

# 1) Découvrir Symfony aujourd'hui

Un framework vous aide à mieux travailler en structurant son développement et plus rapidement (en réutilisant des modules génériques). Il facilite la maintenance et l'adaptabilité sur le long terme en se conformant aux règles de développement. La conformité avec les standards de développement simplifie aussi l'intégration et la connexion de l'application avec le reste du système d'information.

## Symfony est PHP open source.

C'est un ensemble de composants PHP réutilisables, sur lequel les meilleures applications PHP sont développées.

Choisissez l'un des [50 composants](#) disponible pour vos applications.

Symfony est reconnue internationalement comme étant un environnement de développement stable. Il est soutenu par SensioLabs, une entreprise de plus de 13 ans d'expérience dans le développement Web, et par une communauté internationale prospère, qui assure sa longévité.

: <https://symfony.com/explained-to-a-developer> <https://symfony.com/stats/downloads>

## Le process de publication

### Le Versionnement Sémantique

Le Versionnement Sémantique ( abrégé [SemVer](#)), est un système de versionnement de plus en plus utilisés ces dernières années.

Avec de nouveaux plug-ins, add-ons, de nouvelles extensions, et bibliothèques construites tous les jours, avoir une méthode universelle de versionnement de nos logiciels et projets de développement est une bonne chose pour nous aider à suivre efficacement ce qui se passe.

: <https://www.sitepoint.com/semantic-versioning-why-you-should-using/>

### Qu'est-ce que SemVer?

SemVer est un système à 3 composants au format **x.y.z** où :

- **x** représente une version majeure
- **y** représente une version mineure

- **z** représente un correctif

vous avez: **Majeur.Mineur.Correctif**.

## Les versions de Symfony :

Symfony gère ses versions avec la stratégie de [Semantic Versioning](#) :

- Une nouvelle version mineure de Symfony (e.g. 2.8, 3.2, 4.1) tous les *six mois*: une en *Mai* et une en *Novembre*;
- Une nouvelle version majeure de Symfony (e.g., 3.0, 4.0) sort tous les *deux ans* et est publiée en même temps que la dernière version mineure de la précédente version majeure.

Voir : <https://symfony.com/doc/current/contributing/community/releases.html> et <https://symfony.com/roadmap?version=4.1>

## Symfony Flex, 3.3 +

[Symfony Flex](#) est la nouvelle manière d'installer et de gérer ses composants Symfony. Flex n'est pas une nouvelle version de Symfony, mais un outil qui remplace et améliore [Symfony Installer](#) et [Symfony Standard Edition](#).

Symfony Flex **permet d'automatiser les tâches les plus communes des applications Symfony**, comme installer et enlever des packages et autre dépendance Composer. Symfony Flex fonctionne pour Symfony 3.3 et plus. A partir de Symfony 4.0, Flex devrait être utilisé par défaut, mais reste optionnel.

## Recettes

Les recettes Symfony permettent d'automatiser la configuration des packages composer via le plug-in Symfony Flex.

Une recette Symfony consiste en un fichier de configuration **manifest.json** et optionnellement un certain nombre de fichiers et dossiers.

Quand vous installez un nouveau composant Symfony, une recette va s'appliquer et configurer automatiquement votre application Symfony.

**Lorsque vous installez Doctrine la recette va créer pour vous automatiquement une variable d'environnement `database_url` pour la connexion à votre base de données.**

<https://symfony.sh/> et <https://github.com/symfony/recipes/blob/master/doctrine/doctrine-bundle/1.6/manifest.json>

## 2) Créer un Projet Symfony

Si vous deviez commencer à sélectionner manuellement les dépendances que vous vouliez de [Symfony](#), cela deviendrait très compliqué à gérer. Chaque bibliothèque pourrait aussi avoir des dépendances, et cela peut vous conduire vers un désordre, surtout si d'autres personnes travaillent sur votre projet.

### Composer

C'est ici que [Composer](#) intervient. Composer est un gestionnaire de dépendances pour PHP. Composer gère les dépendances que vous avez demandées, projet par projet. Cela signifie que Composer rassemblera toutes les bibliothèques et les dépendances, pour toutes les gérer en un seul endroit.

Voir : <http://www.culttt.com/2013/01/07/what-is-php-composer/>

### Packagist

Packagist est le référentiel de package par défaut de Composer. Il vous permet de trouver les packages et permet à Composer de savoir où se procurer le code. Vous pouvez utiliser Composer pour gérer votre projet ou les dépendances des bibliothèques.

Voir : <https://packagist.org/?q=symfony&p=0>

### Installation de Symfony via Composer :

Composer installera tout ce dont vous avez besoin, pour le bon fonctionnement de votre projet Symfony.

<https://symfony.com/doc/current/index.html>

Étapes de l'installation : <https://symfony.com/doc/current/setup.html>

### Créer un nouveau projet

```
composer create-project symfony/skeleton my-project
```

### Démarrer votre projet

```
cd my-project  
  
composer require server --dev  
  
php bin/console server:run
```

## Ouvrir votre projet dans votre navigateur

Maintenant que **Composer** à tout installé, vous pouvez ouvrir votre nouveau projet Symfony :

<http://localhost:8000/>

## Structure du dossier projet

`config/`

Contient... la configuration bien sûr ! vous configurerez des routes, des *services* et des packages.

`src/`

Tout votre code PHP est placé ici.

`templates/`

Tout vos templates Twig sont placés ici.

La plupart du temps, vous travaillerez dans `src/`, `templates/` ou `config/`. En continuant la lecture, vous apprendrez ce qui peut être fait dans chacun d'entre eux.

Qu'en est-il des autres répertoires du projet ?

`bin/`

Le fameux fichier `bin/console` vit ici (accompagné de d'autres fichiers exécutables moins importants).

`var/`

C'est ici que les fichiers automatiquement créés sont stockés, comme les fichiers cache (`var/cache/`) et les logs (`var/log/`)

`vendor/`

Les bibliothèques tierces sont ici ! Elles sont téléchargées via le [Composer](#) package manager.

public/

C'est le répertoire root de votre projet : vous placez n'importe quel fichier accessible publiquement ici.

voir: [https://symfony.com/doc/current/page\\_creation.html#checking-out-the-project-structure](https://symfony.com/doc/current/page_creation.html#checking-out-the-project-structure)

## 3) Routes et contrôleurs

See : <https://symfony.com/doc/current/routing.html>

Symfony nous offre quelques possibilités nous permettant de gérer le routage d'application selon nos préférences. Si vous préférez configurer vos routes en **YAML**, **XML** ou **PHP**, ce n'est pas un problème !

### YAML

**YAML** est un langage de sérialisation des données. Vous pouvez le penser comme une alternative légère à XML ou JSON. Néanmoins, contrairement à XML ou JSON, sa structure est définie par indentation (pensez Python)

Même si ce format n'est pas très populaire, les fichiers de configuration YAML sont beaucoup plus simple à lire et à prendre en main.

La majorité de la configuration de Symfony est formatée en YAML.

Allez à :

config / routes.yaml

**Voici un exemple :**

```
index:

    path: /

    defaults: { _controller: 'App\Controller\DefaultController::index' }
```

- **index** : nom unique de la route\*
- **path** : Chemin URL.
- **defaults** : Contrôleurs et actions à exécuter.

\*Une *route* est un chemin entre une URL et un contrôleur.

## Contrôleur

Voir : <https://symfony.com/doc/current/controller.html>

Un contrôleur est une fonction PHP que vous créez qui lit les informations provenant de l'objet `Request` et crée en retour un objet `Response`. La réponse peut être une page HTML, JSON, XML, un téléchargement de fichier, une redirection, une erreur 404, ou tout autre chose que vous pourrez imaginer. Le contrôleur exécute toute logique arbitraire dont *votre application* a besoin pour afficher le contenu d'une page.

Créez un nouveau fichier `src / Controller / DefaultController.php`

Un **Contrôleur Symfony** est aussi simple qu'une classe PHP :

```
class DefaultController { }
```

**NOTA BENE** : Grâce à Symfony Autowiring, votre contrôleur sera automatiquement détecté par Symfony. Vous pourrez donc l'utiliser immédiatement sans avoir à le configurer.

Voir : [http://symfony.com/doc/current/service\\_container/autowiring.html](http://symfony.com/doc/current/service_container/autowiring.html)

## Écrivez votre première action

Bien qu'un contrôleur peut être une fonction, une méthode, un objet ou une `Closure`, il est généralement une méthode à l'intérieur d'une classe contrôleur :

Créez votre première action dans la classe `DefaultController` :

```
public function index() {

    return new Response("<html><body><h1>HOME PAGE</h1></body></html>");

}
```

Essayez la ensuite dans votre navigateur : <http://localhost:8000>

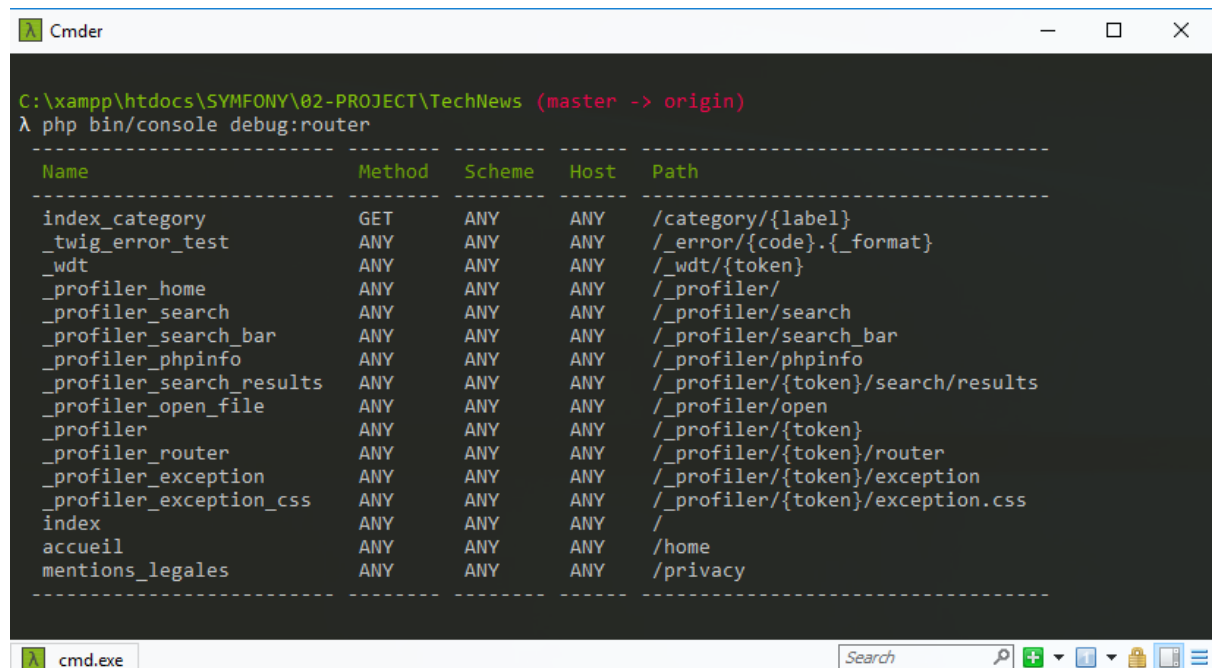
## Router Debug

La console de Symfony peut vous aider à corriger les erreurs de votre application.

Essayez la commande suivante dans la console de votre projet :

```
php bin/console debug:router --env=prod
```

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



```
C:\xampp\htdocs\SYMFONY\02-PROJECT\TechNews (master -> origin)
λ php bin/console debug:router
```

Name	Method	Scheme	Host	Path
index_category	GET	ANY	ANY	/category/{label}
_twig_error_test	ANY	ANY	ANY	/_error/{code}.{_format}
_wdt	ANY	ANY	ANY	/_wdt/{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
index	ANY	ANY	ANY	/
accueil	ANY	ANY	ANY	/home
mentions_legales	ANY	ANY	ANY	/privacy

## Annotations

Les annotations sont un autre moyen de créer une logique de routage.

Voir : <https://symfony.com/doc/current/routing.html#creating-routes>

Dans votre console :

composer require annotations

Vous pouvez maintenant définir vos routes en utilisant les annotations.

```
/**
 *
 * Display category articles.
 *
 * @Route("/categorie/{category}", name="index_category", methods={"GET"})
 */
```

```
* @param $category
* @return Response
*/

public function categorie($category = 'all') {

    return new Response("<html><body><h1>CATEGORY : $category</h1></body></html>");

}
```

Comme vous le constatez, vous pouvez déclarer votre route, directement à partir de votre PHP DocBlock :

```
@Route("/categorie/{category}", name="index_category")
```

Avec ceci, vous n'avez plus besoin de la déclaration YAML.

Corrigez votre route via la console Symfony :

```
php bin/console router:match /category/business
```

```
c:\symfony\Symfony_TestProject (master -> origin)
λ php bin/console router:match /HighTech
```

[OK] Route "index\_category" matches

Property	Value
Route Name	index_category
Path	/ {label} / {currentPage}
Path Regex	#^(?P<label>[\w-]+)(?:/(?P<currentPage>[^\d/]+))?\$#s
Host	ANY
Host Regex	
Scheme	ANY
Method	GET
Requirements	label: [\w-]+
Class	Symfony\Component\Routing\Route
Defaults	_controller: App\Controller\TechNews\CategoryController::category currentPage: 1
Options	compiler_class: Symfony\Component\Routing\RouteCompiler

[Aller plus loin : Redirection et contrôleur de templates](#)

Symfony fournit deux contrôleur spécifiques :

## RedirectController



Quelquefois, une URL doit rediriger vers une autre URL. Vous pouvez faire cela en créant une nouvelle action contrôleur dont l'unique tâche sera de rediriger, mais en utilisant

le `[RedirectController]`(<http://api.symfony.com/4.0/Symfony/Bundle/FrameworkBundle/Controller/RedirectController.html> "`Symfony\Bundle\FrameworkBundle\Controller\RedirectController`") du

`FrameworkBundle`, c'est encore plus simple.

Vous pouvez rediriger vers un emplacement spécifique (e.g. `/home`) ou vers une route spécifique en utilisant son nom (e.g. `index`).

Voir

: [https://symfony.com/doc/current/routing/redirect\\_in\\_config.html#redirecting-using-a-route](https://symfony.com/doc/current/routing/redirect_in_config.html#redirecting-using-a-route)

Ajoutez le code suivant dans votre `routes.yaml` :

```
home:

    path: /home

    controller: Symfony\Bundle\FrameworkBundle\Controller\RedirectController::redirectAction

    defaults:

        route: index

        permanent: true
```

Grâce à **redirectAction**, créer un contrôleur personnalisé n'est maintenant plus nécessaire.

TemplateController

Voir

: [https://symfony.com/doc/current/templating/render\\_without\\_controller.html](https://symfony.com/doc/current/templating/render_without_controller.html)

Habituellement, quand vous avez besoin de créer une page, vous devez créer un contrôleur et générer un template depuis ce contrôleur. Mais si vous générez un simple template ne nécessitant pas de transmission de données, vous pouvez utiliser le contrôleur

incorporé `Symfony\Bundle\FrameworkBundle\Controller\TemplateController::templateAction`, au lieu de créer entièrement un contrôleur.

Supposons que vous voulez générer un template `static/privacy.html.twig`, qui ne nécessite pas l'envoi de variable. Vous pouvez le faire sans créer de contrôleur :

privacy:

path: /privacy

controller: Symfony\Bundle\FrameworkBundle\Controller\TemplateController::templateAction

defaults:

template: static/privacy.html.twig

Symfony request flow :

