**DOCUMENTATION SUR SYMFONY**

**1 – Notion de Framework**

Framework est formé des mots « frame » et « work » et signifie donc « cadre de travail ». Concrètement, il s’agit d’un ensemble de composants servant à créer les fondations, l'architecture et les grandes lignes d'un logiciel. Il a pour but d'améliorer la productivité des développeurs qui l'utilisent.

**a – Etude comparative des différents Framework PHP**

Il existe plusieurs Frameworks php mais cette étude portera sur les plus utilisés :

**CakePHP**

CakePHP est relativement simple à prendre en main comparé à certains autres frameworks PHP tels que Zend, pour ne citer que l’un des plus connus. Il permet le développement rapide d’applications (RAD : Rapid Application Development) et simplifie leur maintenance. Les principes de base de CakePHP sont avant tout sa robustesse et sa simplicité de mise en œuvre quel que soit le type d’application. Il la rend facilement extensible et personnalisable avec l’architecture MVC.

Les principales caractéristiques de CakePHP sont :

* Compatible PHP 4 et PHP 5.
* Pas de configuration nécessaire.
* Utilisation d’une simplicité peu commune.
* Communauté conséquente et active.
* Licence MIT (toute application développée peut être vendue).
* Entièrement conçu et maintenu par l’équipe de développeurs.
* Intègre l’architecture MVC.
* Gestion facilitée des bases de données grâce à Active Record.
* Entièrement Orienté Objet (OO).
* Nombreux outils de développement (ajax, formulaires…).
* Génération d’applications CRUD (Create/Read/Update/Delete)

Son architecture extensible permet de développer et maintenir les applications en optimisant les tâches du développeur et favorisant un gain de productivité non négligeable. Ces avantages sont rendus possibles grâce à l’adoption des systèmes MVC (Modèle-Vue-Contôleur) et ORM (Object-Relational Mapping).  
Framework d’applications rapides (RAD) CakePHP est avant tout de conception robuste. Il prend en charge tous les aspects d’une page web, de la requête initiale à l’interface finale. En appliquant les principes de l’architecture MVC, il permet toute forme de personnalisation et d'extension de chaque application. Reposant sur un concept simple et puissant, une organisation structurée, CakePHP se positionne parmi les meilleurs frameworks et offre aux développeurs toutes les facilités nécessaires à l’élaboration rapide d’applications sérieuses et complexes en leur assurant consistance et logique.

**CodeIgniter**

CodeIgniter est un framework PHP puissant, bien que léger, basé sur le fonctionnement de Ruby on Rails et adoptant l’architecture MVC (Modèle-Vue-Contrôleur). Développé à l’origine pour les propres besoins de l’entreprise EllisLab en 2006, il a depuis adapté le statut open source.  
CodeIgniter est particulièrement simple à utiliser, performant et d’une vitesse d’exécution remarquable. Il est fortement recommandé aux développeurs débutants.

Les principales caractéristiques de CodeIgniter :

* Compatible PHP 4 et PHP 5
* Simplicité de prise en main
* Intégration de l’architecture MVC
* Nombreuses librairies
* Complètement orienté objet
* Importante communauté très active (anglophone en majorité)
* Abondante documentation
* Souplesse et évolution constante

CodeIgniter est un framework PHP qui comme tout framework permet de réduire le temps de développement et d’optimiser le code d’une application.  
Il est particulièrement souple et adapté aux débutants car aisément compréhensible. Moins connu que CakePHP ou Zend, CodeIgniter est maintenu par la Société EllisLab et remarquablement documenté. Son avenir semble prometteur, à en juger par l’importance de sa communauté. CodeIgniter est puissant bien que nécessitant peu de ressources. Il est en constante évolution et son importante collection de librairies contribue largement aux gains de productivité que l’on attend d’un framework, en matière de développement d’applications de toutes natures.. Sa simplicité d’utilisation par rapport à Symfony ou Zend contribue à sa popularité au sein des développeurs PHP.  
CodeIgniter est le framework idéal pour le développement de petites et moyennes applications. En effet il nécessite peu de ressources tout en offrant un maximum de fonctionnalités. Son système de cache lui procure d’excellentes performances et minimise l’impact sur le chargement des pages.

Ses principaux atouts : extrême légèreté, robustesse et fiabilité. Facile à installer et acceptant de multiples environnements, CodeIgniter à tout pour séduire.  
  
Selon un test réalisé il y a quelques mois par Richard Goutorbe, consultant informatique à Montpellier, l’utilisation de CodeIgniter pour le développement d'applications PHP offre un gain de productivité de 50%. La perte de performance liée à son exécution est de l'ordre de 10%, très largement compensée par une augmentation de fiabilité et de maintenabilité due à la réutilisation du code.  
  
**Yii**  
Le Yii Framework ("Yes, It Is") est un framework pour PHP 5 et utilise le paradigme de programmation orientée objet. Il est destiné au développement d'applications Web. Yii nécessite minimalement la version 5.1.0 de PHP. Une documentation complète est disponible. La communauté autour du projet est très active. Le créateur et développeur principal de Yii est Qiang Xu, qui a également développé et maintenu le framework PHP Prado pendant 3 ans. Yii est d'ailleurs le successeur officiel de Prado.

Il offre notamment les fonctionnalités suivantes :

* de hautes performances ;
* une architecture Modèle-Vue-Contrôleur ;
* un accès aux bases de données par DAO/ActiveRecord ;
* les fonctions nécessaires pour la gestion de l'internationalisation (I18N/L10N) ;
* la gestion de caches ;
* le support de AJAX via l'intégration de jQuery ;
* le contrôle d'accès par la gestion de rôles utilisateurs (RBAC) ;
* la génération automatique du code PHP pour les opérations de base (création, lecture, mise à jour et suppression) sur la base de données (scaffolding) ;
* le contrôle des saisies utilisateurs sur les formulaires ;
* la notion de widgets ;
* les événements sur les éléments des pages (boutons, liens...) ;
* la gestion de thèmes pour l'habillage des sites ;
* le support des services Web ;
* la possibilité d'ajouter des fonctions via un système de plugins ;
* le support des tests unitaires et fonctionnels.
* la migration de bases de données ;

Son architecture extensible permet de développer et maintenir les applications en optimisant les tâches du développeur et favorisant un gain de productivité non négligeable. Ces avantages sont rendus possibles grâce à l’adoption des systèmes MVC (Modèle-Vue-Contrôleur) et ORM (Object-Relational Mapping).

**Symfony**

Symfony est un framework open source écrit en PHP 5, utilisant l’architecture MVC.  
Conçu dès l’origine en 2005 par une société française, Sensio Labs, bien connue pour ses idées novatrices dans le développement web, Symfony est robuste, fiable, très puissant, et s’adapte à un très grand nombre de configurations.  
Développé par une équipe de professionnels expérimentés, il est spécialement dédié à la conception d’applications moyennes à très lourdes, c’est pourquoi il trouve tout naturellement sa place dans l’entreprise. Ses multiples fonctionnalités et son abondante documentation ont fait de Symfony, en quelques années, l’un des frameworks PHP les plus utilisés.

Les principales caractéristiques de Symfony :

* Compatible PHP 5 totalement orienté objet.
* Intègre l’architecture MVC et la méthode ORM.
* Licence MIT (tout ce qui est réalisé avec Symfony peut être vendu).
* Extensible et modulaire.
* Supporte ajax.
* Système de templates.
* Importante communauté très active.
* Très grand nombre de traductions.
* Conçu pour une utilisation professionnelle.

Comme chaque framework le but de Symfony est d’abord de faciliter et d’accélérer le temps de développement par sa puissance, tout en optimisant le code en en permettant sa réutilisation, en toute sécurité. Un des nombreux avantages de Symfony est sa facilité d’installation sur la plupart des configurations existantes. Sa compatibilité avec la quasi-totalité des bases de données fait de Symfony un framework que les développeurs web confirmés ont peu de difficultés à prendre en main rapidement. Bien plus qu’un framework MVC, Symfony réunit les meilleurs outils de développement PHP afin d’aborder avec cohérence la réalisation d’applications web. L’architecture, outils et composants de Symfony favorisent et facilitent la conception d’applications complexes avec une aisance et une rapidité certaine.  
  
Basé sur l’expérience, fiable et multi-fonctionnel, Symfony gagne de jour en jour en notoriété et reconnaissance. Les nouveaux arrivants trouvent auprès de l’importante communauté, appui, documentation et contributions. En effet, le développement actuel de Symfony repose pour une bonne partie sur l’ajout de modules ou plugins (plus de 600 aujourd’hui), très faciles à produire et utiliser. Fabien Potencier, créateur et responsable du développement de Symfony et PDG de Sensio Labs, confirme l’adoption de cette méthode dans les évolutions futures. Il désigne Zend comme son principal concurrent.  
  
  
**Laravel**

Laravel a été créé par Taylor Otwell1 en juin 20112. Le référentiel Larave/laravel présent sur le site GitHub contient le code source des premières versions de Larave. À partir de la quatrième version, le framework est développé au sein du référentiel Laravl/framework. En peu de temps, une communauté d'utilisateurs du framework s'est constituée. La quatrième version de Laravel est écrite en PHP 5.3.73 et son installation est basée sur le gestionnaire de paquets Composer. Dans une syntaxe élégante, Laravel fournit des fonctionnalités en termes de routage de requête, de mapping objet-relationnel (un système baptisé Eloquent implémentant Active Record), d'authentification, de migration de base de données, de gestion des exceptions et de test unitaire1. L'ensemble des éléments constitué de la documentation du framework, de tutoriels et de copies d'écran permettent l'apprentissage du framework.

**Zend**

Zend Framework est un framework PHP 5 créé par la société Zend Technologies, appelée aussi The PHP Company car fondée par les pères du PHP moderne : Andi Gutmans et Zeev Suraski. Il est distribué open source sous la New BSD license. De par son histoire, Zend tient une place particulière dans l’univers des Frameworks PHP. Bénéficiant d’une communauté massive à travers le monde, il est aujourd’hui à n’en pas douter le plus connu et utilisé des frameworks. Notons que son utilisation requiert toutefois une connaissance approfondie de PHP. Zend Technologies l’a développé en privilégiant les « bonnes pratiques ». Il est totalement orienté objet et intègre l’architecture MVC.

Les principales caractéristiques de Zend Framework :

* Compatible PHP 5
* Conception totalement orientée objet
* Intègration de l’architecture MVC (Modèle-Vue-Contrôleur)
* Faible dépendance des composants entre eux, permet leur utilisation individuelle (use-at-will)
* Puissant, extensible et modulaire
* Imposante communauté hyper-active
* Fortement adapté à l’environnement professionnel
* Développeurs externes respectant la licence open source
* Alliés et contributeurs hors normes : Microsoft, Google…

Zend Framework peut être considéré actuellement comme le leader des Frameworks PHP. Sa souplesse est telle, qu’à partir d’une base quelconque, il est possible de créer une liaison avec d’autres frameworks ou composants.  
D’une puissance, robustesse et fiabilité à toute épreuve, la conception de Zend Framework est particulière et le différencie des autres frameworks PHP. En effet, ses composants ont la faculté d’être utilisés individuellement, car très peu dépendants des uns envers les autres. Par contre, utilisés ensemble dans la librairie standard de Zend Framework, ils donnent au framework sa redoutable puissance. De même, bien que l’architecture MVC soit présente nativement dans Zend Framework, elle n’est pas imposée mais optionnelle. Toujours à la pointe du développement, de multiples composants permettent l’accès aux services web les plus modernes. Le principal moteur du projet reste évidemment Zend Technologies, mais certaines sociétés extérieures ont développé leur propres composants et les ont mis ensuite à la disposition de la communauté. Parmi ces contributeurs exceptionnels, ils convient de citer Google et Microsoft, deux des plus importants acteurs du web. Zend Framework est en perpétuel développement, et chaque entreprise intéressée peut développer ses composants, à la seule condition de les mettre gracieusement à la disposition des utilisateurs, dans le respect de la licence. Le concept PHP reste l’esprit de Zend Framework : libre, ouvert, simple, efficace et puissant !!!

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Nom | PHP | Modèle | ORM | Template | Cache | Url conviviales | Validation de formulaire | Ajax | Extension | Générateur de code |
| Cake PHP | 4/5 | MVC | AR | PHP | 0 | 0 | 0 | 0 | 0 | 0 |
| CodeIgniter | 4/5 | MVC | AR | PHP | 0 | 0 | 0 | N | 0 | N |
| Yii | 5 | MVC | DAO | Twig ou Smarty | 0 | 0 | 0 | 0 | 0 | 0 |
| Laravel | 5 | MVC | AR | PHP | 0 | 0 | N | N | 0 | N |
| Symfony | 5 | MVC | Propel | PHP ou smarty | 0 | 0 | 0 | 0 | 0 | 0 |
| Zend | 5 | MVC | / | PHP | 0 | 0 | N | N | N | N |

Légende

0 = oui

N= non

AR= Active Record

MVC = Modèle-Vue-Contrôleur

Propel= Mapping objet relationnel (ORM)

DAO= Data Access Object

**b – Historique de symfony**

La première version de Symfony sort le 18 octobre 2005. Elle est développé par SensioLabs à l'époque une simple agence web qui à force de recréer les mêmes fonctionnalités de gestion d'utilisateurs, gestion ORM, etc. va décider de développer un framework pour ses propres besoins. Le projet prend alors le nom de Symfony suite à la volonté de Fabien Potencier (le créateur du framework) de conserver les initiales S et F de Sensio Framework. Dans cette première version, le framework compatible à partir de la version de PHP 5.2.4 embarque les fonctionnalités suivante :

- Une séparation en trois couches selon le modèle MVC ce qui permet une meilleure maintenabilité et évolutivité.

- Des performances optimisées et un système de cache afin d'assurer des temps de réponse optimaux.

- Une gestion des URL parlante, permettant à une page d'avoir une URL distincte de sa position dans l'arborescence.

- Un système de configuration en cascade utilisant le langage YAML.

- Un générateur de back-office et un lanceur de module.

- L'internationalisation native.

- Une couche de mapping objet-relationnel (ORM) et une abstraction de données.

- Le support d'AJAX.

- Une architecture extensible permettant la créations et l'utilisations de plugins.

La version 2 de Symfony casse la compatibilité avec la branche 1.x. et est disponible à partir de PHP 5.3.3. Sortie le 28 juillet 2011. En effet, la deuxième mouture du framework vient par défaut avec une boîte à outils impressionnante pour permettre aux développeurs d'être immédiatement opérationnel. Le support pour des librairies telles qu'Assetic, Twig, Imagine et Monolog est une illustration de la volonté de pouvoir réutiliser tout ce qui se fait de mieux en termes de performances et de standards. Le concept de « Bundle » commence à arriver, d'ailleurs Symfony2 est un Bundle.

La troisième version de Symfony arrive le 30 novembre 2015. Elle est développé pour fonctionné avec une version minimum de PHP en 5.5.9. Si l'arrivé de Symfony 2 avait soufflé un vent de panique dans la communauté avec la difficulté de porter un projet Symfony 1 sur Symfony 2, la troizième version permet une mise à niveau plus sereine. La bonne nouvelle est donc que le code Symfony 2 est compatible avec Symfony 3.

Symfony 4 est sorti le 30 novembre 2017. Elle représente une refonte complète de ses idées et fonctionnalités pour correspondre aux pratiques de l'industrie: les bundles d'application ont disparu, les paramètres de configuration sont désormais des variables d'environnement, la structure du répertoire de l'application est plus facile à parcourir. Le résultat est moins de concepts Symfony et plus de pratiques standard. Plus facile à apprendre, plus facile à configurer, plus facile à installer et à déployer et plus facile à maîtriser.

Symfony 5 est sorti le 21 novembre 2019 et représente la version la plus récente de Symfony à ce jour. Elle poursuit la révolution entamée par Symfony 4 pour créer la meilleure version Symfony jamais publiée. Encore plus facile à apprendre, plus facile à configurer et plus facile à installer, à déployer et à maîtriser. Symfony 5 s'intègre de manière transparente à Symfony Flex pour automatiser les tâches les plus courantes effectuées par les applications. L'activation des bundles ou la création de leur configuration initiale est oubliée: Symfony Flex est là.

**2 – Composer**

**a – Quoi ?**

Composer est un outil permettant de gérer les dépendances en PHP. Une dépendance est une bibliothèque dont votre projet dépend pour fonctionner. Par exemple si un projet utilise la bibliothèque SwiftMail pour envoyer des e-mails alors SwiftMail est une dépendance dans ce projet.

**b – Avantages et inconvénients**

Composer est utile dans le sens où il télécharge automatiquement les librairies dont un projet a besoin pour fonctionner, il dispose également d’un autoloader permettant automatiquement de charger tout ce dont vous avez besoin et tout ce qu’il y a à faire est d'inclure un fichier. Il permet également d’utiliser les namespace psr-4 pour charger un chemin spécifique sur le projet et le faire inclure dans le fichier du chargeur automatique, ce namespace pourra simplement être utilisé et sera ensuite disponible dans toute l’application.

**c – Installation**

Composer étant un outil PHP, il faut vérifier l’environnement de PHP et vérifier qu’il soit bien configuré pour pouvoir s'exécuter via la console.

Sous Windows il faut lancer l’invite de commande et taper la commande php -v

Sous Linux et Mac lancer le terminal et exécuter la même commande

Si cette commande retourne la version de PHP et d'autres informations, c'est tout bon pour vous. Il faut aussi vérifier que la version de PHP est 7.1.3 au minimum sinon il faut d'abord le mettre à jour.

En cas d’erreur c’est-à-dire que Windows ne sait pas où trouver le dossier PHP, voici la démarche à suive :

1. Aller dans les paramètres système avancés : *Démarrer > Panneau de configuration > Système et sécurité > Système > Paramètres système avancés* ;
2. Cliquer sur le bouton  Variables d'environnement…  ;
3. Dans la section  Variables utilisateurs trouver l'entrée  Path  et la sélectionner ;
4. Cliquez sur  Modifier puis sur  Nouveau  puis entrer votre répertoire PHP à la fin, c'est le répertoire dans lequel se trouve le fichier php.exe. Par exemple sur **WAMP** : C:\wamp\bin\php\php7.4.0 ou sous **XAMPP** : C:\xampp\php

Confirmez en cliquant sur OK. Vous devez ensuite redémarrer l'invite de commandes pour prendre en compte les changements.

Si vous êtes sous Linux, vérifiez votre installation de PHP. Vous devez notamment avoir le paquet  php7-cli, qui est la version console de PHP.

Passons à l’installation de Composer : si vous êtes sous Windows, la méthode la plus facile est d'installer Composer via son exécutable d'installation, comme n'importe quel autre logiciel c’est-à-dire depuis un lien comme celui-ci par exemple : [getcomposer.org/Composer-Setup.exe](https://getcomposer.org/Composer-Setup.exe) puis suivre les instructions.

Sous Linux et Mac exécuter simplement la commande suivante :

php -r "eval('?>'.file\_get\_contents('http://getcomposer.org/installer'));"

Si tout est ok le terminal devrait afficher :

All settings correct for using Composer

Downloading...

Composer (version 1.6.3) successfully installed to: C:\wamp\www\admin\composer.phar

Use it: php composer.phar

Cependant pour récupérer certaines bibliothèques Composer utilise git qu’il faudra installer aussi.

**3 – Symfony**

**a – installation**

Installation à partir d'une archive Le moyen le plus simple d'installer symfony est de télécharger l'archive pour la version que vous avez choisie sur le site de symfony. Aller à la page d'installation de la version que vous venez de choisir, symfony 1.4 par exemple.

Sous la section "Download as an Archive", vous trouverez l'archive sous le format .tgz ou .zip. Téléchargez l'archive, puis mettez-la sous le nouveau répertoire créé lib/vendor/, décompressez la, et renommez le répertoire en symfony :

$ cd lib/vendor$ tar zxpf symfony-1.4.8.tgz$ mv symfony-1.4.8 symfony$ rm symfony1.4.8.tgz

Sous Windows, décompressez le fichier zip en utilisant l'Explorateur Windows. Après avoir renommé le répertoire en symfony, vous devriez y avoir une structure de répertoire similaire à c:\dev\sfproject\lib\vendor\symfony.

Installation à partir de Subversion (recommandée). Si vous utilisez Subversion, il est même préférable d'utiliser la propriété svn:externals pour bien intégrer symfony dans votre projet dans le répertoire lib/vendor/ :

$ svn pe svn:externals lib/vendor/

Si tout va bien, cette commande va lancer votre éditeur favori pour vous donner la possibilité de configurer les sources extérieures Subversion.

TIP

Sur Windows, vous pouvez utiliser des outils comme TortoiseSVN pour tout faire sans avoir besoin d'utiliser la console.

Si vous êtes conservateur, liez votre projet à une version release (un tag de subversion) :

svn checkout http://svn.symfony-project.com/tags/RELEASE\_1\_4\_8 symfony

Chaque fois qu'une nouvelle version sort (comme annoncé sur le blog de Symfony), vous devrez modifier l'URL vers la nouvelle version.

Si vous voulez aller sur la route de l'avant-garde, utilisez la branche 1.4 :

svn checkout http://svn.symfony-project.com/branches/1.4/ symfony

Grâce à l'utilisation de cette branche, votre projet bénéficie des corrections de bogues automatiquement chaque fois que vous exécutez svn update.

**Le fichier de configuration app.yml 1.4**

Le framework symfony fournit un fichier de configuration intégré pour les paramètres spécifiques de l'application, le fichier de configuration app.yml. Ce fichier YAML peut contenir n'importe quel paramètre que vous voulez et qui semble raisonnable pour votre application spécifique. Dans le code, ces paramètres sont disponibles via la classe globale sfConfig, et les clés sont préfixés de la chaîne app\_ :

sfConfig::get('app\_active\_days');

Tous les paramètres sont préfixés par app\_ parce que la classe sfConfig donne également accès aux paramètres de symfony et aux répertoires de projet. Comme indiqué dans l'introduction, le fichier app.yml est sensible à l'environnement, et bénéficie du mécanisme de configuration en cascade. Le fichier de configuration app.yml est un endroit idéal pour définir les paramètres qui changement selon l'environnement (une clé API par exemple), ou les paramètres qui peuvent évoluer au fil du temps (une adresse email par exemple). Il est également le meilleur endroit pour définir les paramètres qui doivent être modifiées par quelqu'un qui ne connait pas nécessairement symfony ou PHP (un administrateur système par exemple).

**TIP**

Abstenez-vous d'utiliser app.yml pour grouper la logique d'application.

**NOTE**

Le fichier de configuration app.yml est mis en cache dans un fichier PHP, le processus est automatiquement géré par la classe sfDefineEnvironmentConfigHandler.

Ce framework est structuré en 4 dossier :

app, src, vendor, web Le répertoire /app Ce répertoire contient tout ce qui concerne le site internet sauf son code source c’est à dire la configuration, le cache, les fichiers logs. Le répertoire /src Le répertoire dans lequel se trouve le code source. Dans ce répertoire, le code

est organisé en bundles, des briques de l’application. Le répertoire /vendor Ce répertoire contient toutes les bibliothèques externes à l’application. Dans ces bibliothèques externes nous pouvons y trouver Syphony2, Doctrine, Twig, SwiftMailer, etc. Le répertoire /web Ce répertoire contient tous les fichiers destinés aux visiteurs : images, fichiers CSS et JavaScript, etc. Il contient également le contrôleur frontal (app.php).C'est le seul répertoire qui devrait être accessible aux visiteurs. Le contrôleur frontal Le contrôleur frontal est le point d'entrée de votre application. C'est le fichier par lequel passent toutes les pages. Dans Symfony2, le contrôleur frontal se situe dans le répertoire /web, il s'agit de app.php ou app\_dev.php.

b – Symfony Flex : Notion de bundle

Un Bundle est un répertoire dans un projet symfony qui intègre une structure bien définie, ce répertoire permet d’implémenter plusieurs fonctionnalités qui peuvent être utilisées dans d’autre projet symfony. On peut voir un Bundle comme un plug-in dans symfony. Ce framework propose des milliers de Bundles réutilisables quasiment dans tous les projets symfony qu’on peut avoir. il existe même des sites qui proposent de télécharger des Bundles, il y’a par exemple : [KnpBundle](http://knpbundles.com/" \t "_blank). Pour créer un bundle on peut procéder de trois manières :

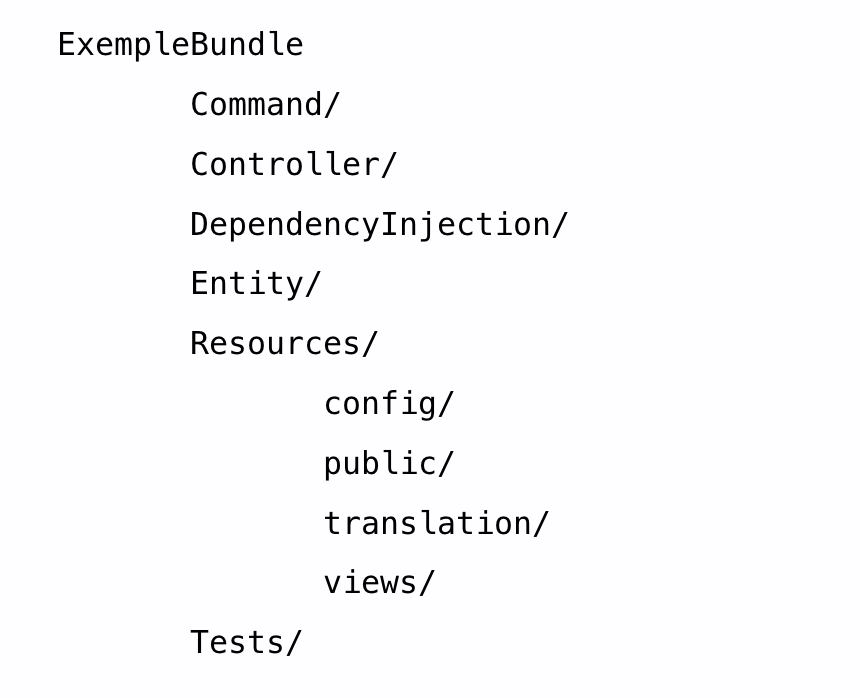
Via Composer on utilisant cette commande

$ composer.phar require nomdubundle/exemple-bundle "version", en ligne de commande ou manuellement.

Utiliser un Bundle peut apporter plusieurs avantages et surtout faire gagner beaucoup de temps, ça permet aussi de travailler sur une base solide, car la plupart des Bundles existants ont été construits pour être flexible et réutilisable en plus d'être sécurisé.

**Architecture d’un bundle**

Un bundle a généralement toujours la même structure, image ci-dessous, mais ne se limite pas qu’au dossier indiqué on peut aussi rajouter des dossiers dans le bundle afin d’organiser le projet plus proprement et de faire en sorte que chaque partie du dossier soit dédier aune tache particulière.



**Eléments d’un Bundle**

**Command** : dans ce répertoire, on aura les classes qui vont permettre de construire des commandes du Bundle qui vont être utilisées dans le terminal dans le projet.

**Controller** : c’est ici qu'il aura des classes et fonctions appelées controlleur. Le rôle d'un contrôleur est de créer et de retourner une réponse aux vues de l'application

**DependanceInjection** : dans ce dossier, il y'a des méthodes qui vont permettre de gérer la dépendance d'injection. Ceci permet surtout de rendre utilisable nos services symfony dans toute l'application et de laisser à symfony de s'occuper des taches répétitives qu'on pourra faire notamment instancier des classes.

**Resources**

- **config** : Contiens tous les fichiers de configuration du Bundle (service, routing)

- **doc** : éventuelles documentations du bundle

- **translation** : Ce dossier va contenir les différents fichiers de traduction du bundle si l'application est développée en plusieurs langues.

- **public** : Ce répertoire contient tous les astes du Bundle (css, js, images etc.)

- **views** toutes les vues qui vont être retournées par le contrôller.

**Entity** : le modèle de l'application il y'aura des classes qui vont contenir des attributs et leurs getters/setters pour gérer la couche métier du Bundle ainsi de faire correspondre chaque attribut de la classe par un champ de la table de la base de données tout ceci est géré par doctrine.

**Tests** : Dossier qui contient tous les tests de l'application.

**c - Commandes de Génération**

Controller Un contrôleur est une fonction PHP que vous créez qui lit les informations de l' Requestobjet et crée et renvoie un Responseobjet. La réponse pourrait être une page HTML, JSON, XML, un téléchargement de fichier, une redirection, une erreur 404 ou autre chose. Le contrôleur exécute la logique arbitraire dont votre application a besoin pour afficher le contenu d'une page. Pour gagner du temps, vous pouvez installer Symfony Maker et demander à Symfony de générer une nouvelle classe de contrôleur: Pour générer un CRUD complet à partir d'une entité Doctrine , utilisez: • php bin/console make:crud Product Pour créer un Controller, on utilise la commande symfony console make:controller nomController. Entity et des Relations Symfony fournit tous les outils dont vous avez besoin pour utiliser des bases de données dans vos applications grâce à Doctrine , le meilleur ensemble de bibliothèques PHP pour travailler avec des bases de données. Ces outils prennent en charge les bases de données

relationnelles comme MySQL et PostgreSQL ainsi que les bases de données NoSQL comme MongoDB. Installer Doctrine pour la génération du code • composer require symfony/orm-packcomposer require --dev symfony/maker-bundle Configuration de la base de données Les informations de connexion à la base de données sont stockées sous la forme d'une variable d'environnement appelée DATABASE\_URL. Pour le développement, vous pouvez trouver et personnaliser cela à l'intérieur .env. • DATABASE\_URL="mysql://db\_user:db\_password@127.0.0.1:3306/db\_name" Sous MySQL, on peut modifier cette ligne en mettant • DATABASE\_URL="mysql://root:’’@127.0.0.1:3306/nom\_de\_la\_base\_de\_donnée" Apres avoir configuré les paramètres de connexion, doctrine nous crée la base de données via la commande suivante : • php bin/console doctrine:database:create Création d'une classe d'entité Supposons que vous créez une application dans laquelle les produits doivent être affichés. Sans même penser à Doctrine ou aux bases de données, vous savez déjà que vous avez besoin d'un Productobjet pour représenter ces produits. Vous pouvez utiliser la make:entity commande pour créer cette classe et tous les champs dont vous avez besoin. La commande vous posera quelques questions - répondez-y comme ci-dessous: • $ php bin/console make:entity Class name of the entity to create or update:> Product New property name (press <return> to stop adding fields):> name Field type (enter ? to see all types) [string]:> string Field length [255]:> 255 Can this field be null in the database (nullable) (yes/no) [no]:> no New property name (press <return> to stop adding fields):> price Field type (enter ? to see all types) [string]:> integer Can this field be null in the database (nullable) (yes/no) [no]:> no New property name (press <return> to stop adding fields):> (press enter again to finish) Apres cette étape, doctrine nous crée un fichier Exemple : src/Entity/Product.php Ici, le fichier créer Product.php dans src/Entity Voici le contenu du fichier : // src/Entity/Product.phpnamespace App\Entity; use Doctrine\ORM\Mapping as ORM; /\*\* \* @ORM\Entity(repositoryClass="App\Repository\ProductRepository") \*/class Product{ /\*\* \* @ORM\Id \* @ORM\GeneratedValue \* @ORM\Column(type="integer") \*/ private $id; /\*\* \* @ORM\Column(type="string", length=255) \*/ private $name; /\*\* \* @ORM\Column(type="integer") \*/ private $price; public function getId() { return $this->id; } // ... getter and setter methods}

Concernant les relations, il existe les relations

ManyToMany

OneTOMany

ManyToOne

OneToOne

Symfony Maker vous aide à créer des commandes vides, des contrôleurs, des classes de formulaire, des tests et plus encore afin que vous puissiez oublier d'écrire du code passe

partout. Ce bundle est une alternative à SensioGeneratorBundle pour les applications Symfony modernes et nécessite l'utilisation de Symfony 3.4 ou plus récent. Ce bundle suppose que vous utilisez une structure de répertoire Symfony 4 standard, mais de nombreuses commandes peuvent générer du code dans n'importe quelle application. • php bin/console list make make:command Creates a new console command class make:controller Creates a new controller class make:entity Creates a new Doctrine entity class.

**d – Rooting**

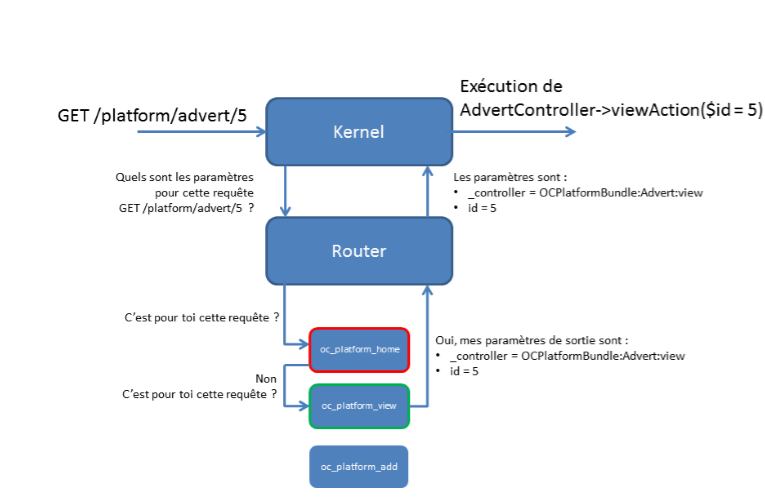
Une route est composée au minimum de deux éléments : l'URL à faire correspondre (son path), et le contrôleur à exécuter (paramètre \_controller). • Le routeur essaie de faire correspondre chaque route à l'URL appelée par l'internaute, et ce dans l'ordre d'apparition des routes : la première route qui 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 routes peuvent être configurés en YAML, XML, PHP ou en utilisant des annotations. Tous les formats offrent les mêmes fonctionnalités et performances, alors choisissez votre favori. Symfony recommande des annotations car il est pratique de placer la route et le contrôleur au même endroit.

Chaque route fait la correspondance entre une URL et un jeu de paramètres.

chaque route prend : • Une entrée (ligne path) : c'est l'URL à capturer ; • Une sortie (ligne defaults) : ce sont les paramètres de la route, notamment celui qui dit quel est le contrôleur à appeler.

Le but du routeur est donc, à partir d'une URL, de trouver la route correspondante et de retourner les paramètres de sortie que définit cette route, dont le contrôleur. Pour trouver la bonne route, le routeur va les parcourir une par une, dans l'ordre du fichier, et s'arrêter à la première route qui fonctionne.



Cheminement du routeur

1. On appelle l'URL /platform/advert/5.

2. Le routeur essaie de faire correspondre cette URL avec le path de la première route. Ici, /platform/advert/5 ne correspond pas du tout à /platform (ligne path de la première route).

3. Le routeur passe donc à la route suivante. Il essaie de faire correspondre /platform/advert/5 avec /platform/advert/{id}. Nous le verrons plus loin, mais {id} est un paramètre, une sorte de joker « je prends tout ». Cette route correspond, car nous avons bien : o /platform/advert (URL) = /platform/advert (route) ; o 5 (URL) = {id} (route).

4. Le routeur s'arrête donc, il a trouvé sa route.

5. Il demande à la route : « Quels sont tes paramètres de sortie ? », la route répond : « Mes paramètres sont 1/ le contrôleur OCPlatformBundle:Advert:view, et 2/ la valeur $id = 5. »

6. Le routeur renvoie donc ces informations au Kernel.

7. Le noyau va exécuter le bon contrôleur avec les bons paramètres !

Dans le cas où le routeur ne trouve aucune route correspondante, le noyau de Symfony va déclencher une erreur 404.

Pour chaque page, il est possible de visualiser toutes les routes que le routeur essaie une à une, et celle qu'il utilise finalement. C'est le Profiler qui s'occupe de tracer cela, accessible depuis la barre d'outils : cliquez sur le nom de la route dans la barre d'outils, « oc\_platform\_home » si vous êtes sur la page /platform. Ce lien vous amène dans l'onglet « Request » du Profiler, mais allez dans l'onglet « Routing » qui nous intéres

**e - ORM Doctrine**

• ORM signifie Object-Relationnal Mapper. Un ORM sert à offrir une couche d’abstraction de connexion à toutes les BD relationnelles (comme PDO) mais aussi des facilités pour réaliser les requêtes courantes sans descendre au niveau des requêtes SQL et pour générer automatiquement des entités dans le langage utilisé avec les getters et setters correspondants.

• installation

• Il suffit de taper :

• composer req orm

• symfony maker-bundle

• Il permet de fabriquer des entités, des contrôleurs, des CRUD, des tests etc.

• installation • composer req maker-bundle --dev

• sous-commandes disponibles

• php bin/console list make

• ce qui donne :

• Symfony 5.0.7 (env: dev, debug: true) Usage:command [options] [arguments] Options:-h, --help Display this help message-q, --quiet Do not output any message-V, --version Display this application version --ansi Force ANSI output --no-ansi Disable ANSI output-n, --no-interaction Do not ask any interactive question-e, --env=ENV The Environment name. [default: "dev"] -no-debug Switches off debug mode.-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug Available commands for the "make" namespace:make:auth Creates a Guard authenticator of different flavorsmake:command Creates a new console command classmake:controller Creates a new controller classmake:crud Creates CRUD for Doctrine entity classmake:entity Creates or updates a Doctrine entity class, and optionally an API Platform resourcemake:fixtures Creates a new class to load Doctrine fixturesmake:form Creates a new form classmake:functional-test Creates a new functional test classmake:message Creates a new message and handlermake:messenger-middleware Creates a new messenger middlewaremake:migration Creates a new migration based on database changesmake:registration-form Creates a new registration form systemmake:reset-password Create controller, entity, and repositories for use with symfonycasts/reset-password-bundle.make:serializer:encoder Creates a new serializer encoder classmake:serializer:normalizer Creates a new serializer normalizer classmake:subscriber Creates a new event subscriber classmake:twigextension Creates a new Twig extension classmake:unit-test Creates a new unit test classmake:user Creates a new security user classmake:validator Creates a new validator and constraint classmake:voter Creates a new security voter class • Entités

• Créons une entité Personne de ce type :

• php bin/console make:entity

• Répondez aux questions pour ajouter des champs nom et prenom de type string dans Personne.

• Vérifiez la création du code correspondant dans src/Entity.

• <?php namespace App\Entity; use Doctrine\ORM\Mapping as ORM; /\*\*\* @ORM\Entity(repositoryClass="App\Repository\PersonneRepository")\*/class Personne{ /\*\* \* @ORM\Id() \* @ORM\GeneratedValue() \* @ORM\Column(type="integer") \*/ private $id; /\*\* \* @ORM\Column(type="string", length=255) \*/ private $nom; /\*\* \* @ORM\Column(type="string", length=255) \*/ private $prenom; public function getId(): ?int { return $this->id; } public function getNom(): ?string { return $this->nom; } public function setNom(string $nom): self { $this->nom = $nom; return $this; } public function getPrenom(): ?string { return $this->prenom; } public function setPrenom(string $prenom): self { $this->prenom = $prenom; return $this; }}

• Config

• Réglons la configuration de la base dans .env sur SQLite:

• ###> doctrine/doctrine-bundle ###DATABASE\_URL=sqlite:///%kernel.project\_dir%/var/carnet.db?charset=utf8mb 4###< doctrine/doctrine-bundle ###

• Interaction avec la BD • Initialisons la base :

• ./bin/console doctrine:database:create./bin/console doctrine:schema:update --force

• easy\_admin

• Pour faciliter l’interaction avec la BD, installons easy\_admin :

• composer req admin

• Puis visitez : localhost:8000/admin

• Nous obtenons une erreur, complétons donc config/packages/easy\_admin.yaml :

• easy\_admin: entities: # List the entity class name you want to manage - App\Entity\Personne

• et vous pourrez ensuite ajouter facilement des personnes dans l’admin !

• générateur de CRUD

• Nous aurions pu aussi utiliser le générateur de CRUD :

• php bin/console make:crud

• en choisissant l’entité Personne.

• Cela fabrique automatiquement un CRUD avec le contrôleur et les templates nécessaires. Intéressant pour inspecter le code généré même si visuellement easy\_admin est plus abouti …

• profiler

• Puis installons la barre du profiler :

• composer req profiler

• Pour tester le CRUD et le profiler : localhost:8000/personne

• On obtient :

• debug des routes

• On peut lister toutes les routes :

• php bin/console debug:router • sécurité • On peut utiliser le security-checker pour vérifier tous les bundles installés : • composer require sec-checker --dev

**f - Forms**

La création et le traitement de formulaires HTML sont difficiles et répétitifs. Vous devez gérer le rendu des champs de formulaire HTML, la validation des données soumises, le mappage des données de formulaire en objets et bien plus encore. Symfony comprend une puissante fonctionnalité de formulaire qui fournit toutes ces fonctionnalités et bien d'autres pour des scénarios vraiment complexes.

**Installation**

Dans les applications utilisant [Symfony Flex](https://symfony.com/doc/current/setup.html#symfony-flex) , exécutez cette commande pour installer la fonctionnalité de formulaire avant de l'utiliser :



USAGE

Le flux de travail recommandé lors de l'utilisation de formulaires Symfony est le suivant :

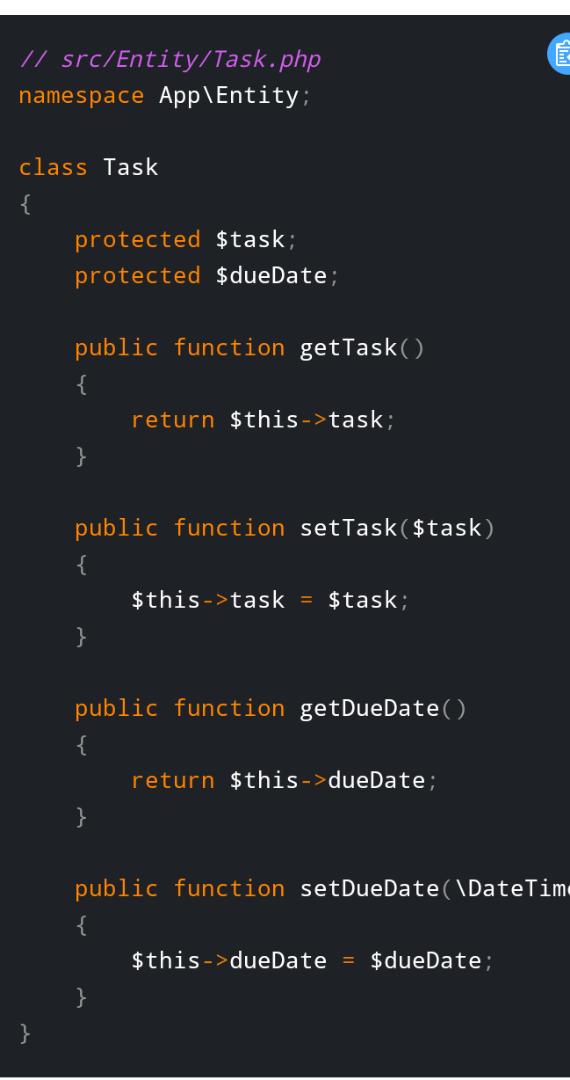
Créez le formulaire dans un contrôleur Symfony ou en utilisant une classe de formulaire dédiée ;

Rendez le formulaire dans un modèle afin que l'utilisateur puisse le modifier et le soumettre ;

Traitez le formulaire pour valider les données soumises, transformez-les en données PHP et faites-en quelque chose (par exemple, conservez-les dans une base de données).

Chacune de ces étapes est expliquée en détail dans les sections suivantes. Pour rendre les exemples plus faciles à suivre, ils supposent tous que vous créez une application de liste Todo simple qui affiche des « tâches ».

Les utilisateurs créent et modifient des tâches à l'aide de formulaires Symfony. Chaque tâche est une instance de la Taskclasse suivante :



Cette classe est un « objet PHP simple et ancien » car, jusqu'à présent, elle n'a rien à voir avec Symfony ou toute autre bibliothèque. C'est un objet PHP normal qui résout directement un problème à l'intérieur de votre application (c'est-à-dire la nécessité de représenter une tâche dans votre application). Mais vous pouvez également modifier les [entitésDoctrine](https://symfony.com/doc/current/doctrine.html) De la même manière.

Types de formulaires

Avant de créer votre premier formulaire Symfony, il est important de comprendre le concept de « type de formulaire ». Dans d'autres projets, il est courant de faire la différence entre « formulaires » et « champs de formulaire ». Dans Symfony, tous sont des « types de formulaires » :

Un seul champ de formulaire est un « type de formulaire » (par exemple) ;<input type="text">TextType

Un groupe de plusieurs champs HTML utilisés pour saisir une adresse postale est un « type de formulaire » (par exemple PostalAddressType);

Un ensemble <form>avec plusieurs champs pour modifier un profil utilisateur est un « type de formulaire » (par exemple UserProfileType).

Cela peut être déroutant au début, mais cela vous semblera naturel assez tôt. En outre, il simplifie le code et facilite la mise en œuvre des champs de formulaire de « composition » et « d'intégration ».

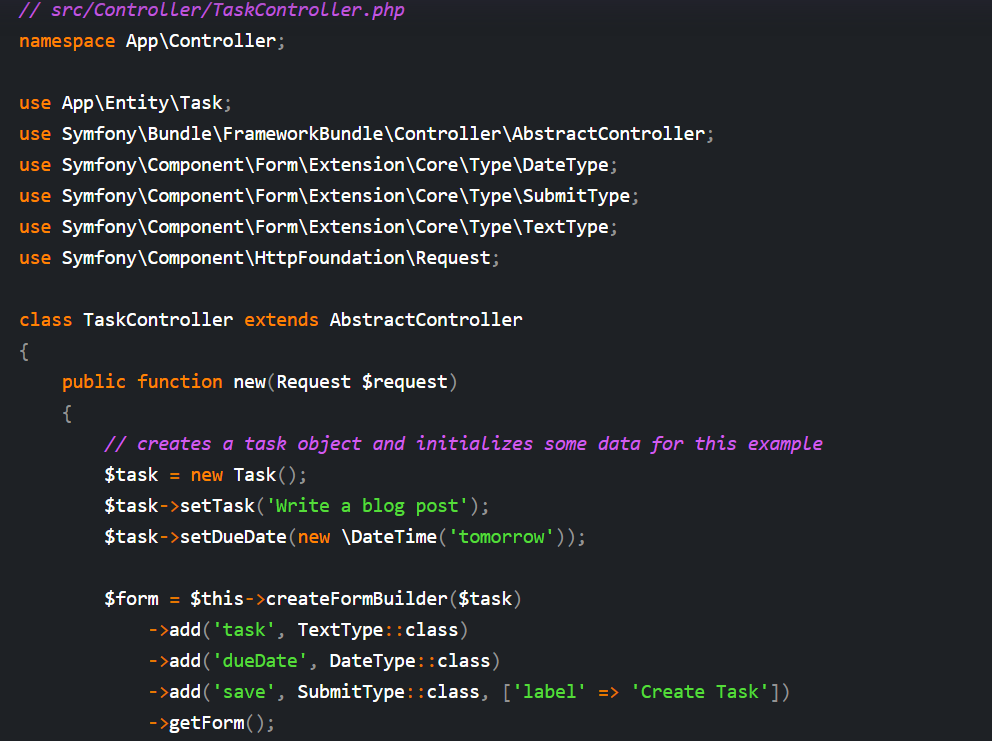
[Symfony propose](https://symfony.com/doc/current/reference/forms/types.html) des dizaines de [types](https://symfony.com/doc/current/reference/forms/types.html) de [formulaires](https://symfony.com/doc/current/reference/forms/types.html) et vous pouvez également [créer vos proprestypes de formulaires](https://symfony.com/doc/current/form/create_custom_field_type.html) .

**Création de formulaires**

Symfony fournit un objet «form builder» qui vous permet de décrire les champs du formulaire à l'aide d'une interface fluide. Plus tard, ce générateur crée l'objet de formulaire réel utilisé pour rendre et traiter le contenu.

Création de formulaires dans les contrôleurs

Si votre contrôleur s’étend du [AbstractController](https://symfony.com/doc/current/controller.html#the-base-controller-class-services) , utilisez l’ createFormBuilder()assistant:



Si votre contrôleur ne s'étend pas depuis AbstractController, vous devrez [récupérer les services dans votre contrôleur](https://symfony.com/doc/current/controller.html#controller-accessing-services) et utiliser la createBuilder()méthode du form.factoryservice.

Dans cet exemple, vous avez ajouté deux champs à votre formulaire - tasket dueDate - correspondant aux propriétés tasket dueDatede la Task classe. Vous avez également attribué à chacun un [type de formulaire](https://symfony.com/doc/current/forms.html#form-types) (par exemple, TextType et DateType), représenté par son nom de classe complet. Enfin, vous avez ajouté un bouton d'envoi avec une étiquette personnalisée pour soumettre le formulaire au serveur.

**Création de classes de formulaires**

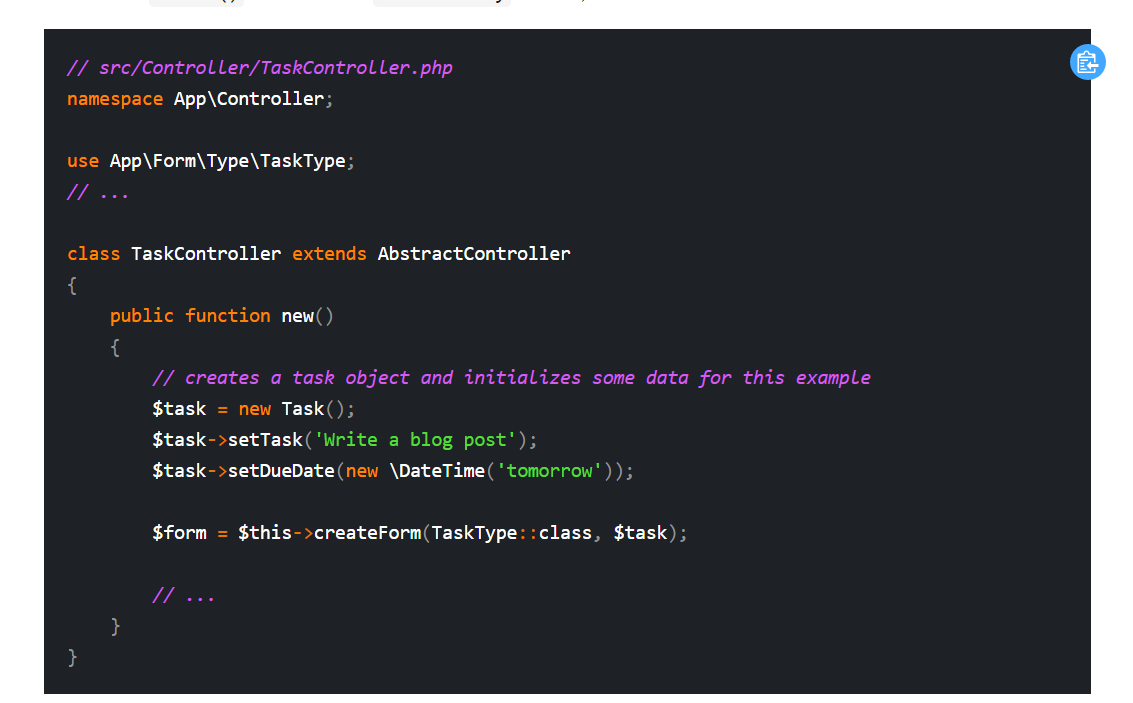
Symfony recommande de mettre le moins de logique possible dans les contrôleurs. C'est pourquoi il est préférable de déplacer des formulaires complexes vers des classes dédiées plutôt que de les définir dans des actions de contrôleur. De plus, les formulaires définis dans les classes peuvent être réutilisés dans plusieurs actions et services.

Les classes de formulaire sont des [types de formulaire](https://symfony.com/doc/current/forms.html#form-types) qui implémentent [FormTypeInterface](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Form/FormTypeInterface.php). Cependant, il est préférable d'étendre à partir de [AbstractType](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Form/AbstractType.php), qui implémente déjà l'interface et fournit certains utilitaires :



Installez le [MakerBundle](https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html) Dans votre projet pour générer des classes de formulaire à l'aide des commandes make:formet make:registration-form.

La classe de formulaire contient toutes les instructions nécessaires pour créer le formulaire de tâche. Dans les contrôleurs s'étendant de [AbstractController](https://symfony.com/doc/current/controller.html#the-base-controller-class-services) , utilisez l' createForm()assistant (sinon, utilisez la create()méthode du form.factoryservice):



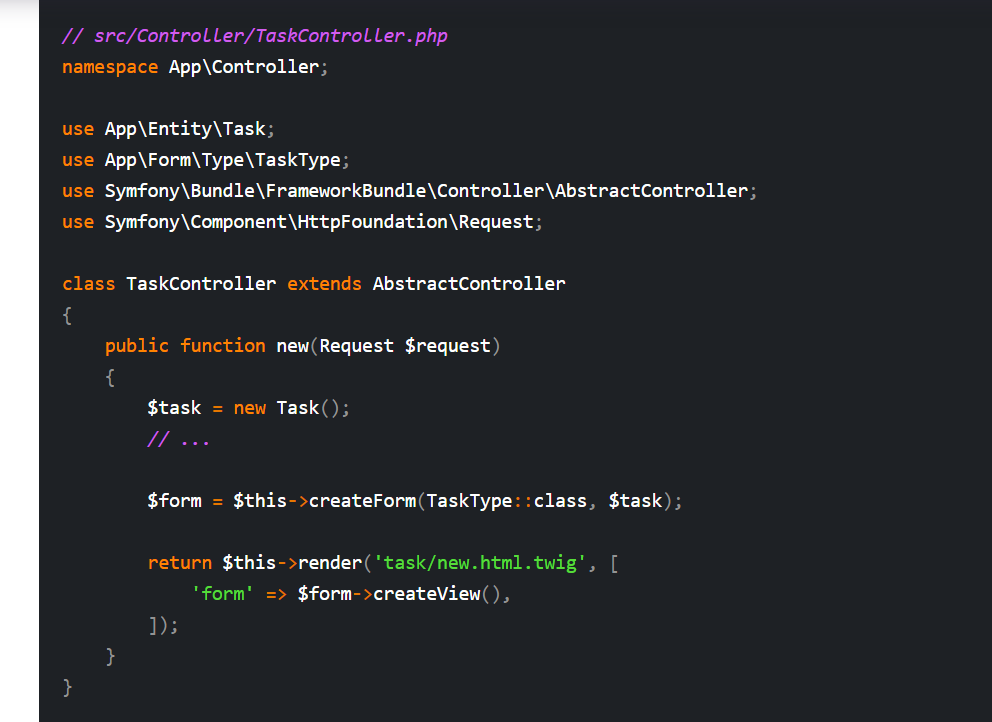
Chaque formulaire doit connaître le nom de la classe qui contient les données sous-jacentes (par exemple App\Entity\Task). Habituellement, cela est juste supposé en fonction de l'objet passé au deuxième argument createForm() (c'est- à -dire $task). Plus tard, lorsque vous commencerez à [incorporer des formulaires](https://symfony.com/doc/current/form/embedded.html) , cela ne sera plus suffisant.

Ainsi, bien que cela ne soit pas toujours nécessaire, il est généralement préférable de spécifier explicitement l' data\_classoption en ajoutant ce qui suit à votre classe de type de formulaire :



Formulaires de rendu

Maintenant que le formulaire a été créé, l'étape suivante consiste à le rendre. Au lieu de passer l'intégralité de l'objet de formulaire au modèle, utilisez la createView()méthode pour créer un autre objet avec la représentation visuelle du formulaire:

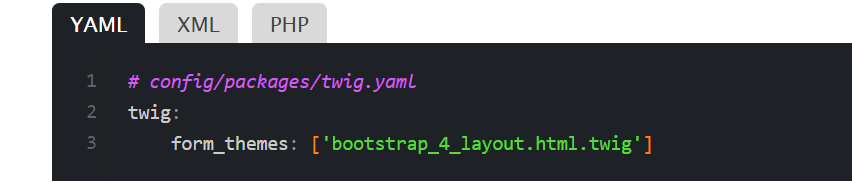


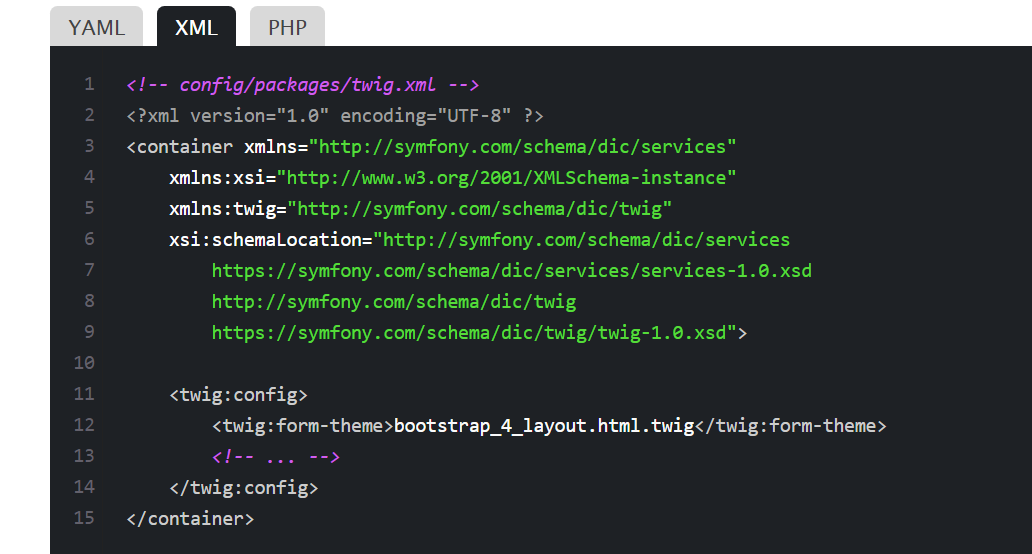
Ensuite, utilisez des [fonctions d'assistance de formulaire](https://symfony.com/doc/current/form/form_customization.html#reference-form-twig-functions) pour rendre le contenu du formulaire:

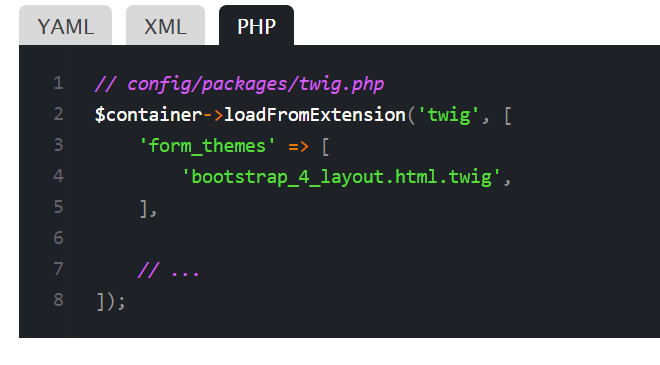
C'est tout ! La fonction [form()](https://symfony.com/doc/current/form/form_customization.html#reference-forms-twig-form) rend tous les champs et les <form>balises de début et de fin. Par défaut, la méthode du formulaire est POSTet l'URL cible est la même que celle affichée pour le formulaire, mais [vous pouvez modifier les deux](https://symfony.com/doc/current/forms.html#forms-change-action-method) .

Remarquez comment le taskchamp de saisie rendu a la valeur de la task propriété de l' $taskobjet (c.-à-d. « Rédiger un article de blog »). Il s'agit du premier travail d'un formulaire : prendre des données d'un objet et les traduire dans un format adapté au rendu dans un formulaire HTML.

Aussi court que soit ce rendu, il n'est pas très flexible. Habituellement, vous aurez besoin de plus de contrôle sur l'apparence de l'ensemble du formulaire ou de certains de ses champs. Par exemple, grâce à l' [intégration de Bootstrap 4 avec les formulaires Symfony,](https://symfony.com/doc/current/form/bootstrap4.html) vous pouvez définir cette option pour générer des formulaires compatibles avec le framework CSS Bootstrap 4 :







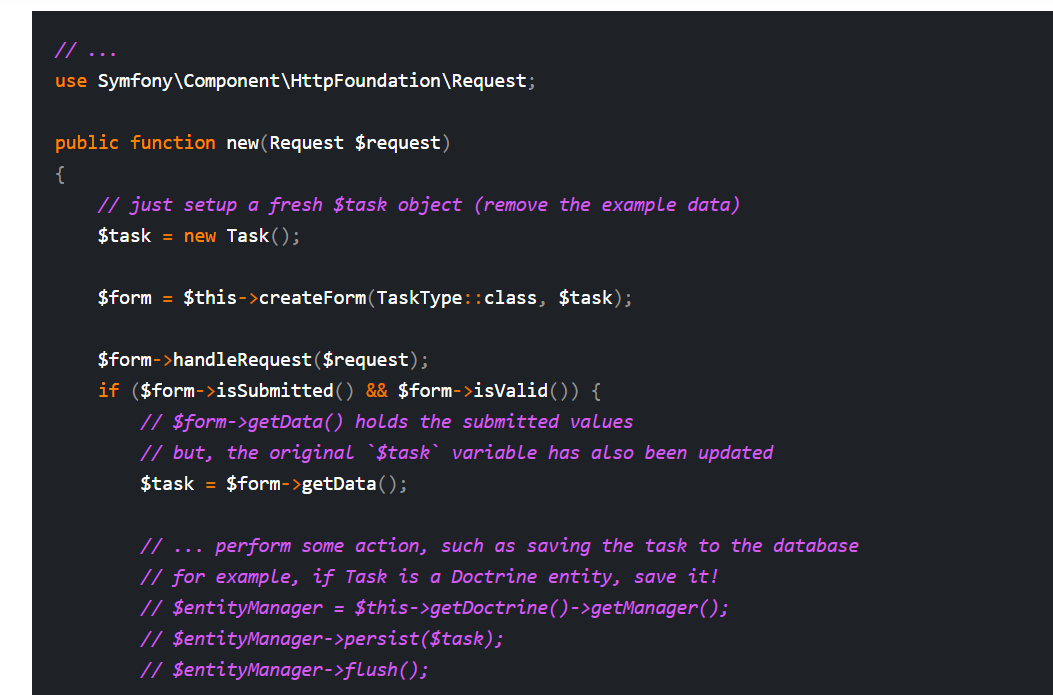
Les [thèmes de formulaire Symfony intégrés](https://symfony.com/doc/current/form/form_themes.html#symfony-builtin-forms) Incluent Bootstrap 3 et 4 et Fondation 5. Vous pouvez également [créer votre propre thème de formulaire Symfony](https://symfony.com/doc/current/form/form_themes.html#create-your-own-form-theme) .

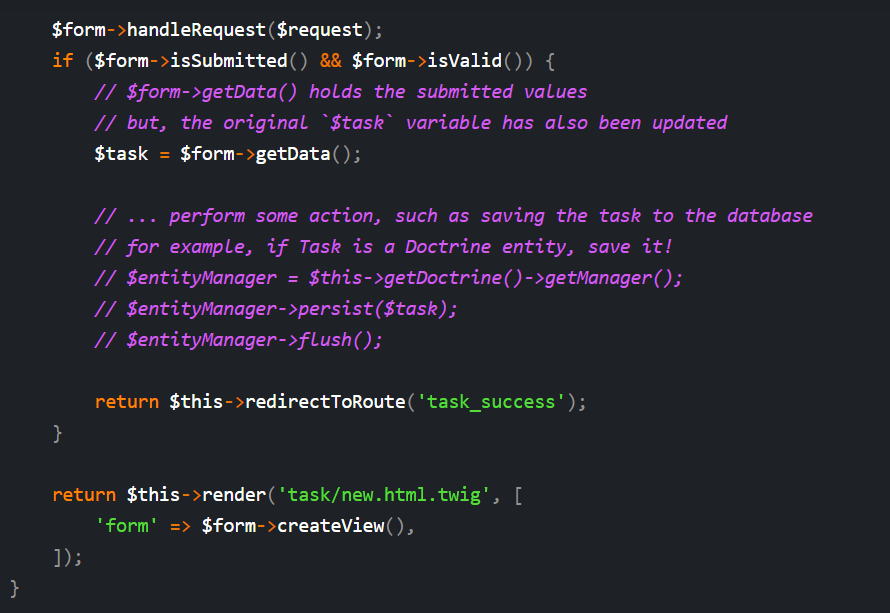
En plus des thèmes de formulaire, Symfony vous permet de [personnaliser la façon dont les champs sont rendus](https://symfony.com/doc/current/form/form_customization.html) avec plusieurs fonctions pour rendre chaque partie de champ séparément (widgets, étiquettes, erreurs, messages d'aide, etc.)

Traitement des formulaires

La [méthode recommandée de traitement des formulaires](https://symfony.com/doc/current/best_practices.html#best-practice-handle-form) consiste à utiliser une seule action pour rendre le formulaire et gérer la soumission du formulaire. Vous pouvez utiliser des actions distinctes, mais l'utilisation d'une seule action simplifie tout tout en gardant le code concis et maintenable.

Le traitement d'un formulaire signifie la conversion des données soumises par l'utilisateur vers les propriétés d'un objet. Pour ce faire, les données soumises par l'utilisateur doivent être écrites dans l'objet de formulaire :





Ce contrôleur suit un modèle commun pour la gestion des formulaires et a trois chemins possibles :

Lors du chargement initial de la page dans un navigateur, le formulaire n'a pas encore été soumis et $form->isSubmitted()retourne false. Ainsi, le formulaire est créé et rendu ;

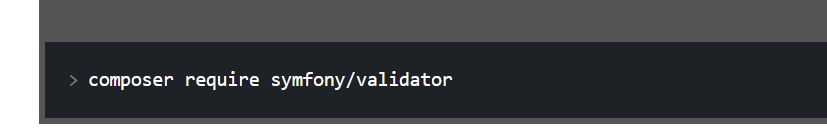
Lorsque l'utilisateur soumet le formulaire, le [handleRequest()](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Form/FormInterface.php) reconnaît et réécrit immédiatement les données soumises dans les propriétés tasket dueDatede l' $taskobjet. Cet objet est ensuite validé (la validation est expliquée dans la section suivante). S'il n'est pas valide, [isValid()](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Form/FormInterface.php)retourne false et le formulaire est rendu à nouveau, mais maintenant avec des erreurs de validation ;

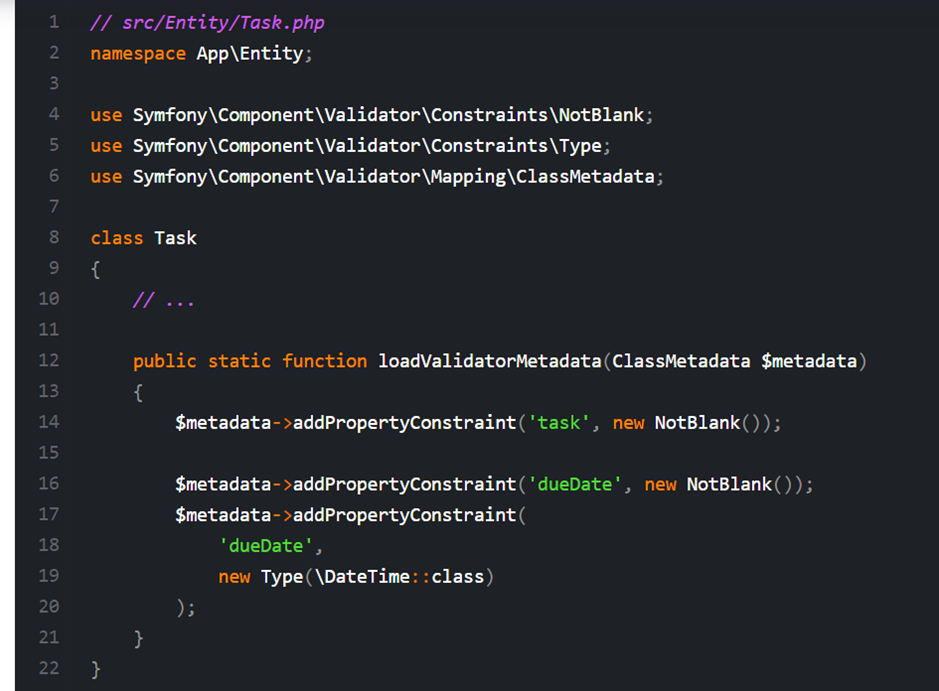
Lorsque l'utilisateur soumet le formulaire avec des données valides, les données soumises sont à nouveau écrites dans le formulaire, mais cette fois [isValid()](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Form/FormInterface.php) revient true. Vous avez maintenant la possibilité d'effectuer certaines actions en utilisant l' $taskobjet (par exemple le persister dans la base de données) avant de rediriger l'utilisateur vers une autre page (par exemple une page de « merci » ou de « succès ») ;

**Validation des formulaires** [**¶**](https://symfony.com/doc/current/forms.html#validating-forms)

Dans la section précédente, vous avez appris comment un formulaire peut être soumis avec des données valides ou non valides. Dans Symfony, la question n'est pas de savoir si le « formulaire » est valide, mais si l'objet sous-jacent ( $taskdans cet exemple) est valide après que le formulaire lui a appliqué les données soumises. L'appel $form->isValid()est un raccourci qui demande à l' $taskobjet s'il possède ou non des données valides.

Avant d'utiliser la validation, ajoutez-en la prise en charge dans votre application :

La validation se fait en ajoutant un ensemble de règles (appelées contraintes) à une classe. Pour voir cela en action, ajoutez des contraintes de validation afin que le taskchamp ne puisse pas être vide et le dueDatechamp ne puisse pas être vide et doit être un objet DateTime valide.

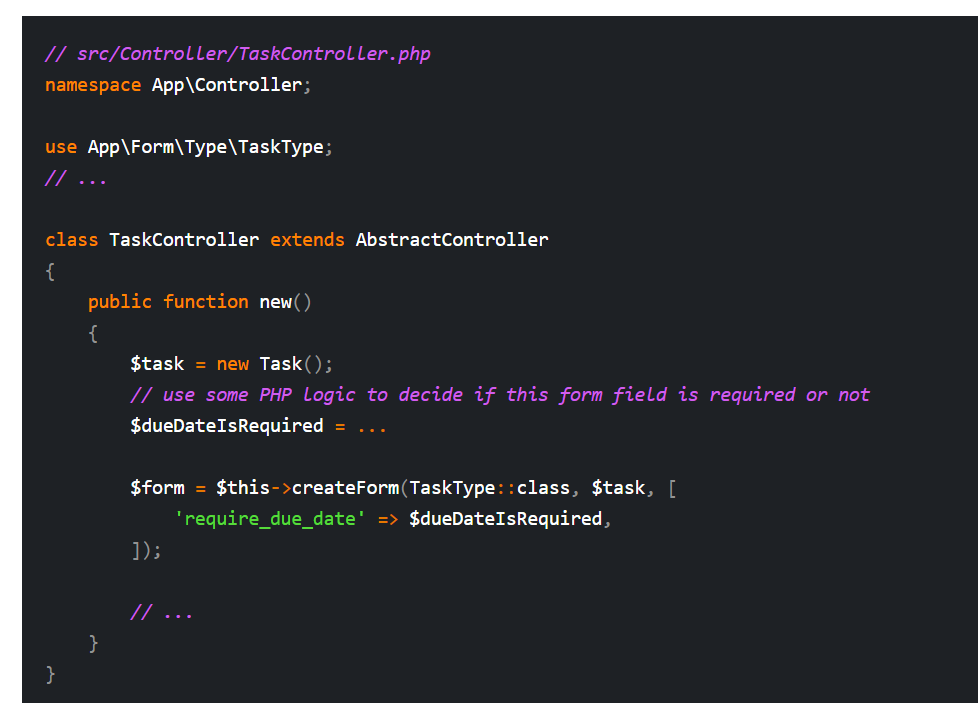


C'est tout! Si vous renvoyez le formulaire avec des données non valides, vous verrez les erreurs correspondantes imprimées avec le formulaire. Lisez la [documentation de validation Symfony](https://symfony.com/doc/current/validation.html) Pour en savoir plus sur cette puissante fonctionnalité.

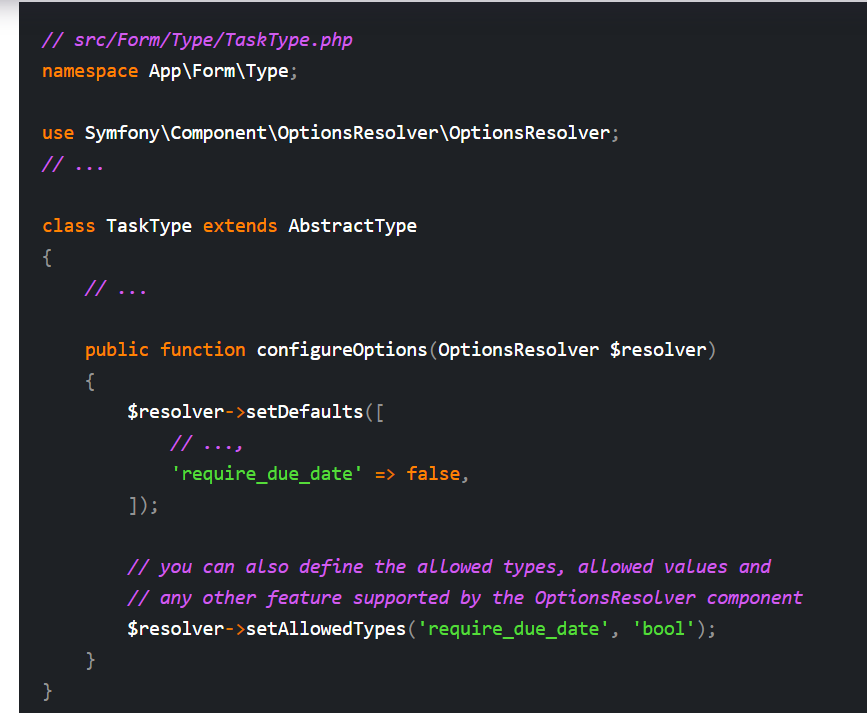
Autres fonctionnalités de formulaire courantes

Passer des options aux formulaires

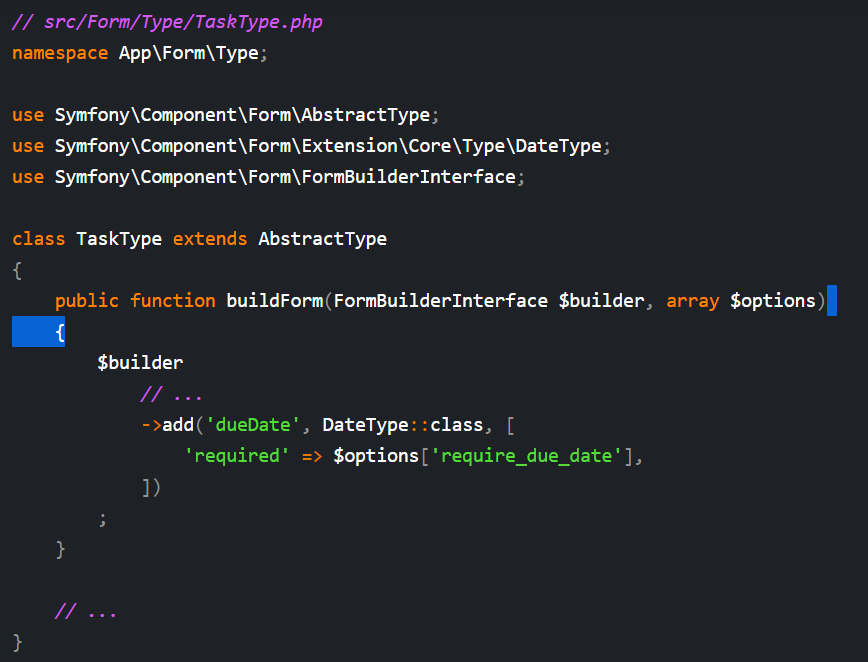
Si vous [créez des formulaires dans des classes](https://symfony.com/doc/current/forms.html#creating-forms-in-classes) , lors de la création du formulaire dans le contrôleur, vous pouvez lui passer des options personnalisées comme troisième argument facultatif de createForm() :



Si vous essayez d'utiliser le formulaire maintenant, vous verrez un message d'erreur : L'option "require\_due\_date" n'existe pas. C'est parce que les formulaires doivent déclarer toutes les options qu'ils acceptent en utilisant la configureOptions()méthode :



Vous pouvez maintenant utiliser cette nouvelle option de formulaire dans la buildForm()méthode:



Options de type de formulaire

Chaque [type de formulaire](https://symfony.com/doc/current/forms.html#form-types) possède un certain nombre d'options pour le configurer, comme expliqué dans la [référence des types de formulaires Symfony](https://symfony.com/doc/current/reference/forms/types.html) . Les deux options couramment utilisées sont requiredet label.

**L' requiredoption**

L'option la plus courante est l' requiredoption, qui peut être appliquée à n'importe quel champ. Par défaut, cette option est définie sur true, ce qui signifie que les navigateurs compatibles HTML5 devront remplir tous les champs avant de soumettre le formulaire.

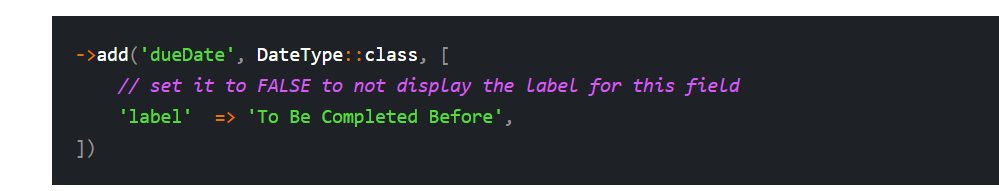
Si vous ne souhaitez pas ce comportement, [désactivez la validation côté client](https://symfony.com/doc/current/forms.html#forms-html5-validation-disable) pour l'ensemble du formulaire ou définissez l' requiredoption false sur un ou plusieurs champs :



L' requiredoption n'effectue aucune validation côté serveur. Si un utilisateur soumet une valeur vide pour le champ (avec un ancien navigateur ou un service Web, par exemple), elle sera acceptée comme valeur valide, sauf si vous utilisez également Symfony NotBlankou NotNullles contraintes de validation.

**L' labeloption**

Par défaut, le libellé des champs de formulaire est la version humanisée du nom de propriété ( user-> User; postalAddress-> ). Définissez l' option sur les champs pour définir explicitement leurs étiquettes:Postal Addresslabel

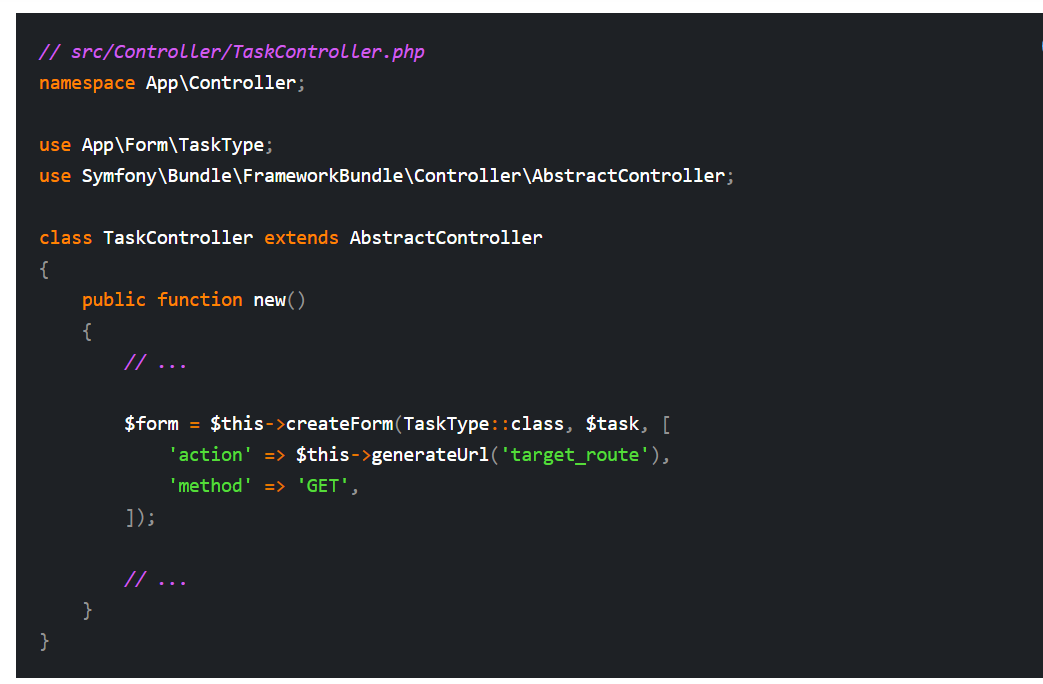


Changer l'action et la méthode HTTP

Par défaut, un formulaire sera envoyé via une requête HTTP POST à la même URL sous laquelle le formulaire a été rendu. Lors de la création du formulaire dans le contrôleur, utilisez les méthodes setAction()et setMethod()pour modifier cela :



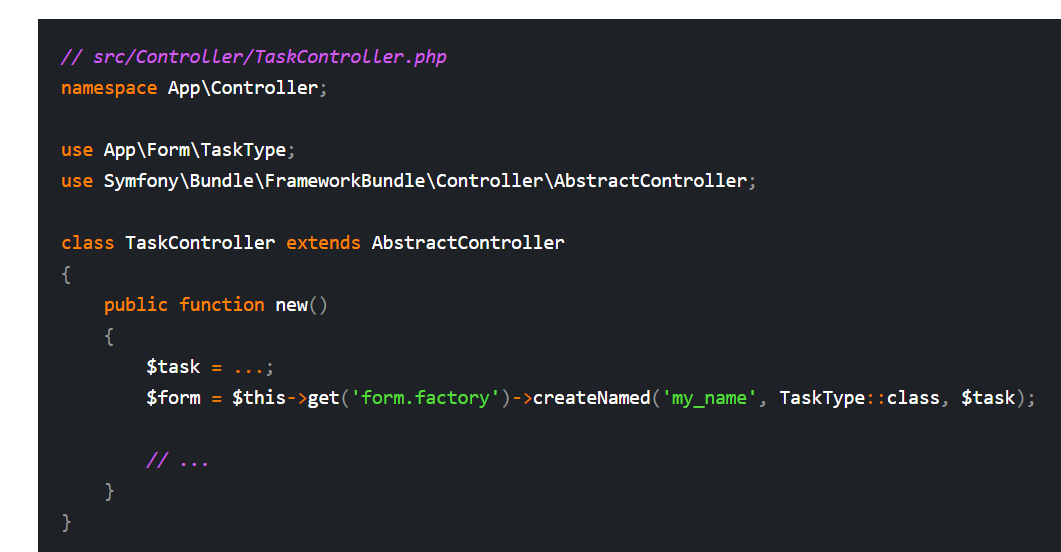
Lors de la création du formulaire dans une classe, transmettez l'action et la méthode comme options de formulaire:



Enfin, vous pouvez remplacer l'action et la méthode dans le modèle en les transmettant à la form()ou aux form\_start()fonctions d'assistance:

Si vous inspectez le contenu HTML du formulaire rendu, vous verrez que le <form>nom et les noms de champ sont générés à partir du nom de la classe de type (par exemple et ).<form name="task" ...><select name="task[dueDate][date][month]" ...>

Si vous souhaitez modifier cela, utilisez la [createNamed()](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Form/FormFactoryInterface.php) méthode:

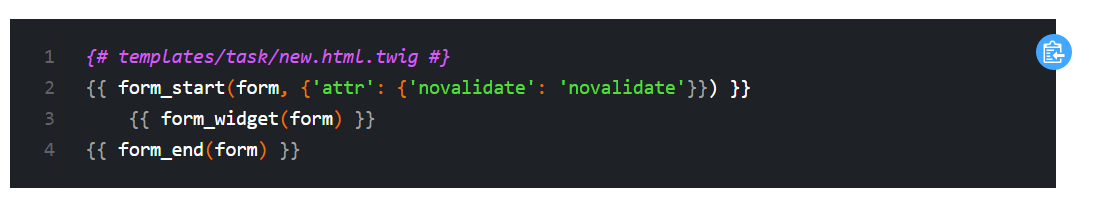


Vous pouvez même supprimer complètement le nom en le définissant sur une chaîne vide.

**Validation HTML côté client**

Grâce à HTML5, de nombreux navigateurs peuvent appliquer nativement certaines contraintes de validation côté client. La validation la plus courante est activée en ajoutant un requiredattribut aux champs obligatoires. Pour les navigateurs prenant en charge HTML5, cela entraînera l'affichage d'un message de navigateur natif si l'utilisateur essaie de soumettre le formulaire avec ce champ vide.

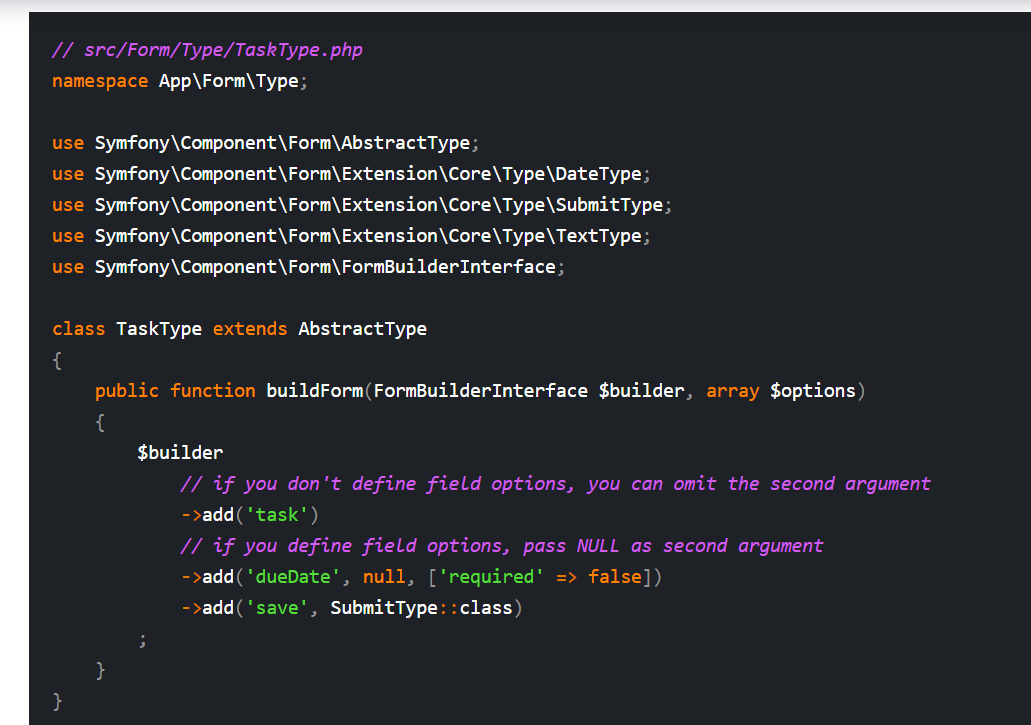
Les formulaires générés tirent pleinement parti de cette nouvelle fonctionnalité en ajoutant des attributs HTML sensibles qui déclenchent la validation. Cependant, la validation côté client peut être désactivée en ajoutant l' novalidateattribut à la <form>balise ou formnovalidateà la balise d'envoi. Cela est particulièrement utile lorsque vous souhaitez tester vos contraintes de validation côté serveur, mais que votre navigateur ne peut pas, par exemple, soumettre des champs vides.



**Type de formulaire Deviner**

Si l'objet géré par le formulaire comprend des contraintes de validation, Symfony peut introspecter ces métadonnées pour deviner le type de votre champ et le configurer pour vous. Dans l'exemple ci-dessus, Symfony peut deviner à partir des règles de validation que le taskchamp est un TextTypechamp normal et le dueDatechamp est un DateTypechamp.

Lors de la construction du formulaire, omettez le deuxième argument de la add()méthode, ou passez- nully, pour activer le «mécanisme de devinettes» de Symfony:



**Options de type de formulaire Deviner**

Lorsque le mécanisme de devinettes est activé pour certains champs (c'est-à-dire que vous omettez ou passez nullcomme deuxième argument add()), en plus de son type de formulaire, les options suivantes peuvent également être devinées:

required

L' requiredoption peut être devinée en fonction des règles de validation (c'est-à-dire le champ NotBlankou NotNull) ou des métadonnées Doctrine (c'est-à-dire le champ nullable). Ceci est très utile, car votre validation côté client correspondra automatiquement à vos règles de validation.

maxlength

Si le champ est une sorte de champ de texte, l' maxlengthattribut option peut être deviné à partir des contraintes de validation (si Lengthou Rangeest utilisé) ou des métadonnées [Doctrine](https://symfony.com/doc/current/doctrine.html) (via la longueur du champ).

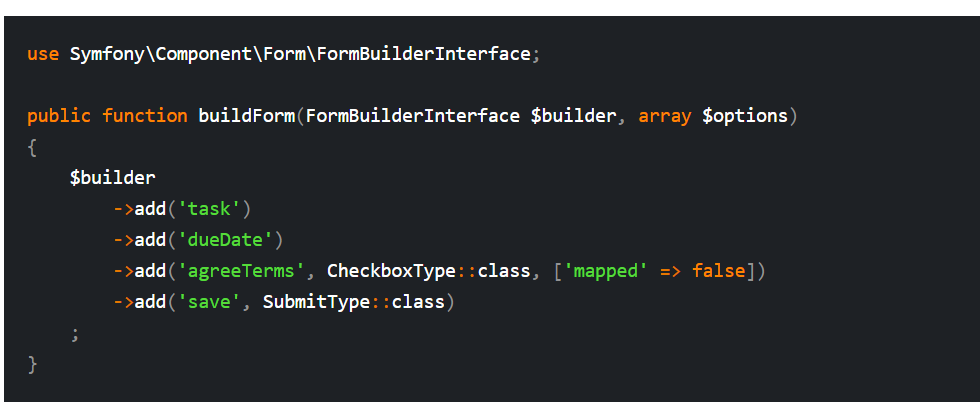
Si vous souhaitez modifier l'une des valeurs devinées, remplacez-la en passant l'option dans le tableau de champs d'options :



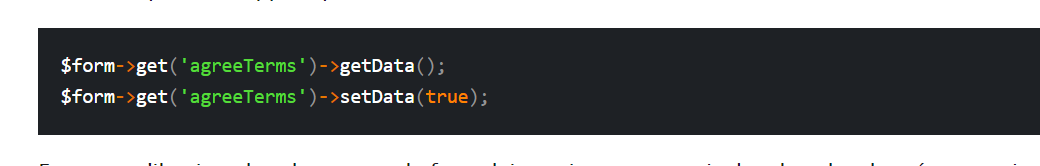
**Champs non mappés**

Lors de la modification d'un objet via un formulaire, tous les champs du formulaire sont considérés comme des propriétés de l'objet. Tous les champs du formulaire qui n'existent pas sur l'objet provoquent la levée d'une exception.

Si vous avez besoin de champs supplémentaires dans le formulaire qui ne seront pas stockés dans l'objet (par exemple pour ajouter une case à cocher «Je suis d'accord avec ces termes» ), définissez l' mapped option falsedans ces champs:

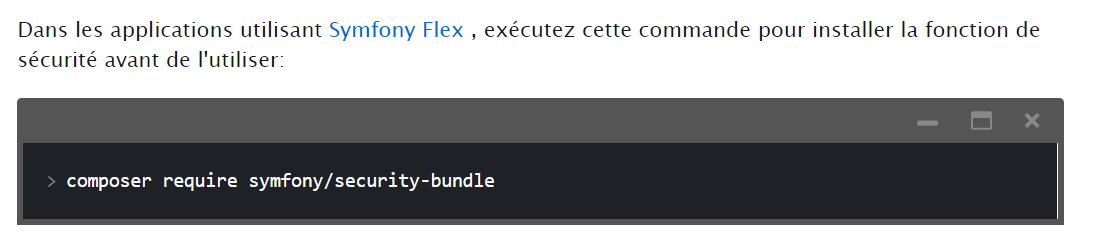


Ces «champs non mappés» peuvent être définis et accessibles dans un contrôleur avec:



En outre, s'il existe des champs sur le formulaire qui ne sont pas inclus dans les données soumises, ces champs seront explicitement définis sur null.

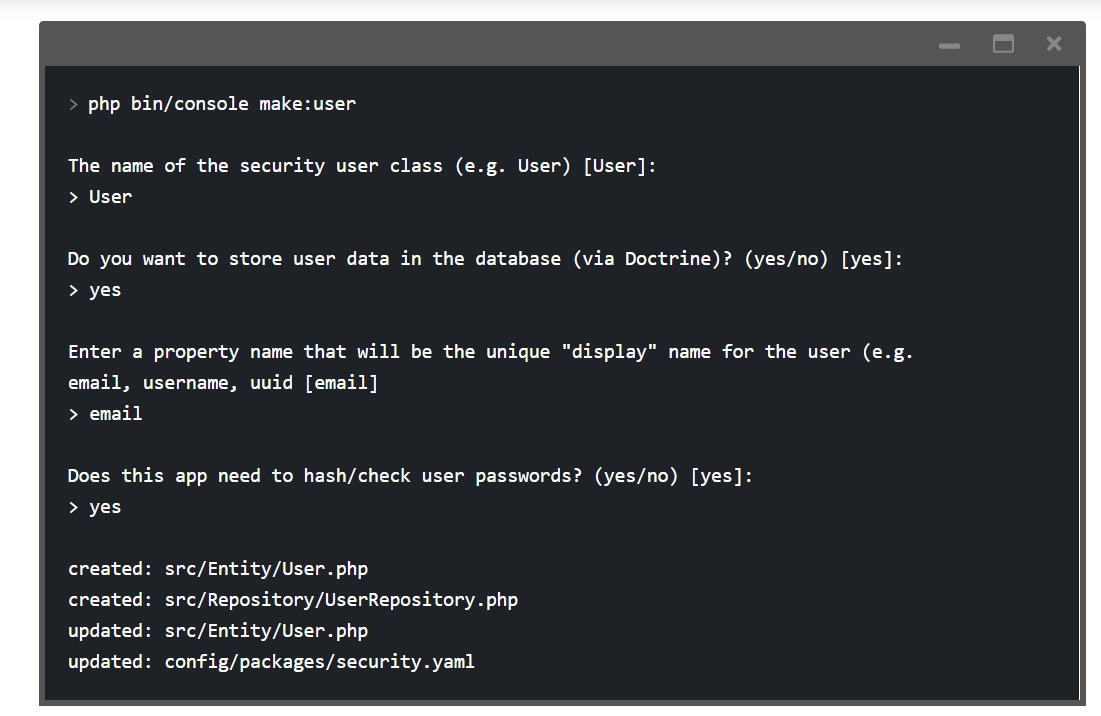
**SECURITE**



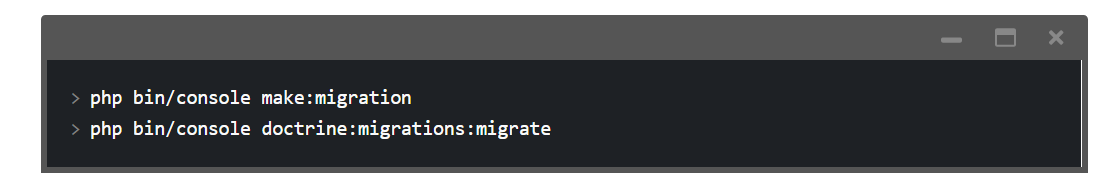
**CREATION DE LA CLASSE D’UTILISATION**

Peu importe comment vous vous authentifierez (par exemple, formulaire de connexion ou jetons d'API) ou où vos données d'utilisateur seront stockées (base de données, authentification unique), l'étape suivante est toujours la même : créez une classe « Utilisateur ». La façon la plus simple est d'utiliser le [MakerBundle](https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html) .

Supposons que vous souhaitiez stocker vos données utilisateur dans la base de données avec Doctrine:



C'est tout ! La commande pose plusieurs questions afin de générer exactement ce dont vous avez besoin. Le plus important est le User.phpfichier lui-même. La seule règle concernant votre Userclasse est qu'elle doit être implémentée [UserInterface](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Security/Core/User/UserInterface.php). N'hésitez pas à ajouter tout autre champ ou logique dont vous avez besoin. Si votre Userclasse est une entité (comme dans cet exemple), vous pouvez utiliser la [commande make: entity](https://symfony.com/doc/current/doctrine.html#doctrine-add-more-fields) pour ajouter plus de champs. Assurez-vous également d'effectuer et d'exécuter une migration pour la nouvelle entité:



**Le « fournisseur d'utilisateur »**

En plus de votre Userclasse, vous avez également besoin d'un « fournisseur d'utilisateur » : une classe qui aide avec quelques choses, comme le rechargement des données utilisateur de la session et certaines fonctionnalités optionnelles, comme se [souvenir de moi](https://symfony.com/doc/current/security/remember_me.html) et l' [emprunt d'identité](https://symfony.com/doc/current/security/impersonating_user.html) .

Heureusement, la make:usercommande en a déjà configuré un dans votre security.yamlfichier sous la providersclé.

Si votre Userclasse est une entité, vous n'avez rien d'autre à faire. Mais si votre classe n'est pas une entité, elle make:useraura également généré une UserProviderclasse que vous devez terminer. En savoir plus sur les fournisseurs d'utilisateurs ici : [Fournisseurs d'utilisateurs](https://symfony.com/doc/current/security/user_provider.html) .

**2c) Encodage des mots de passe**

Toutes les applications n'ont pas « d'utilisateurs » qui ont besoin de mots de passe. Si vos utilisateurs ont des mots de passe, vous pouvez contrôler la façon dont ces mots de passe sont encodés security.yaml. La make:user commande pré-configurera ceci pour vous:



**g - Sécurité**

Comme beaucoup de frameworks, Symfony comprend des fonctionnalités de sécurité. Symfony est plutôt robuste par défaut, mais ce serait tout de même une erreur de s’appuyer entièrement sur cette réputation pour assurer la sécurité d’une application. Parce que chaque application est différente, le contexte de sécurité est différent lui aussi. Les développeurs ont donc toujours un rôle important à jouer pour s’assurer que tout est bien configuré et testé.

Filtrage des inputs, authentification, gestion des sessions… Cette série d’articles aborde les différentes mécaniques que tout développeur Symfony peut utiliser, afin d’assurer la sécurité et la fiabilité de l’application qu’il développe. Et parce que “built-in security” ne signifie pas que tout est déjà fait, nous verrons quelles sont les choses réellement importantes, et les pièges classiques.

Protéger votre application Symfony contre les injections Protéger votre application Symfony contre les injections est assez simple.

Comme pour tout framework ou langage de programmation, la règle numéro #1 est la validation des inputs : Vous vous attendez à ce qu’un nombre soit fourni par l’utilisateur dans un de vos formulaires ? Vérifiez que la donnée fournie est effectivement un nombre. Par chance, Symfony propose une façon simple de mettre en place des contraintes, et de créer des contraintes personnalisées.

• Contraintes de validation (“validation constraints”) : <http://symfony.com/doc/current/reference/constraints.html>

• Créer des contraintes personnalisées : http://symfony.com/doc/current/cookbook/validation/custom\_constraint.html

TOUTES les inputs doivent êtres filtrées. Champs de formulaires, paramètres d’URLs, mais également les cookies ou les headers HTTP, si certains d’entre eux sont traités par votre application. Un enseignement tiré des audits de sécurité que nous avons réalisés sur des applications Symfony est que la plupart des applications procèdent à des validations d’inputs. Mais quelques paramètres passent généralement à la trappe. Malheureusement une erreur suffit pour ouvrir une faille. Une bonne pratique consiste donc à maintenir une liste des inputs utilisés par l’application, et leur contrôles de sécurité correspondants (par exemple dans les spécifications de l’application).

Concernant la base de données, utiliser les mécaniques proposées par Symfony est une bonne pratique.

Si vous utilisez Doctrine (qui est fourni avec la version standard de Symfony), alors vous êtes protégé, à minima pour les requêtes “classiques” sur des objets. Lors de l’utilisation de DQL pour des requêtes un peu plus personnalisées, vous devez vous assurer que les valeurs provenant d’entrées utilisateurs sont bien communiquées comme des placeholders, avec la méthode setParameter. Ainsi, les requêtes personnalisées deviennent des “requêtes paramétrées”, et évitent les injections SQL. DQL: http://symfony.com/doc/current/book/doctrine.html#querying-for-objects-with-dql

Doctrine n’est pas la seule solution, les autres ORMs comme Propel protègent également des injections SQL, si vous les utilisez correctement.

Si vous envoyez des commandes vers le système d’exploitation (OS commands) ou d’autres requêtes sensibles, vous devez vous assurer que les données passées en paramètres sont bien également “échappées” (input escaping), en fonction du contexte.

Gérer l’authentification et les sessions de manière sécurisée L’authentification est un sujet assez fourni. Nous n’allons pas parcourir Le fichier biblio.php, placé à la racine de l'application web, doit récupérer les données, initialiser Twig, stocker la configuration, charger et compiler le template biblio.twig et retourner le résultat au navigateur.

Ensuite, certains points peuvent être renforcés, en particulier concernant la protection contre les attaques brute-force ou attaques par dictionnaire. Mettre en place une politique de mots de passe solide est une première stratégie, afin d’éviter que des attaquants trouvent les mots de passe les plus évidents. Trouver le juste équilibre entre commodité et sécurité est le plus difficile ! Si votre application gère des données très sensibles, ajouter un pare-feu applicatif (WAF – Web Application Firewall) est une option à considérer. En fonction de votre budget et contexte, différentes solutions sont possibles. Certains pare-feux sont capables de détecter et bloquer des attaquants, même si leur adresse IP change.

Protéger une application Symfony contre le Cross Site Scripting (XSS) Deux règles de base contre le XSS : 1. filtrer les entrées (inputs) 2. échapper les données en sortie (output)

Filtrer les entrées peut être fait avec les contraintes, comme détaillé précédemment dans la section injections. Vous devez d’abord vous assurer que vous recevez le bon type de données de vos utilisateurs. Mais pour les données “complexes”, comme des champs de texte libre (pour un commentaire ou une description, par exemple), l’utilisateur peut fournir presque n’importe quel type de données. Si vous devez accepter certains caractères spéciaux, ou même des balises html (attention !), alors vous pouvez utiliser des “data transformers” pour implémenter des traitements de données puissants et spécifiques. Implémenter des data transformers : http://symfony.com/doc/current/cookbook/form/data\_transformers.html

Échapper les données en sortie est une obligation. Les moteurs de templates comme Twig font un bon travail dans ce domaine, en vous permettant d’échapper les données (avec la syntaxe {{ user.username|escape }} par exemple). En savoir plus sur ce sujet : http://symfony.com/doc/current/book/templating.html#outputescaping-in-twig

Les headers Content Security Policy sont aussi des solutions que vous pouvez implémenter sur votre projet, pour donner des instructions claires aux navigateurs web. Ces headers sont généralement implémentés au niveau du serveur (nginx, httpd…) ou directement dans Symfony avec HttpFoundation. En savoir plus sur la CSP : https://developer.mozilla.org/en-US/docs/Web/Security/CSP Et plus généralement sur les headers HTTP relatifs à la sécurité : Sécurisez votre site web avec les headers HTTP

Encore une fois, appliquer ces bonnes pratiques est une bonne chose, mais les appliquer partout dans votre application est un must.

**h - Template : TWIG**

**PHASE 1 – INSTALLATION**

Twig est le moteur de template intégré dans le Framework Symfony.

Pour exécuter le moteur de template, il faut un serveur web Apache avec PHP qui ne doit pas être antérieur à la version 5.2.7.

Si le gestionnaire de paquets Composer est déjà installé, se placer dans le répertoire de l’application web (dans cet exemple, /var/www/html/appli), puis exécuter la commande composer pour télécharger la dernière version de Twig :

$ cd /var/www/html/appli

$ composer require "twig/twig:~1.0"

Sinon, créer un répertoire TWIG, et se placer dans ce répertoire. Télécharger et installer composer, puis télécharger Twig. Enfin, placer le répertoire vendor dans le répertoire appli à la racine de l’application web, c'est-à-dire /var/www/html :

$ mkdir TWIG

$ cd TWIG

$ curl -s http://getcomposer.org/installer | php

$ php composer.phar require "twig/twig:~1.0"

$ mv vendor /var/www/html/appli

Le répertoire du projet web contient à présent un répertoire vendor comportant la bibliothèque Twig ainsi qu’un script autoload.php. La version de Twig testée dans ce document est la 1.26.1 du 5 octobre 2016.

Attention aux droits : Twig utilise un répertoire cache pour stocker les fichiers PHP générés à partir des templates et donc le répertoire /var/www/html/appli/cache doit être accessible en écriture pour le serveur web.

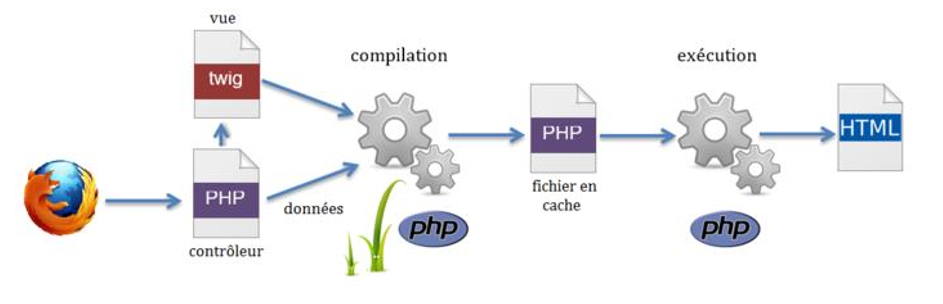
**PHASE 2 - CRÉATION D'UN TEMPLATE**

Le template est utilisé pour générer l'interface utilisateur, car séparer la présentation des traitements facilite le développement du projet et sa maintenance. La page web reçue par le navigateur est générée côté serveur à partir de deux fichiers :

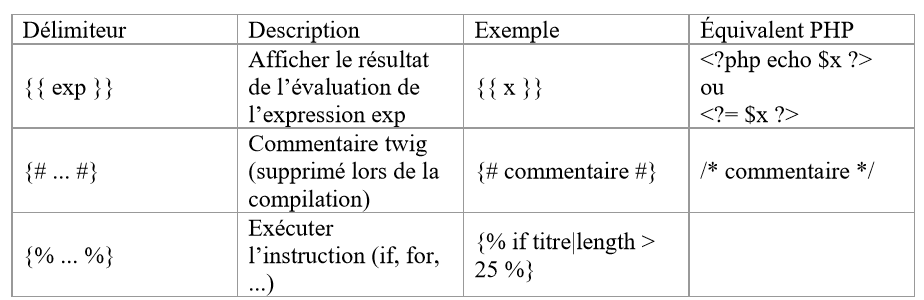
- Le template ou gabarit (vue) est un fichier avec l'extension twig, qui contient des parties statiques (code HTML) ainsi que des parties dynamiques permettant de sélectionner et intégrer des données dans la page HTML. Il utilise un langage simple et concis pour parcourir les données et extraire l’information à afficher (boucles, conditions, variables, fonctions, filtres).

- Un script PHP (contrôleur) qui définit les données qui seront passées au template.

Le script PHP appelé par le navigateur, récupère les données et les met à disposition du template (figure 1). Le template est analysé et compilé par le moteur de template en un fichier

PHP. Ce dernier est placé dans le répertoire de cache afin de limiter la consommation de ressources sur le serveur. Une fois ce fichier exécuté, le code HTML est généré et transmis au navigateur.

**Fig.1 : Principe de fonctionnement de Twig.**

Nous allons créer un fichier biblio.twig qui affichera les données transmises par le script biblio.php, qui sera écrit dans la phase suivante. Placer le fichier biblio.twig dans le répertoire vue de l’application web. Il comportera du code HTML ainsi que des parties dynamiques marquées par deux types de délimiteurs {{ exp }} et {# ... #}. Le premier affiche le résultat de l’évaluation de l’expression exp, ce qui permettra d'afficher les données de manière dynamique. Le second ajoute une ligne de commentaire.

Le fichier biblio.twig (figure 2) contient un code HTML statique, ainsi que du code twig surligné en rouge. Les parties dynamiques du code entre délimiteur double accolades seront remplacées lors de l'exécution par les valeurs des variables titre\_doc, titre\_page et date. Il est possible d'écrire des commentaires dans les templates en les plaçant entre accolades et dièses. Le commentaire dans ce fichier indique l'emplacement où les titres et noms de livres seront mis en page dans les phases suivantes.



**Fig.2 : Template biblio.twig.**

**PHASE 3 - CRÉATION DU CODE PHP**

Le fichier biblio.php, placé à la racine de l'application web, doit récupérer les données, initialiser Twig, stocker la configuration, charger et compiler le template biblio.twig et retourner le résultat au navigateur.

Pour simplifier l'exemple, la fonction getData()du script data.php, retourne des données statiques plutôt que d'interroger une base de données. Ce fichier, placé dans le répertoire modele, est inclus dans biblio.php. Twig travaille avec des tableaux de données. Dans le template Twig {{ titre\_doc }} fait référence à la valeur de la clé titre\_doc du tableau associatif des données.

<?php

// data.php

function getData(){

// tableau associatif des données

$data = array(

'titre\_doc' => 'Bibliotheque',

'titre\_page' => 'Liste des livres',

'date' => date("d/m/Y"),

// pour simplifier l'exemple, les données sont définies

// statiquement (généralement elles sont extraites d'une BD)

'biblio' => array(

array('titre'=>'N ou M', 'nom'=>'Christie', 'prenom'=>'Agatha'),

array('titre'=>'1984', 'nom'=>'Orwell', 'prenom'=>'George'),

array('titre'=>'Dune', 'nom'=> 'Herbert', 'prenom'=>'Frank')

)

);

return $data;

}

Une fois les données récupérées et stockées dans la variable $donnees dans biblio.php, le script inclut l'autoloader, puis définit le mode de chargement du template et son emplacement, en créant une instance de la classe Twig\_Loader\_Filesystem. Le template sera chargé depuis le chemin passé en paramètre, c'est-à-dire le répertoire vue placé au même niveau de l'arborescence que biblio.php. Nous avons défini deux tableaux d'options pour les phases de développement et de production. Pendant le développement, le système de cache est désactivé en attribuant la valeur false à l'option cache. Lorsque l'application sera en production, Twig stockera dans le répertoire cache les fichiers PHP générés. L'option autoescape permet de protéger automatiquement les sorties contre les attaques de type XSS.