



Simon GAUTIER



Cours Symfony 5

Qui suis-je ? 😊

- Formation : Master 2 STIC à l'Université de Poitiers

- 2006 à 2017 : ESN



- Octobre 2017 → Juillet 2020 : responsable équipe de développement chez PGA Motors devenu Emil Frey France depuis le 1^{er} février 2019
- Depuis juillet 2020 : en charge de l'hébergement des solutions web



Quelques prérequis

- Connaissance des concepts objet
- Connaissance de PHP
- Connaissance du XML / YML
- Connaissance des principes du modèle MVC
- Motivation : ce cours n'est qu'une présentation rapide de Symfony
→ il est nécessaire de creuser ensuite chaque notion
- Cours **orienté backend** : même si nous abordons le moteur de template Twig, nous ne travaillons pas à la construction d'un site frontend Symfony (gestion fine des assets, ...)

Fonctionnement du cours

- Présentation des aspects théoriques
- Présentation d'exemples concrets (beaucoup de code)
➔ Conseil : s'exercer avec les exemples présentés afin de bien assimiler les notions + approfondir au besoin avec la doc en ligne
- Tout le long du cours, TPs : construction d'un mini-blog
- Evaluation :
 - Sur la base des TPs produits ➔ Commentez votre code !
 - Un commentaire permet au développeur de structurer sa pensée et au lecteur de plus vite le comprendre
 - Le code présenté dans les diapos est commenté, inspirez-vous en ;-)
 - Dans un premier temps, beaucoup de commentaires sont nécessaires lorsqu'on découvre un nouveau framework / une nouvelle techno. Puis, on ne rédige des commentaires au sein du code que pour les cas à expliciter
 - Assiduité / participation lors des TPs

Légende

- Slides pour les TP :



- Slides obsolètes (conservées pour l'historique) :



ARCHIVE

- Slides non abordées en cours qui traitent de concepts plus avancés :



Pour aller plus loin

Focus sur le mini-blog que nous allons développer



- **/ →** Une page d'accueil de notre site qui permet de se présenter
 - Le blog que nous développons n'est qu'une rubrique du site, on pourrait en avoir d'autres (ex : boutique, forum, ...)
- **/blog/list/xxx →** Une page d'accueil du blog qui liste les articles du plus récent au plus ancien avec pagination (xxx est le numéro de page)
- **/blog/article/xxx →** La page de consultation de l'article xxx
- **/blog/article/add →** Un formulaire d'ajout d'article
- **/blog/article/edit/xxx →** Un formulaire de modification de l'article xxx
- **/blog/article/delete/xxx →** Une action de suppression de l'article xxx qui redirige vers la liste des articles du blog après suppression

Sommaire

- Introduction
- Installation
- Architecture
- Hello World (routes et contrôleurs)
- Twig
- Doctrine
- Doctrine - Relations entre entités
- Gestion des formulaires
- Services Symfony / Injection de dépendances
- Sécurité (excepté les voters)
- i18n : internationalisation
- Sécurité (voters)
- Doctrine - Evènements et extensions
- Gestion d'évènements
- Les formulaires (avancé)
- La console Symfony
- Le profiler Symfony
- Cache HTTP
- Gestion des tests

Sommaire

Objectifs promo 2020/2021

J5 et J10

Temps réservé pour la pratique

- Introduction **J1**
- Installation **J1**
- Architecture **J1**
- Hello World (routes et contrôleurs) **J1**
- Twig **J2**
- Doctrine **J3**
- Doctrine - Relations entre entités **J4**
- Gestion des formulaires **J6**
- Services Symfony / Injection de dépendances **J7**
- Sécurité (excepté les voters) **J8**
- i18n : internationalisation **J8**
- Sécurité (voters) **J9**
- Doctrine - Evènements et extensions **J9**
- Gestion d'évènements
- Les formulaires (avancé)
- La console Symfony
- Le profiler Symfony
- Cache HTTP
- Gestion des tests



Sommaire

Répartition des journées avec la promo 2017/2018

- Introduction **J1**
- Installation **J1**
- Architecture **J1**
- Hello World (routes et contrôleurs) **J1**
- Twig (excepté la création d'extension) **J2**
- Sécurité (excepté les voters) **J7**
- i18n : internationalisation **J8**
- Services Symfony / Injection de dépendances **J3**
- Gestion d'évènements
- Doctrine **J4**
- Doctrine - Relations entre entités **J5**
- Doctrine - Evènements et extensions **J8**
- Gestion des formulaires **J6**
- Les formulaires (avancé)
- La console Symfony
- Le profiler Symfony
- Cache HTTP
- Gestion des tests



Sommaire

Répartition des journées avec la promo 2018/2019

- Introduction **J1**
- Installation **J1**
- Architecture **J1**
- Hello World (routes et contrôleurs) **J1**
- Twig **J2**
- Doctrine **J3**
- Doctrine - Relations entre entités **J4**
- Gestion des formulaires **J5**
- Services Symfony / Injection de dépendances **J6**
- Sécurité (excepté les voters) **J7**
- Doctrine - Evènements et extensions **J8**
- i18n : internationalisation **J8**
- Gestion d'évènements
- Les formulaires (avancé)
- La console Symfony
- Le profiler Symfony
- Cache HTTP
- Gestion des tests



Sommaire

Répartition des journées avec la promo 2019/2020

- Introduction **J1**
- Installation **J1**
- Architecture **J1**
- Hello World (routes et contrôleurs) **J1**
- Twig **J2**
- Doctrine **J3**
- Doctrine - Relations entre entités **J4**
- Gestion des formulaires **J5**
- Services Symfony / Injection de dépendances **J6**
- Sécurité (excepté les voters) **J7**
- i18n : internationalisation **J7**
- Sécurité (voters) **J8**
- Doctrine - Evènements et extensions **J8**
- Gestion d'évènements
- Les formulaires (avancé)
- La console Symfony
- Le profiler Symfony
- Cache HTTP
- Gestion des tests



Simon GAUTIER



Introduction

Qu'est-ce que Symfony ?

Framework MVC opensource made in France par SensioLabs

- Symfony = des composants ET AUSSI un framework
- Symfony est un ensemble de composants indépendants qui traitent chacun un sujet précis → on peut les utiliser indépendamment dans des projets PHP
 - Quelques composants : HttpFoundation, Console, Templating, Validator, ...
- Symfony est aussi un framework full-stack PHP développé à partir de ces composants
- Framework ?
 - Littéralement, Framework = cadre de travail
 - <https://fr.wikipedia.org/wiki/Framework>

Pourquoi un framework ?

- Pour produire du code **propre** (car on suit son architecture), **maintenable**, **réutilisable**, **fiable** (éprouvé par une communauté), **sécurisé**
 - Quand on respecte les bonnes pratiques du Framework 😊
- Pour **travailler en équipe**
 - Code normé et connu → facile d'intégrer un nouveau développeur à l'équipe (contrairement au développement « maison »)
 - Modèle en couche : séparer à minima la logique métier de la logique d'affichage (l'intégrateur sait travailler dans sa partie du framework sans être perturbé par le code PHP)
 - Communauté
 - Documentation, tutoriaux, forums, ...
 - Respect des standards de programmation, support, mises à jour, montées en version, ...
- Framework vs. solution toute faite (ex : CMS) ?
 - Le framework n'est pas utilisable en tant que tel
 - Le framework nécessite le travail du développeur
 - Remarque : certains produits du marché sont construits avec Symfony :
 - Thelia (solution e-commerce), eZ Platform (CMS), Sylius (solution e-commerce), ...

MVC = Modèle / Vue / Contrôleur

- **C**ontrôleur

- Fournir une réponse à la requête HTTP demandée
- Vérifier les données en entrée (paramètres, ...)
- Appeler les autres couches

- **M**odèle :

- Fournir une couche d'abstraction pour l'accès aux données (en général : accès à un SGBD mais potentiellement n'importe quoi qui contient des données !)

- **V**ue :

- Générer l'affichage, dans notre cas le code HTML des pages
- Accessible en théorie aux intégrateurs qui n'ont pas nécessairement de compétences de développement back-end

Politique de releases de Symfony

- symfony.com/roadmap
- Patches tous les mois : X.Y.1, X.Y.2, ...
- Versions mineures 2 fois par an : X.1.0, X.2.0, ...
 - 5.1 est sortie en mai 2020
 - 5.2 doit sortir en novembre 2020
- Versions majeures tous les 2 ans : 3.0.0, 4.0.0, ...
 - 4.0 est sortie en novembre 2017
 - 5.0 est sortie en novembre 2019
 - 6.0 sortira théoriquement en novembre 2021

Quels outils pour le développeur ?

- En environnement local, nécessite uniquement PHP car un serveur est intégré
- Git (comment travailler sans ???!)
- Composer (évoqué dans le prochain chapitre)
- IDE conseillé par SensioLabs : PhpStorm + plugin Symfony



Ressources

- Documentation : symfony.com/doc
 - À bookmarker !!
- Support : symfony.com/support
- Remonter des anomalies / contribuer : github.com/symfony/symfony
- Actualités officielles : symfony.com/blog



Simon GAUTIER



Installation

Composer

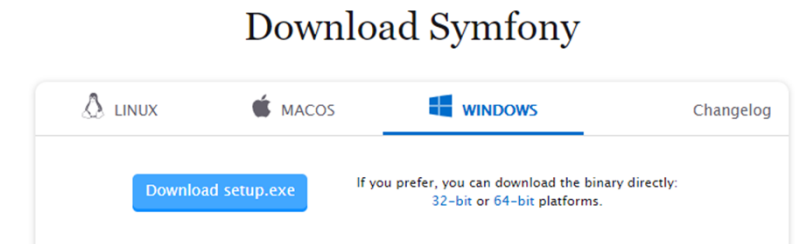
Flex

Installer Symfony

Installer une dépendance

Philosophie du micro-framework

- Installation de base de Symfony : framework minimaliste
- Les composants nécessaires au projet ne sont installés qu'à la demande au fur et à mesure
- ➔ 2 projets Symfony n'ont potentiellement « pas le même code source » en fonction des composants installés
- Installation de Symfony :
 - Avant la version 3.3 : via le Symfony installer
 - En 3.4 et en 4.x : via **Composer**
 - Depuis la 5.0 : utilisation du setup Symfony ➔ <https://symfony.com/download>
 - Permet d'avoir la commande « **symfony** »



Composer



- Gestionnaire de dépendances PHP
- Installer Composer → getcomposer.org/download
- Pour mettre à jour composer : **\$ composer self-update**
- Dépôt principal de Composer : Packagist → packagist.org
 - Il est possible de définir son propre dépôt
- Principe de Composer :
 - composer.json : lister les composants souhaités (et leurs versions) sur le projet
 - composer.lock : liste des composants et leurs dépendances à leurs versions exactes à l'instant T
 - Gestion de l'autoload pour définir comment inclure chaque dépendance à notre projet
→ cf. résultat dans **vendor/autoload.php**



Composer

Exemple : faire un projet PHP qui utilisera le moteur de template Twig



- Préparer un composer.json dans un répertoire vide :
 - Lister les librairies utiles et les versions demandées

```
{  
  "require": {  
    "twig/twig": "^3.0"  
  }  
}
```

- Mettre à jour les composants via la commande **composer update**

```
Loading composer repositories with package information  
Updating dependencies (including require-dev)  
Package operations: 3 installs, 0 updates, 0 removals  
  - Installing symfony/polyfill-ctype (v1.20.0): Downloading (100%)  
  - Installing symfony/polyfill-mbstring (v1.20.0): Downloading (100%)  
  - Installing twig/twig (v3.1.1): Downloading (100%)  
Writing lock file  
Generating autoload files  
3 packages you are using are looking for funding.  
Use the 'composer fund' command to find out more!
```

- Actions de Composer :
 - Installation de la librairie demandée et de ses éventuelles dépendances
 - Génération du autoload.php
 - ➔ Regarder le répertoire « vendor » en détail (notamment répertoire « composer »)
 - Il reste à écrire son code en incluant simplement le autoload.php généré par composer
- Exemple de projet : https://github.com/sgautier/composer_test

Composer – versions



- **"xxxx/yyyy" : "2.4.2"**

1.2.0	1.3.0	1.4.0	2.0.0	2.1.0	2.2.0	2.3.0	2.4.0	2.5.0	3.0.0	3.1.0	3.2.0
1.2.1	1.3.1	1.4.1	2.0.1	2.1.1	2.2.1	2.3.1	2.4.1	2.5.1	3.0.1	3.1.1	3.2.1
1.2.2	1.3.2	1.4.2	2.0.2	2.1.2	2.2.2	2.3.2	2.4.2	2.5.2	3.0.2	3.1.2	3.2.2
1.2.3	1.3.3	1.4.3	2.0.3	2.1.3	2.2.3	2.3.3	2.4.3	2.5.3	3.0.3	3.1.3	3.2.3
1.2.4	1.3.4	1.4.4	2.0.4	2.1.4	2.2.4	2.3.4	2.4.4	2.5.4	3.0.4	3.1.4	3.2.4
1.2.5	1.3.5	1.4.5	2.0.5	2.1.5	2.2.5	2.3.5	2.4.5	2.5.5	3.0.5	3.1.5	3.2.5
1.2.6	1.3.6	1.4.6	2.0.6	2.1.6	2.2.6	2.3.6	2.4.6	2.5.6	3.0.6	3.1.6	3.2.6
1.2.7	1.3.7	1.4.7	2.0.7	2.1.7	2.2.7	2.3.7	2.4.7	2.5.7	3.0.7	3.1.7	3.2.7
1.2.8	1.3.8	1.4.8	2.0.8	2.1.8	2.2.8	2.3.8	2.4.8	2.5.8	3.0.8	3.1.8	3.2.8
1.2.9	1.3.9	1.4.9	2.0.9	2.1.9	2.2.9	2.3.9	2.4.9	2.5.9	3.0.9	3.1.9	3.2.9

Composer – versions



- **"xxxx/yyyy" : "2.4.*"**

1.2.0	1.3.0	1.4.0	2.0.0	2.1.0	2.2.0	2.3.0	2.4.0	2.5.0	3.0.0	3.1.0	3.2.0
1.2.1	1.3.1	1.4.1	2.0.1	2.1.1	2.2.1	2.3.1	2.4.1	2.5.1	3.0.1	3.1.1	3.2.1
1.2.2	1.3.2	1.4.2	2.0.2	2.1.2	2.2.2	2.3.2	2.4.2	2.5.2	3.0.2	3.1.2	3.2.2
1.2.3	1.3.3	1.4.3	2.0.3	2.1.3	2.2.3	2.3.3	2.4.3	2.5.3	3.0.3	3.1.3	3.2.3
1.2.4	1.3.4	1.4.4	2.0.4	2.1.4	2.2.4	2.3.4	2.4.4	2.5.4	3.0.4	3.1.4	3.2.4
1.2.5	1.3.5	1.4.5	2.0.5	2.1.5	2.2.5	2.3.5	2.4.5	2.5.5	3.0.5	3.1.5	3.2.5
1.2.6	1.3.6	1.4.6	2.0.6	2.1.6	2.2.6	2.3.6	2.4.6	2.5.6	3.0.6	3.1.6	3.2.6
1.2.7	1.3.7	1.4.7	2.0.7	2.1.7	2.2.7	2.3.7	2.4.7	2.5.7	3.0.7	3.1.7	3.2.7
1.2.8	1.3.8	1.4.8	2.0.8	2.1.8	2.2.8	2.3.8	2.4.8	2.5.8	3.0.8	3.1.8	3.2.8
1.2.9	1.3.9	1.4.9	2.0.9	2.1.9	2.2.9	2.3.9	2.4.9	2.5.9	3.0.9	3.1.9	3.2.9

Composer – versions



- "**xxxx/yyyy**" : "**~2.3**"

1.2.0	1.3.0	1.4.0	2.0.0	2.1.0	2.2.0	2.3.0	2.4.0	2.5.0	3.0.0	3.1.0	3.2.0
1.2.1	1.3.1	1.4.1	2.0.1	2.1.1	2.2.1	2.3.1	2.4.1	2.5.1	3.0.1	3.1.1	3.2.1
1.2.2	1.3.2	1.4.2	2.0.2	2.1.2	2.2.2	2.3.2	2.4.2	2.5.2	3.0.2	3.1.2	3.2.2
1.2.3	1.3.3	1.4.3	2.0.3	2.1.3	2.2.3	2.3.3	2.4.3	2.5.3	3.0.3	3.1.3	3.2.3
1.2.4	1.3.4	1.4.4	2.0.4	2.1.4	2.2.4	2.3.4	2.4.4	2.5.4	3.0.4	3.1.4	3.2.4
1.2.5	1.3.5	1.4.5	2.0.5	2.1.5	2.2.5	2.3.5	2.4.5	2.5.5	3.0.5	3.1.5	3.2.5
1.2.6	1.3.6	1.4.6	2.0.6	2.1.6	2.2.6	2.3.6	2.4.6	2.5.6	3.0.6	3.1.6	3.2.6
1.2.7	1.3.7	1.4.7	2.0.7	2.1.7	2.2.7	2.3.7	2.4.7	2.5.7	3.0.7	3.1.7	3.2.7
1.2.8	1.3.8	1.4.8	2.0.8	2.1.8	2.2.8	2.3.8	2.4.8	2.5.8	3.0.8	3.1.8	3.2.8
1.2.9	1.3.9	1.4.9	2.0.9	2.1.9	2.2.9	2.3.9	2.4.9	2.5.9	3.0.9	3.1.9	3.2.9

Composer – versions



- "**xxxx/yyyy**" : "**~2.3.2**"

1.2.0	1.3.0	1.4.0	2.0.0	2.1.0	2.2.0	2.3.0	2.4.0	2.5.0	3.0.0	3.1.0	3.2.0
1.2.1	1.3.1	1.4.1	2.0.1	2.1.1	2.2.1	2.3.1	2.4.1	2.5.1	3.0.1	3.1.1	3.2.1
1.2.2	1.3.2	1.4.2	2.0.2	2.1.2	2.2.2	2.3.2	2.4.2	2.5.2	3.0.2	3.1.2	3.2.2
1.2.3	1.3.3	1.4.3	2.0.3	2.1.3	2.2.3	2.3.3	2.4.3	2.5.3	3.0.3	3.1.3	3.2.3
1.2.4	1.3.4	1.4.4	2.0.4	2.1.4	2.2.4	2.3.4	2.4.4	2.5.4	3.0.4	3.1.4	3.2.4
1.2.5	1.3.5	1.4.5	2.0.5	2.1.5	2.2.5	2.3.5	2.4.5	2.5.5	3.0.5	3.1.5	3.2.5
1.2.6	1.3.6	1.4.6	2.0.6	2.1.6	2.2.6	2.3.6	2.4.6	2.5.6	3.0.6	3.1.6	3.2.6
1.2.7	1.3.7	1.4.7	2.0.7	2.1.7	2.2.7	2.3.7	2.4.7	2.5.7	3.0.7	3.1.7	3.2.7
1.2.8	1.3.8	1.4.8	2.0.8	2.1.8	2.2.8	2.3.8	2.4.8	2.5.8	3.0.8	3.1.8	3.2.8
1.2.9	1.3.9	1.4.9	2.0.9	2.1.9	2.2.9	2.3.9	2.4.9	2.5.9	3.0.9	3.1.9	3.2.9

Composer – versions



- "**xxxx/yyyy**" : "**^2.3**"

1.2.0	1.3.0	1.4.0	2.0.0	2.1.0	2.2.0	2.3.0	2.4.0	2.5.0	3.0.0	3.1.0	3.2.0
1.2.1	1.3.1	1.4.1	2.0.1	2.1.1	2.2.1	2.3.1	2.4.1	2.5.1	3.0.1	3.1.1	3.2.1
1.2.2	1.3.2	1.4.2	2.0.2	2.1.2	2.2.2	2.3.2	2.4.2	2.5.2	3.0.2	3.1.2	3.2.2
1.2.3	1.3.3	1.4.3	2.0.3	2.1.3	2.2.3	2.3.3	2.4.3	2.5.3	3.0.3	3.1.3	3.2.3
1.2.4	1.3.4	1.4.4	2.0.4	2.1.4	2.2.4	2.3.4	2.4.4	2.5.4	3.0.4	3.1.4	3.2.4
1.2.5	1.3.5	1.4.5	2.0.5	2.1.5	2.2.5	2.3.5	2.4.5	2.5.5	3.0.5	3.1.5	3.2.5
1.2.6	1.3.6	1.4.6	2.0.6	2.1.6	2.2.6	2.3.6	2.4.6	2.5.6	3.0.6	3.1.6	3.2.6
1.2.7	1.3.7	1.4.7	2.0.7	2.1.7	2.2.7	2.3.7	2.4.7	2.5.7	3.0.7	3.1.7	3.2.7
1.2.8	1.3.8	1.4.8	2.0.8	2.1.8	2.2.8	2.3.8	2.4.8	2.5.8	3.0.8	3.1.8	3.2.8
1.2.9	1.3.9	1.4.9	2.0.9	2.1.9	2.2.9	2.3.9	2.4.9	2.5.9	3.0.9	3.1.9	3.2.9

Composer – versions



- "**xxxx/yyyy**" : "**^2.3.2**"

1.2.0	1.3.0	1.4.0	2.0.0	2.1.0	2.2.0	2.3.0	2.4.0	2.5.0	3.0.0	3.1.0	3.2.0
1.2.1	1.3.1	1.4.1	2.0.1	2.1.1	2.2.1	2.3.1	2.4.1	2.5.1	3.0.1	3.1.1	3.2.1
1.2.2	1.3.2	1.4.2	2.0.2	2.1.2	2.2.2	2.3.2	2.4.2	2.5.2	3.0.2	3.1.2	3.2.2
1.2.3	1.3.3	1.4.3	2.0.3	2.1.3	2.2.3	2.3.3	2.4.3	2.5.3	3.0.3	3.1.3	3.2.3
1.2.4	1.3.4	1.4.4	2.0.4	2.1.4	2.2.4	2.3.4	2.4.4	2.5.4	3.0.4	3.1.4	3.2.4
1.2.5	1.3.5	1.4.5	2.0.5	2.1.5	2.2.5	2.3.5	2.4.5	2.5.5	3.0.5	3.1.5	3.2.5
1.2.6	1.3.6	1.4.6	2.0.6	2.1.6	2.2.6	2.3.6	2.4.6	2.5.6	3.0.6	3.1.6	3.2.6
1.2.7	1.3.7	1.4.7	2.0.7	2.1.7	2.2.7	2.3.7	2.4.7	2.5.7	3.0.7	3.1.7	3.2.7
1.2.8	1.3.8	1.4.8	2.0.8	2.1.8	2.2.8	2.3.8	2.4.8	2.5.8	3.0.8	3.1.8	3.2.8
1.2.9	1.3.9	1.4.9	2.0.9	2.1.9	2.2.9	2.3.9	2.4.9	2.5.9	3.0.9	3.1.9	3.2.9



Créer un projet Symfony – Symfony 4

```
$ composer create-project symfony/skeleton mon_projet
```

Crée le projet Symfony à la dernière version stable dans le répertoire « mon_projet »

```
$ composer create-project symfony/skeleton:^4.0 mon_projet
```

Crée le projet Symfony à la version 4.x.y (cf. règles sur les versions)

```
$ php bin/console --version
```

En se plaçant dans le répertoire où Symfony est installé, permet d'afficher la version courante

```
$ composer require requirements-checker
```

```
$ sh vendor/bin/requirements-checker
```

En se plaçant dans le répertoire où Symfony est installé, permet d'afficher la version courante. Puis, on peut supprimer le composant requirements-checker :

```
$ composer remove requirements-checker
```

Créer un projet Symfony – Symfony 5



Prérequis : installation du binaire « **symfony** »

\$ **symfony check:requirements**

Permet de savoir si votre système permet d'installer Symfony

\$ **symfony new mon_projet_cours_symfony**

Crée le projet Symfony à la dernière version stable dans le répertoire
« mon_projet_cours_symfony »

\$ **php bin/console --version OU symfony console --version**

En se plaçant dans le répertoire où Symfony est installé, permet d'afficher la version courante



Travailler avec un projet existant

- Récupérer le projet existant (ex : avec git) :

```
$ git clone https://mon-serveur-git/mon-project.git
```

```
$ cd my-project/
```

```
$ composer install
```
- ➔ Installe les composants dans les versions du composer.lock
- Mettre à jour Symfony : **\$ composer update**
 - Met à jour les composants du projet dans leurs versions à jour en respectant les règles du composer.json
 - Ne pas oublier de commiter le composer.lock pour le reste de l'équipe ! Il suffira à chacun de faire un **\$ composer install**

Installer un composant



- Vérifier sur Packagist la compatibilité avec son projet



requires

- php: ^5.5.9|^7.0
- symfony/framework-bundle: ~2.7|~3.0|~4.0
- symfony/console: ~2.7|~3.0|~4.0
- symfony/dependency-injection: ~2.7|~3.0|~4.0
- doctrine/dbal: ^2.5.12
- jdorn/sql-formatter: ~1.1
- symfony/doctrine-bridge: ~2.7|~3.0|~4.0
- doctrine/doctrine-cache-bundle: ~1.2

- Puis, dans le répertoire du projet :
\$ composer require doctrine/doctrine-bundle
- Ou :
\$ composer req doctrine/doctrine-bundle
- Afficher les dépendances directes de Composer installées (celles présentes dans composer.json) :
\$ composer info -D

Intégrer une lib à Symfony si elle n'est pas sur Packagist



- Déposer la lib dans `vendor` (elle doit respecter le standard PSR-4)
 - Modifiez le `.gitignore` si votre lib doit être versionnée
- Dire à Composer d'intégrer votre lib dans le calcul de l'autoload (dans `composer.json`):

```
{
    ...
    "autoload": {
        "psr-4": {
            "App\\": "src/",
            "MyLib": "path/to/my/lib"
        }
    },
    ...
}
```

- Enfin, mettre à jour l'autoload : **\$ composer dump-autoload**
- Attention : NE JAMAIS modifier le fichier **`vendor/composer/autoload_namespaces.php`** car il peut être écrasé à n'importe quel moment par Composer

Symfony Flex

- Composer ne sait travailler que dans le répertoire « vendor »
- Flex s'appuie sur composer mais permet en plus d'automatiser l'intégration de composants dans Symfony
 - Il permet de créer des répertoires et fichiers en dehors de vendor (ex : configuration, routing, ...)
 - Techniquement, Flex surcharge le mécanisme « install » et « update » de composer
- Pour pouvoir utiliser Flex, il faut que le package Flex soit intégré au projet
- Installer Flex sur un projet existant :
\$ composer req symfony/flex

Symfony Flex

- L'installation de symfony/skeleton embarque le composant Flex + le strict minimum (routing, kernel, ...)
- C'est grâce à Flex qu'un projet Symfony installé initialement est presque « vide ». Avant Flex, il y avait tous les packages Symfony y compris ceux dont on a potentiellement pas besoin...
- Pour ajouter des composants à un projet Flex, on utilise les recipes
 - Un recipe contient les fichiers et répertoires à déployer pour un composant
 - Le travail du recipe est d'automatiser l'intégration d'un composant
 - Fichier manifest.json : description des actions du recipe pour installation
 - Désinstallation d'un recipe : capacité à revenir à l'état initial en supprimant tout fichier / répertoire ajouté

Symfony Flex – Les recipes

- github.com/symfony/recipes : recipes officiels
- github.com/symfony/recipes-contrib : recipes de la communauté
- flex.symfony.com : serveur des recipes Symfony
 - Contient les recipes de la communauté et les recipes officiels

Symfony Flex – Exemple avec twig

```
$ composer req twig
```

- Il existe un recipe pour lequel l'alias est « twig »

- Contenu d'un recipe pour exemple

Branch: master ▾ recipes / symfony / twig-bundle / 3.3 /

garak unalign	
..	
config	unalign
templates	Removing unnecessary closing /
manifest.json	removes templating as an alias (#219)

- Zoom sur le manifest.json du recipe

```
1 {
2     "bundles": {
3         "Symfony\\Bundle\\TwigBundle\\TwigBundle": ["all"]
4     },
5     "copy-from-recipe": {
6         "config/": "%CONFIG_DIR%",
7         "templates/": "templates/"
8     },
9     "aliases": ["twig", "template", "templates"]
10 }
```

Symfony Recipes Server

Search: twig

Symfony Flex is the way to manage Symfony applications.

It is based on **Symfony Recipes**, which are a set of automated instructions to integrate third-party packages into Symfony applications.

This page lists these great building blocks for your Symfony applications.

Learn More

- Using recipes in your Symfony applications
- Create a recipe for a public package
- Source code for official recipes, community recipes and Symfony Flex itself

12 recipes found

basster/twig-base64-extension contrib Package details Recipe	drupol/ecl-twig-bundle contrib Package details Recipe	kerox/twig-image-placeholder-extension contrib Package details Recipe	nicohaase/twig-isbn-formatter contrib Package details Recipe
ravenflux/env-in-twig contrib Package details Recipe	symfony/twig-bridge official Aliases twig-bridge Package details	symfony/twig-bundle official Aliases twig-bundle Package details Recipe	symfony/twig-pack official Aliases template templates twig twig-pack Package details Recipe

TP – Installation de Symfony



- Installer les composants suivants :
 - PHP : <https://windows.php.net/download/> (prendre la version ZIP) → Ajouter le chemin vers PHP à la variable d'environnement « Path » si besoin (voir slide suivante)
 - Git : <https://git-scm.com/downloads> → conseil : utiliser ensuite la console fournie par Git bash à la place de la console Windows native
 - Composer : <https://getcomposer.org/>
 - PHPStorm : <https://www.jetbrains.com/phpstorm/download/> + plugin Symfony (voir paramétrage dans les slides suivantes)
 - Symfony afin de disposer de la commande « **symfony** »
- Via la commande symfony :
 - Vérifier que tout est OK via la commande : **symfony local:php:list** → Symfony doit retrouver le PHP que vous avez installé
 - Vérifier que votre système est compatible avec Symfony
 - Créer votre premier projet
- Lancer votre serveur local ! (pratique, mais uniquement pour développer → pas de production avec)
 - Toujours avec la commande « **symfony** » → **symfony serve**
 - Et dans votre navigateur : <http://127.0.0.1:8000/>
- Ouvrir le projet avec PHPStorm et activer le plugin Symfony
- Et installez votre premier recipe annotations qui vous sera très utile ;-)
\$ composer req annotations

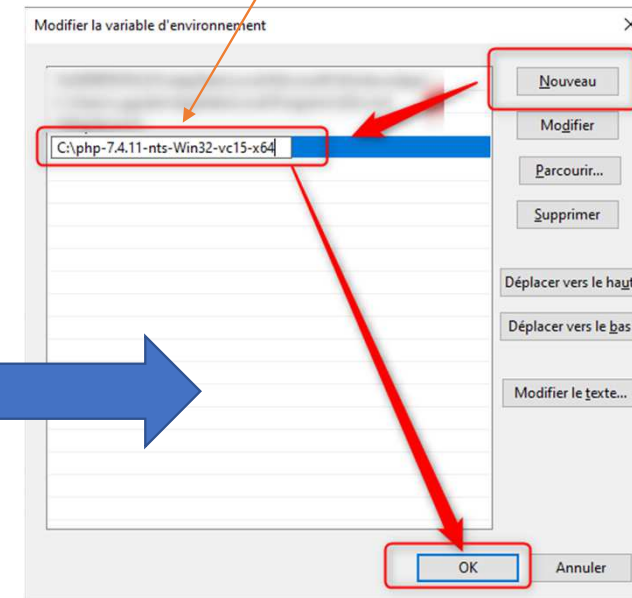
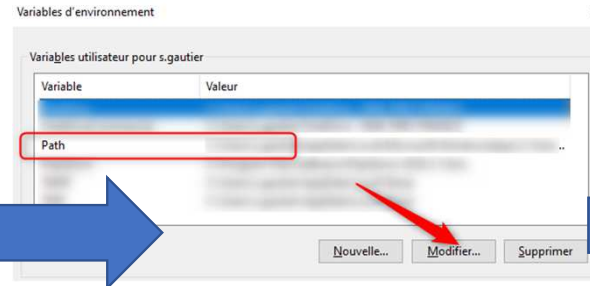
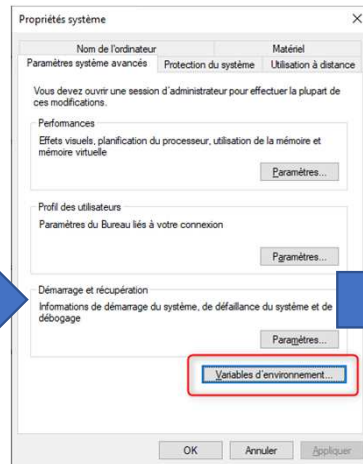
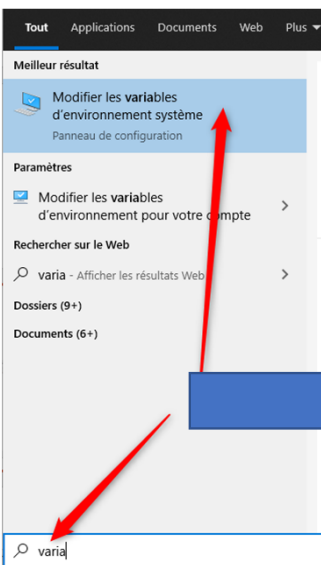


TP – Installation de Symfony

- Focus sur l'ajout du chemin vers PHP dans le Path



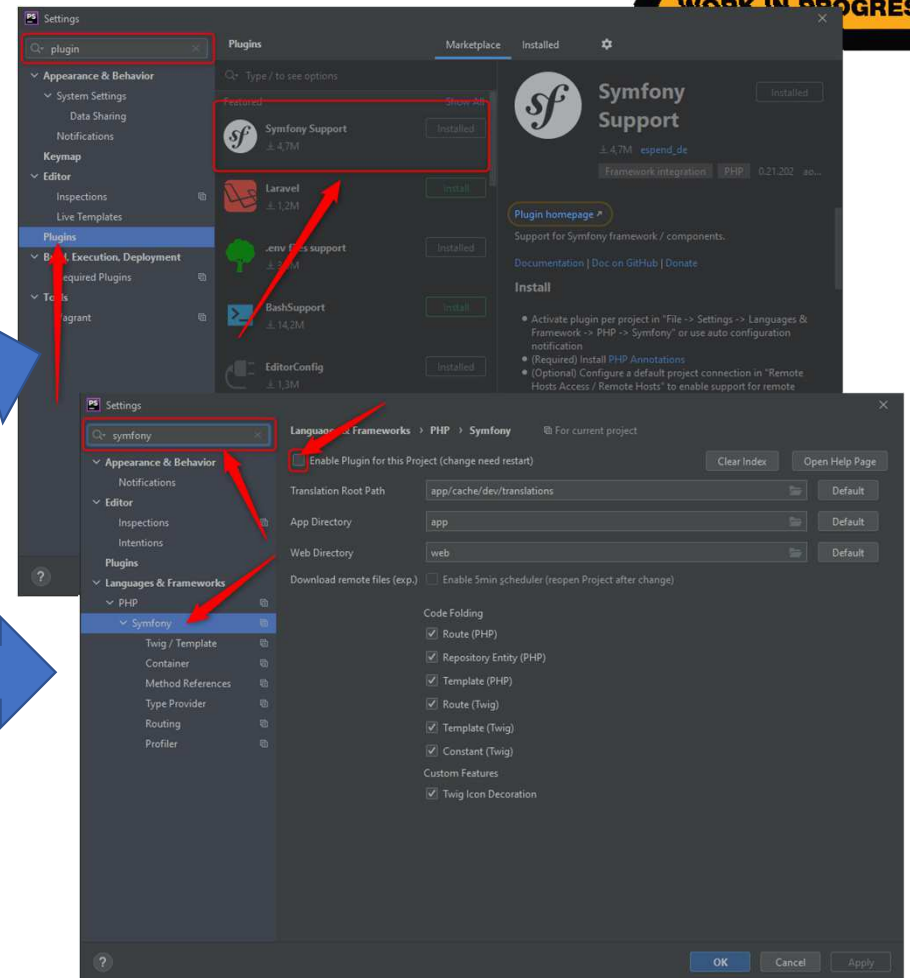
Saisir le chemin où l'archive PHP a été décompressée (contient le binaire php.exe)



TP – Paramétrage de PHPStorm

Gestion de l'extension Symfony

- Activer l'extension Symfony sur chaque projet Symfony sur lequel vous travaillez !
- L'extension est censée être installée de base dans PHPStorm





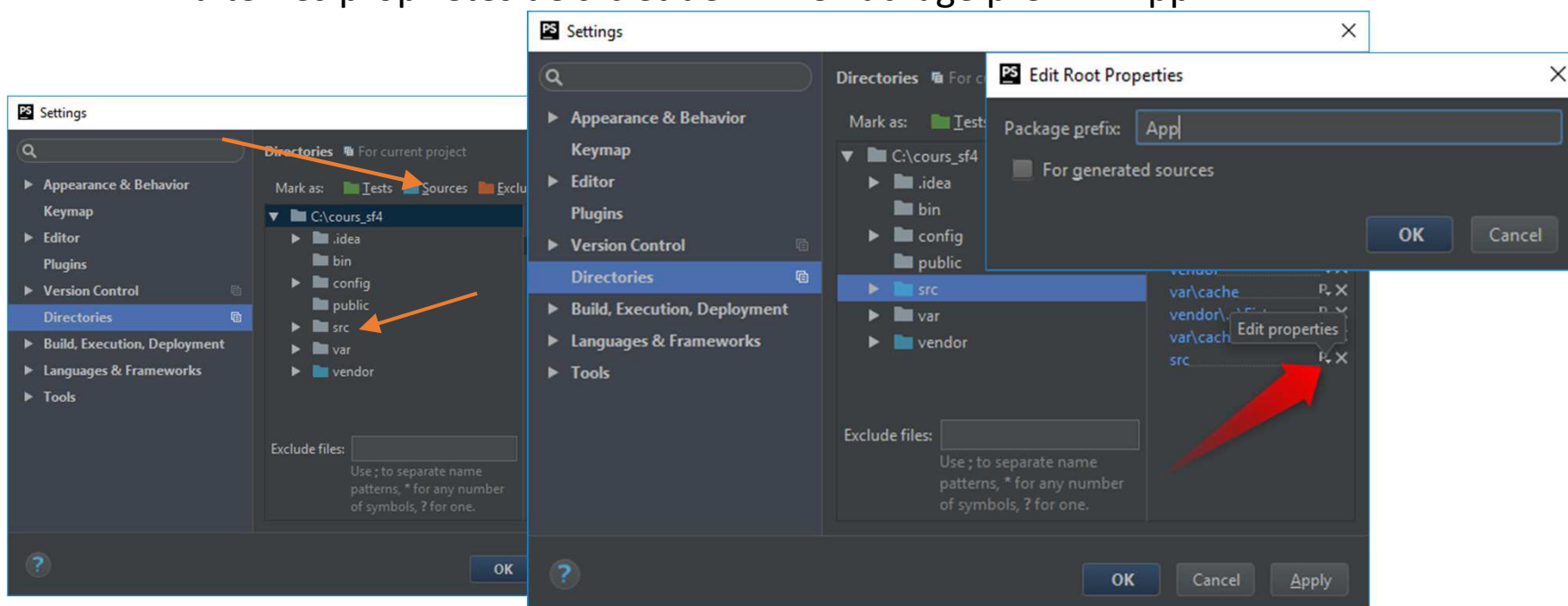
ARCHIVE

Inutile dans les versions récentes de PhpStorm
à vérifier tout de même !

TP – Paramétrage de PhpStorm



- Editer les paramètres du projet > menu « Directories »
- Puis, sélectionner le répertoire src et le définir comme « Sources »
- Editer les propriétés de src et définir le Package prefix « App »





Simon GAUTIER



Architecture

Répertoires de base de Symfony

Configuration

Variables d'environnement / Environnements

Répertoires de base de Symfony

<mon-projet>

└ **bin/**

└ **config/**

└ **public/**

└ **src/**

└ **var/**

└ **vendor/**

Répertoire **config**

<mon-projet>

```
└─ config/
   │ bootstrap.php
   │ bundles.php
   │ services.yaml
   │ routes.yaml
   └─ packages/
      │ framework.yaml
      │ dev/
      │ prod/
      └─ test/
```

- Contient la configuration principale de Symfony (bundles.php, services.yaml, routes.yaml)
- Contient la configuration de chaque dépendance installée (répertoire packages)
 - Noter que les configurations peuvent varier d'un environnement à l'autre (répertoires dev, prod, test) → cf. environnements

Répertoire **var**

<mon-projet>

└ **var/**

└ **cache/**

└ **log/**

└ **sessions/**

- Répertoire technique qui contient les fichiers générés par Symfony (fichiers volatiles) :
 - Cache
 - Logs
 - Sessions
- Pour redéfinir ces répertoires, modifier le fichier `src/Kernel.php` ➔ cf. symfony.com/doc/current/configuration/override_dir_structure.html

Répertoire **src**

<mon-projet>

└ **src/**

└ **Kernel.php**

└ **Controller/**

└ **Entity/**

└ **...**

- Contient le fichier Kernel.php utilisé par Symfony pour initialiser l'application
- Contient tout le code de votre projet !
- Important : depuis Symfony 3.4, il n'y a plus de découpage en bundles dans src

Répertoire **vendor**

<mon-projet>

└ **vendor/**

└ **bin/**

└ **composer/**

└ **symfony/**

└ ...

- Contient toutes les dépendances de votre projet (bundles ou librairies)
- Rappel : c'est composer qui gère ce répertoire
➔ ne jamais y modifier de code
- Ne pas hésiter à aller lire du code pour comprendre comment fonctionne le framework

Répertoire **public**

<mon-projet>

└ **public/**

└ **index.php**

└ **images/**

└ **css/**

└ **js/**

└ **...**

- Contient les seuls fichiers censés être accessibles publiquement :
 - index.php
 - assets
- Bonne pratique : la configuration du serveur doit définir ce répertoire comme le seul accessible (sécurité)

Répertoire **bin**

- Contient les « exécutables » utiles aux développements
- Utilisables avec la commande PHP
- De base : **console** ➔ très utile !
- D'autres binaires peuvent être présents en fonction des bundles installés

Configuration avec Symfony

- Symfony supporte de base les formats de configuration suivants :
 - Via des fichiers : YAML, XML, PHP
 - Via le code : les annotations ➔ utilisation des commentaires `/** ... */`
- Important : quelque soit le mode choisi, les performances restent les mêmes car Symfony précalcule un cache en PHP
- Recommandations de l'éditeur (nous verrons cela au fil des cours) :
 - Annotations : routing, sécurité, persistance et validation de données
 - L'utilisation des annotation nécessite dans certains cas l'installation d'un package dédié (que nous avons déjà installé 😊) :
\$ composer req annotations
 - YAML / XML : pour les services et les options de configuration
 - PHP : dans des cas très précis (ex : configuration conditionnelle)

Variables d'environnement : gestion des fichiers **.env**

- **.env** (versionné) : contient les valeurs par défaut des variables d'environnement utilisées par votre application
- **.env.local** (non versionné) : surcharges liées à l'installation courante de votre application
- **.env.\$APP_ENV** (ex : « .env.dev », versionné) : contient les valeurs des variables d'environnement par défaut pour l'environnement concerné
- **.env.\$APP_ENV.local** (ex : « .env.dev.local, non versionné) : surcharges liées à l'installation courante de votre application pour l'environnement concerné


- Extrait du fichier **.env** :

```
...  
###> symfony/framework-bundle ###  
APP_ENV=dev  
APP_DEBUG=1  
APP_SECRET=ea6ab9fc150d103166262aefcbd70770  
###< symfony/framework-bundle ###  
...
```

Variables d'environnement : utilisation dans un fichier de configuration

- Exemple avec le fichier config/packages/doctrine.yaml (si Doctrine a été installé)
 - Important : l'évaluation de la variable est faite uniquement à l'exécution

```
doctrine:
  dbal:
    url: '%env(resolve:DATABASE_URL)%'
  orm:
    auto_generate_proxy_classes: true
    naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware
    auto_mapping: true
    mappings:
      App:
        is_bundle: false
        type: annotation
        dir: '%kernel.project_dir%/src/Entity'
        prefix: 'App\Entity'
        alias: App
```



Environnements

- Chaque environnement est différent en fonction des variables d'environnement qui lui sont associées
 - En production, la performance est privilégiée
 - En développement, il est important de pouvoir visualiser les logs
- Symfony permet d'exécuter le même projet dans des environnements différents
 - Pour Symfony, un environnement est défini par une chaîne de caractères arbitraire (ex : « dev » ou « prod » ou « preprod », ...)
 - C'est dans le fichier public/index.php que Symfony prend en compte l'environnement courant

Environnements

- Configuration : focus sur le répertoire config/packages :
 - Comme évoqué, il est possible d'y définir le paramétrage de chaque dépendance installée
 - Les répertoires config/packages/[NOM_ENVIRONNEMENT] permettent d'avoir des configurations différentes en fonction de l'environnement
- Exemple : logger les requêtes jouées par Doctrine uniquement en dev
- Le nom du répertoire sous config/packages est le nom de l'environnement

Environnements

<your-project>

└ **config/**

└ **packages/**

└ **doctrine.yaml**

└ **dev/**

└ **doctrine.yaml**

└ **prod/**

└ **doctrine.yaml**

Configuration utilisée par défaut pour Doctrine

Configuration utilisée pour Doctrine uniquement en dev

Configuration utilisée pour Doctrine uniquement en prod

➔ Cf. \App\Kernel::configureContainer()



Simon GAUTIER



Votre premier projet important : Hello World !

Hello World simple / Hello world avec Twig

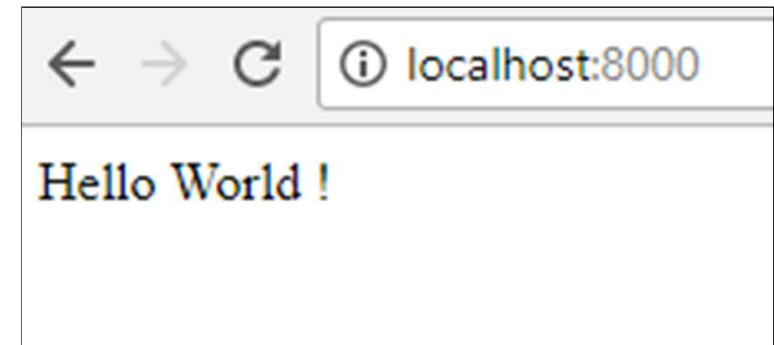
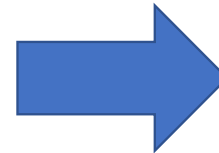
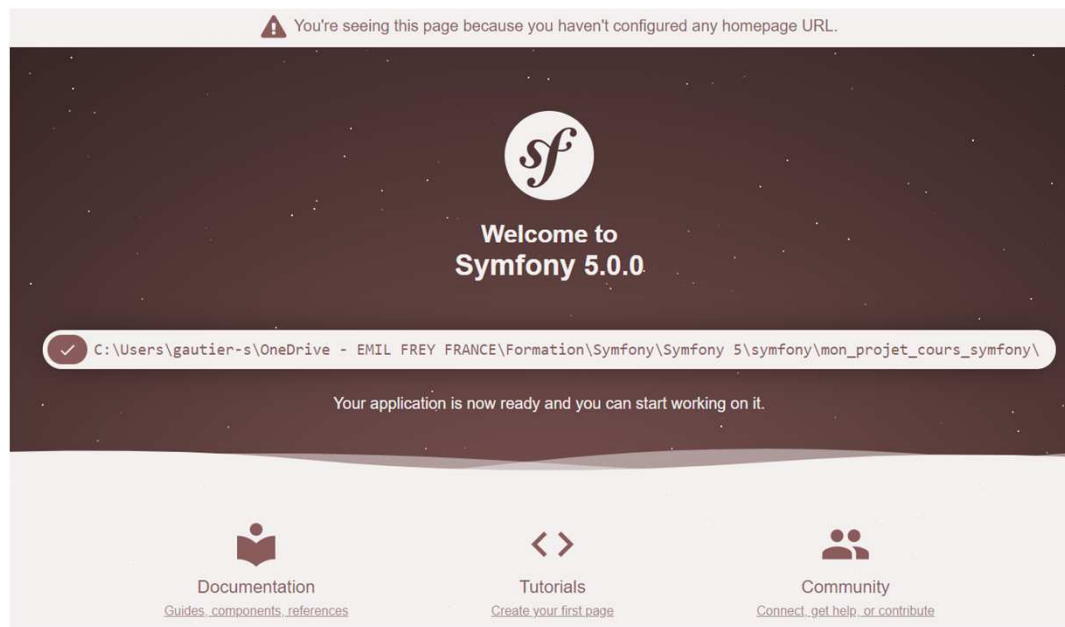
Configuration serveur : bonnes pratiques

Requêtes et réponses HTTP avec Symfony

Routage d'URLs avec Symfony

Les contrôleurs

Hello World – Objectif



Hello World – La solution

```
<?php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class HelloWorldController
{
    /**
     * @Route("/")
     */
    public function helloWorldAction()
    {
        return new Response('Hello World !');
    }
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/HelloWorldController.php

Tester : <https://cours-symfony.argetis.com/>

Pour la route « / », retourner la réponse HTTP contenant le texte brut « Hello World ! »

Par convention :

- Les contrôleurs sont nommés **XxxxController**
- Les actions sont nommées **YyyyAction**

Hello World – Version twig

- Installer Twig : `$ composer req twig`
- Créer un template dans le répertoire « templates » de Symfony :

https://github.com/sgautier/cours_symfony/blob/master/templates/hello_world.html.twig

Hello World Twig !



URL à tester : http..../**twig**

```
<?php
namespace App\Controller;

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

class HelloWorldTwigController extends AbstractController
{
    /**
     * @Route("/twig")
     */
    public function helloWorldAction()
    {
        return $this->render('hello_world.html.twig');
    }
}
```

Hériter du contrôleur standard de Symfony

Faire appel au template

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/HelloWorldTwigController.php

Tester : <https://cours-symfony.argetis.com/twig>

Hello World – Version twig



Je vous en présente souvent, n'oubliez pas de les installer vous aussi !

- Installer Twig : \$ **composer req twig**
- Créer un template dans le répertoire « templates » de Symfony :

https://github.com/sgautier/cours_symfony/blob/master/templates/hello_world.html.twig

Hello World Twig !



URL à tester : http..../**twig**

```
<?php
namespace App\Controller;

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

class HelloWorldTwigController extends AbstractController
{
    /**
     * @Route("/twig")
     */
    public function helloWorldAction()
    {
        return $this->render('hello_world.html.twig');
    }
}
```

Hériter du contrôleur standard de Symfony

Faire appel au template

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/HelloWorldTwigController.php

Tester : <https://cours-symfony.argetis.com/twig>

Configuration serveur – Travailler en local

- Utiliser le serveur intégré à Symfony
- Utiliser Docker
- Utiliser une machine virtuelle



Pour aller plus loin

Configuration serveur

```
<VirtualHost *:80>
  ServerName mon-site-symfony.loc
  DocumentRoot
    "/chemin/vers/mon/projet/public"
  DirectoryIndex index.php
  <Directory
    "/chemin/vers/mon/projet/public">
    AllowOverride None
    Allow from All
  </Directory>
  <IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php [QSA,L]
  </IfModule>
</VirtualHost>
```



Apache

Bonnes pratiques :

- « public » est le seul répertoire qui doit être accessible
- Ne pas permettre l'utilisation des fichiers .htaccess



Pour aller plus loin

Configuration serveur



```
server {  
    server_name mon-site-symfony.loc;  
    root /chemin/vers/mon/projet/public;  
  
    location / {  
        try_files $uri /index.php$is_args$args;  
    }  
  
    error_log /var/log/nginx/mon-projet-symfony_error.log;  
    access_log /var/log/nginx/mon-projet-  
    symfony_access.log;  
}
```

Bonnes pratique :

- « public » est le seul répertoire qui doit être accessible



Pour aller plus loin

Gestion des permissions – Problématique

- Le user Web (souvent **www-data**) a besoin :
 - De lire l'ensemble des fichiers de l'arborescence
 - D'écrire dans **var/cache** et dans **var/log**
- Un utilisateur de la machine (ex : **symfony**) a besoin en parallèle :
 - De lire l'ensemble des fichiers de l'arborescence
 - D'écrire dans **var/cache** et dans **var/log**
- Comment faire cohabiter les 2 besoins ? Si par exemple **www-data** crée un fichier, comment s'assurer que **symfony** peut le modifier ?



Pour aller plus loin

Gestion des permissions – Solution risquée

Dans **bin/console**

- Utiliser **umask ()** pour forcer le masque des fichiers créés :

```
#!/usr/bin/env php
<?php

use App\Kernel;
use Symfony\Bundle\FrameworkBundle\Console\Application;
use Symfony\Component\Console\Input\ArgvInput;
use Symfony\Component\Debug\Debug;
use Symfony\Component\Dotenv\Dotenv;

// 0000 correspond à des fichiers créés en 777 sur le serveur, donc accessibles par tous
umask(0000);

set_time_limit(0);

require __DIR__.'/../vendor/autoload.php';

// ...
```



Pour aller plus loin

Gestion des permissions – Solution risquée

Dans **public/index.php**

- Utiliser **umask ()** pour forcer le masque des fichiers créés :

```
<?php

use App\Kernel;
use Symfony\Component\Debug\Debug;
use Symfony\Component\Dotenv\Dotenv;
use Symfony\Component\HttpFoundation\Request;

require __DIR__.'../vendor/autoload.php';

// 0000 correspond à des fichiers créés en 777 sur le serveur, donc accessibles par tous
umask(0000);

// ...
```



Pour aller plus loin

Permissions

Zoom sur la fonction **umask()**

```
<?php
// Affiche le masque par défaut au démarrage du script
var_dump("Masque par défaut :");
var_dump(umask()); // En décimal
var_dump(decoct(umask())); // En octal
// Crée le fichier "default.txt" avec le masque par défaut
file_put_contents('default.txt', 'Le contenu...');

// Change le masque en 0000
umask(0000);
// Affiche le masque par défaut au démarrage du script
var_dump("Masque 0000 :");
var_dump(umask()); // En décimal
var_dump(decoct(umask())); // En octal
// Crée le fichier "0000.txt" avec le masque par défaut
file_put_contents('0000.txt', 'Le contenu...');

// Change le masque en 0777
umask(0777);
var_dump("Masque 0777 :");
var_dump(umask()); // En décimal
var_dump(decoct(umask())); // En octal
// Crée le fichier "0777.txt" avec le masque par défaut
file_put_contents('0777.txt', 'Le contenu...');
```

```
$ php toto.php
```

```
string(20) "Masque par défaut :"  
int(18)  
string(2) "22"  
string(13) "Masque 0000 :"  
int(0)  
string(1) "0"  
string(13) "Masque 0777 :"  
int(511)  
string(3) "777"
```

```
$ ls -l *.txt
```

```
-rw-rw-rw- 1 root root 13 févr. 17 14:52 0000.txt  
----- 1 root root 13 févr. 17 14:52 0777.txt  
-rw-r--r-- 1 root root 13 févr. 17 14:52 default.txt
```

Cf. doc : <http://php.net/manual/fr/function.umask.php>



Pour aller plus loin

Permissions – Bonne pratique

- Changer le user du serveur web afin qu'il soit le même que celui de la machine qui exécute la console Symfony
- Avec Apache, fichier [...] **/conf/httpd.conf** :
User symfony
Group team
- Avec Nginx, fichiers [...] **/nginx.conf** et [...] **/php-fpm.conf** :
user symfony
group team



Pour aller plus loin

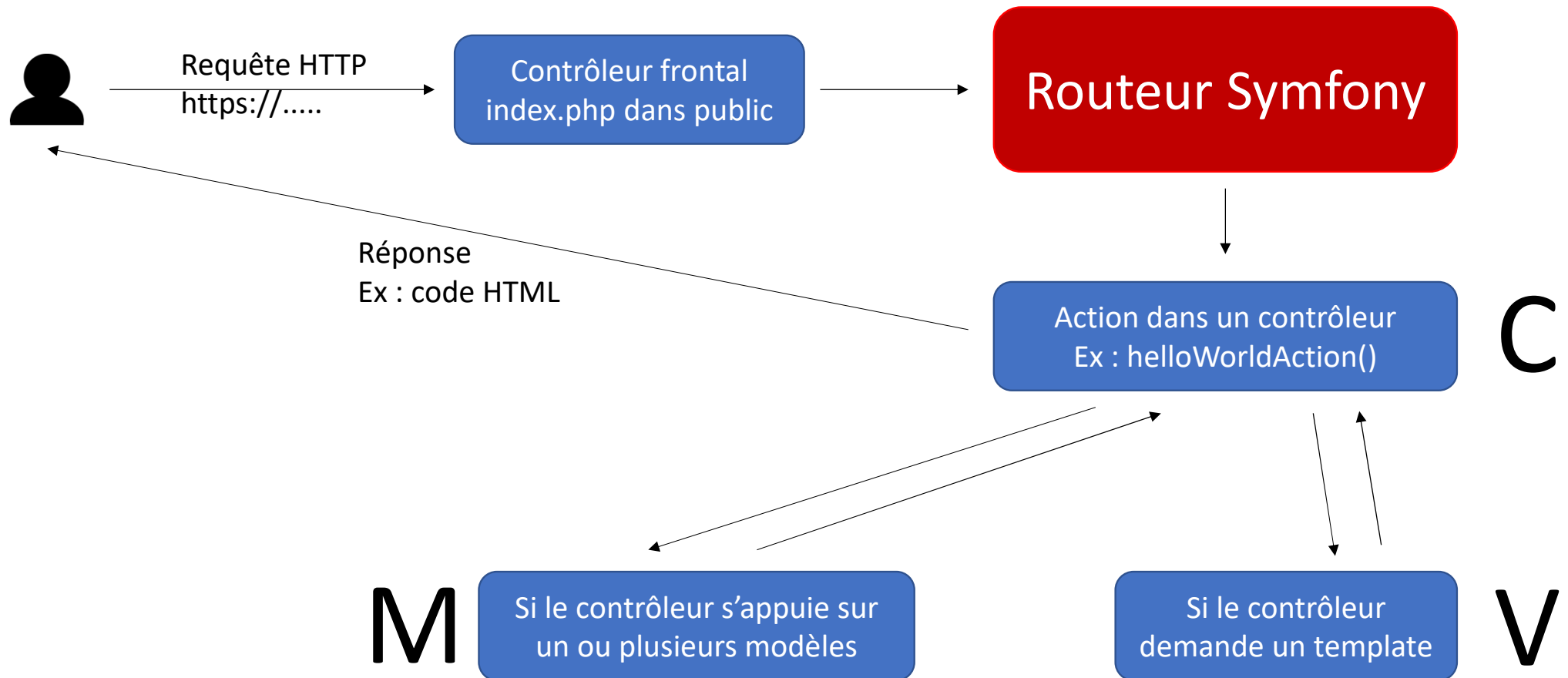
Permissions – Bonne pratique

Alternatives

- Solution 1 : mettre l'utilisateur **symfony** dans le groupe du serveur web **www-data**
- Solution 2 : utiliser setfacl pour autoriser les 2 utilisateurs à écrire dans les répertoires concernés ➔ cf. doc de setfacl

Routeur Symfony

Illustration du modèle MVC



Routeur Symfony

- Le routeur Symfony a 2 missions :
 - Savoir quel contrôleur / action appeler à partir d'une URL
 - Générer une URL à partir d'une route
- Définitions possibles dans Symfony :
 - Via les annotations ➔ solution recommandée car les routes sont dans les mêmes fichiers que les actions des contrôleurs
 - YAML ➔ anciennement la solution recommandée
 - XML ➔ trop verbeux
 - PHP ➔ trop bas niveau

Exemple de route

- Sans la définition du name (qui doit être unique), Symfony attribue une route construite à partir du nom du contrôleur / de l'action → cf. profiler
- **IMPORTANT** : installer le profiler → `$ composer req profiler`

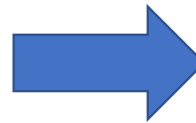


```
<?php
namespace App\Controller;

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

class RouteTestController extends AbstractController
{
    /**
     * @Route("/route/test", name="nom-de-ma-route")
     */
    public function testAction()
    {
        return new Response('<body>Contenu de la réponse</body>');
    }
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/RouteTestController.php

A screenshot of the Symfony Profiler interface. The top bar shows the URL 'https://127.0.0.1:8000/route/test' and the method 'GET'. The left sidebar has a menu with 'Routing' selected. The main content area shows the route 'nom-de-ma-route' as a 'Matched route'. Below this, the 'Route Parameters' section is empty. The 'Route Matching Logs' section shows a table with the following data:

#	Route name	Path	Log
1	_preview_error	/_error/{code}.{_format}	Path does not match
2	_wdt	/_wdt/{token}	Path does not match
3	nom-de-ma-route	/route/test	Path does not match

Tester : <https://cours-symfony.argetis.com/route/test>

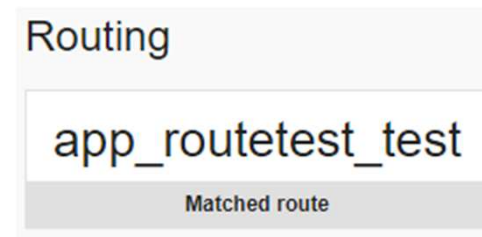
Nom de la route

- Sans nom explicite, un nom est créé par Symfony → non recommandé car on a besoin du nom pour créer des URLs !
- Remarque : le contenu de la réponse doit contenir « </body> » pour que le profiler puisse fonctionner

```
/**  
 * @Route("/route/test", name="nom-de-ma-route")  
 */
```



```
/**  
 * @Route("/route/test")  
 */
```



Routes avec variables

```
<?php
namespace App\Controller;

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

class RouteTestController extends AbstractController
{
    /**
     * @Route("/route/with-variable/{id}", name="nom-de-ma-route-2")
     */
    public function routeWithVariableAction($id)
    {
        // $id est accessible ici de manière automatique
        return new Response("<body>$id</body>");
    }
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/RouteTestController.php

Tester : <https://cours-symfony.argetis.com/route/with-variable/25>

Routes avec variables et valeurs par défaut

Solution 1

```
<?php
namespace App\Controller;

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

class RouteTestController extends AbstractController
{
    /**
     * @Route(
     *     "/route/with-variable-and-default-value/{page}",
     *     defaults={"page": "1"},
     *     name="nom-de-ma-route-3"
     * )
     */
    public function withDefaultValuesAction($page)
    {
        return new Response("<body>$page</body>");
    }
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/RouteTestController.php

Tester : <https://cours-symfony.argetis.com/route/with-variable-and-default-value>

Routes avec variables et valeurs par défaut

Solution 2

```
<?php
namespace App\Controller;

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

class RouteTestController extends AbstractController
{
    /**
     * @Route(
     *     "/route/with-variable-and-default-value-bis/{page}",
     *     name="nom-de-ma-route-4"
     * )
     */
    public function withDefaultValuesBisAction($page=1)
    {
        return new Response("<body>$page</body>");
    }
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/RouteTestController.php

Tester : <https://cours-symfony.argetis.com/route/with-variable-and-default-value-bis>

Routes avec contraintes

Contrainte sur une variable

```
<?php
namespace App\Controller;

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

class RouteTestController extends AbstractController
{
    /**
     * @Route(
     *     "/route/with-constraint/{page}",
     *     defaults={"page": "1"},
     *     requirements={"page": "\d+"},
     *     name="nom-de-ma-route-5"
     * )
     */
    public function withConstraintAction($page)
    {
        return new Response("<body>$page</body>");
    }
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/RouteTestController.php

Tester : <https://cours-symfony.argetis.com/route/with-constraint/25>

Routes avec contraintes

Contrainte sur plusieurs variables

```
/**
 * @Route(
 *     "/route/with-multiple-constraints/{year}/{month}/{filename}.{extension}",
 *     defaults={
 *         "extension": "html"
 *     },
 *     requirements={
 *         "year": "\d{4}",
 *         "month": "\d{2}",
 *         "extension": "html|xml|css|js"
 *     },
 *     name="nom-de-ma-route-6"
 * )
 */
public function withMultipleConstraintsAction($year, $month, $filename, $extension)
{
    return new Response("<body>$year, $month, $filename, $extension</body>");
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/RouteTestController.php

Tester : <https://cours-symfony.argetis.com/route/with-multiple-constraints/2020/11/test.html>

- Toutes les URLs définies comme suit matchent cette route :
 - /my-page/[0-9]{4}/[0-9]{2}/*.html|xml|css|js
 - OU /my-page/[0-9]{4}/[0-9]{2}/*

Routes avec contraintes

Contrainte sur la méthode HTTP

```
<?php
namespace App\Controller;

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

class RouteTestController extends AbstractController
{
    /**
     * @Route(
     *     "/route/with-http-method-constraint/{page}",
     *     defaults={"page": "1"},
     *     requirements={"page": "\d+"},
     *     methods={"GET"},
     *     name="nom-de-ma-route-7")
     */
    public function withHttpMethodConstraintAction($page)
    {
        return new Response("<body>$page</body>");
    }
}
```

Tester : <https://cours-symfony.argetis.com/route/with-http-method-constraint/25>

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/RouteTestController.php

Autoriser plusieurs méthodes : `methods={"GET", "POST"}`

Préfixe sur toutes les routes d'un contrôleur

```
/**
 * Class RouteTestBisController
 * @package App\Controller
 *
 * @Route("/route-bis")
 */
class RouteTestBisController extends AbstractController
{
    /**
     * @Route("/test", name="nom-de-ma-route-bis")
     * En réalité, la route de cette action sera /route-bis/test
     */
    public function testAction()
    {
        return new Response('<body>Contenu de la réponse</body>');
    }

    /**
     * @Route("/with-variable/{id}", name="nom-de-ma-route-2-bis")
     * En réalité, la route de cette action sera /route-bis/with-variable/{id}
     */
    public function routeWithVariableAction($id)
    {
        // $id est accessible ici de manière automatique
        return new Response("<body>$id</body>");
    }

    // ...
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/RouteTestBisController.php

Tester : <https://cours-symfony.argetis.com/route-bis/test>
<https://cours-symfony.argetis.com/route-bis/with-variable/25>



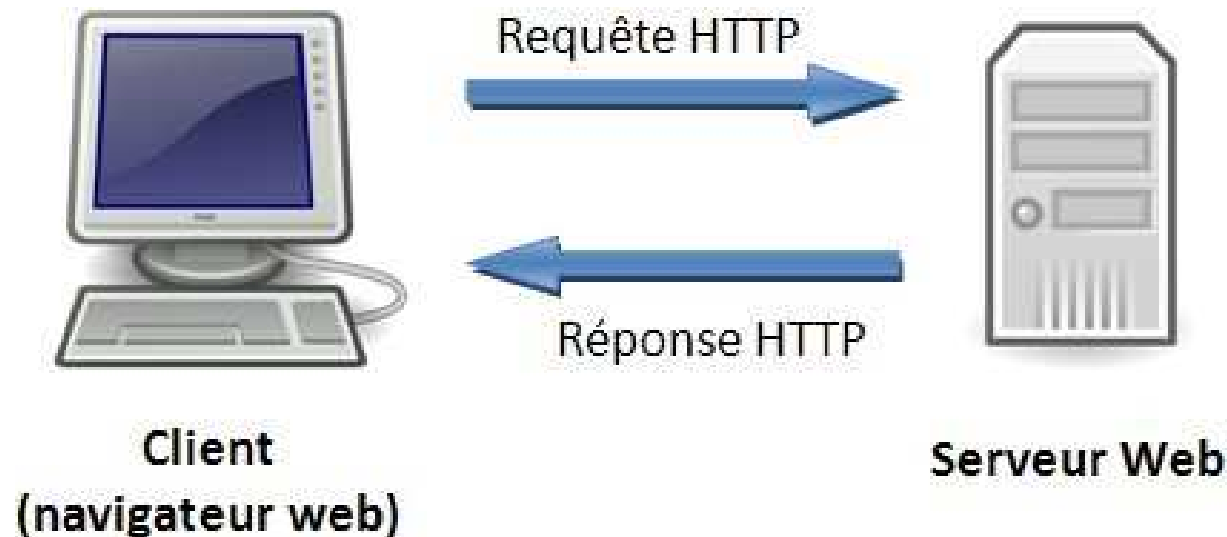
Routes – aller plus loin

symfony.com/doc/current/routing.html

- Voir les équivalents des annotations en YAML / XML / PHP
- Découvrir les variables spéciales :
 - `_controller`
 - `_format`
 - `_fragment`
 - `_locale`
- ...

Les contrôleurs – Concepts

- Symfony utilise les concepts du protocole HTTP
 - En entrée : une requête (objet Request)
 - En sortie : une réponse (objet Response)



Les contrôleurs – Exemple de requête

The screenshot shows a web browser at `localhost:8000/hello` displaying a page with the text "Hello !" and "Ma page Hello world !". The page source code is visible, showing an HTML document with a title "Hello World !". The Network tab in the developer tools shows a list of requests, with "hello" selected. The "Response" tab is active, displaying the request details. Red boxes and arrows highlight specific parts of the network request and response.

Page Content:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Hello World !</title>
5 </head>
6 <body>
7   <h1>Hello !</h1>
8   <p>
9     Ma page Hello world !
10  </p>
11 </body>
12 </html>
```

Network Request Details:

- Request URL:** `http://localhost:8000/hello`
- Request Method:** GET
- Status Code:** 200 OK
- Remote Address:** `:::1]:8000`
- Referrer Policy:** no-referrer-when-downgrade

Response Headers:

- Cache-Control: max-age=0, private, must-revalidate
- Cache-Control: no-cache, private
- Connection: close
- Content-Type: text/html; charset=UTF-8
- Date: Wed, 10 Jan 2018 13:02:57 +0000
- Date: Wed, 10 Jan 2018 13:02:57 GMT
- Host: localhost:8000
- X-Debug-Token: b4a6d7
- X-Debug-Token-Link: `http://localhost:8000/_profiler/b4a6d7`
- X-Powered-By: PHP/7.1.11

Request Headers:

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
- Accept-Encoding: gzip, deflate, br
- Accept-Language: fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
- Cache-Control: max-age=0
- Connection: keep-alive
- Host: localhost:8000
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36

Contrôleurs – Objet Request

Les paramètres « hors route »

- Exemple : comment récupérer toto dans l'url suivante :

<https://localhost:8000/hello?toto=123> →

```
public function helloWorldBisAction(Request $request)
{
    dump($request->query->get('toto'));
    return new Response('Hello World !');
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/HelloWorldController.php

Tester : <https://cours-symfony.argetis.com/hello?toto=123>

- Tous les types de paramètres :

- `$_GET` → `$request->query->get('...')` → permet également de récupérer les paramètres de la route
- `$_POST` → `$request->request->get('...')`
- `$_COOKIE` → `$request->cookies->get('...')`
- `$_SERVER` → `$request->server->get('...')`
- `$_SERVER['HTTP_...']` → `$request->headers->get('...')`
→ Récupérer les variables d'entête

- Récupérer le tableau complet : méthode `all()`

- Exemple : tous les paramètres GET : `$request->query->all()`

Contrôleurs – Objet Request

Quelques méthodes utiles

- Récupérer la méthode de la requête HTTP

```
$request->getMethod()
```

- Savoir si la requête est en POST

```
if ($request->isMethod('POST')) { /* Requête POST */ }
```

- Savoir si la requête est une requête Ajax

```
if ($request->isXmlHttpRequest()) { /* Requête Ajax */ }
```

- Et toutes les autres ! ➔ voir documentation officielle

Contrôleurs – Objet Response

Décomposition d'une réponse

The image illustrates the relationship between a Symfony controller action and the HTTP response it generates. On the left, a code snippet shows the `my404Action()` method, which creates a `Response` object, sets its status to `HTTP_NOT_FOUND`, and sets its content to "Ma page 404". Two orange arrows point from the code to the browser interface on the right: one from the `@Route("/my-404", name="my_404")` annotation to the browser's address bar, and another from the `$response->setContent('Ma page 404');` line to the browser's page content.

```
/**
 * @Route("/my-404", name="my_404")
 * @return Response
 */
public function my404Action()
{
    $response = new Response(); // Initialisation de la réponse
    $response->setStatusCode(Response::HTTP_NOT_FOUND); // Code de la réponse
    $response->setContent('Ma page 404'); // Contenu de la réponse
    return $response; // Retour de la réponse
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Tester : <https://cours-symfony.argetis.com/my-404>

The browser interface on the right shows the URL `localhost:8000/my-404` and the page content "Ma page 404". The Network tab displays a list of requests, with `my-404` selected. The `General` tab for this request shows the following details:

- Request URL: `http://localhost:8000/my-404`
- Request Method: `GET`
- Status Code: `404 Not Found`
- Remote Address: `[::1]:8000`
- Referrer Policy: `no-referrer-when-downgrade`
- Response Headers:
 - Cache-Control: `max-age=0, private, must-revalidate`
 - Cache-Control: `no-cache, private`
 - Connection: `close`

Contrôleurs – déclencher une erreur 404

- Le contrôleur peut avoir besoin de déclencher une erreur 404 en fonction de tests applicatifs

```
/**
 * @Route("/display-results/{page}", defaults={"page": 1}, requirements={"page": "\\d+"}, name="display_results")
 */
public function displayResultsAction($page)
{
    // La page DOIT être un entier strictement positif (les requirements s'assurent uniquement du fait que la
    // variable est composée de chiffres)
    if($page < 1) {
        throw new NotFoundException("La page $page n'existe pas");
    }

    return new Response("<body>Affichage de la page $page</body>");
}
```

Tester : <https://cours-symfony.argetis.com/display-results/toto>
<https://cours-symfony.argetis.com/display-results/0>
<https://cours-symfony.argetis.com/display-results/25>

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php



- Les 2 URLs suivantes génèrent une erreur 404 mais la seconde est générée au niveau du contrôleur par le développeur
 - <http://localhost:8000/display-results/toto>
 - <http://localhost:8000/display-results/0>

Contrôleurs – Objet Response

Redirections

- Nécessité dans certains cas (ex : action CRUD) de rediriger après traitement vers une page de résultats
- Une redirection est une réponse HTTP comme une autre
 - Il est possible de passer des paramètres (second argument : tableau associatif PHP)
- 3 façons de faire équivalentes dans notre contrôleur :

```
public function testRedirectAction()  
{  
    return new RedirectResponse($this->get('router')->generate('hello'));  
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

→ Ne pas oublier dans ce cas la routine suivante :

Tester : <https://cours-symfony.argetis.com/test-redirect>

use Symfony\Component\HttpFoundation\RedirectResponse;

```
public function testRedirectBisAction()  
{  
    return $this->redirect($this->get('router')->generate('hello'));  
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Tester : <https://cours-symfony.argetis.com/test-redirect-bis>

```
public function testRedirectTerAction()  
{  
    return $this->redirectToRoute('hello');  
}
```

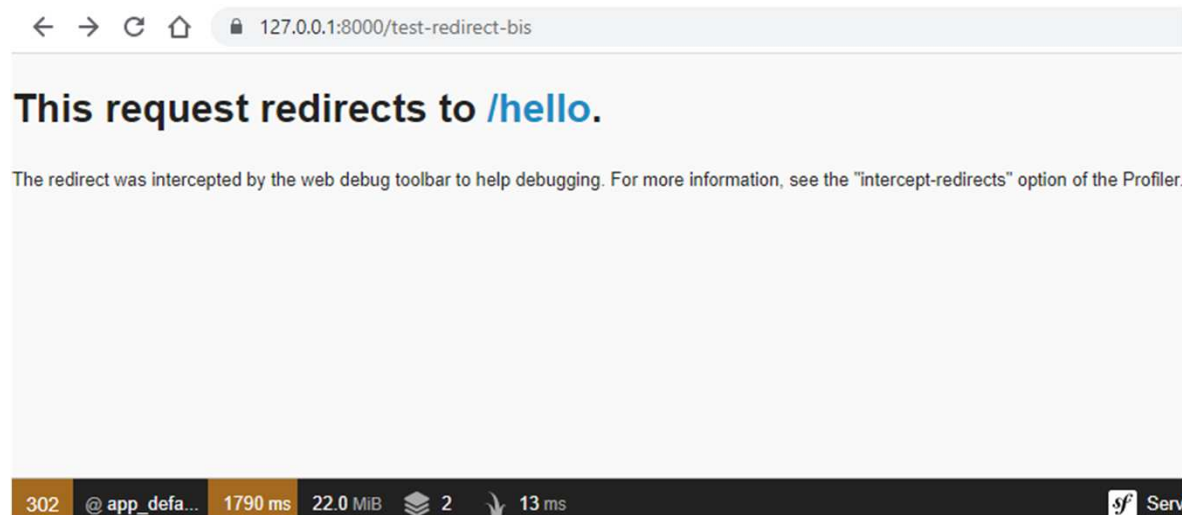
https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Tester : <https://cours-symfony.argetis.com/test-redirect-ter>

Contrôleurs – Objet Response

Capter les redirections

- Passer dans « **config/packages/dev/web_profiler.yaml** » le paramètre « **intercept_redirects** » à **true** et tester de nouveau la redirection
 - ➔ Symfony ne génère pas l'entête de redirection mais affiche une page indiquant vers quelle destination la redirection va se faire. Intérêt : le Profiler Symfony nous permet d'avoir la trace complète de cette redirection



Contrôleurs – Objet Response

Changer le content-type

- Exemple : retourner un contenu JSON ➔ 2 méthodes équivalentes :

```
public function sendJsonAction()
{
    // Ne pas oublier dans ce cas le use nécessaire : Symfony\Component\HttpFoundation\JsonResponse;
    return new JsonResponse(['toto' => 'titi']);
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Tester : <https://cours-symfony.argetis.com/test-response-json>

```
public function sendJsonAction()
{
    // Initialisation de la réponse avec le contenu attendu : du JSON
    $response = new Response(json_encode(['toto' => 'titi']));

    // Définition du Content-Type
    $response->headers->set('Content-Type', 'application/json');

    // Retour de la réponse
    return $response;
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Tester : <https://cours-symfony.argetis.com/test-response-json-bis>

Résultat identique
dans les 2 cas :

▼ Response Headers [view source](#)
Cache-Control: max-age=0, private, must-revalidate
Cache-Control: no-cache, private
Connection: close
Content-Type: application/json

Contrôleurs – La session

Etape préliminaire : s'assurer que les sessions sont actives (dans `config/packages/framework.yaml`) :

```
framework:
# ...
session: # La ligne ne doit pas être commentée
```

2 solutions équivalentes

```
public function testSessionAction(Request $request)
{
    // Récupération de la session
    $session = $request->getSession();

    // Accès à une variable de la session
    // Uniquement à la première exécution, variable vide (cf. instruction suivante)
    dump($session->get('toto'));

    // Définition d'une variable en session
    // Une fois l'action exécutée au moins une fois, cette ligne peut être retirée et le dump()
    // ci-dessus continuera à afficher la valeur 1234 (tant que la session ne sera pas détruite)
    $session->set('toto', 1234);

    return new Response('<body></body>');
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Tester : <https://cours-symfony.argetis.com/test-session>

```
public function testSessionBisAction(Session $session)
{
    // Autre manière de récupérer la session : paramètre du contrôleur
    dump($session->get('toto'));
    $session->set('toto', 1234);
    return new Response('<body></body>');
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Tester : <https://cours-symfony.argetis.com/test-session-bis>

Utiliser le profiler pour voir le contenu de la session :

The screenshot shows the Symfony Profiler interface. The top bar indicates the URL `http://localhost:8000/session`, method `GET`, status `200`, and IP `::1`. The left sidebar has a menu with items like 'Request / Response', 'Performance', 'Forms', 'Exception', 'Logs', 'Events', 'Routing', 'Cache', 'Translation', and 'Security'. The 'Request / Response' tab is selected and highlighted with a red box. The main content area shows the 'DefaultController :: testSessionAction' action. It has tabs for 'Request', 'Response', 'Cookies', 'Session', and 'Flashes'. The 'Session' tab is selected and highlighted with a red box. Below the tabs, there is a 'Session Metadata' table with columns 'Key' and 'Value'. The table contains three rows: 'Created' with value 'Sat, 13 Jan 18 15:55:18 +0000', 'Last used' with value 'Sat, 13 Jan 18 15:55:23 +0000', and 'Lifetime' with value '0'. Below this is a 'Session Attributes' table with columns 'Attribute' and 'Value'. It contains one row: 'toto' with value '1234'. This table is also highlighted with a red box.

Key	Value
Created	"Sat, 13 Jan 18 15:55:18 +0000"
Last used	"Sat, 13 Jan 18 15:55:23 +0000"
Lifetime	"0"

Attribute	Value
toto	1234

Contrôleurs – La session

Utiliser les messages flash

- Exemple : afficher un message de confirmation d'un traitement en particulier (suite à une action CRUD)
- A l'actualisation de la page de destination, le message disparaît
 - ➔ Twig : la variable `{{ app }}` est une variable globale qui permet entre autre d'accéder à la session

```
/**
 * @Route("/test-flash")
 * @param Session $session
 * @return Response
 */
public function testFlashCrudAction(Session $session)
{
    // ... faire un traitement particulier

    // valeur 'info' libre : c'est le développeur qui choisit ses étiquettes
    $session->getFlashBag()->add('info', "Tout s'est bien passé :-");

    // Méthode équivalente :
    $this->addFlash('info', "Tout s'est vraiment bien passé :-");

    return $this->redirectToRoute('test_flash_result');
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Tester : <https://cours-symfony.argetis.com/test-flash>

```
/**
 * @Route("/test-flash-result", name="test_flash_result")
 * @param Session $session
 * @return void
 */
public function testFlashResultAction(Session $session)
{
    return $this->render('test_flash_bag.html.twig');
}
```

https://github.com/sgautier/cours_symfony/blob/master/src/Controller/DefaultController.php

Twig : petit spoiler du chapitre suivant :

```
{% for msg in app.session.flashBag.get('info') %}
<p>{{ msg }}</p>
{% endfor %}
```

https://github.com/sgautier/cours_symfony/blob/master/templates/test_flash_bag.html.twig

TP – Contrôleur et routes



- Créer un contrôleur avec les actions suivantes :
 - **listAction** : action permettant de lister les articles du blog
 - URL : /blog/list/xxx → prévoir un paramètre de numéro de page (entier) et facultatif (page 1 par défaut) !
 - Test au niveau du code la valeur du numéro de page (doit être un entier strictement positif)
 - **viewAction** : consultation d'un article. URL : /blog/article/xxx (où xxx est un entier)
 - **addAction** : action CRUD d'ajout d'un article (si on est en train de valider le formulaire, cf. prochain chapitre) suivie d'une redirection vers la liste, affichage formulaire sinon. URL : /blog/article/add
 - **editAction** → même principe que **addAction** mais redirection vers la consultation d'un article. URL : /blog/article/edit/xxx (où xxx est un entier)
 - **deleteAction** → action CRUD de suppression d'un article qui fait une redirection vers la page de liste après suppression. URL : /blog/article/delete/xxx (où xxx est un entier)
- Utiliser « blog » comme préfixe à toutes les routes du contrôleur
- Prévoir d'ores et déjà les messages flash dans le cas de la validation d'un ajout, d'une modification, d'une suppression
- Pour le moment :
 - Pas d'ajout / modification / suppression effectifs (notions vues plus tard)
 - Pas de templates Twig (créés au prochain chapitre) → retourner des objets **Response ()**