Introduction à Symfony 5

Introduction

Ce support de cours introduit les concepts de base du framework Symfony dans sa version 5 (Sf5) dans un environnement Linux.

Le framework Symfony est un framework PHP développé par l'éditeur de logiciels Sensiolabs.

Un *framework* ou cadriciel est un cadre de développement modulaire qui permet de réaliser plus rapidement et de manière plus efficace une application. Il est basé sur une ou plusieurs bibliothèques de classes permettant une implémentation facile et rapide de fonctionnalités. Il fournit un guide d'architecture en partitionnant le domaine visé en modules avec définition des responsabilités de chacun des modules et classes. Tous ses composants sont organisés pour être utilisés en interaction les uns avec les autres.

En général un *framework* est spécialisé vis à vis d'une couche applicative particulière. Il peut s'agir de la couche de persistance, de présentation, d'accès aux données, MVC, … Mais certains offrent un périmètre plus large qui couvre l'ensemble des problématiques liées à une architecture applicative, notamment les applications Web avec IHM et les services Web.

Programmer au sein d'un *framework* impose de respecter les règles d'un environnement contraint par les hiérarchies et dépendances des services proposés et par sa «vision». De nombreux *frameworks* existent, un tableau non exhaustif de différents frameworks est présent sur le site de Wikipédia, voir http://fr.wikipedia.org/wiki/Framework.

Symfony est un framework PHP qui est spécialisé dans le développement Web, c'est un projet communautaire. La version 1 du produit a connu des développements jusqu'à la fin 2012, elle a été développée en PHP 5.1 et 5.2.
Symfony2 a vu le jour depuis juin 2012. C'est une réécriture complète de la version 1 en PHP5.3 et intègre de nouvelles fonctionnalités comme l'espace de nommage.
Le développement de Sf3 a commencé depuis novembre 2015 mais la première version LTS (Long Term Support) a vu le jour en mai 2017.
Aujourd'hui, on en est à la version 6, mais la version stable est la 5.4. Nous allons donc travailler avec cette version LTS.

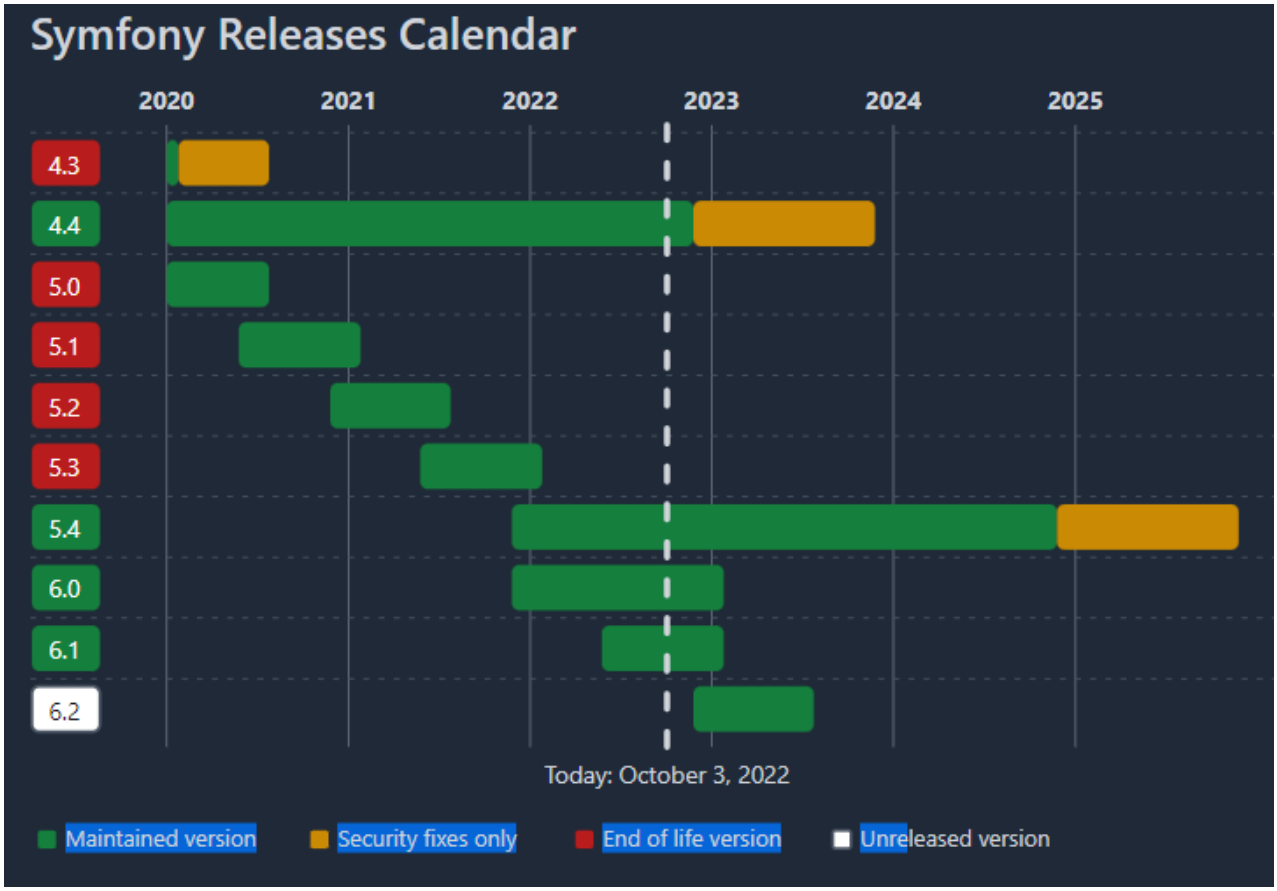


Figure 1. Source https://symfony.com/releases.

1. Hiérarchie d'une application Symfony

La hiérarchie de Symfony a évolué au fil des versions :

- Le dossier **bin** : dans la version 2, il y était uniquement pour des raisons de compatibilité avec les anciennes versions de Symfony. Dans la version Sf3, il contenait les exécutables et les dépendances du projet. Il contient notamment la console de Symfony (avant, elle se trouvait dans le répertoire app). Depuis, il contient uniquement le script PHP exécutable **`console`**.
- Le dossier **config** (dans les versions 2, il s'appelait **app**) contient les fichiers de configuration de l'application. C'est ici que seront configurés les routes et les services.
- Le dossier **src** contient les fichiers PHP de l'application à développer (contrôleurs). Il contient, au départ, la classe Kernel et le répertoire Controller.
- Le dossier **vendor** contient les composants et bibliothèques de Symfony (doctrine, sensio, symfony…). Il contient également les dépendances créées avec Composer. Vous ne devez en aucun cas versionner ce répertoire (Composer s'occupe de le générer à chaque installation/ déploiement de votre application). **A ne pas modifier**.
- Le dossier **public** (anciennement **web**) est destiné à contenir toutes les ressources publiques telles que les fichiers images, les feuilles de style CSS et les scripts JavaScript. Ce répertoire est le seul qui doit être accessible aux internautes. Il contient également le contrôleur frontal de symfony : **`index.php`**. Toutes les requêtes passeront par ce fichier.
- **templates** : nouveauté des versions 4 et suivantes. C'est ici qu'on mettra les vues de l'application. Avant, elles étaient dans src/.
- **tests** : tests des fonctionnalités. Etaient dans src dans les versions précédentes.
- **var** : contient le répertoire cache et log.
- **.env** : définit les variables d'environnement de l'application (dev/prod, infos de connexion à la BDD...).
- Le fichier **composer.json** : contient tous les paquets installés dans le projet qui se trouvent dans les sections **require** et **require-dev**.
- Le fichier **composer.lock** contient également les pré-requis et un ensemble de propriétés.
- Le fichier **symfony.lock** : nouveau depuis la version 4. En effet, depuis la version 4, on a Symfony Flex qui est un moyen d'installer et de gérer des applications Symfony. C'est un plugin composer qui modifie le comportement des commandes require, update et remove. A chacune de ces commandes, l'application adresse la requête au serveur de Symfony Flex avant d'essayer d'installer le package avec composer. S'il n'y a pas d'informations dans Flex, le paquet est installé par composer. Si Flex a des informations sur le paquet, il les envoie sous forme de fichier (recette ou recipe) que l'application va exploiter pour décider quel package utiliser et quelles commandes lancer automatiquement après l'installation. Le fichier symfony.lock journalise la trace des recettes installées par Flex.

2. Composants de Symfony

Symfony, de manière générale, est un ensemble de composants autonomes et de composants tierces qui peuvent être utilisés dans un projet PHP. Tous les composants sont dans le répertoire **Symfony/vendor**.
Les principaux composants Symfony 3 sont :

- **HttpFoundation** : englobe les classes de :
  - requêtes HTTP (Symfony\Component\HttpFoundation\Request)
  - réponses HTTP (Symfony\Component\HttpFoundation\Response),
  - gestion de sessions
  - chargement de fichiers.

Les variables $_GET et $_POST sont accessibles respectivement via les propriétés publiques query et request. Chacun de ces objets est un objet (Symfony\Component\HttpFoundation\ParameterBag) qui a des méthodes comme get(), has(), all(). Obtenir une variable SERVER revient à exécuter l'instruction : $request->server->get('http_HOST'), par exemple.

- **Routing** : système de routage qui effectue la correspondance entre la requête demandée et le contrôleur qui va l'exécuter.
- **Form** : ensemble de composants permettant la gestion de formulaires.
- **Validator** : système de création de règles pour contrôler les données et les valider lors d'envoi de ces données.
- **ClassLoader** ; librairie d'auto-chargement des classes PHP.
- **Templating** : moteur de générations de vues.
- **Security** : librairie de composants pour prendre en charge tout type de sécurité au sein d'une application.

- **Translation** : framework qui prend en charge la translation de chaînes de caractères dans une application.

3. Fonctionnement d'une application Symfony

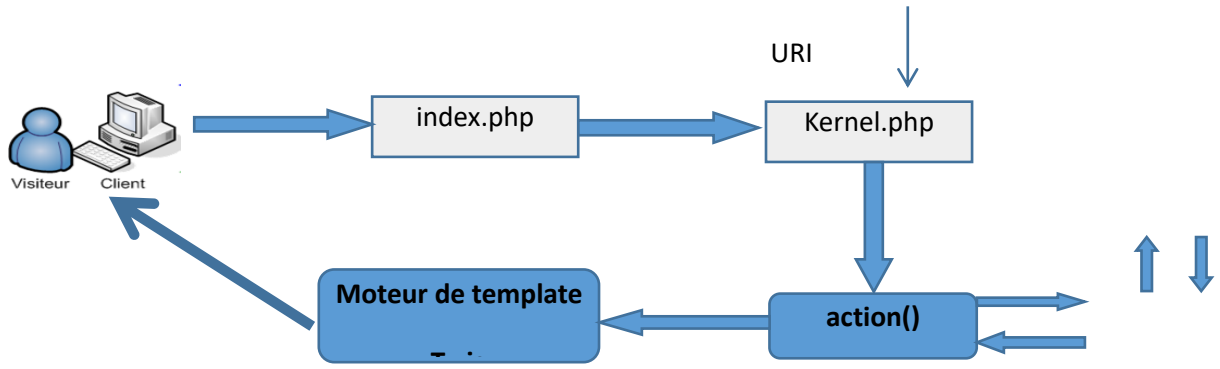
Dans une application Web traditionnelle, chaque page d'un site correspond à un fichier php. Sous Symfony, une application web est construite en respectant une architecture MVC (Modèle – Vue – Contrôleur).
Un contrôleur est une classe PHP dont le nom doit se terminer par Controller et qui doit se trouver dans le répertoire src/Controller.

Une vue peut être un fichier HTML, PHP ou twig. Symfony travaille avec des vues Twig. C'est un langage qui combine le HTML et les structures du PHP selon une syntaxe différente.

Le modèle peut être une classe PHP. Mais Symfony travaille avec l'ORM Doctrine pour la persistance des données en base.

L'appel à une vue se fait dans les méthodes des contrôleurs. La méthode à appeler est précisée dans le fichier routes.yml (routing.yml dans les anciennes versions).

Parcours d'une requête HTTP sous Symfony 5



index.php : création d'une instance du kernel
Kernel.php : chargé de gérer la requête et de retourner une réponse. Il va analyser l'URI et lancer la méthode concernée dans le contrôleur référencé => système de routage : correspondance entre les URI et les méthodes appelées.

4. Installation d'un projet Symfony 5

Symfony respecte le «versionning sémantique » (depuis la version 2.3.0) qui permet de mettre à jour Symfony en toute sécurité. Chaque numéro de version comporte trois valeurs : MAJOR, MINOR et PATCH.

Une version MAJOR est créée lorsque Symfony ne peut assurer une rétrocompatibilité avec des projets de versions précédentes.

La version MINOR est mise à jour lorsque de nouvelles fonctionnalités ont été ajoutées à la version MAJOR.

La version PATCH est mise à jour pour les corrections de bugs.

4.1. Installation avec composer :

La procédure d'installation suivante est réalisée dans un environnement :

- Linux, plus précisément la distribution Debian 9 ou 10
- doté du serveur Apache2
- avec une BDD mysql,
- PHP 7. La configuration PHP : date.timezone, realpath_cache_size à 5M

Composer est l'outil recommandé par SensioLabs pour installer Symfony. L'argument *nomProjet* est à remplacer par la valeur du chemin où doit être installée l'application, à savoir `/var/www/html/nomProjet` et *version* est à remplacer par le numéro de version souhaitée. Si la version n'est pas indiquée, il installe la dernière version stable.

Dans notre cas, on veut installer la version LTS avec les derniers correctifs.

```
$ cd /var/www/html
$ composer create-project symfony/skeleton:"5.4.*" projet1
$ cd projet1
$ composer require webapp
Ou
$ symfony new demo1 --version=lts
```

Un répertoire du nom demo1 sera créé avec des sous-répertoires. Le répertoire vendor ne doit pas être supprimé. Le répertoire src sera le répertoire où sera hébergé le projet php.

Un projet Symfony respecte l'architecture MVC. Le squelette Symfony devient de plus en plus léger et on le personnalise selon les besoins. En effet, Symfony nous permet d'installer des paquets composer, des bundles Symfony et des recettes Flex (recipes ou recettes). Flex est un plugin composer et les recipes contiennent des instructions qui indiquent à Flex ce qu'il doit faire pour chaque paquet. Elles permettent ainsi d'automatiser la configuration des bundles et des plugins qu'on rajoute à notre squelette. https://afsy.fr/avent/2017/08-symfony-flex-la-nouvelle-facon-de-developper-avec-symfony

On va installer 4 recettes essentielles :

- requirements-checker : permet de vérifier la présence des prérequis nécessaires à Symfony
- annotations : commentaires PHP qui seront interprétés par PHP
- make : outil de génération de code.

```
$ cd /var/www/html/demo1
$ composer req annotations requirements-checker make
```

4.2. Création d'une page Web statique avec Symfony 5

Toute page créée avec Symfony nécessite :

- une route
- un contrôleur : contient les méthodes appelées par le système de routage.

Dans un premier temps, nous allons créer un contrôleur qui renvoie directement une réponse au format Json sans passer par un template (vue).

=> Création du contrôleur

Le fichier contrôleur doit être créé dans le répertoire Controller qui se trouve dans le dossier src. Il s'agit d'une classe dont le nom se termine par Controller et qui doit hériter de la classe Symfony AbstractController.
On peut le créer à la main. Depuis la version 5, on peut le créer en ligne de commande et générer automatiquement la vue également. Pour cela, il faut, au préalable lancer les commandes suivantes :

```
ahe@debian:/var/www/html/demo1$ composer require symfony/maker-bundle --dev
$ php bin/console make:controller DefaultController
```

La commande va créer une classe contrôleur DefaultController dans le répertoire src/Controller :

```
<?php

namespace App\Controller;
```

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

class DefaultController extends AbstractController
{
    /**
     * @Route("/default", name="default")
     */
    public function index() : Response
    {
        return $this->json([
            'message' => 'Welcome to your new Controller !',
            'path' => 'src/Controller/DefaultController.php',
        ]);
    }
}
```

Pour tester ce programme, on va activer le serveur interne de Symfony. Lancer le serveur :

```
ahe@debian:/var/www/html/demo1$ php bin/console server:start


 [OK] Server listening on http://127.0.0.1:8000
```

Si le serveur ne se lance pas, il est possible que le bundle WebServer ne soit pas installé. Lancer la commande suivante :

```
ahe@debian:/var/www/html/demo1$ composer require symfony/web-server-bundle
```

Lancer le navigateur et saisir l'URL suivante : **http://localhost:8000/default**

=> Création du contrôleur et de la vue correspondante

Pour créer un contrôleur qui envoie une vue comme réponse, il faut activer le moteur de template Twig :

```
ahe@debian:/var/www/html/demo1$ composer require twig
```

Le répertoire templates sera alors créé dans le répertoire du projet avec un fichier de base : base.html.twig

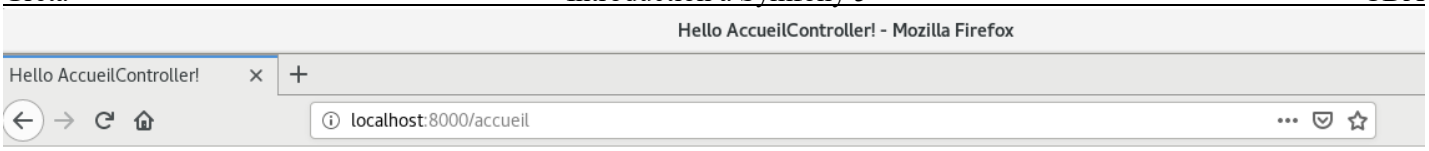
Travail à faire :

Créer un 2ème contrôleur en ligne de commande.
Ecrire votre commande de façon à avoir le résultat escompté :

```
ahe@debian:/var/www/html/cours$ ..........
 created: src/Controller/AccueilController.php
 created: templates/accueil/index.html.twig
  Success!
 Next: Open your new controller class and add some pages!
```

Le contenu du contrôleur :

```
<?php
```



Hello AccueilController! ✅

This friendly message is coming from:

- Your controller at src/Controller/AccueilController.php
- Your template at templates/accueil/index.html.twig

```
namespace App\Controller;

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

class AccueilController extends AbstractController
{
    /**
     * @Route("/accueil", name="accueil")
     */
    public function index()
    {
        return $this->render('accueil/index.html.twig', [
            'controller_name' => 'AccueilController',
        ]);
    }
}
```

On remarque alors que :

- **le fichier twig a l'extension .html.twig**
- **il est créé dans un sous-dossier de templates nommé comme le contrôleur mais sans majuscule.**
- le fichier twig a le même nom que la méthode.

Lancer le navigateur et taper l'URL suivante : **http://localhost:8000/accueil**
La page web s'affiche :

=> Création de la route dans un fichier yaml

Dans l'exemple précédent, la route a été générée à l'aide d'annotation. Les routes peuvent être également générées à l'aide de fichier yaml.

Travail à faire

1. Dans le fichier contrôleur, retirer les annotations concernant la route.

```
<?php

namespace App\Controller;

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

```
class AccueilController extends AbstractController
{
    /**
     * @Route("/accueil", name="accueil")
     */
    public function index()
    {
        return $this->render('accueil/index.html.twig', [
            'controller_name' => 'AccueilController',
        ]);
    }
}
```

2. Rafraîchir la page. Quel est le message d'erreur affiché ?

3. Modifier le fichier config/routes.yaml :

```
accueil:
     path: /accueil
     controller: App\Controller\AccueilController::index
```

4. Rafraîchir la page. Est-ce que l'URL est opérationnelle ?

En résumé :

- Symfony est un framework PHP basé sur une architecture MVC
- La brique de base de Symfony2/3 est le bundle. Ce qui n'est plus le cas en symfony 5
- Le code des contrôleurs est dans le répertoire Controller.
- Les vues sont dans le répertoire templates
- Symfony recommande le langage twig pour la création des vues
- Une URI correspond à une méthode dans un contrôleur. Cette association peut être indiquée directement dans le contrôleur ou dans le fichier routes.yaml (anciennement routing.yml).