



Symfony

L'architecture de Symfony

Symfony



Symfony

Introduction

L'architecture de Symfony

Un framework en quelques mots...

- L'objectif principal d'un **framework** consiste à faciliter le travail des développeurs
- Un **framework** représente un ensemble de **composants** permettant de créer les fondations d'un logiciel informatique
- En proposant une **architecture** « prête à l'emploi », un **framework** permet de ne pas avoir à réinventer la roue à chaque nouveau projet en utilisant des **composants** développés par d'autres
- Un **framework** est donc perpétuellement enrichi par l'expériences de nombreux développeurs
- En outre, il facilite la **réutilisation de code** tout en offrant un **standard** à la programmation ainsi qu'une **architecture**

Pourquoi Symfony ?

- Symfony est un **framework** écrit en **PHP** représentant un ensemble de **composants**
- Considéré comme l'un des meilleurs **frameworks PHP** du marché, Symfony a été développé par la société française SensioLabs 
- Symfony se distingue de la concurrence par la qualité qu'il offre, tant en terme de **sécurité** web que de **performance**
- Reconnu mondialement, il dispose d'une très large **communauté internationale** développant plusieurs milliers de **composants libres et gratuits**



Symfony

Introduction à Symfony

L'architecture de Symfony

Symfony en quelques mots... (1/2)

- Symfony est un **framework PHP Modèle-Vue-Contrôleur (MVC)** libre et gratuit offrant un accès à un ensemble de **composants PHP**
- Il fournit des **fonctionnalités modulables et adaptables** accélérant et facilitant la construction de sites et d'applications web
- Le projet a émergé de l'agence web française **SensioLabs** sous le nom de **Sensio Framework**
- Pour ne pas avoir à réécrire toujours les mêmes fonctionnalités, l'entreprise a développé cet outil pour satisfaire ses propres besoins avant de le partager à la **communauté PHP** et de le renommer **Symfony**

Symfony en quelques mots... (2/2)

- Laravel et Symfony sont les **frameworks PHP** les plus utilisés
- De base, Symfony propose **50 composants** autonomes à mettre en place dans des sites et applications web
- Il permet donc d'accélérer leur **création** et leur **maintenance** tout en évitant de répéter du code en permettant de garder un contrôle sur ce dernier
- Symfony dispose d'une large **communauté** avec plus **600 000 développeurs** dans plus de **120 pays** pour faire évoluer **PHP** et répondre à vos questions



Symfony

Installation

L'architecture de Symfony

Pré-requis - PHP

- Si vous n'avez pas une version de **PHP >= 7.2.5** installée sur votre machine, téléchargez et installez la dernière version disponible de **PHP**
- Si vous êtes sous **Windows**, vous pouvez suivre ce [guide](#)
- Sous **Ubuntu**, rendez-vous à cette [adresse](#)
- Sous **Mac OS**, vous pouvez suivre ce [tutoriel](#)
- La commande `php -v` permet de connaître la version de **PHP** installée sur votre ordinateur

```
Chris_Chevalier@Macbook ~ % php -v
PHP 7.3.26 (cli) (built: Jan  8 2021 13:03:03) ( NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.3.26, Copyright (c) 1998-2018 Zend Technologies
with Zend OPcache v7.3.26, Copyright (c) 1999-2018, by Zend Technologies
```

Pré-requis - Composer

- Une fois une version **PHP >= 7.2.5** installée sur votre machine, installez **Composer** en suivant les instructions dispensées sur le [site officiel](#)
- **Composer** est un logiciel de gestion de **dépendances PHP**
- Il nous permettra d'installer les différents **composants** dont notre projet aura besoin
- La **commande** `composer -v` vous permettra de vous assurer que **Composer** a été correctement installé sur votre machine

```
Chris_Chevalier@Macbook ~ % composer -v
```



Pré-requis - Symfony CLI

- Téléchargez et installez l'**application en ligne de commande (CLI) Symfony** en suivant ce [guide](#)
- Cette installation est optionnelle mais va nous permettre d'avoir accès à tous les outils **Symfony** permettant le développement et l'exécution d'applications en local
- Une fois les pré-requis installés, saisissez la **commande** `symfony check:requirements` pour vérifier si toutes vos installations se sont bien passées

```
[OK]
Your system is ready to run Symfony projects
```



Symfony

Créer et lancer une application

L'architecture de Symfony

Créer une application Symfony

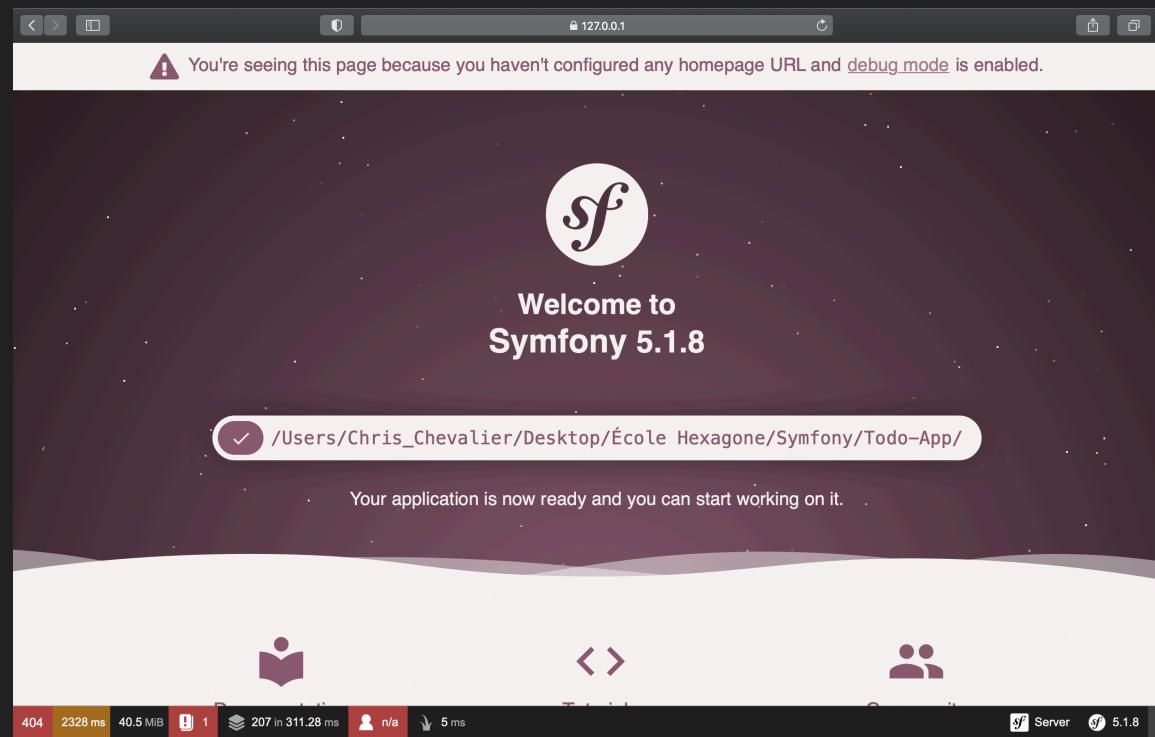
- Pour créer un nouveau projet **Symfony**, il suffit de lancer la **commande** `symfony new nom_du_projet` dans le répertoire de votre choix
- Associée à cette **commande**, le **drapeau** `--full` permet d'installer les **composants** habituellement nécessaires à la construction d'applications web
- Cette **commande** crée un nouveau dossier « `nom_du_projet` », télécharge les **dépendances** et génère les fichiers et dossiers de base nécessaires pour lancer une application **Symfony**
- Notre application est ainsi prête à l'emploi !

Lancer une application Symfony

- **Symfony CLI** permet également d'utiliser un **serveur web en local**
- Pour cela, on saisit simplement cette **commande** dans le répertoire du projet : `symfony server:start` ou `symfony serve`
- L'application se lance à l'adresse <http://localhost:8000>
- Pour arrêter le serveur, il suffit de stopper la **commande** lancée depuis le **terminal** via le raccourci `Ctrl+C`

La page d'accueil Symfony

- Si tout s'est bien passé, **cet écran d'accueil** apparaît à l'adresse <http://localhost:8000> ou <http://127.0.0.1:8000>





Symfony

Installer des paquets

L'architecture de Symfony

Symfony Flex en quelques mots...

- Durant le développement d'une application web **Symfony**, il est commun de devoir installer des **paquets (dépendances)**
- Ils fournissent des **fonctionnalités** prêtes à l'emploi et nécessitent généralement une certaine **configuration** pour pouvoir les utiliser
- Bien heureusement, cette **configuration** peut être **automatisée** grâce à l'outil **Symfony Flex** simplifiant l'installation et la suppression de **paquets**
- Il s'agit d'une **extension Composer**, installée par défaut lors de la création d'une application **Symfony**, automatisant les tâches les plus courantes du **framework**
- En outre, le **plugin** modifie le comportement des **commandes Composer** de base

Installer un paquet

- Pour installer un **paquet**, il suffit de saisir la **commande** `composer require nom-du-paquet` dans le répertoire de votre projet **Symfony** (Exemple : `composer require logger`)
- Cette **commande** installe et active tous les **paquets** nécessaires à l'utilisation du **paquet** spécifié
- Cette pratique est rendue possible grâce au fait que de nombreux **paquets Symfony** définissent des **recettes** décrivant un ensemble d'**instructions automatisées** pour **installer** et **activer** plusieurs **paquets**
- Par ailleurs, **Flex** garde la trace des **recettes** des **paquets** installés dans un fichier `symfony.lock` à la **racine** d'un projet **Symfony**
- Les **dépendances** installées dans un projet **Symfony** apparaissent dans les sections `require` et `require-dev` du fichier `composer.json`, situé à la **racine**

Le fichier `composer.json`

```
1  {
2      "type": "project",
3      "license": "proprietary",
4      "minimum-stability": "dev",
5      "prefer-stable": true,
6      "require": {
7          "php": ">=7.2.5",
8          "ext-ctype": "*",
9          "ext-iconv": "*",
10         "symfony/console": "5.2.*",
11         "symfony/dotenv": "5.2.*",
12         "symfony/flex": "^1.3.1",
13         "symfony/framework-bundle": "5.2.*",
14         "symfony/yaml": "5.2.*"
15     },
16     "require-dev": {
17         "symfony/phpunit-bridge": "^5.2"
18     },
19 }
```

Installer plusieurs paquets

- Il arrive qu'une fonctionnalité nécessite l'installation de **plusieurs paquets**
- Plutôt que de les installer individuellement, **Symfony** fournit des **méta-paquets Composer** comprenant **plusieurs dépendances**
- Lancer la commande `composer require debug --dev` ajoute des fonctionnalités de **déboggage** à une application
- En réalité, cette commande installe le **paquet** `symfony/debug` qui installe à son tour plusieurs autres **dépendances** : `symfony/debug-bundle`, `symfony/monolog-bundle`, `symfony/var-dumper`, etc.

La sécurité dans Symfony (1/2)

- L'**application en ligne de commande Symfony** fournit, entre autres, une fonctionnalité permettant de vérifier si les **dépendances** de votre projet contiennent une **vulnérabilité de sécurité connue**
- Pour cela, il suffit de saisir la **commande** `symfony check:security` au sein de votre projet
- Une **bonne pratique de sécurité** consiste à **exécuter cette commande régulièrement** afin de **mettre à jour ou remplacer les dépendances compromises** au plus vite
- Ce contrôle de sécurité est réalisé **localement** en se référant à la base de données publique des avis de sécurité **PHP**

La sécurité dans Symfony (2/2)

```
Chris_Chevalier@Macbook demo-app % symfony check:security
Symfony Security Check Report
=====
```

```
No packages have known vulnerabilities.
```

```
Note that this checker can only detect vulnerabilities that are referenced in the security advisories database.
Execute this command regularly to check the newly discovered vulnerabilities.
```

- Dans ce projet, il n'y a aucune vulnérabilité connue parmi les paquets installés



Symfony

TP - L'application de démonstration Symfony

L'architecture de Symfony

TP - L'application de démonstration Symfony (1/2)

- Installez l'**application de démonstration** fournit par **Symfony** en saisissant, dans le répertoire de votre choix, la **commande d'installation**, suivi du **drapeau** —demo - Nommez le projet demo-app
- Ouvrez le dossier ainsi créé dans votre **éditeur de texte** favoris (Visual Studio Code, Atom, etc.)
- Lancez l'application à l'adresse <http://127.0.0.1:8000>
- Naviguez sur l'**interface utilisateur**
- Parcourez les **fichiers sources** sans les modifier

TP - L'application de démonstration Symfony (2/2)

- Vérifiez la vulnérabilité des paquets installés => Interprétez la sortie rendue par la console
- Dans quel **fichier** s'affiche la **liste des dépendances** de votre projet **Symfony** ?
- Les réponses à ces 2 questions doivent apparaître dans un nouveau fichier *questions.md* situé à la racine du projet - Voici un [guide](#) indiquant comment écrire un fichier **Markdown**
- **Supprimez** le dossier *vendor* de votre projet (il contient les dépendances installées ; il est donc très volumineux) avant d'**envoyer votre dossier compressé au format ZIP à chevalier@chris-freelance.com**



Symfony

Créer une page Symfony

L'architecture de Symfony

Créer une page Symfony

- Créer une page **Symfony** se fait toujours en 2 étapes :
 - Construire un **contrôleur** (une **fonction PHP**) - Il utilise les informations envoyées dans la **requête HTTP** pour créer un **objet Symfony** de type *Response* contenant du code **HTML**, **JSON** ou encore **un fichier**
 - Définir une **route** - il s'agit de l'**URL** de la page qui pointe vers le **contrôleur** (Exemple : `/about`)

Un exemple de contrôleur

- Voici un exemple de **contrôleur Symfony** renvoyant un **nombre aléatoire** compris entre 0 et 100 au format **HTML**

```
1  <?php
2  // src/Controller/LuckyController.php
3
4  // On définit un namespace pour chacun de nos classe afin de
5  // les rendre accessible depuis d'autres fichiers du projet
6  namespace App\Controller;
7
8  // Ici, on importe le(s) composants utiles à la classe
9  use Symfony\Component\HttpFoundation\Response; // Représente une réponse HTTP
10
11 class LuckyController {
12     public function number(): Response { // Un contrôleur renvoie TOUJOURS une réponse HTTP
13         // On stocke le résultat de la fonction PHP random_int() dans la variable $number
14         $number = random_int(0, 100);
15         // La réponse HTTP affiche du HTML basique contenant la variable $number
16         return new Response(
17             "<html><body>{$number}</body></html>"
18         );
19     }
20 }
21
22 ?>
```

Un exemple de route

- Pour associer cette **méthode de contrôleur** à un **route** (ici, *lucky/number*) et que celle-ci soit appelée quand l'utilisateur atteint cette **URL**, il suffit de définir une **route** dans le fichier *config/routes.yaml*

```
1 # config/routes.yaml
2
3 app_lucky_number: # On nomme la route
4   path: lucky/number # On fait pointer la route sur une URL
5   controller: App\Controller\LuckyController::number # On associe la route à un contrôleur
```

- En se rendant à l'adresse <http://localhost:8000/lucky/number>, on obtient cet affichage avec un nombre changeant à chaque recharge de la page (à chaque appel de la méthode *number()*)

Lucky number: 93

À vous de jouer ! (1/2)

- Créez un nouveau projet Symfony « *my-first-project* » et suivez les étapes décrites précédemment pour créer une nouvelle page affichant un **nombre aléatoire** à chaque actualisation
- Dans le dossier *src/Controller* du projet, créez une classe *LuckyController* contenant la méthode *number()* renvoyant un nombre aléatoire choisi entre 0 et 100
- Configurez le fichier *config/routes.yaml* pour faire pointer l'URL */lucky/number* sur le contrôleur créé à l'étape précédente
- Lancez votre application **Symfony**, rendez-vous à l'adresse <http://localhost:8000/lucky/number> pour visualiser et tester votre première page **Symfony**

Le paquet annotations

- Pour ne pas avoir à définir vos **routes** en permanence dans le fichier config/routes.yaml, **Symfony** propose d'utiliser des **annotations** grâce au bien nommé **paquet annotations**
- Pour l'utiliser, il suffit de l'installer à l'aide de la **commande** composer require annotations et de définir les **routes** au dessus de la **fonction contrôleur** souhaitée

```
11 // On importe le composant pour définir la route de notre contrôleur à l'aide d'une annotation
12 use Symfony\Component\Routing\Annotation\Route;
13
14 class LuckyController {
15     // On place l'annotation @Route au dessus de méthode contrôleur.
16     // On lui donne une URL et éventuellement un nom
17
18     /**
19      * @Route("/lucky/number", name="app_lucky_number")
20      */
21     public function number(): Response {
22         // ...
```

Débogguer avec Symfony (1/3)

- Un projet **Symfony** tel que le notre possède déjà un puissant outil pour **déboguer** une application
- Il s'agit de la **commande** `php bin/console`
- En saisissant cette **commande** au sein d'un projet **Symfony**, une **liste de commandes** apparaît
- Elles permettent de vous donner des **informations de déboggage**, à **générer du code**, à **générer des migrations de bases de données**, etc.
- Dans notre cas, il peut être intéressant de faire apparaître la **liste de toutes les routes disponibles** dans votre application avec la **commande** `php bin/console debug:router`

Déboguer avec Symfony (2/3)

```
Chris_Chevalier@Macbook my-first-project % php bin/console debug:router
```

Name	Method	Scheme	Host	Path
_preview_error	ANY	ANY	ANY	/_error/{code}.{_format}
app_lucky_number	ANY	ANY	ANY	/lucky/number

Débogguer avec Symfony (3/3)

- Pour débogguer une application, Symfony propose également une **barre d'outils** disposant d'un certain nombre d'informations - Elle est disponible **en bas de votre page de développement**
- Pour avoir accès à cette dernière, il suffit d'installer le **paquet symfony/profiler-pack**
- Elle permet, entre autres, d'obtenir des informations sur le **routage**, les **performances** et les **logs**

Le rendu de pages avec Twig (1/3)

- Twig est un **langage de template** pour **PHP** utilisé par défaut par **Symfony**
- La **commande** `composer require twig` permet d'**installer le paquet** et crée automatique un dossier *templates* situé à la **racine** du projet
- En mettant à jour notre **contrôleur** `number()` situé dans la **classe** `LuckyController`, nous pouvons retourner une **réponse** associée à un fichier **Twig**

Le rendu de pages avec Twig (2/3)

```
1  <?php
2  // src/Controller/LuckyController.php
3
4  namespace App\Controller;
5
6  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController; // Classe incluant des méthodes utiles aux contrôleurs
7  use Symfony\Component\HttpFoundation\Response;
8  use Symfony\Component\Routing\Annotation\Route;
9
10 class LuckyController extends AbstractController { // LuckyController hérite du bundle AbstractController pour utiliser ses méthodes publiques
11     /**
12      * @Route("/lucky/number", name="app_lucky_number")
13      */
14     public function number(): Response {
15         $number = random_int(0, 100);
16         // La méthode render(), héritée de la classe AbstractController, renvoie une réponse HTTP sous la forme d'un fichier Twig
17         return $this->render('/lucky/number.html.twig', [
18             'number' => $number // On envoie la valeur de la variable $number au fichier Twig
19         ]);
20     }
21 }
22
23 ?>
```

Le rendu de pages avec Twig (3/3)

- Il ne nous reste plus qu'à créer et remplir notre fichier **Twig**

```
1  {%# templates/lucky/number.html.twig #}
2  {%# Un fichier Twig s'écrit avec des balises HTML auxquelles on peut ajouter des variables Twig #}
3  <h2>Votre nombre fétiche est le {{ number }}</h2>
```

- Vos **templates** doivent être enregistrés dans le répertoire `./templates` automatiquement créé lors de l'installation du paquet `twig`
- Ici, on trouve le fichier `number.html.twig` dans un répertoire `lucky`
- Les **doubles accolades** représentent la syntaxe utilisée pour afficher des variables **Twig** => `{{ maVariable }}`

À vous de jouer ! (2/2)

- Suivez les étapes décrites précédemment pour recréer la **page affichant un nombre aléatoire** à chaque actualisation en utilisant un template **Twig** :
 - Installez le **paquet twig**
 - Mettez à jour la **classe** *LuckyController* afin de lui faire hériter du bundle *AbstractController* et renvoyer le fichier **Twig** *templates/lucky/number.html.twig* en **réponse** avec le nombre aléatoire stocké dans la variable `$number`
 - Créez et rédigez le fichier *templates/lucky/number.html.twig* affichant un message « **Votre nombre fétiche est le <votre-nombre>** »
- Vous trouverez plus de détails concernant la **syntaxe** de **Twig** sur sa **documentation officielle** (Nous y reviendrons dans le cours **3 - Twig le moteur de template**)



Symfony

La structure d'un projet Symfony

L'architecture de Symfony

La structure d'un projet Symfony (1/2)

- À ce stade du cours, nous avons déjà travaillé avec les **répertoires** les plus importants d'un projet **Symfony**
- C'est dans le dossier *config* que l'on peut **configurer** les **routes**, **services** et **paquets** utilisés
- Tout votre code **PHP** se trouve dans le dossier *src*
- Les modèles **Twig** se trouvent dans le répertoire *templates*

La structure d'un projet Symfony (2/2)

- Le fichier *bin/console* permettant d'utiliser les **commandes de déboggage** se trouve, sans surprise, dans le dossier *bin*
- Les fichiers de **cache** et de **logs**, créés automatiquement, sont stockés dans le répertoire *var*
- Les **bibliothèques tierces (dépendances)** téléchargées par **Composer** sont installées dans le dossier *vendor*
- La **racine** de votre projet se trouve dans le dossier *public*
- Comme pour **Twig**, l'installation de certains **paquets crée des dossiers automatiquement**



Symfony

TP - Une page de bienvenue

L'architecture de Symfony

TP - Une page de bienvenue (1/2)

- Toujours dans le projet « *my-first-project* », créez une **classe** `WelcomeController` héritant du **bundle** `AbstractController` et dont la seule **méthode** `welcome()` représente un **contrôleur Symfony**
- Utilisez les **annotations de route** pour faire pointer le contrôleur sur la route racine de votre application (/) - Nommez cette route « `welcome` »
- En réponse, le contrôleur doit afficher un **message de bienvenue** avec le nom de votre application passé à un **template** `/templates/welcome.html.twig` sous la forme d'une variable => « **Bienvenue sur <nom-de-mon-appli>** »

TP - Une page de bienvenue (2/2)

- Faites apparaître les différentes **routes** disponibles pour votre application depuis un terminal à l'aide d'une **commande** (Cf. **diapo 32**)
- Copiez-les manuellement pour les afficher sous la forme d'une **liste HTML non ordonnée** () dans votre modèle **Twig**
- Chaque élément de la liste doit correspondre au **nom** de la route associé à un lien hypertexte pointant vers son **URL** quand elle ne contient pas de paramètre

TP - Une page de bienvenue (3/3)

- Dans quel dossier d'un projet **Symfony** se trouvent les **dépendances installées** ?
- Quel dossier est créé automatiquement par la **commande** `composer require twig` ?
- Les réponses à ces 2 questions doivent apparaître dans un nouveau fichier `README.md` situé à la racine du projet
- **Envoyez votre dossier** de projet (vendor exclus) **compressé** au **format ZIP** à chevalier@chris-freelance.com

Pour aller plus loin...

- Site officiel **Symfony** : <https://symfony.com>
- Documentation **Symfony** : <https://symfony.com/doc/current/index.html>
- Site officiel **SensioLabs** : <https://sensiolabs.com>
- Site officiel **PHP** : <https://www.php.net>
- Site officiel **Composer** : <https://getcomposer.org>
- Documentation **Twig** : <https://twig.symfony.com>



Symfony

Des questions ?