Support de cours Symfony 4

Jérôme AMBROISE

Support de cours Symfony 4 - Partie 1

Sommaire

Sommaire

1.

Présentation générale

Qu’est-ce qu’un Framework ?

Quels outils sont présents dans Symfony ?

“

Qu’est-ce qu’un Framework ?

Quelles sont les différences entre un Framework et une librairie ?

Présentation générale

Définition d'un Framework

Un framework :

- Est un ensemble non-spécialisé cohérent d'outils
- Propose un cadre de développement

Un framework peut donc intégrer plusieurs bibliothèques, outils.

Présentation générale
Définition d'un Framework

Le Framework a pour objectif d'accélérer le temps de développement des tâches répétitives.

- Structure des dossiers
- Routing
- Accès à la BDD
- Gestion des utilisateurs
- Mise en cache
- ...

Présentation générale

Définition d'un Framework

En fait au lieu de refaire toujours les codes répétitifs à la main, le framework le fait pour nous.

On peut donc s'intéresser aux besoins clients spécifiques à valeur ajoutée.

Présentation générale
Symfony

Symfony est un Framework Full-Stack MVC codé en PHP.

- 3000 contributeurs
- 600 000 développeurs Symfony
- 48 000 000 de téléchargements mensuels

Symfony propose une cinquantaine d'outils (Components).

Présentation générale
Symfony

Exemple de Components

- **Cache** : PSR-6, PSR-16
- **ClassLoader** : chargement automatique de classes
- **Console** : accès à des commandes en console exécutant du PHP
- **DependencyInjection**
- **Dotenv** : lecture des fichiers “.env”

Présentation générale
Symfony

Exemple de Components

- **EventDispatcher** : système d'événements
- **ExpressionLanguage** : évaluation d'expression
- **Form** : génération HTML et vérification PHP de formulaires
- **Guard** : garde pour l'authentification
- **HttpFoundation** : implémentation classe Request et Response (équivalent PSR-7)

Présentation générale
Symfony

Exemple de Components

- **Messenger** :
- **PHPUnit Bridge**
- **Process** : execution de commande dans des sous-programmes
- **Routing**
- **Security** : gestion des autorisations utilisateurs (utilisateurs, rôles, contrôles d'accès)

Présentation générale
Symfony

Exemple de Components

- **Serializer** : conversion Objet ⇔ JSON (XML, array, …)
- **Templating**
- **Translation** : internationalisation
- **Validator** : contraintes sur des entités
- **YAML** : lecture de fichiers YAML pour la configuration (traduction en PHP)

Présentation générale

Symfony

Symfony propose des components pour intégrer des librairies externes

- **Intégration de TWIG**
- **Intégration de Monolog**
- **Intégration de SwiftMailer**
- **Intégration de Doctrine**
- **Profiler**
- **...**

Présentation générale
Symfony

En plus de ses components, la communauté développe des “Bundles” (components tiers)

- **Paginator**
- **Upload de fichiers**
- **Back-office automatique**
- **WYSIWYG**
- **API automatique**

Présentation générale
Symfony

En résumé :

- Symfony propose des composants déjà codés
- Symfony propose la configuration automatique de librairies externes
- La communauté développe des composants externes que l'on peut utiliser

2.

Environnement de développement

De quoi a besoin Symfony pour fonctionner ?

Environnement de développement
Prérequis

Pour fonctionner, Symfony a besoin :

- D'un serveur web
  - Version PHP minimum : 7.1.3
- De composer
- Éventuellement d'un serveur de base de données

Environnement de développement
Encodage de la base de données

Symfony (le composant Doctrine en fait) préconise l'utilisation de l'encodage "utf8mb4_unicode_ci" pour la base de données.

Nous allons configurer MySQL afin de définir l'encodage par défaut.

Environnement de développement
Encodage de la base de données

Le fichier “my.ini” permet de configurer MySQL.

Tout en bas de ce fichier, nous allons ajouter l’encodage par défaut.

```
[mysqld]
port = 3306
character-set-server=utf8mb4
collation-server=utf8mb4_unicode_ci
```

3.

Premier projet Symfony

Comment récupérer l'architecture de base de Symfony ?

Que représente chaque dossier/fichier ?

Comment accéder aux pages web de Symfony ?

Comment installer un composant de Symfony ?

Comment optimiser le serveur web pour Symfony ?

Premier projet Symfony
Récupération de l'architecture

Composer va nous permettre de récupérer l'architecture de base d'un projet Symfony. Il existe 2 "squelettes".

- "symfony/skeleton"
  - Version minimale (équivalent à un MicroFramework)
- "symfony/website-skeleton"
  - Pré-installation des composants récurrents pour faire un site web (TWIG, FORM, Validation, Doctrine, PHPUnit, …).

Les 2 squelettes respectent la même architecture.

Premier projet Symfony
Récupération de l'architecture

Pour récupérer l'architecture nous allons utiliser la commande "create-project" de composer.

Il faut au préalable ouvrir un invite de commandes/ terminal dans le dossier dans lequel sera créé le dossier du projet Symfony.

composer create-project #skeleton# #nom du dossier#

- "skeleton" : nom du squelette
- "nom du dossier" : nom du dossier qui sera **créé** par composer (où l'architecture sera copiée)

Premier projet Symfony
Récupération de l'architecture

Exemple :

- "symfony/skeleton"

  composer create-project symfony/skeleton sf4-project

- "symfony/website-skeleton"

  composer create-project symfony/website-skeleton sf4-project

Remarque : n'oubliez pas d'ouvrir un invite de commande dans le dossier de travail voulu.

Premier projet Symfony

Architecture Symfony4

- ***/bin*** : executable de la console
- ***/config*** : dossier de configuration
  - ***bundles.php*** : liste des bundles (composants)
  - ***packages*** : fichiers de config des packages
- ***/public*** : dossier public de l'application
- ***/src*** : dossier des sources (namespace App)
- ***/var*** : cache, logs
- ***/vendor*** : les bundles (vendors) externes

“

Comment accéder aux pages web de Symfony ?

“

Comme d’habitude !

Premier projet Symfony
Lancement du site web

Rappel des possibilités d'accès à une application PHP.

- **Dossier "www"** (htdocs)
  - URL : localhost/
- Méthode avec les **virtualhosts**
  - URL : nom du virtualhost
- **Serveur web intégré** : lancement d'une commande dans le dossier du projet
    - php -S 127.0.0.1:8000 -t public
  - URL : localhost:8000/
  - (Symfony propose un composant pour cela)

Premier projet Symfony
Optimisation du serveur web

Symfony met à notre disposition un composant pour optimiser notre serveur web à l'utilisation de Symfony.

“

Comment utiliser un composant de Symfony ?

Premier projet Symfony

Installation d'un composant de Symfony

Un composant Symfony est une librairie (ou un ensemble de librairies) PHP.

- Ces librairies sont listées sur packagist
- Ces librairies sont disponibles au téléchargement via composer
- Une fois installées, les librairies doivent être configurées
- Les librairies sont alors utilisables comme n'importe quelle autre librairie PHP.
- Les librairies respectent l'architecture d'un bundle Symfony

“

Qu’est-ce que Symfony flex ?

Premier projet Symfony

Installation d'un composant de Symfony

Symfony Flex est nouvelle façon d'installer et de gérer les composants (Symfony ou tiers).

Quelles sont les nouveautés ?

- Les alias
- Les "recipes"

Premier projet Symfony

Installation d'un composant de Symfony

Les alias de Symfony Flex

De manière générale les alias permettent de nommer autrement quelque chose, généralement pour trouver un nom raccourci.

- SQL : alias des tables et des colonnes
- Les "use … as … " dans PHP : alias des classes PHP

Symfony Flex permet d'aliasser d'une librairie.

Premier projet Symfony

Installation d'un composant de Symfony

Les alias de Symfony Flex

Qu'est-ce que ça change ?

Nous pouvons utiliser les alias lorsque nous installons des librairies, lorsque nous faisons les "composer require…".

Où trouver les alias ?

https://flex.symfony.com/

Premier projet Symfony

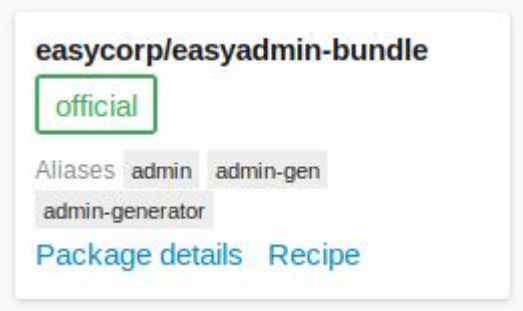
Installation d'un composant de Symfony

Les alias de Symfony Flex

Prenons un exemple : [easycorp/easyadmin-bundle]

Les alias disponibles pour ce package sont :

- admin
- admin-gen
- admin-generator



Premier projet Symfony

Installation d'un composant de Symfony

Les alias de Symfony Flex

Comment installer ce package avec composer ?

- composer require easycorp/easyadmin-bundle
- composer require admin
- composer require admin-gen
- composer require admin-generator

Ces 4 lignes sont équivalentes (c'est quand même plus simple à retenir…).

Remarque : pour que cela fonctionne, il faut que l'invite de commandes soit ouvert dans un projet Symfony.

Premier projet Symfony

Installation d'un composant de Symfony

Les recipes de Symfony Flex

Les recipes (recettes) servent à effectuer des actions automatiques lorsque l'on fait un "composer require …" (require/update/remove/install).

Ces actions sont listées dans un fichier appelé "manifest.json".

- Nous avons accès aux "manifestes" sur https://flex.symfony.com/ (il faut cliquer sur "recipe" pour accéder au recipe d'un package)

Premier projet Symfony

Installation d'un composant de Symfony

Les recipes de Symfony Flex

Voici le manifest de EasyAdminBundle : Manifest du recipe EasyAdminBundle.

Nous y retrouvons la définition des alias que nous avons vu précédemment.

En fait, chaque clef représente une action, les clefs sont donc des mots précis définis par Symfony Flex.

Chaque package peut éventuellement définir une valeur par clef.

Premier projet Symfony

Installation d'un composant de Symfony

Les recipes de Symfony Flex

Action liées aux clefs :

- **"bundles"** : enregistrement du bundle (du package)
  - Ajout d'une ligne dans "/config/bundles.php"
- **"copy-from-recipe"** : copier-coller de dossier/fichier dans notre projet (en dehors du dossier vendor)
- **"env"** : "ajout" de variables d'environnement
  - Ajout de lignes dans "/.env"
- **"aliases"** : définition des alias
  - Détection lors de commandes composer

Premier projet Symfony

Installation d'un composant de Symfony

Les recipes de Symfony Flex

Action liées aux clefs :

- **"composer-scripts"** : ajout d'une commande dans composer
  - Le script s'executera automatiquement lorsque nous ferons des commandes composer.

Premier projet Symfony
Optimisation du serveur web

Maintenant que nous savons comment fonctionne Symfony Flex, nous allons pouvoir installer des packages.

- ***“requirements-checker”*** : optimisation du serveur web

“

Que fait le recipe du package “requirements-checker”

Premier projet Symfony
Optimisation du serveur web

Utilisation du “requirements-checker”

Requirements-checker propose des optimisations pour le serveur web, ces optimisations sont différentes selon l’environnement sur lequel on est (serveur local, serveur hébergeur, pc au travail, pc au domicile, …)

- Pour accéder aux optimisations il faut se rendre sur la page “check.php”

Il faut alors résoudre les recommandations ;)

Premier projet Symfony
Optimisation du serveur web

Utilisation du “requirements-checker”

Une fois que toutes les recommandations sont effectuées, nous pouvons désinstaller le package.

Il faudra néanmoins le réinstaller lorsque nous mettrons notre site en production afin de vérifier que le serveur de l’hébergeur respectent les recommandations.

Premier projet Symfony
Sécurité des librairies

- ***“security-checker”*** : vérification des failles de sécurités connues des packages

“

Que fait le recipe du package “security-checker”

Premier projet Symfony
Sécurité des librairies

Utilisation du “security-checker”

Security-checker executera sa commande de vérification lorsque nous ferons des commandes composer.

Premier projet Symfony
Sécurité des librairies

Résumé :

- On a récupéré l'architecture de base de Symfony
- On sait utiliser Symfony Flex pour installer de nouveaux packages
- Nous avons effectué les recommandations de Symfony pour optimiser notre serveur web
- Nous avons mis en place une vérification de sécurité qui se déclenche automatiquement lorsque nous installons de nouveaux packages.

Nous pouvons désormais afficher des pages.

Premier projet Symfony
Git et PHPcs

N’oublions pas de mettre en place Git et PHPcs avant de passer à la suite =)

4.

Création d'une page

Comment créer une page avec Symfony 4 ?

Comment intégrer TWIG ?

Comment intégrer une barre de debug ?

Création d'une page
Méthodologie

Pour créer un page nous allons suivre 3 étapes :

- Création d'un contrôleur
- Création d'une route
- Création d'une vue

Pour notre première page nous n'allons pas utiliser de vue, nous l'ajouterons juste après avec TWIG.

Création d'une page
Méthodologie

Avant de commencer, nous allons configurer notre IDE pour l'utilisation de Symfony. Cela se fait via l'utilisation de plugin pour gagner du temps ;)

Peu importe l'IDE/éditeur, un plugin officiel doit exister.

Création d'une page

Méthodologie

Si par hasard vous avez PHPStorm, voici comment installer et paramétrer le plugin Symfony.

- Installation
  - File => Settings => Plugins
  - Télécharger "Symfony Plugin"
- Paramétrage
  - File => Settings => Langages & Frameworks => PHP => Symfony
  - Cocher "Enable Plugin for this Project"
  - Translation Root Path : "translations"
  - App Directory : "src"
  - Web Directory : "public"

Création d'une page
Création d'un contrôleur

Les contrôleurs se trouvent dans le dossier "/src/Controller".

Ce sont des classes PHP classiques.

- Création du contrôleur Home :
  - Création fichier :
    - /src/Controller/HomeController.php
- Création d'une action home :
  - Création méthode :
    - home()

Création d'un contrôleur

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

class HomeController
{
    public function home()
    {

    }
}
```

Création d'une page
Création d'un contrôleur

Comment retourner une réponse ?

- Symfony nous propose une implémentation d'une Request et d'une Response dans son composant "HttpFoundation"

Ces classes n'implémente ntpas le PSR7 mais se comportent de la même manière.

Il existe un package afin d'adapter ces classes pour respecter le PSR7 (documentation).

Création d'une page
Création d'un contrôleur

Création d'une Response de Symfony

Namespace de la classe

- Symfony\Component\HttpFoundation\Response

Nous allons instancier une réponse et la renvoyer.

Création d'un contrôleur

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;

class HomeController
{
   public function home(): Response
   {
      return new Response('<h1>Réponse Symfony</h1>');
   }
}
```

Création d'une page
Création d'une route

Symfony embarque un routeur. Nous pouvons le configurer de plusieurs façons (nous le verrons en détail dans le prochain chapitre).

Pour l'instant nous allons le paramétrer en YAML.

“

Qu’est-ce que le YAML?

Création d'une page
Le YAML

Le YAML est un format de représentation de données comme le JSON, le XML, le CSV.

Les fichiers YAML ont l'extension ".yml" ou ".yaml"

- Il propose une syntaxe sans accolades qui permet d'être le plus lisible possible par les humains
- Le YAML est souvent utilisé pour le paramétrage

Symfony utilise le YAML, le composant YAML de Symfony permet de convertir le YAML en tableau PHP.

Création d'une page
Le YAML

Le YAML en comparaison des autres formats de représentation de données

- Plus lisible que du JSON
- Moins verbeux que le XML
- Moins compacte que le CSV

Création d'une page
Le YAML

Pour représenter les données, le YAML se base sur 2 principes :

- L'indentation, par des espaces, manifeste une arborescence.
- Les tableaux sont de la forme clé: valeur, à raison d'un couple par ligne.

Création d'une page

Le YAML

Version PHP

```php
$project = [
    "name" => "Projet Symfony",
    "languages" => [
        "php" => [
            "Acronyme" => "Hypertext Preprocessor",
            "Back" => true
        ],
        "html" => [
            "Acronyme" => "HyperText Markup Language",
            "Back" => false
        ]
    ]
];
```

Version YAML

```yaml
project:
    name: Projet Symfony
    languages:
        php:
            Acronyme: HyperText Preprocessor
            Back: true
        html:
            Acronyme: HyperText Markup Language
            Back: false
```

Création d'une page
Le YAML

Le paramétrage des routes peut se faire en YAML dans fichier /config/routes.yaml.

Pour définir une route en YAML, nous devons fournir minimum 3 informations :

- Le nom de la route
- Le path
- L'action de contrôleur

Création d'une page
Le YAML

- Le nom de la route
  - Utilité : créer des liens dynamiques dans l'HTML
- Le path
  - Utilité : ce que l'on doit renseigner dans l'URL du navigateur
- L'action de contrôleur
  - Utilité : fonction à appeler lors le path est détecté
  - Syntaxe :
    - Namespace complet du contrôleur
    - ::
    - Le nom de la méthode du contrôleur

Création d'une route en YAML

Emplacement : "/config/routes.yaml"

```yaml
homepage:
  path: /
  controller: App\Controller\HomeController::home
```

Création d'une page
Le YAML

On peut désormais accéder à la page d'accueil du site.

Création d'une page
Intégration de TWIG

Jusque là nous définissons l'HTML dans le contrôleur, pour externaliser le HTML dans des vues séparées, il convient d'utiliser un moteur de templating.

Symfony préconise un moteur de templating : TWIG.

“

Nous voulons intégrer TWIG dans Symfony, où peut-on se renseigner ?

Création d'une page
Intégration de TWIG

Pour installer un package dans Symfony, on parlera de bundle, nous pouvons nous renseigner sur le site de Symfony Flex.

Le bundle "symfony/twig-bundle" permet l'intégration de TWIG dans Symfony.

Nous pouvons l'installer grâce à l'un de ses alias.

Exemple :

composer require twig

Création d'une page
Intégration de TWIG

Comment utiliser TWIG ?

- Définition des vues dans le dossier "templates"
  - Extension des vues : ".html.twig"
- Utilisation de la méthode "render()" de TWIG
  - La méthode "render()" est définie dans la classe \Twig\Environment

“

Comment utiliser la méthode “render()” de \Twig\Environment dans notre contrôleur ?

Création d'une page
Intégration de TWIG

Comment utiliser \Twig\Environment?

- Injection de dépendances (constructeur)
- Injection de dépendances (directement depuis la méthode)
- Utilisation de la classe mère de Controller de Symfony

Création d'une page
Intégration de TWIG

Comment utiliser \Twig\Environment?

- Injection de dépendances (constructeur)
  - Création d'un constructeur
  - On demande en paramètre \Twig\Environment
  - On stocke le paramètre en tant que propriété privée de notre contrôleur
  - On utilise la méthode "render()" de TWIG dans notre action (notre méthode "home()")

Injection de \Twig\Environment (1/4)

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;

class HomeController
{
    public function __construct()
    {
    }

    public function home(): Response
    {
        return new Response('<h1>Réponse Symfony</h1>');
    }
}
```

Injection de \Twig\Environment (2/4)

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Twig\Environment;

class HomeController
{
  public function __construct(Environment $twig)
  {
  }

  public function home(): Response
  {
     return new Response('<h1>Réponse Symfony</h1>');
  }
}
```

Injection de \Twig\Environment (3/4)

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Twig\Environment;

class HomeController
{
    /**
     * @var Environment
     */
    private $twig;

    public function __construct(Environment $twig)
    {
        $this->twig = $twig;
    }

    public function home(): Response
    {
        return new Response('<h1>Réponse Symfony</h1>');
    }
}
```

Injection de \Twig\Environment (4/4)

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Twig\Environment;

class HomeController
{
  /**  @var Environment   */
  private $twig;
  public function __construct(Environment $twig)
  {
     $this->twig = $twig;
  }
  /**
   * @return Response
   * @throws \Twig\Error\LoaderError
   * @throws \Twig\Error\RuntimeError
   * @throws \Twig\Error\SyntaxError
   */
  public function home(): Response
  {
     return new Response(
        $this->twig->render('home.html.twig')
     );
  }
}
```

Création d'une page
Intégration de TWIG

Comment utiliser \Twig\Environment?

- Injection de dépendances (directement depuis la méthode)

Symfony met en place un système d'injection de dépendances directement dans une méthode.

Qu'est-ce que ça signifie ?

Jusque là on demandait des classes seulement dans le constructeur, on peut désormais demander des classes dans une méthode de contrôleur.

Injection de \Twig\Environment

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Twig\Environment;

class HomeController
{
    /**
     * @return Response
     * @throws \Twig\Error\LoaderError
     * @throws \Twig\Error\RuntimeError
     * @throws \Twig\Error\SyntaxError
     */
    public function home(Environment $twig): Response
    {
        return new Response(
            $twig->render('home.html.twig')
        );
    }
}
```

Création d'une page
Intégration de TWIG

Comment utiliser \Twig\Environment?

- Utilisation de la classe mère de Controller de Symfony

Symfony met à notre disposition une classe mère de Controller qui contient des méthodes pour nous faire gagner du temps.

- Symfony\Bundle\FrameworkBundle\Controller\AbstractController

Nous allons donc étendre de cette classe et utiliser sa méthode "render()" dans notre contrôleur.

Injection de \Twig\Environment

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;

class HomeController extends AbstractController
{
    /**
     * @return Response
     */
    public function home(): Response
    {
        return $this->render('home.html.twig');
    }
}
```

Création d'une page
Intégration de TWIG

Nous pouvons désormais mettre en place un système de layout avec "base.html.twig".

- Définition des blocks
- Utilisation de CSS/JS

Création d'une page
Intégration de TWIG

Afin de pouvoir debugger notre application, Symfony met à notre disposition un "profiler".

C'est une barre de debug qui apparait sur chaque page HTML afin de nous donner diverses informations.

Pour l'installer, nous allons utiliser son alias "profiler" (trouvé sur flex.symfony.com)

composer require profiler

Création d'une page
Intégration de TWIG

Désormais, à chaque fois qu'une page HTML se charge, une barre de debug apparaît en bas de la page.

Elle contient diverses informations :

- Routing
- Performances

Lorsque nous cliquons sur un des blocks, nous avons accès à davantage d'informations (variables serveurs, …)

Création d'une page
Intégration de TWIG

Un bundle à la possibilité d'ajouter son block dans la barre de debug.

Par la suite nous aurons d'autres types d'informations :

- Requête SQL
- Utilisateurs connectés
- Internationalisation
- ...

4.

Le routage

Quelles sont les différentes façons de router ?

Comment router dans Symfony?

Que sont les annotations ?

Comment lister les routes dans la console ?

“

Qu’est-ce que le routage?

Le routage
Présentation

Le routage permet de faire correspondre une URL avec un traitement.

L'URL est ce qui apparaît en haut du navigateur au niveau de l'adresse du site (http://www.monsite.fr)

Nous nous occuperons dans Symfony de "router" la nom de domaine.

Exemples :

- http://www.monsite.fr**/contact**
- http://www.monsite.fr**/articles**

Le routage
Présentation

L'URL par défaut est "/", ce qui correspond à :

- http://www.monsite.fr

Désormais lorsque que nous parlerons d'URL, on s'intéressa seulement à la partie après le nom d'hôte.

Exemple :

L'url "/contact" correspond au chemin complet :

- http://www.monsite.fr/contact

Le routage

Les différentes formats de routage

Le routage est géré par un composant de Symfony : Routing Component.

L'ajout des routes dans ce routage peut se faire de 4 façons différentes :

- **YAML** : dans le fichier /config/routes.yaml
- **Annotations** : directement dans le contrôleur
- **XML**
- **PHP**

Le routage

Les différentes formats de routage

Quel format de routage choisir ?

- YAML :
  - Avantages :
    - Lisibilité de lecture
    - Séparation routes/contrôleur
  - Inconvénients :
    - Syntaxe avec indentation (il faut faire attention)

Le routage

Les différentes formats de routage

Quel format de routage choisir ?

- Annotations :
  - Avantage :
    - Simplicité d'utilisation
  - Inconvénients :
    - Testabilité
    - La route et l'action de contrôleur sont définies dans le même fichier

Le routage

Les différentes formats de routage

Quel format de routage choisir ?

- XML et PHP :
  - Avantage :
    - Pas de nouvelles choses à apprendre
  - Inconvénients :
    - Lisibilité de lecture

Le routage

Les différentes formats de routage

Comment je fais pour me décider ?

Il faut tester les différents formats et voir celui qui vous convient le mieux, il n'y en a pas vraiment de "meilleur".

- L'annotation est très utilisée pour sa simplicité
- Le YAML est très utilisé pour sa lisibilité et sa séparation du contrôleur
- Le XML/PHP sera plutôt utilisé si on ne veut pas utiliser les autres et pour éviter des "surcouches".

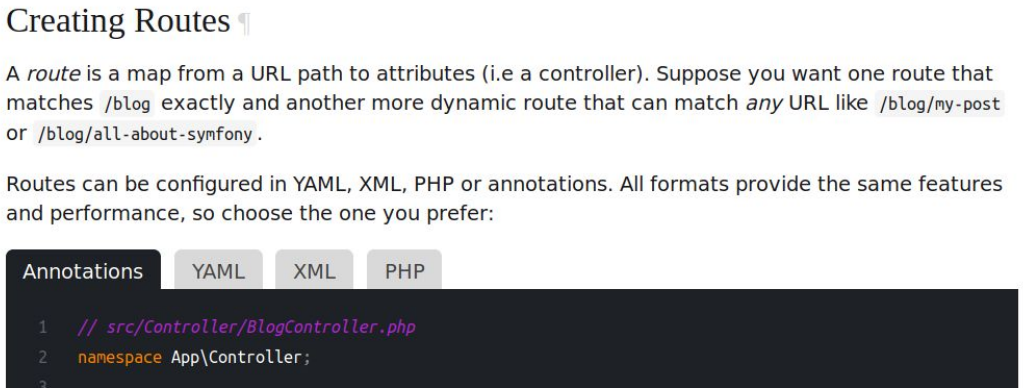
Le routage

Les différentes formats de routage

Il faut retenir que les différents formats de routage font la même chose.

Le documentation détaille chaque format.

- Tutoriel officiel sur le routage



Creating Routes ¶

A *route* is a map from a URL path to attributes (i.e a controller). Suppose you want one route that matches `/blog` exactly and another more dynamic route that can match *any* URL like `/blog/my-post` or `/blog/all-about-symfony`.

Routes can be configured in YAML, XML, PHP or annotations. All formats provide the same features and performance, so choose the one you prefer:

Annotations | YAML | XML | PHP

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

Le routage

Les différentes formats de routage

Pendant cette semaine, nous allons nous intéresser principalement à créer des routes des routes en YAML mais surtout en annotations.

Nous avons vu comment router une route en YAML.

Nous allons désormais voir comment router une route en annotation.

“

Qu’est-ce qu’une annotation?

Le routage
Les annotations

Les annotations permettent de mettre du code PHP dans les commentaires spéciaux de la PHPDoc.

Ces instructions seront exécutées même si elles sont dans des commentaires.

Exemple fausse annotation

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;

class HomeController extends AbstractController
{
    /**
     * @AnnotationPerso()
     * @return Response
     */
    public function home(): Response
    {
        return $this->render('home.html.twig');
    }
}
```

Le routage
Les annotations

- Les annotations sont lues par certains composants et permettent d'effectuer des traitements PHP.

Elles permettent d'avoir une abstraction : nous n'avons pas besoin de savoir comment est codée l'annotation, tant qu'on connait son nom et ses paramètres, on peut l'utiliser.

- L'annotation est une fonction ou une classe appelée par le composant capable de lire l'annotation.

Le routage
Les annotations

- Les annotations se situent dans les **commentaires spéciaux** (/** … */) comme la PHPDoc
- **Les annotations ne sont pas de la PHPDoc**, ils provoquent l'exécution de fonctions PHP
- Les annotations sont préfixées d'un arobase "@"
- Les annotations sont **situés directement au dessus** de l'élément qu'elle décore (une classe, une méthode, une fonction, un propriété, …)

Le routage
Les annotations

Nous pouvons router avec des annotations

- Les annotations ne se limitent pas au routage, nous verrons que l'on va utiliser les annotations aussi avec Doctrine
- D'autres composants peuvent aussi utiliser les annotations

Le routage
Les annotations

Dans le cas du routage, nous allons décorer les méthodes avec l'annotation "@Route".

- Cette "Route" est en fait une classe, nous devons donc l'importer :
  - use Symfony\Component\Routing\Annotation\Route;

Le routage
Les annotations

Pour router en annotation, nous allons créer une annotation au dessus de la méthode qui devra être exécutée.

Nous devrons simplement fournir l'URL de détection.

Remarque : on utilise obligatoirement les quotes doubles (les guillemets) dans les annotations.

Les quotes simples ne sont pas reconnues, attention ;)

Routage avec annotations

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

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

class HomeController extends AbstractController
{
    /**
     * @Route("/")
     * @return Response
     */
    public function home(): Response
    {
        return $this->render('home.html.twig');
    }
}
```

Le routage
Les annotations

Nous pouvons désormais router en YAML ou en annotations.

- Il est possible d'utiliser les deux formats dans le même projet

“

Quel est le nom d’une route créée en annotation ?

Le routage
Les annotations

Le nom de la route créée est automatiquement attribué par rapport au contexte

- Namespace (App => app)
- Nom du contrôleur (HomeController => home)
- Nom de la méthode (home() => home

Les différents éléments sont séparés d'un underscore.

Exemple pour notre cas :

app_home_home

Le routage

Lister les différentes routes

Comment être sûr du nom d'une route ?

- Avec la console !

Symfony possède un composant qui permet de lancer des fonctions PHP dans la console.

- Chaque composant que nous installons peut définir des commandes en console

C'est le cas du composant de routage.

Le routage
Lister les différentes routes

Voici la commande pour lister les différentes routes de notre application en console :

php bin/console debug:router

On y trouve les différentes informations des routes de notre applications (nom, méthode HTTP autorisée, URL, ...).

Le routage
Lister les différentes routes

Les annotations de routes permettent aussi de renommer la route si nous le souhaitons.

- Pour cela on peut préciser un paramètre “name” dans l’annotation Route en précisant le nouveau nom de la route

Changement du nom d'une route

Emplacement : "/src/Controller/HomeController.php"

```php
<?php
namespace App\Controller;

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

class HomeController extends AbstractController
{
    /**
     * @Route("/", name="app_homepage")
     * @return Response
     */
    public function home(): Response
    {
        return $this->render('home.html.twig');
    }
}
```

Le routage
.htaccess

Lorsque nous utilisons une route différente de la racine, nous sommes obligés d'indiquer "index.php" dans le navigateur.

Pour éviter cela, nous pouvons récupérer un ".htaccess" qui nous évitera de saisir "index.php" dans l'URL.

- Ce package possède un alias "apache-pack"

Le routage
Les variables de routes

L'utilisation des variables dans l'URL s'effectue grâce à **l'ajout d'accolades** dans la définition de la route (comme avec Slim).

Dans les exemples j'utiliserai les annotations pour router mais la logique est similaire avec les autres formats ; la syntaxe pourra néanmoins changer.

Prenons un exemple :

Nous voudrions accéder aux détails d'un projet avec son slug.

Routage avec variable

Emplacement : "/src/Controller/ProjectController.php"

```php
<?php
namespace App\Controller;

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

class ProjectController extends AbstractController
{
    /**
     * @Route("/projet/{slug}")
     * @return Response
     */
    public function show(): Response
    {
        return $this->render('project/show.html.twig');
    }
}
```

Le routage
Les variables de routes

Comment récupérer la **variable de l'URL** dans notre méthode de contrôleur ?

- Il suffit d'ajouter un **paramètre à notre action de contrôleur** portant le nom de la variable de l'URL, elle sera remplie automatiquement avec la valeur de la variable saisie dans l'URL du navigateur.

Routage avec variable

Emplacement : "/src/Controller/ProjectController.php"

```php
<?php
namespace App\Controller;

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

class ProjectController extends AbstractController
{
    /**
     * @Route("/projet/{slug}", name="detail_projet")
     * @param string $slug
     * @return Response
     */
    public function show(string $slug): Response
    {
        var_dump($slug);
        die('Récupération du slug de l\'URL');

        return $this->render('project/show.html.twig');
    }
}
```

Le routage
Les variables de routes

Il est possible de rendre la variable de l'URL facultatif.

- Pour cela il faut rendre le paramètre de la méthode correspondant à la variable de l'URL facultatif.

Routage avec variable

Emplacement : "/src/Controller/ProjectController.php"

```php
<?php
namespace App\Controller;

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

class ProjectController extends AbstractController
{
    /**
     * @Route("/projet/{slug}", name="detail_projet")
     * @param string $slug
     * @return Response
     */
    public function show(string $slug = null): Response
    {
        var_dump($slug);
        die('Récupération du slug de l\'URL');

        return $this->render('project/show.html.twig');
    }
}
```

Le routage
Les variables de routes

On peut ajouter des contraintes sur les variables d'URL grâce aux REGEX.

Les contraintes peuvent avoir 2 formes.

- Ajout d'un paramètre "requirements" sur l'annotation "@Route"
- Précision de la REGEX directement dans la définition de l'URL

Routage avec variable

Emplacement : "/src/Controller/ProjectController.php"

```php
<?php
namespace App\Controller;

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

class ProjectController extends AbstractController
{
    /**
     * @Route("/projet/{id}", requirements={"id"="\d+"})
     * @param int $id
     * @return Response
     */
    public function show(int $id = null): Response
    {
        var_dump($id);
        die('Récupération de l\'id de l\'URL');

        return $this->render('project/show.html.twig');
    }
}
```

Routage avec variable

Emplacement : "/src/Controller/ProjectController.php"

```php
<?php
namespace App\Controller;

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

class ProjectController extends AbstractController
{
    /**
     * @Route("/projet/{id<\d+>}")
     * @param int $id
     * @return Response
     */
    public function show(int $id = null): Response
    {
        var_dump($id);
        die('Récupération de l\'id de l\'URL');

        return $this->render('project/show.html.twig');
    }
}
```

Le routage
Les variables de routes

On peut limiter les méthodes permettant d'accéder à l'URL grâce à l'ajout du paramètre "methods" dans l'annotation "@Route". On fournit alors un tableau listant les méthodes disponibles.

Routage avec variable

Emplacement : "/src/Controller/ProjectController.php"

```php
<?php
namespace App\Controller;

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

class ProjectController extends AbstractController
{
    /**
     * @Route("/projet/{id<\d+>}", methods={"GET"})
     * @param int $id
     * @return Response
     */
    public function show(int $id = null): Response
    {
        var_dump($id);
        die('Récupération de l\'id de l\'URL');

        return $this->render('project/show.html.twig');
    }
}
```

Merci de votre attention !

Pour la suite : Contrôleur, TWIG, Doctrine, Bundles tiers