Symfony

Développez votre site web avec le framework Symfony

## Symfony, un framework PHP

#### En résumé

* Le mot « framework » signifie « cadre de travail » en français.
* L'objectif principal d'un framework est d'améliorer la productivité des développeurs qui l'utilisent.
* Contrairement aux CMS, un framework est destiné à des développeurs, et non à des novices en informatique.
* L'apprentissage d'un framework est un investissement : il y a un certain effort à fournir au début, mais les résultats se récoltent ensuite sur le long terme !
* Symfony est un framework PHP très populaire, français, et très utilisé dans le milieu des entreprises.

## Vous avez dit Symfony ?

#### En résumé

* Symfony est organisé en six répertoires : app, bin , src, var , vendor et web.
* Le répertoire dans lequel on passera le plus de temps est src, il contient le code source de notre site.
* Il existe deux environnements de travail :
  + L'environnement « prod » est destiné à vos visiteurs : il est rapide à exécuter, et ne divulgue pas les messages d'erreur.
  + L'environnement « dev » est destiné au développeur, c'est-à-dire vous : il est plus lent, mais offre plein d'informations utiles au développement.
* Symfony utilise l'architecture MVC pour bien organiser les différentes parties du code source.
* Un bundle est une brique de votre application : il contient tout ce qui concerne une fonctionnalité donnée. Cela permet de bien organiser les différentes parties de votre site.
* Il existe des milliers de bundles développés par la communauté, pensez à vérifier qu'il n'existe pas déjà un bundle qui fait ce que vous souhaitez faire !

## Utilisons la console pour créer un bundle

#### À retenir

Ce qu'il faut retenir de tout cela, c'est que pour qu'un bundle soit opérationnel, il faut :

* Son code source, situé dans src/Application/Bundle, et dont le seul fichier obligatoire est la classe à la racine OCPlatformBundle.php ;
* Enregistrer le bundle dans le noyau pour qu'il soit chargé, en modifiant le fichier app/AppKernel.php ;
* Enregistrer les routes (si le bundle en contient) dans le Routeur pour qu'elles soient chargées, en modifiant le fichier app/config/routing.yml.

Ces trois points sont bien sûr effectués automatiquement lorsqu'on utilise le générateur. Mais vous pouvez tout à fait créer un bundle sans l'utiliser, et il faudra alors remplir cette petite checklist manuellement.

Par la suite, tout notre code source sera situé dans des bundles. Un moyen très propre de bien structurer son application.

#### En résumé

* Les commandes Symfony disponibles en ligne de commande ont pour objectif de nous faciliter la vie en automatisant certaines tâches.
* Les commandes sont faites, comme tout Symfony, en PHP uniquement. La console n'est qu'un moyen différent du navigateur pour exécuter du code PHP.
* La commande pour générer un nouveau bundle est php bin/console generate:bundle.
* Le code du cours tel qu'il doit être à ce stade est disponible sur [la branche iteration-1](https://github.com/winzou/mooc-symfony/tree/iteration-1) du dépot Github.

## Mon premier « Hello World ! » avec Symfony

#### En résumé

* Le rôle du routeur est de déterminer quel route utiliser pour la requête courante.
* Le rôle d'une route est d'associer une URL à une action du contrôleur.
* Le rôle du contrôleur est de retourner au noyau un objet Response, qui contient la réponse HTTP à envoyer à l'internaute (page HTML ou redirection).
* Le rôle des vues est de mettre en forme les données que le contrôleur lui donne, afin de construire une page HTML, un flux RSS, un e-mail, etc.
* Le code du cours tel qu'il doit être à ce stade est disponible sur la branche [iteration-2](https://github.com/winzou/mooc-symfony/tree/iteration-2) du dépôt Github.

## Le routeur de Symfony

#### Convention pour le nom du contrôleur

Vous l'avez vu, lorsque l'on définit le contrôleur à appeler dans la route, il y a une convention à respecter : la même que pour appeler un template (nous l'avons vue au chapitre précédent). Un rappel ne fait pas de mal : lorsque vous écrivez « OCPlatformBundle:Advert:view », vous avez trois informations :

* « OCPlatformBundle » est le nom du bundle dans lequel aller chercher le contrôleur. En terme de fichier, cela signifie pour Symfony2 : « Va voir dans le répertoire de ce bundle. ». Dans notre cas, Symfony2 ira voir danssrc/OC/PlatformBundle.
* « Advert » est le nom du contrôleur à ouvrir. En terme de fichier, cela correspond àController/AdvertController.phpdans le répertoire du bundle. Dans notre cas, nous avons comme chemin absolusrc/OC/PlatformBundle/Controller/AdvertController.php.
* « view » est le nom de l'action à exécuter au sein du contrôleur. Attention, lorsque vous définissez cette méthode dans le contrôleur, vous devez la faire suivre du suffixe « Action », comme ceci :public function viewAction().

#### Pour conclure

Ce chapitre est terminé, et vous savez maintenant tout ce qu'il faut savoir sur le routeur et les routes.

Retenez que ce système de routes vous permet premièrement d'avoir des belles URL, et deuxièmement de découpler le nom de vos URL du nom de vos contrôleurs. Ajoutez à cela la génération d'URL, et vous avez un système extrêmement flexible et maintenable.

Le tout sans trop d'efforts !

Pour plus d'informations sur le système de routes, n'hésitez pas à lire la [documentation officielle](https://symfony.com/fr/doc/current/book/routing.html).

#### En résumé

* Une route est composée au minimum de deux éléments : l'URL à faire correspondre (son path), et le contrôleur à exécuter (paramètre \_controller).
* Le routeur essaie de faire correspondre chaque route à l'URL appelée par l'internaute, et ce dans l'ordre d'apparition des routes : la première route qui correspond est sélectionnée.
* Une route peut contenir des paramètres, facultatifs ou non, représentés par les accolades {paramètre}, et dont la valeur peut être soumise à des contraintes via la section requirements.
* Le routeur est également capable de générer des URL à partir du nom d'une route, et de ses paramètres éventuels.

## Les contrôleurs avec Symfony

#### 

#### À retenir

##### L'erreur 404

Je vous ai donné un exemple qui vous montre comment déclencher une erreur 404. C'est quelque chose que l'on fera souvent, par exemple dès qu'une annonce n'existera pas, qu'un argument ne sera pas bon (page = 0), etc. Lorsque l'on déclenche cette exception, le noyau l'attrape et génère une belle page d'erreur 404. Vous pouvez aller voir l'annexe « [Comment personnaliser ses pages d'erreur](https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony2/personnaliser-les-pages-d-erreur) ».

##### La définition des méthodes

Nos méthodes vont être appelées par le noyau : elles doivent donc respecter le nom et les arguments que nous avons définis dans nos routes et se trouver dans le scope « public ». Vous pouvez bien entendu rajouter d'autres méthodes, par exemple pour exécuter une fonction que vous réutiliserez dans deux actions différentes. Dans ce cas, vous ne devez pas les suffixer de « Action » (afin de ne pas confondre).

#### Pour conclure

Créer un contrôleur à ce stade du cours n'est pas évident, car vous ne connaissez et ne maîtrisez pas encore tous les services nécessaires. Seulement, vous avez pu comprendre son rôle et voir un exemple concret.

Rassurez-vous, dans la partie 4 du cours, on apprendra tout le nécessaire pour construire l'intérieur de nos contrôleurs.    
En attendant, rendez-vous au prochain chapitre pour en apprendre plus sur les templates.

Pour plus d'informations concernant les contrôleurs, n'hésitez pas à lire la [documentation officielle](https://symfony.com/fr/doc/current/book/controller.html).

#### En résumé

* Le rôle du contrôleur est de retourner un objet Response : ceci est obligatoire !
* Le contrôleur construit la réponse en fonction des données qu'il a en entrée : paramètre de route et objet Request.
* Le contrôleur se sert de tout ce dont il a besoin pour construire la réponse : la base de données, les vues, les différents services, etc.

## Le moteur de templates Twig

#### À savoir

Première chose à savoir sur Twig : vous pouvez afficher des variablesetpouvez exécuter des expressions. Ce n'est pas la même chose :

* {{ … }} affiche quelque chose ;
* {% … %} fait quelque chose ;
* {# … #}  n'affiche rien et ne fait rien : c'est la syntaxe pour les commentaires, qui peuvent être sur plusieurs lignes.

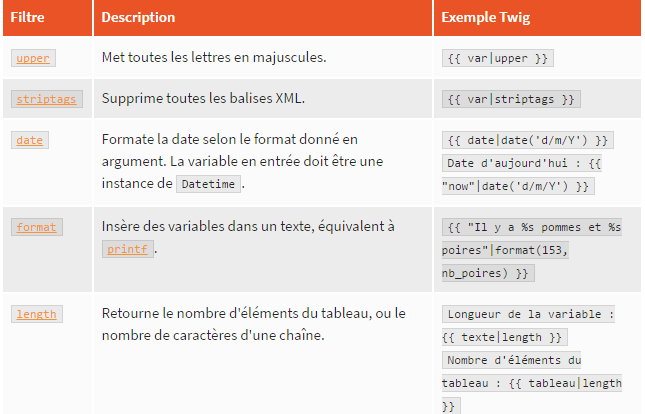
L'objectif de la suite de ce chapitre est donc :

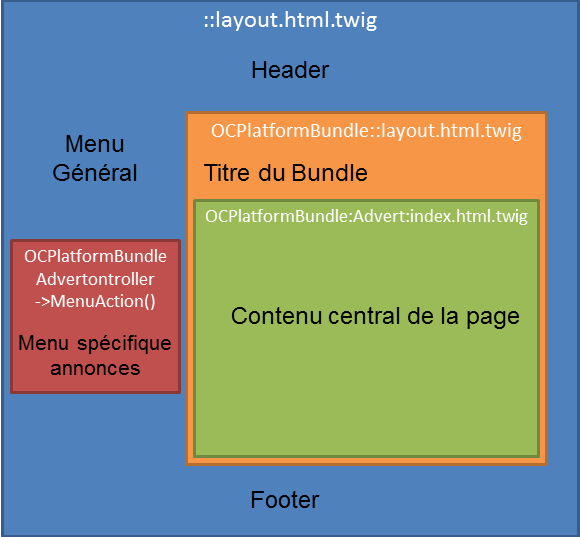


#### Précisions sur la syntaxe {{ objet.attribut }}

Le fonctionnement de la syntaxe {{ objet.attribut }} est un peu plus complexe qu'il n'en a l'air. Elle ne fait pas seulement objet->getAttribut. En réalité, voici ce qu'elle fait exactement :

* Elle vérifie si objet est un tableau, et si attribut en est un index valide. Si c'est le cas, elle affiche objet['attribut'].
* Sinon, et si objet est un objet, elle vérifie si attribut en est un attribut valide (public donc). Si c'est le cas, elle affiche objet->attribut.
* Sinon, et si objet est un objet, elle vérifie si attribut() en est une méthode valide (publique donc). Si c'est le cas, elle affiche objet->attribut().
* Sinon, et si objet est un objet, elle vérifie si getAttribut() en est une méthode valide. Si c'est le cas, elle affiche objet->getAttribut().
* Sinon, et si objet est un objet, elle vérifie si isAttribut() en est une méthode valide. Si c'est le cas, elle affiche objet->isAttribut().
* Sinon, elle n'affiche rien et retourne null.





#### Pour conclure

Et voilà, nous avons créé presque tous nos templates. Bien sûr, ils sont encore un peu vides, car on ne sait pas utiliser les formulaires ni récupérer les annonces depuis la base de données. Cependant vous savez maintenant les réaliser et c'était une étape importante ! Je vais vous laisser créer les templates manquants ou d'autres afin que vous vous fassiez la main. Bon code !

Cela termine ce chapitre : vous savez afficher avec mise en forme le contenu de votre site. Vous avez maintenant presque toutes les billes en main pour réaliser un site internet. Bon, OK, c'est vrai, il vous manque encore des concepts clés tels que les formulaires, la base de données, etc. Néanmoins vous maîtrisez pleinement la base du framework Symfony2, et apprendre ces prochains concepts sera bien plus facile !

Pour plus d'informations concernant Twig et ses possibilités, n'hésitez pas à lire la [documentation officielle](http://twig.sensiolabs.org/documentation).

#### En résumé

* Un moteur de templates tel que Twig permet de bien séparer le code PHP du code HTML, dans le cadre de l'architecture MVC ;
* La syntaxe {{ var }}‌ affiche la variable var ;
* La syntaxe {% if %}  exécute quelque chose, ici une condition ;
* Twig offre un système d'héritage (via {% extends %}) et d'inclusion (via {{ include() }} et {{ render() }}) très intéressant pour bien organiser les templates ;
* Le modèle triple héritage est très utilisé pour des projets avec Symfony.
* Le code du cours tel qu'il doit être à ce stade est disponible sur la branche [iteration-5](https://github.com/winzou/mooc-symfony/tree/iteration-5) du dépot Github.

## Installer un bundle grâce à Composer

#### Comment Composer sait où trouver les bibliothèques ?

Très bonne question. En effet, il est évident que ce système de gestion ne peut fonctionner que si on peut centraliser les informations de chaque bibliothèque. C'est le rôle du site [www.packagist.org](https://www.packagist.org/).

Par exemple, voici la page pour la bibliothèque Symfony (eh oui, c'est une bibliothèque comme une autre !) : <https://packagist.org/packages/symfony/symfony>. Vous pouvez voir les informations comme le mainteneur principal, le site de la bibliothèque, etc. Mais ce qui nous importe le plus, ce sont les sources ainsi que les dépendances (dans Requires).

#### Pour conclure

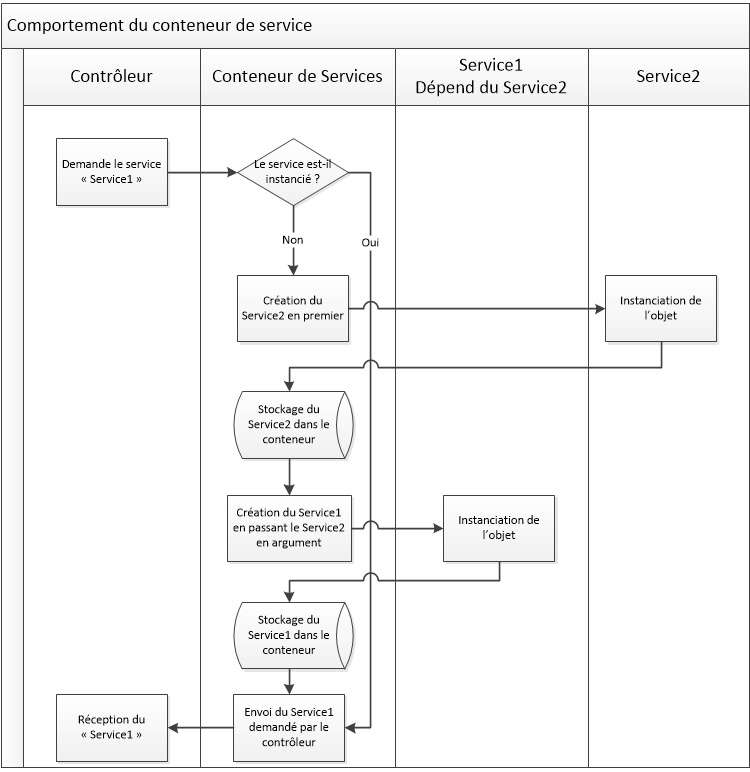
Ce chapitre-parenthèse sur Composer touche à sa fin. S'il vous semble un peu décalé aujourd'hui, vous me remercierez un peu plus tard de vous en avoir parlé, lorsque vous voudrez installer des bundles trouvés à droite ou à gauche. D'ailleurs, on a déjà installé DoctrineFixtureBundle, un bundle bien pratique dont nous nous resservirons dès la prochaine partie sur Doctrine !

Sachez également que je n'ai absolument pas tout dit sur Composer, car cela ferait trop long et ce n'est pas tellement l'objet de ce tutoriel. Cependant, Composer a sa propre documentation et je vous invite à vous y référer. Par curiosité, par intérêt, en cas de problème, n'hésitez pas : [http://getcomposer.org](https://getcomposer.org/) !

#### En résumé

* Composer est un outil pour gérer les dépendances d'un projet en PHP, qu'il soit sous Symfony ou non.
* Le fichier composer.json permet de lister les dépendances que doit inclure Composer dans votre projet.
* Composer détermine la meilleure version possible pour vos dépendances, les télécharge, et configure leur autoload tout seul.
* Composer trouve toutes les bibliothèques sur le site http://www.packagist.org, sur lequel vous pouvez envoyer votre propre bibliothèque si vous le souhaitez.
* La très grande majorité des bundles Symfony sont installables avec Composer, ce qui simplifie énormément leur utilisation dans un projet.
* Le code du cours tel qu'il doit être à ce stade est disponible sur la branche [iteration-6](https://github.com/winzou/mooc-symfony/tree/iteration-6) du dépot Github.

## Les services, théorie et création



#### Pour conclure

Je me permets d'insister sur un point : **les services et leur conteneur** sont l'élément crucial et inévitable de Symfony. Les services sont utilisés intensément par le cœur même du framework, et nous serons amenés à en créer assez souvent dans la suite de ce cours.

Gardez en tête que leur intérêt principal est de bien découpler les fonctions de votre application. Tout ce que vous comptez utiliser à plusieurs endroits dans votre code mérite un service. Gardez vos contrôleurs les plus simples possibles, et n'hésitez pas à créer des services qui contiennent la logique de votre application.

Ce chapitre vous a donc apporté les connaissances nécessaires pour définir et utiliser simplement les services. Bien sûr, il y a bien d'autres notions à voir, mais nous les verrons un peu plus loin dans [un prochain chapitre](https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony/les-services-utilisation-poussee-1).

Si vous souhaitez aborder plus en profondeur les notions théoriques abordées dans ce chapitre, je vous propose les lectures suivantes :

* [Introduction à l'injection de dépendances en PHP (OpenClassrooms)](https://openclassrooms.com/courses/introduction-a-l-injection-de-dependances-en-php) de vincent1870 ;
* [Les design patterns : l'injection de dépendances (OpenClassrooms)](https://openclassrooms.com/courses/programmez-en-oriente-objet-en-php/les-design-patterns#r-1668102) de Victor Thuillier ;
* [Architecture orientée services (Wikipédia).](https://fr.wikipedia.org/wiki/Architecture_orient%C3%A9e_services)

En attendant, la prochaine partie abordera la gestion de la base de données !

#### En résumé

* Un service est une simple classe associée à une certaine configuration.
* Le conteneur de services organise et instancie tous vos services, grâce à leur configuration.
* Les services sont la base de Symfony, et sont très utilisés par le cœur même du framework.
* L'injection de dépendances est assurée par le conteneur, qui connaît les arguments dont a besoin un service pour fonctionner, et les lui donne donc à sa création.
* Le code du cours tel qu'il doit être à ce stade est disponible sur la branche [iteration-7](https://github.com/winzou/mooc-symfony/tree/iteration-7) du dépot Github.

## La couche métier : les entités