: ['PUT'])]
    public function edit(int $id): Response
    {
        // ... edite le post
    }

    #[Route('/api/posts', methods: ['POST'])]
    public function new(int $id): Response
    {
        // ... crée un nouveau post
    }
```

Lister les routes d'une application

```
bin/console debug:router
```

NB : Seules les routes ayant un « name » sont affichées.
Pour plus de détail sur la route blog_list :

```
bin/console debug:router blog_list
```

Prefixes de routes

Il est possible préfixer toutes les routes au niveau de la classe

```
#[Route('/blog', name: 'blog_')]
class BlogController extends AbstractController
{
}
```

Exercice

Créer une classe ProductController, avec quelques routes préfixées par / product/

Pour aller plus loin

Tuto

https://openclassrooms.com/fr/courses/8264046-construisez-un-site-web-a-laide-du-framework-symfony-7