1)NOTION DE FRAMEWORK

Tout d’abord, qu’est-ce qu’un framework ? Un framework est un “cadre de travail”, il permet de simplifier le travail des développeurs en leur offrant une boîte à outils permettant de gagner du temps dans le développement. En clair, il vous permet de réaliser des applications complexes, de façon structurée et rapide.

a) Etude comparative des différents FrameWork PHP

Symfony est donc un framework PHP (langage de programmation web) utilisé afin de développer des applications ou sites web. Il est considéré comme l’un des meilleurs framework dans la création d’application web car il permet d’obtenir plus de flexibilité. Il est également assez facile à prendre en main, grâce aux nombreuses documentations disponibles en ligne. De plus, l’outil de débogage sur Symfony est puissant. Le débogage est utilisé afin d’identifier et corriger vos fautes dans le code. Cet outil vous fait donc gagner un temps considérable dans la phase de développement de votre application.

De plus, la technologie Symfony intègre aussi des mesures de sécurité innées afin de lutter contre des failles et attaques XSS, CSRF et SQL. Ce sont des failles permettant d’injecter du code HTML dans des variables mal protégées. C’est aussi un avantage, par rapport à l’utilisation du simple PHP. En effet, lorsque que vous développez un site internet en PHP, vous êtes obligé(e) de protéger chaque formulaire manuellement. Grâce à Symfony, vous n’avez plus à vous soucier de cela, il le fait systématiquement pour vous. De ce fait, votre application ou site web est mieux protégé des pirates, par rapport aux CMS par exemple (WordPress, Wings ou Joomla) qui eux sont assez répandus et dont leur architecture est plus facilement attaquable.

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. Les problèmatiques auxquels répond le framework sont souvent les mêmes pour d'autres développeurs, le code est donc par la suite partagé avec la communauté des développeurs de PHP.

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 suivantes :

- 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.

Symfony 2 :

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, quand on regarde ce qui a changé, la réponse est principalement « TOUT ». 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 framework se professionnalise donc dans cette deuxième version avec des injections de dépendance, le concept de « Bundle » commence a arriver, d'ailleurs Symfony2 est un Bundle. Les ESI (Edge Side Include), le cache HTTP ou encore le support d'un reverse proxy comme Varnish font leurs arrivés ainsi qu'une barre de débug plus complète que la précédente.

Symfony 3 :

La troizième 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.

Composer

[Composer](https://getcomposer.org/) est le gestionnaire de packages utilisé par les applications PHP modernes. Utilisez Composer pour gérer les dépendances dans vos applications Symfony et installer les composants Symfony dans vos projets PHP.

Depuis la première publication de cet article, l'installation de Composer s'est beaucoup améliorée. Par conséquent, le contenu d'origine de cet article a été supprimé et vous êtes invité à installer Composer comme expliqué dans la documentation officielle de Composer.

Les avantages :

Les frameworks, en général, permettent un gain de productivité mais pas seulement. Ils permettent d'avoir une bonne organisation du code et on sait qu'un code bien organisé est plus maintenable et évolutif. L'utilisation de Symfony assure une longévité à nos projets, en effet, à partir du moment où un développeur a été formé sur Symfony, il pourra prendre en main n'importe quel projet même s'il n'a pas participé à son élaboration.

Comme dit précédemment, Symfony est open source, ce qui est un bon avantage. On n'a donc pas de contraintes imposées et on peut développer des solutions propriétaires.

Symfony est considéré comme un des framework PHP les plus puissants et les plus flexibles. Il permet également d'utiliser certains de ses composants sans utiliser obligatoirement le framework entier, ce qui offre une réelle flexibilité.

Symfony est très populaire, reconnu au niveau international. En effet, c'est un des frameworks les plus utilisés dans le monde, notamment dans les entreprises. Les exemples de sites développés sous Symfony ne manquent pas, parmi lesquels Yahoo !, Dailymotion.

Symfony bénéficie d'une bonne communauté ainsi on s'assure de ''ne jamais être tout seul avec l'écran''. En effet, grâce au support de cette communauté, on pourra toujours trouver des réponses à nos questions.

Symfony permet de construire des applications qui répondent précisément à nos besoins et respectant les standards de PHP.

Les inconvénients :

Un des principaux inconvénients d'un framework est le temps d'apprentissage qui est bien plus élevé que pour un langage en lui-même.

Symfony, plus particulièrement nécessite d'apprendre plusieurs autres technologies qui lui sont propres comme YAML ou Doctrine.

Symfony est un framework assez lourd et n'est pas idéal pour les petits projets.

Un des autres inconvénients de Symfony, pour ceux qui ne sont pas à l'aise avec, est qu'il faut utiliser des lignes de commandes, pour l'installer, pour créer une base de données.

INSTALLATION

[Composer](https://getcomposer.org/) est le gestionnaire de packages utilisé par les applications PHP modernes. Utilisez Composer pour gérer les dépendances dans vos applications Symfony et installer les composants Symfony dans vos projets PHP.

Depuis la première publication de cet article, l'installation de Composer s'est beaucoup améliorée. Par conséquent, le contenu d'origine de cet article a été supprimé et vous êtes invité à [installer Composer](https://getcomposer.org/download/) comme expliqué dans la documentation officielle de Composer.

SYMFONY

Installation

Avant de créer votre première application Symfony, vous devez:

* Installez PHP 7.2.5 ou supérieur et ces extensions PHP (qui sont installées et activées par défaut dans la plupart des installations PHP 7): [Ctype](https://www.php.net/book.ctype) , [iconv](https://www.php.net/book.iconv) , [JSON](https://www.php.net/book.json) , [PCRE](https://www.php.net/book.pcre) , [Session](https://www.php.net/book.session) , [SimpleXML](https://www.php.net/book.simplexml) et [Tokenizer](https://www.php.net/book.tokenizer) ;
* [Installez Composer](https://getcomposer.org/download/) , qui est utilisé pour installer les packages PHP.

En option, vous pouvez également [installer Symfony CLI](https://symfony.com/download) . Cela crée un binaire appelé symfonyqui fournit tous les outils dont vous avez besoin pour développer et exécuter votre application Symfony localement.

Le symfony binaire fournit également un outil pour vérifier si votre ordinateur répond à toutes les exigences. Ouvrez votre terminal de console et exécutez cette commande:

|  |
| --- |
| symfony check:requirements |

**REMARQUE**

Le binaire Symfony est développé en interne chez Symfony. Si vous souhaitez signaler un bug ou suggérer une nouvelle fonctionnalité, veuillez créer un problème sur [symfony / cli](https://github.com/symfony/cli) .

## **Création d'applications Symfony**[**¶**](https://symfony.com/doc/current/setup.html#creating-symfony-applications)

Ouvrez votre terminal de console et exécutez l'une de ces commandes pour créer une nouvelle application Symfony:

|  |
| --- |
| symfony new my\_project\_name --full  symfony new my\_project\_name |

La seule différence entre ces deux commandes est le nombre de packages installés par défaut. L' --fulloption installe tous les packages dont vous avez généralement besoin pour créer des applications Web, de sorte que la taille de l'installation sera plus grande.

Si vous n'utilisez pas le binaire Symfony, exécutez ces commandes pour créer la nouvelle application Symfony à l'aide de Composer:

|  |
| --- |
| composer create-project symfony/website-skeleton my\_project\_name  composer create-project symfony/skeleton my\_project\_name |

Quelle que soit la commande que vous exécutez pour créer l'application Symfony. Tous créeront un nouveau my\_project\_name/répertoire, y téléchargeront certaines dépendances et généreront même les répertoires et fichiers de base dont vous aurez besoin pour commencer. En d'autres termes, votre nouvelle application est prête!

**REMARQUE**

Le répertoire cache et logs du projet (par défaut, <project>/var/cache/ et <project>/var/log/) doit être accessible en écriture par le serveur Web. Si vous avez un problème, lisez comment [configurer les autorisations pour les applications Symfony](https://symfony.com/doc/current/setup/file_permissions.html) .

## **Exécuter des applications Symfony**[**¶**](https://symfony.com/doc/current/setup.html#running-symfony-applications)

En production, vous devez installer un serveur Web comme Nginx ou Apache et le [configurer pour exécuter Symfony](https://symfony.com/doc/current/setup/web_server_configuration.html) . Cette méthode peut également être utilisée si vous n'utilisez pas le serveur Web local Symfony pour le développement.

Cependant, pour le développement local, le moyen le plus pratique d'exécuter Symfony est d'utiliser le [serveur Web local](https://symfony.com/doc/current/setup/symfony_server.html) fourni par le symfony binaire. Ce serveur local fournit entre autres la prise en charge de HTTP / 2, les requêtes simultanées, TLS / SSL et la génération automatique de certificats de sécurité.

Ouvrez votre terminal de console, accédez à votre nouveau répertoire de projet et démarrez le serveur Web local comme suit:

|  |
| --- |
| cd my-project/  symfony server:start |

Ouvrez votre navigateur et accédez à http://localhost:8000/. Si tout fonctionne, vous verrez une page d'accueil. Plus tard, lorsque vous avez terminé de travailler, arrêtez le serveur en appuyant sur Ctrl+C depuis votre terminal.

**POINTE**

Le serveur Web fonctionne avec n'importe quelle application PHP, pas seulement avec les projets Symfony, c'est donc un outil de développement générique très utile.

## **Configurer un projet Symfony existant**[**¶**](https://symfony.com/doc/current/setup.html#setting-up-an-existing-symfony-project)

En plus de créer de nouveaux projets Symfony, vous travaillerez également sur des projets déjà créés par d'autres développeurs. Dans ce cas, il vous suffit d'obtenir le code du projet et d'installer les dépendances avec Composer. En supposant que votre équipe utilise Git, configurez votre projet avec les commandes suivantes:

|  |
| --- |
| cd projects/  git clone ...  cd my-project/  composer install |

Vous devrez probablement également personnaliser votre [fichier .env](https://symfony.com/doc/current/configuration.html#config-dot-env) et effectuer quelques autres tâches spécifiques au projet (par exemple, créer une base de données). Lorsque vous travaillez sur une application Symfony existante pour la première fois, il peut être utile d'exécuter cette commande qui affiche des informations sur le projet:

|  |
| --- |
| php bin/console about |

## **Installation de packages**[**¶**](https://symfony.com/doc/current/setup.html#installing-packages)

Une pratique courante lors du développement d'applications Symfony consiste à installer des packages (Symfony les appelle des [bundles](https://symfony.com/doc/current/bundles.html) ) qui fournissent des fonctionnalités prêtes à l'emploi. Les packages nécessitent généralement une configuration avant de les utiliser (modification d'un fichier pour activer le bundle, création d'un fichier pour ajouter une configuration initiale, etc.)

La plupart du temps, cette configuration peut être automatisée et c'est pourquoi Symfony inclut [Symfony Flex](https://github.com/symfony/flex) , un outil pour simplifier l'installation / la suppression des packages dans les applications Symfony. Techniquement parlant, Symfony Flex est un plugin Composer qui est installé par défaut lors de la création d'une nouvelle application Symfony et qui **automatise les tâches les plus courantes des applications Symfony** .

**POINTE**

Vous pouvez également [ajouter Symfony Flex à un projet existant](https://symfony.com/doc/current/setup/flex.html) .

Symfony Flex modifie le comportement des requière, update et des remove commandes Composer pour fournir des fonctionnalités avancées. Prénoms l'exemple suivant:

|  |
| --- |
| cd my-project/  composer require logger |

Si vous exécutez cette commande dans une application Symfony qui n'utilise pas Flex, vous verrez une erreur Composer expliquant qu'il loggerne s'agit pas d'un nom de package valide. Cependant, si l'application a installé Symfony Flex, cette commande installe et active tous les packages nécessaires pour utiliser l'enregistreur officiel Symfony.

Ceci est possible car de nombreux packages / bundles Symfony définissent des **«recettes»** , qui sont un ensemble d'instructions automatisées pour installer et activer les packages dans les applications Symfony. Flex garde une trace des recettes qu'il a installées dans un symfony.lockfichier, qui doit être validé dans votre référentiel de code.

Les recettes Symfony Flex sont apportées par la communauté et sont stockées dans deux référentiels publics:

* [Le référentiel de recettes principal](https://github.com/symfony/recipes) est une liste organisée de recettes pour des emballages de haute qualité et maintenus. Symfony Flex recherche uniquement dans ce référentiel par défaut.
* [Dépôt de recettes Contrib](https://github.com/symfony/recipes-contrib) , contient toutes les recettes créées par la communauté. Tous sont garantis pour fonctionner, mais leurs packages associés pourraient ne pas être maintenus. Symfony Flex vous demandera votre permission avant d'installer l'une de ces recettes.

Lisez la [documentation](https://github.com/symfony/recipes/blob/master/README.rst) de [Symfony Recipes](https://github.com/symfony/recipes/blob/master/README.rst) pour tout savoir sur la création de recettes pour vos propres packages.

### Packs Symfony [¶](https://symfony.com/doc/current/setup.html#symfony-packs)

Parfois, une seule fonctionnalité nécessite l'installation de plusieurs packages et bundles. Au lieu de les installer individuellement, Symfony fournit des **packs** , qui sont des métapaquets Composer qui incluent plusieurs dépendances.

Par exemple, pour ajouter des fonctionnalités de débogage dans votre application, vous pouvez exécuter la commande. Ceci installe le , qui à son tour installe plusieurs paquets comme , , , etc.composer require --dev debugsymfony/debug-packsymfony/debug-bundlesymfony/monolog-bundlesymfony/var-dumper

Par défaut, lors de l'installation des packs Symfony, votre composer.jsonfichier affiche la dépendance du pack (par exemple ) au lieu des packages réellement installés. Pour afficher les packages, ajoutez l' option lors de l'installation d'un pack (par exemple ) ou exécutez cette commande pour décompresser les packs déjà installés: (par exemple )."symfony/debug-pack": "^1.0"--unpackcomposer require debug --dev –unpack composer unpack PACK\_NAME composer unpack debug

## **Vérification des vulnérabilités de sécurité**[**¶**](https://symfony.com/doc/current/setup.html#checking-security-vulnerabilities)

Le symfony binaire créé lorsque vous [installez Symfony CLI](https://symfony.com/download) fournit une commande pour vérifier si les dépendances de votre projet contiennent une vulnérabilité de sécurité connue:

|  |
| --- |
| symfony check:security |

Une bonne pratique de sécurité consiste à exécuter cette commande régulièrement pour pouvoir mettre à jour ou remplacer les dépendances compromises dès que possible. Le contrôle de sécurité se fait localement en clonant la [base de données](https://github.com/FriendsOfPHP/security-advisories) publique [des avis de sécurité PHP](https://github.com/FriendsOfPHP/security-advisories) , afin que votre composer.lockfichier ne soit pas envoyé sur le réseau.

**POINTE**

La check: Security commande se termine avec un code de sortie différent de zéro si l'une de vos dépendances est affectée par une vulnérabilité de sécurité connue. De cette façon, vous pouvez l'ajouter à votre processus de construction de projet et à vos flux de travail d'intégration continue pour les faire échouer en cas de vulnérabilité.

## **Versions Symfony LTS**[**¶**](https://symfony.com/doc/current/setup.html#symfony-lts-versions)

Selon le [processus de publication de Symfony, des versions de](https://symfony.com/doc/current/contributing/community/releases.html) «prise en charge à long terme» (ou LTS pour faire court) sont publiées tous les deux ans. Consultez les [versions de Symfony](https://symfony.com/releases) pour savoir quelle est la dernière version de LTS.

Par défaut, la commande qui crée de nouvelles applications Symfony utilise la dernière version stable. Si vous souhaitez utiliser une version LTS, ajoutez l' –version option:

|  |
| --- |
| symfony new my\_project\_name --version=lts  symfony new my\_project\_name --version=next  symfony new my\_project\_name --version=4.4 |

Les raccourcis ltset nextne sont disponibles que lorsque vous utilisez Symfony pour créer de nouveaux projets. Si vous utilisez Composer, vous devez indiquer la version exacte:

|  |
| --- |
| composer create-project symfony/website-skeleton:^4.4 my\_project\_name |

## **L'application Symfony Demo**[**¶**](https://symfony.com/doc/current/setup.html#the-symfony-demo-application)

[L'application de démonstration Symfony](https://github.com/symfony/demo) est une [application](https://github.com/symfony/demo) entièrement fonctionnelle qui montre la manière recommandée de développer des applications Symfony. C'est un excellent outil d'apprentissage pour les nouveaux arrivants de Symfony et son code contient des tonnes de commentaires et de notes utiles.

Exécutez cette commande pour créer un nouveau projet basé sur l'application Symfony Demo:

|  |
| --- |
| symfony new my\_project\_name --demo |

## **Commencez à coder!**[**¶**](https://symfony.com/doc/current/setup.html#start-coding)

Avec la configuration derrière vous, il est temps de [créer votre première page dans Symfony](https://symfony.com/doc/current/page_creation.html) .

ROLE

Les rôles sont des moyens de charger automatiquement certains fichiers vars, tâches et gestionnaires en fonction d'une structure de fichier connue. Le regroupement du contenu par rôles permet également de partager facilement les rôles avec d'autres utilisateurs.

# **Installation du Projet**

1.4

Dans symfony, les **applications** partageant le même modèle de données sont regroupés dans des **projets**. Pour la plupart des projets, vous avez deux applications différentes : un frontend et un backend.

## **Création du projet**

Depuis le répertoire sfproject/, exécuter la tâche symfony generate:project pour créer le projet symfony:

$ php lib/vendor/symfony/data/bin/symfony generate:project PROJECT\_NAME

Sur Windows:

c:\> php lib\vendor\symfony\data\bin\symfony generate:project PROJECT\_NAME

La tâche generate:project génére la structure par défaut des répertoires et les fichiers nécessaires pour un projet symfony :

| **Répertoire** | **Description** |
| --- | --- |
| apps/ | Accueille toutes les applications du projet |
| cache/ | Les fichiers mis en cache par le framework |
| config/ | Les fichiers de configuration du projet |
| data/ | Les fichiers de données comme les jeux de données initiales |
| lib/ | Les bibliothèque et les classes du projet |
| log/ | Les fichiers log du framework |
| plugins/ | Les plugins installés |
| test/ | Les fichiers de test unitaire et fonctionnel |
| web/ | Le répertoire racine Web (voir ci-dessous) |

**NOTE**

Pourquoi symfony génère autant de dossiers ? L'un des principaux avantages d'un framework full-stack consiste à normaliser les développements. Grâce à la structure par défaut des fichiers et des répertoires de symfony, tout développeur ayant une certaine connaissance de symfony peut prendre en charge la maintenance d'un projet symfony. En quelques minutes, il sera capable de parcourir le code, de corriger des bugs et d'ajouter de nouvelles fonctionnalités.

La tâche generate: project a également créé un raccourci symfony dans le répertoire racine du projet pour diminuer le nombre de caractères que vous allez écrire lors de l'exécution d'une tâche.

Ainsi, à partir de maintenant, au lieu d'utiliser le chemin complet pour le programme symfony, vous pouvez utiliser le raccourci symfony.

### Vérification de l'installation

Maintenant que symfony est correctement installé, vérifiez la en utilisant la ligne de commande de symfony pour afficher la version de symfony (notez que le V est en majuscule) :

$ cd ../..

$ php lib/vendor/symfony/data/bin/symfony -V

Sur Windows:

c:\> cd ..\..

c:\> php lib\vendor\symfony\data\bin\symfony -V

L'option -V affiche également le chemin vers le répertoire d'installation de symfony, qui est stocké dans config/ProjectConfiguration.class.php.

Si le chemin de symfony est absolue (ce qui ne devrait pas l'être par défaut si vous suivez les instructions ci-dessus), changez-le de sorte qu'il soit lu comme l'exemple suivant pour une meilleure portabilité :

*// config/ProjectConfiguration.class.php*

require\_once dirname(\_\_FILE\_\_).'/../lib/vendor/symfony/lib/autoload/sfCoreAutoload.class.php';

De cette façon, vous pouvez déplacer le répertoire du projet n'importe où sur votre machine ou une autre, et cela fonctionnera bien.

**TIP**

Si vous êtes curieux de savoir ce que cet outil en ligne de commande peut faire pour vous, tapez symfony pour lister les options et les tâches disponibles :

$ php lib/vendor/symfony/data/bin/symfony

Sur Windows:

c:\> php lib\vendor\symfony\data\bin\symfony

La ligne de commande symfony est le meilleur ami du développeur. Il fournit de nombreux utilitaires qui permettent d'améliorer votre productivité sur les activités quotidiennes comme le vidage du cache, la génération de code, et bien plus encore.

## **Configuration de la base de données**

Le framework Symfony supporte toutes les [PDO](http://www.php.net/PDO)-soutenus par des bases de données (MySQL, PostgreSQL, SQLite, Oracle, MSSQL, ...) hors de la boîte. Au sommet de PDO, symfony est livré avec deux outils ORM: Propel et Doctrine.

Lorsque vous créez un nouveau projet, Doctrine est activé par défaut. La configuration de la base de données employée par Doctrine est aussi simple en utilisant la tâche configure:database:

$ php symfony configure: database "mysql:host=localhost;dbname=dbname" root mYsEcret

La tâche configure:database comporte 3 arguments: le [PDO DSN](http://www.php.net/manual/en/pdo.drivers.php), le nom de l'utilisateur, et le mot de passe pour accéder à la base de données. Si vous n'avez pas besoin d'un mot de passe pour accéder à votre base de donnée sur le serveur de développement, omettez simplement le troisième argument.

**TIP**

Si vous souhaitez utiliser Propel au lieu de Doctrine, ajoutez --orm=Propel lors de la création du projet avec la tâche generate:project. Et si vous ne voulez pas utiliser un ORM, passer juste --orm=none.

## **Création de l'application**

Maintenant, créez l'application frontend en exécutant la tâche generate:app :

$ php symfony generate:app frontend

**TIP**

Parce que le raccourci symfony est exécutable, les utilisateurs Unix peuvent remplacer toutes les occurrences de 'php symfony' par './symfony' à partir de maintenant.

Sur Windows vous pouvez copier le fichier 'symfony.bat' vers votre projet et utilisez 'symfony' à la place de 'php symfony' :

c:\> copy lib\vendor\symfony\data\bin\symfony.bat .

Basé sur le nom de l'application donné en argument, la tâche generate:app crée par défaut la structure du répertoire nécessaire à l'application sous le répertoire apps/frontend/ :

| **Répertoire** | **Description** |
| --- | --- |
| config/ | Les fichiers de configuration de l'application |
| lib/ | Les bibliothèques et les classes de l'application |
| modules/ | Le code de l'application (MVC) |
| templates/ | Les fichiers template globaux |

**SIDEBAR**

Securité

Par défaut, la tâche generate:app a sécurisé notre application sur les deux vulnérabilités les plus répandues que l'on trouve sur le web. C'est vrai, symfony prend automatiquement des mesures de sécurité à notre place.

Pour prévenir des attaques XSS, l'output escaping a été activé; et pour prévenir des attaques CSRF, un secret CSRF a été créé aléatoirement.

Bien sûr, vous pouvez modifier ces paramètres grâce aux options suivantes :

* --escaping-strategy: Active ou désactive l'output escaping
* --csrf-secret: active les jetons de session dans les formulaires

Si vous ne savez rien sur [XSS](http://fr.wikipedia.org/wiki/Cross-site_scripting) ou [CSRF](http://fr.wikipedia.org/wiki/Cross-site_request_forgery), prenez le temps d'en apprendre d'avantage sur ces failles de sécurité.

## **Droits sur les répertoires structurés**

Avant d'essayer d'accéder à votre projet nouvellement créé, vous devez configurer l'écriture sur les répertoires cache/ et log/ à des niveaux appropriés, afin que votre serveur web puisse écrire dedans :

$ chmod 777 cache/ log/

**SIDEBAR**

Conseils pour les personnes utilisant un outil de SCM

symfony écrit seulement dans deux répertoires du projet symfony : cache/ et log/. Le contenu de ces répertoires peut-être ignoré par votre SCM (En utilisant la propriété svn:ignore, si vous utilisez Subversion par exemple).

Notion 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/). Pour créer un bundle on peut procéder de trois manières:

Via Composer on utilisant cette commande

$ composer.phar require nomdubundle/exemple-bundle "version"

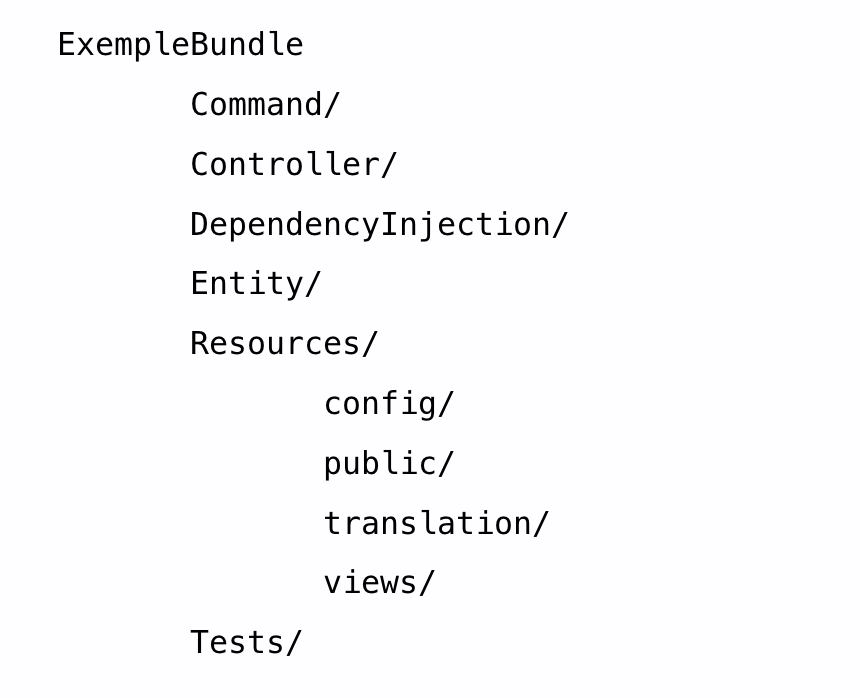
En ligne de commande

### Pourquoi utiliser un Bundle

Comme on le dit souvent «  On n’invente pas la roue » utiliser un Bundle peut nous apporter plusieurs avantages et surtout de nous 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. Il existe plusieurs bundle qui peuvent aider à développer un projet efficacement notamment [FOSUserBundle](https://github.com/FriendsOfSymfony/FOSUserBundle) qui permet entre autres de faire une gestion assez poussée des utilisateurs d'un site par exemple créer des groupes d'utilisateurs avec des rôles différents. Il y'a aussi [SonataAdminBundle](https://sonata-project.org/bundles/admin/2-3/doc/index.html) qui est un puissant système d'Administration de site sous symfony.

### 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.



### Éléments 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

**Dependance Injection** : 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.

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**[**¶**](https://symfony.com/doc/current/forms.html#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:

|  |
| --- |
| composer require symfony/form |

## **Utilisation**[**¶**](https://symfony.com/doc/current/forms.html#usage)

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

1. **Créez le formulaire** dans un contrôleur Symfony ou en utilisant une classe de formulaire dédiée;
2. **Rendez le formulaire** dans un modèle afin que l'utilisateur puisse le modifier et le soumettre;
3. **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 :

*// src/Entity/Task.php*

namespace App\Entity;

class Task

{

protected $task;

protected $dueDate;

public function getTask()

{

return $this->task;

}

public function setTask($task)

{

$this->task = $task;

}

public function getDueDate()

{

return $this->dueDate;

}

public function setDueDate(\DateTime $dueDate = null)

{

$this->dueDate = $dueDate;

}

}

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 [entités Doctrine](https://symfony.com/doc/current/doctrine.html) de la même manière.

### Types de formulaires [¶](https://symfony.com/doc/current/forms.html#form-types)

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 formularies»:

* 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 propres types de formulaires](https://symfony.com/doc/current/form/create_custom_field_type.html) .

## **Création de formulaires**[**¶**](https://symfony.com/doc/current/forms.html#building-forms)

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 [¶](https://symfony.com/doc/current/forms.html#creating-forms-in-controllers)

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

*// src/Controller/TaskController.php*

namespace App\Controller;

use App\Entity\Task;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\HttpFoundation\Request;

class TaskController extends AbstractController

{

public function new(Request $request)

{

*// creates a task object and initializes some data for this example*

$task = new Task();

$task->setTask('Write a blog post');

$task->setDueDate(new \DateTime('tomorrow'));

$form = $this->createFormBuilder($task)

->add('task', TextType::class)

->add('dueDate', DateType::class)

->add('save', SubmitType::class, ['label' => 'Create Task'])

->getForm();

*// ...*

}

}

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 [¶](https://symfony.com/doc/current/forms.html#creating-form-classes)

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:

*// src/Form/Type/TaskType.php*

namespace App\Form\Type;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\Form\FormBuilderInterface;

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options)

{

$builder

->add('task', TextType::class)

->add('dueDate', DateType::class)

->add('save', SubmitType::class)

;

}

}

**POINTE**

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):

*// src/Controller/TaskController.php*

namespace App\Controller;

use App\Form\Type\TaskType;

*// ...*

class TaskController extends AbstractController

{

public function new()

{

*// creates a task object and initializes some data for this example*

$task = new Task();

$task->setTask('Write a blog post');

$task->setDueDate(new \DateTime('tomorrow'));

$form = $this->createForm(TaskType::class, $task);

*// ...*

}

}

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:

*// src/Form/Type/TaskType.php*

namespace App\Form\Type;

use App\Entity\Task;

use Symfony\Component\OptionsResolver\OptionsResolver;

*// ...*

class TaskType extends AbstractType

{

*// ...*

public function configureOptions(OptionsResolver $resolver)

{

$resolver->setDefaults([

'data\_class' => Task::class,

]);

}

}

## **Formulaires de rendu**[**¶**](https://symfony.com/doc/current/forms.html#rendering-forms)

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:

*// src/Controller/TaskController.php*

namespace App\Controller;

use App\Entity\Task;

use App\Form\Type\TaskType;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Request;

class TaskController extends AbstractController

{

public function new(Request $request)

{

$task = new Task();

*// ...*

$form = $this->createForm(TaskType::class, $task);

return $this->render('task/new.html.twig', [

'form' => $form->createView(),

]);

}

}

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:

|  |  |
| --- | --- |
| 1  2 | *{# templates/task/new.html.twig #}*  {{ form(form) }} |

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.

**POINTE**

Le système de formulaire est suffisamment intelligent pour accéder à la valeur de la taskpropriété protégée via les méthodes getTask()et setTask()sur la Taskclasse. À moins qu'une propriété ne soit publique, elle doit avoir une méthode «getter» et «setter» afin que Symfony puisse obtenir et mettre des données sur la propriété. Pour une propriété booléenne, vous pouvez utiliser une méthode «isser» ou «hasser» (par exemple is Published()ou has Reminder()) au lieu d'un getter (par exemple get Published()ou get Reminder()).

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:

* [YAML](https://symfony.com/doc/current/forms.html)

|  |  |
| --- | --- |
| 1  2  3 | *# config/packages/twig.yaml*  twig:  form\_themes: ['bootstrap\_4\_layout.html.twig'] |

* [XML](https://symfony.com/doc/current/forms.html)

* [PHP](https://symfony.com/doc/current/forms.html)

Les [thèmes de formulaire Symfony intégrés](https://symfony.com/doc/current/form/form_themes.html#symfony-builtin-forms) incluent Bootstrap 3 et 4 et Foundation 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**[**¶**](https://symfony.com/doc/current/forms.html#processing-forms)

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 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:

*// ...*

use Symfony\Component\HttpFoundation\Request;

public function new(Request $request)

{

*// just setup a fresh $task object (remove the example data)*

$task = new Task();

$form = $this->createForm(TaskType::class, $task);

$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {

*// $form->getData() holds the submitted values*

*// but, the original `$task` variable has also been updated*

$task = $form->getData();

*// ... perform some action, such as saving the task to the database*

*// for example, if Task is a Doctrine entity, save it!*

*// $entityManager = $this->getDoctrine()->getManager();*

*// $entityManager->persist($task);*

*// $entityManager->flush();*

return $this->redirectToRoute('task\_success');

}

return $this->render('task/new.html.twig', [

'form' => $form->createView(),

]);

}

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

1. 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;
2. Lorsque l'utilisateur soumet le formulaire, le [handle Request()](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 falseet le formulaire est rendu à nouveau, mais maintenant avec des erreurs de validation;
3. 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 remerciement ou de réussite);

**REMARQUE**

Rediriger un utilisateur après une soumission de formulaire réussie est une meilleure pratique qui empêche l'utilisateur de cliquer sur le bouton "Actualiser" de son navigateur et de publier à nouveau les données.

**MISE EN GARDE**

La createView()méthode doit être appelée après avoir handle Request() été appelée. Sinon, lors de l'utilisation d' [événements de formulaire](https://symfony.com/doc/current/form/events.html) , les modifications apportées aux \*\_SUBMIT événements ne seront pas appliquées à la vue (comme les erreurs de validation).

**VOIR ÉGALEMENT**

Si vous avez besoin de plus de contrôle sur le moment exact où votre formulaire est soumis ou sur les données qui lui sont transmises, vous pouvez [utiliser la méthode submit () pour gérer les soumissions de formulaire](https://symfony.com/doc/current/form/direct_submit.html) .

## **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:

|  |
| --- |
| composer require symfony/validator |

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.

* [Annotations](https://symfony.com/doc/current/forms.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | *// src/Entity/Task.php*  namespace App\Entity;  use Symfony\Component\Validator\Constraints as Assert;  class Task  {  */\*\**  *\* @Assert\NotBlank*  *\*/*  public $task;  */\*\**  *\* @Assert\NotBlank*  *\* @Assert\Type("\DateTime")*  *\*/*  protected $dueDate;  } |

* [YAML](https://symfony.com/doc/current/forms.html)

* [XML](https://symfony.com/doc/current/forms.html)

* [PHP](https://symfony.com/doc/current/forms.html)

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**[**¶**](https://symfony.com/doc/current/forms.html#other-common-form-features)

### Passer des options aux formulaires [¶](https://symfony.com/doc/current/forms.html#passing-options-to-forms)

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():

*// src/Controller/TaskController.php*

namespace App\Controller;

use App\Form\Type\TaskType;

*// ...*

class TaskController extends AbstractController

{

public function new()

{

$task = new Task();

*// use some PHP logic to decide if this form field is required or not*

$dueDateIsRequired = ...

$form = $this->createForm(TaskType::class, $task, [

'require\_due\_date' => $dueDateIsRequired,

]);

*// ...*

}

}

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:

*// src/Form/Type/TaskType.php*

namespace App\Form\Type;

use Symfony\Component\OptionsResolver\OptionsResolver;

*// ...*

class TaskType extends AbstractType

{

*// ...*

public function configureOptions(OptionsResolver $resolver)

{

$resolver->setDefaults([

*// ...,*

'require\_due\_date' => false,

]);

*// you can also define the allowed types, allowed values and*

*// any other feature supported by the OptionsResolver component*

$resolver->setAllowedTypes('require\_due\_date', 'bool');

}

}

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

*// src/Form/Type/TaskType.php*

namespace App\Form\Type;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\FormBuilderInterface;

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options)

{

$builder

*// ...*

->add('dueDate', DateType::class, [

'required' => $options['require\_due\_date'],

])

;

}

*// ...*

}

### Options de type de formulaire [¶](https://symfony.com/doc/current/forms.html#form-type-options)

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 [¶](https://symfony.com/doc/current/forms.html#the-required-option)

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 falsesur un ou plusieurs champs:

->add('dueDate', DateType::class, [

'required' => false,

])

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 à moins que vous n'utilisiez également Symfony NotBlankou NotNullles contraintes de validation.

#### L' labeloption [¶](https://symfony.com/doc/current/forms.html#the-label-option)

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

->add('dueDate', DateType::class, [

*// set it to FALSE to not display the label for this field*

'label' => 'To Be Completed Before',

])

**POINTE**

Par défaut, les <label>balises des champs obligatoires sont rendues avec une requiredclasse CSS, vous pouvez donc afficher un astérisque pour les champs obligatoires en appliquant ces styles CSS:

|  |  |
| --- | --- |
| 1  2  3 | label.required:before {  content: "\*";  } |

### Changer l'action et la méthode HTTP [¶](https://symfony.com/doc/current/forms.html#changing-the-action-and-http-method)

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:

*// src/Controller/TaskController.php*

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

class TaskController extends AbstractController

{

public function new()

{

*// ...*

$form = $this->createFormBuilder($task)

->setAction($this->generateUrl('target\_route'))

->setMethod('GET')

*// ...*

->getForm();

*// ...*

}

}

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

*// src/Controller/TaskController.php*

namespace App\Controller;

use App\Form\TaskType;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class TaskController extends AbstractController

{

public function new()

{

*// ...*

$form = $this->createForm(TaskType::class, $task, [

'action' => $this->generateUrl('target\_route'),

'method' => 'GET',

]);

*// ...*

}

}

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:

|  |  |
| --- | --- |
| 1  2 | *{# templates/task/new.html.twig #}*  {{ form\_start(form, {'action': path('target\_route'), 'method': 'GET'}) }} |

**REMARQUE**

Si la méthode du formulaire n'est pas GETou POST, mais PUT, PATCHou DELETE, Symfony insérera un champ masqué avec le nom \_method qui stocke cette méthode. Le formulaire sera soumis dans une POST demande normale , mais [le routage de Symfony](https://symfony.com/doc/current/routing.html) est capable de détecter le \_methodparamètre et l'interprétera comme une PUT, PATCHou une DELETEdemande. Voir l' option [http\_method\_override](https://symfony.com/doc/current/reference/configuration/framework.html#configuration-framework-http-method-override) .

### Changer le nom du formulaire [¶](https://symfony.com/doc/current/forms.html#changing-the-form-name)

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:

*// src/Controller/TaskController.php*

namespace App\Controller;

use App\Form\TaskType;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class TaskController extends AbstractController

{

public function new()

{

$task = ...;

$form = $this->get('form.factory')->createNamed('my\_name', TaskType::class, $task);

*// ...*

}

}

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

### Validation HTML côté client [¶](https://symfony.com/doc/current/forms.html#client-side-html-validation)

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.

|  |  |
| --- | --- |
| 1  2  3  4 | *{# templates/task/new.html.twig #}*  {{ form\_start(form, {'attr': {'novalidate': 'novalidate'}}) }}  {{ form\_widget(form) }}  {{ form\_end(form) }} |

### Type de formulaire Deviner [¶](https://symfony.com/doc/current/forms.html#form-type-guessing)

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:

*// src/Form/Type/TaskType.php*

namespace App\Form\Type;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\Form\FormBuilderInterface;

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options)

{

$builder

*// if you don't define field options, you can omit the second argument*

->add('task')

*// if you define field options, pass NULL as second argument*

->add('dueDate', null, ['required' => false])

->add('save', SubmitType::class)

;

}

}

**MISE EN GARDE**

Lors de l'utilisation d'un [groupe de validation de formulaire](https://symfony.com/doc/current/form/validation_groups.html) spécifique , le devineur de type de champ tiendra toujours compte de toutes les contraintes de validation lors de la détermination de vos types de champ (y compris les contraintes qui ne font pas partie du ou des groupes de validation utilisés).

#### Options de type de formulaire Deviner [¶](https://symfony.com/doc/current/forms.html#form-type-options-guessing)

Lorsque le mécanisme de devinettes est activé pour un champ (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:

->add('task', null, ['attr' => ['maxlength' => 4]])

**VOIR ÉGALEMENT**

En plus de deviner le type de formulaire, Symfony devine également [les contraintes de validation](https://symfony.com/doc/current/forms.html#validating-forms) si vous utilisez une entité Doctrine. Lisez le guide [Validation des objets](https://symfony.com/doc/current/doctrine.html#automatic-object-validation) pour plus d'informations.

### Champs non mappés [¶](https://symfony.com/doc/current/forms.html#unmapped-fields)

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:

use Symfony\Component\Form\FormBuilderInterface;

public function buildForm(FormBuilderInterface $builder, array $options)

{

$builder

->add('task')

->add('dueDate')

->add('agreeTerms', CheckboxType::class, ['mapped' => false])

->add('save', SubmitType::class)

;

}

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

$form->get('agreeTerms')->getData();

$form->get('agreeTerms')->setData(true);

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.

Le système de sécurité de Symfony est incroyablement puissant, mais sa configuration peut également prêter à confusion. Ne t'inquiète pas! Dans cet article, vous apprendrez à configurer le système de sécurité de votre application étape par étape:

1. [Installer le support de sécurité](https://symfony.com/doc/current/security.html#security-installation) ;
2. [Créez votre classe d'utilisateurs](https://symfony.com/doc/current/security.html#create-user-class) ;
3. [Authentification et pare-feu](https://symfony.com/doc/current/security.html#security-yaml-firewalls) ;
4. [Refuser l'accès à votre application (autorisation)](https://symfony.com/doc/current/security.html#security-authorization) ;
5. [Récupération de l'objet utilisateur actuel](https://symfony.com/doc/current/security.html#retrieving-the-user-object) .

Quelques autres sujets importants sont abordés par la suite.

## **1) Installation**[**¶**](https://symfony.com/doc/current/security.html#installation)

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

|  |
| --- |
| composer require symfony/security-bundle |

**POINTE**

Une [nouvelle sécurité expérimentale a](https://symfony.com/doc/current/security/experimental_authenticators.html) été introduite dans Symfony 5.1, qui remplacera finalement la sécurité dans Symfony 6.0. Ce système est presque entièrement rétrocompatible avec la sécurité Symfony actuelle, ajoutez cette ligne à votre configuration de sécurité pour commencer à l'utiliser:

* [YAML](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4 | *# config/packages/security.yaml*  security:  enable\_authenticator\_manager: true  *# ...* |

* [XML](https://symfony.com/doc/current/security.html)

* [PHP](https://symfony.com/doc/current/security.html)

## **2a) Créez votre classe d'utilisateurs**[**¶**](https://symfony.com/doc/current/security.html#a-create-your-user-class)

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:

|  |
| --- |
| php bin/console make:user  The name of the security user class (e.g. User) [User]:  > User  Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:  > yes  Enter a property name that will be the unique "display" name for the user (e.g.  email, username, uuid [email]  > email  Does this app need to hash/check user passwords? (yes/no) [yes]:  > yes  created: src/Entity/User.php  created: src/Repository/UserRepository.php  updated: src/Entity/User.php  updated: config/packages/security.yaml |

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:

|  |
| --- |
| php bin/console make:migration  php bin/console doctrine:migrations:migrate |

## **2b) Le «fournisseur d'utilisateur»**[**¶**](https://symfony.com/doc/current/security.html#b-the-user-provider)

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**[**¶**](https://symfony.com/doc/current/security.html#c-encoding-passwords)

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:

* [YAML](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | *# config/packages/security.yaml*  security:  *# ...*  encoders:  *# use your user class name here*  App\Entity\User:  *# Use native password encoder*  *# This value auto-selects the best possible hashing algorithm*  *# (i.e. Sodium when available).*  algorithm: auto |

* [XML](https://symfony.com/doc/current/security.html)

* [PHP](https://symfony.com/doc/current/security.html)

Maintenant que Symfony sait comment vous voulez encoder les mots de passe, vous pouvez utiliser le UserPasswordEncoderInterfaceservice pour ce faire avant d'enregistrer vos utilisateurs dans la base de données.

Par exemple, en utilisant [DoctrineFixturesBundle](https://symfony.com/doc/current/testing/database.html#doctrine-fixtures) , vous pouvez créer des utilisateurs de base de données factices:

|  |
| --- |
| php bin/console make:fixtures  The class name of the fixtures to create (e.g. AppFixtures):  > UserFixtures |

Utilisez ce service pour coder les mots de passe:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | // src/DataFixtures/UserFixtures.php  + use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;  // ...  class UserFixtures extends Fixture  {  + private $passwordEncoder;  + public function \_\_construct(UserPasswordEncoderInterface $passwordEncoder)  + {  + $this->passwordEncoder = $passwordEncoder;  + }  public function load(ObjectManager $manager)  {  $user = new User();  // ...  + $user->setPassword($this->passwordEncoder->encodePassword(  + $user,  + 'the\_new\_password'  + ));  // ...  }  } |

Vous pouvez encoder manuellement un mot de passe en exécutant:

|  |
| --- |
| php bin/console security:encode-password |

## **3a) Authentification et pare-feu**[**¶**](https://symfony.com/doc/current/security.html#a-authentication-firewalls)

**Nouveau dans la version 5.1:**L' option a été introduite dans Symfony 5.1. Avant la version 5.1, il était activé à l'aide delazy: trueanonymous: lazy

Le système de sécurité est configuré dans config/packages/security.yaml. La section la plus importante est firewalls:

* [YAML](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | *# config/packages/security.yaml*  security:  firewalls:  dev:  pattern: ^/(\_(profiler|wdt)|css|images|js)/  security: false  main:  anonymous: true  lazy: true |

* [XML](https://symfony.com/doc/current/security.html)

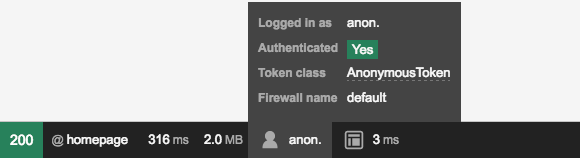
* [PHP](https://symfony.com/doc/current/security.html)

Un «pare-feu» est votre système d'authentification: la configuration ci-dessous définit comment vos utilisateurs pourront s'authentifier (par exemple, formulaire de connexion, jeton d'API, etc.).

Un seul pare-feu est actif sur chaque demande: Symfony utilise la patternclé pour trouver la première correspondance (vous pouvez également faire une [correspondance par hôte ou autre](https://symfony.com/doc/current/security/firewall_restriction.html) ). Le devpare-feu est vraiment un faux pare-feu: il s'assure simplement que vous ne bloquez pas accidentellement les outils de développement de Symfony - qui vivent sous des URL comme /\_profiler et /\_wdt.

Toutes les URL réelles sont gérées par le mainpare - feu (aucune patternclé ne signifie qu'elle correspond à toutes les URL). Un pare-feu peut avoir de nombreux modes d'authentification, c'est-à-dire de nombreuses façons de poser la question «Qui êtes-vous?». Souvent, l'utilisateur est inconnu (c'est-à-dire non connecté) lors de sa première visite sur votre site Web. Le anonymousmode, s'il est activé, est utilisé pour ces demandes.

En fait, si vous allez sur la page d'accueil en ce moment, vous y aurez accès et vous verrez que vous êtes "authentifié" en tant que anon.. Le pare-feu a vérifié qu'il ne connaissait pas votre identité et vous êtes donc anonyme:



Cela signifie que toute demande peut avoir un jeton anonyme pour accéder à certaines ressources, tandis que certaines actions (c'est-à-dire certaines pages ou boutons) peuvent toujours nécessiter des privilèges spécifiques. Un utilisateur peut alors accéder à une connexion de formulaire sans être authentifié en tant qu'utilisateur unique (sinon une boucle de redirection infinie se produirait demandant à l'utilisateur de s'authentifier tout en essayant de le faire).

Vous apprendrez plus tard comment refuser l'accès à certaines URL, contrôleurs ou parties de modèles.

**POINTE**

Le lazymode anonyme empêche le démarrage de la session s'il n'y a pas besoin d'autorisation (c.-à-d. Vérification explicite d'un privilège d'utilisateur). Ceci est important pour garder les requêtes en cache (voir [Cache HTTP](https://symfony.com/doc/current/http_cache.html) ).

**REMARQUE**

Si vous ne voyez pas la barre d'outils, installez le [profileur](https://symfony.com/doc/current/profiler.html) avec:

|  |
| --- |
| composer require --dev symfony/profiler-pack |

Maintenant que nous comprenons notre pare-feu, l'étape suivante consiste à créer un moyen pour vos utilisateurs de s'authentifier!

## **3b) Authentification de vos utilisateurs**[**¶**](https://symfony.com/doc/current/security.html#b-authenticating-your-users)

L'authentification dans Symfony peut sembler un peu «magique» au début. En effet, au lieu de créer une route et un contrôleur pour gérer la connexion, vous activerez un fournisseur d'authentification : du code qui s'exécute automatiquement avant l' appel de votre contrôleur.

Symfony a plusieurs [fournisseurs d'authentification intégrés](https://symfony.com/doc/current/security/auth_providers.html) . Si votre cas d'utilisation correspond exactement à l' un d'eux , tant mieux ! Mais, dans la plupart des cas - y compris un formulaire de connexion - nous vous recommandons de créer un Guard Authenticator : une classe qui vous permet de contrôler chaque partie du processus d'authentification (voir la section suivante).

**POINTE**

Si votre application connecte les utilisateurs via un service tiers tel que Google, Facebook ou Twitter (connexion sociale), consultez le [bundle de la](https://github.com/hwi/HWIOAuthBundle) communauté [HWIOAuthBundle](https://github.com/hwi/HWIOAuthBundle) .

### Authentificateurs de garde [¶](https://symfony.com/doc/current/security.html#guard-authenticators)

Un authentificateur Guard est une classe qui vous donne un contrôle complet sur votre processus d'authentification. Il existe de nombreuses façons différentes de créer un authentificateur; voici quelques cas d'utilisation courants:

* [Comment créer un formulaire de connexion](https://symfony.com/doc/current/security/form_login_setup.html)
* [Système d'authentification personnalisé avec garde (exemple de jeton d'API)](https://symfony.com/doc/current/security/guard_authentication.html) - voir ceci pour la description la plus détaillée des authentificateurs et comment ils fonctionnent

## **4) Refuser l'accès, les rôles et autres autorisations**[**¶**](https://symfony.com/doc/current/security.html#denying-access-roles-and-other-authorization)

Les utilisateurs peuvent désormais se connecter à votre application en utilisant votre formulaire de connexion. Génial! Maintenant, vous devez apprendre à refuser l'accès et à travailler avec l'objet User. C'est ce qu'on appelle l' **autorisation** , et son travail consiste à décider si un utilisateur peut accéder à une ressource (une URL, un objet modèle, un appel de méthode,…).

Le processus d'autorisation a deux aspects différents:

1. L'utilisateur reçoit un ensemble spécifique de rôles lors de la connexion (par exemple ROLE\_ADMIN).
2. Vous ajoutez du code pour qu'une ressource (par exemple URL, contrôleur) nécessite un «attribut» spécifique (le plus souvent un rôle ROLE\_ADMIN) pour être accessible.

### Rôles [¶](https://symfony.com/doc/current/security.html#roles)

Lorsqu'un utilisateur se connecte, Symfony appelle la get Roles() méthode sur votre User objet pour déterminer les rôles de cet utilisateur. Dans la User classe que nous avons générée précédemment, les rôles sont un tableau stocké dans la base de données et chaque utilisateur se voit toujours attribuer au moins un rôle ROLE\_USER::

*// src/Entity/User.php*

*// ...*

class User

{

*/\*\**

*\* @ORM\Column(type="json")*

*\*/*

private $roles = [];

*// ...*

public function getRoles(): array

{

$roles = $this->roles;

*// guarantee every user at least has ROLE\_USER*

$roles[] = 'ROLE\_USER';

return array\_unique($roles);

}

}

C'est une belle valeur par défaut, mais vous pouvez faire ce que vous voulez pour déterminer les rôles qu'un utilisateur doit avoir. Voici quelques directives:

* Chaque rôle **doit commencer par** ROLE\_ (sinon, les choses ne fonctionneront pas comme prévu)
* Autre que la règle ci-dessus, un rôle n'est qu'une chaîne et vous pouvez inventer ce dont vous avez besoin (par exemple ROLE\_PRODUCT\_ADMIN).

Vous utiliserez ces rôles à côté pour accorder l'accès à des sections spécifiques de votre site. Vous pouvez également utiliser une [hiérarchie de](https://symfony.com/doc/current/security.html#security-role-hierarchy) rôles où certains rôles vous donnent automatiquement d'autres rôles.

### Ajouter du code pour refuser l'accès [¶](https://symfony.com/doc/current/security.html#add-code-to-deny-access)

Il existe **deux** façons de refuser l'accès à quelque chose:

1. [access\_control dans security.yaml](https://symfony.com/doc/current/security.html#security-authorization-access-control) vous permet de protéger les modèles d'URL (par exemple/admin/\*). Plus simple, mais moins flexible;
2. [dans votre contrôleur (ou autre code)](https://symfony.com/doc/current/security.html#security-securing-controller) .

#### Sécurisation des modèles d'URL (access\_control) [¶](https://symfony.com/doc/current/security.html#securing-url-patterns-access-control)

La façon la plus simple de sécuriser une partie de votre application est de sécuriser un modèle d'URL complet dans security.yaml. Par exemple, pour exiger ROLE\_ADMINpour toutes les URL commençant par /admin, vous pouvez:

* [YAML](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | *# config/packages/security.yaml*  security:  *# ...*  firewalls:  *# ...*  main:  *# ...*  access\_control:  *# require ROLE\_ADMIN for /admin\**  - { path: '^/admin', roles: ROLE\_ADMIN }  *# or require ROLE\_ADMIN or IS\_AUTHENTICATED\_FULLY for /admin\**  - { path: '^/admin', roles: [IS\_AUTHENTICATED\_FULLY, ROLE\_ADMIN] }  *# the 'path' value can be any valid regular expression*  *# (this one will match URLs like /api/post/7298 and /api/comment/528491)*  - { path: ^/api/(post|comment)/\d+$, roles: ROLE\_USER } |

* [XML](https://symfony.com/doc/current/security.html)

* [PHP](https://symfony.com/doc/current/security.html)

Vous pouvez définir autant de modèles d'URL que vous le souhaitez - chacun est une expression régulière. **MAIS** , **une** seule sera mise en correspondance par demande: Symfony démarre en haut de la liste et s'arrête lorsqu'il trouve la première correspondance:

* [YAML](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | *# config/packages/security.yaml*  security:  *# ...*  access\_control:  *# matches /admin/users/\**  - { path: '^/admin/users', roles: ROLE\_SUPER\_ADMIN }  *# matches /admin/\* except for anything matching the above rule*  - { path: '^/admin', roles: ROLE\_ADMIN } |

* [XML](https://symfony.com/doc/current/security.html)

* [PHP](https://symfony.com/doc/current/security.html)

Le fait de faire précéder le chemin d'accès ^signifie que seules les URL commençant par le modèle sont mises en correspondance. Par exemple, un chemin d'accès /admin(sans le ^) correspondrait /admin/foomais correspondrait également à des URL comme /foo/admin.

Chacun access\_control peut également correspondre à l'adresse IP, au nom d'hôte et aux méthodes HTTP. Il peut également être utilisé pour rediriger un utilisateur vers la https version d'un modèle d'URL. Voir [Comment fonctionne la sécurité access\_control?](https://symfony.com/doc/current/security/access_control.html).

#### Sécurisation des contrôleurs et autre code [¶](https://symfony.com/doc/current/security.html#securing-controllers-and-other-code)

Vous pouvez refuser l'accès depuis l'intérieur d'un contrôleur:

*// src/Controller/AdminController.php*

*// ...*

public function adminDashboard()

{

$this->denyAccessUnlessGranted('ROLE\_ADMIN');

*// or add an optional message - seen by developers*

$this->denyAccessUnlessGranted('ROLE\_ADMIN', null, 'User tried to access a page without having ROLE\_ADMIN');

}

C'est tout! Si l'accès n'est pas accordé, un spécial [AccessDeniedException](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Security/Core/Exception/AccessDeniedException.php) est lancé et plus aucun code dans votre contrôleur n'est exécuté. Ensuite, l'une des deux choses se produira:

1. Si l'utilisateur n'est pas encore connecté, il lui sera demandé de se connecter (par exemple redirigé vers la page de connexion).
2. Si l'utilisateur est connecté, mais n'a pas le ROLE\_ADMINrôle, la page 403 d'accès refusé s'affiche (que vous pouvez [personnaliser](https://symfony.com/doc/current/controller/error_pages.html#controller-error-pages-by-status-code) ).

Grâce au Sensio Framework Extra Bundle, vous pouvez également sécuriser votre contrôleur à l'aide d'annotations:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | // src/Controller/AdminController.php  // ...  + use Sensio\Bundle\FrameworkExtraBundle\Configuration\IsGranted;  + /\*\*  + \* Require ROLE\_ADMIN for \*every\* controller method in this class.  + \*  + \* @IsGranted("ROLE\_ADMIN")  + \*/  class AdminController extends AbstractController  {  + /\*\*  + \* Require ROLE\_ADMIN for only this controller method.  + \*  + \* @IsGranted("ROLE\_ADMIN")  + \*/  public function adminDashboard()  {  // ...  }  } |

Pour plus d'informations, consultez la [documentation FrameworkExtraBundle](https://symfony.com/doc/current/bundles/SensioFrameworkExtraBundle/index.html) .

#### Contrôle d'accès dans les modèles [¶](https://symfony.com/doc/current/security.html#access-control-in-templates)

Si vous souhaitez vérifier si l'utilisateur actuel a un certain rôle, vous pouvez utiliser la is\_granted() fonction d'assistance intégrée dans n'importe quel modèle Twig:

|  |  |
| --- | --- |
| 1  2  3 | {% if is\_granted('ROLE\_ADMIN') %}  <a href="...">Delete</a>  {% endif %} |

#### Sécuriser d'autres services [¶](https://symfony.com/doc/current/security.html#securing-other-services)

Voir [Comment sécuriser un service ou une méthode dans votre application](https://symfony.com/doc/current/security/securing_services.html) .

### Définition des autorisations utilisateur individuelles [¶](https://symfony.com/doc/current/security.html#setting-individual-user-permissions)

La plupart des applications nécessitent des règles d'accès plus spécifiques. Par exemple, un utilisateur ne devrait pouvoir modifier que ses propres commentaires sur un blog. Les électeurs vous permettent d'écrire tout logique métier dont vous avez besoin pour déterminer l' accès. L'utilisation de ces votants est similaire aux contrôles d'accès basés sur les rôles mis en œuvre dans les chapitres précédents. Lisez [Comment utiliser les électeurs pour vérifier les autorisations des utilisateurs](https://symfony.com/doc/current/security/voters.html) pour savoir comment implémenter votre propre électeur.

### Vérification pour voir si un utilisateur est connecté (IS\_AUTHENTICATED\_FULLY) [¶](https://symfony.com/doc/current/security.html#checking-to-see-if-a-user-is-logged-in-is-authenticated-fully)

Si vous souhaitez uniquement vérifier si un utilisateur est connecté (vous ne vous souciez pas des rôles), vous avez deux options. Tout d'abord, si vous avez donné à chaque utilisateur ROLE\_USER, vous pouvez simplement vérifier ce rôle. Sinon, vous pouvez utiliser un «attribut» spécial à la place d'un rôle:

*// ...*

public function adminDashboard()

{

$this->denyAccessUnlessGranted('IS\_AUTHENTICATED\_FULLY');

*// ...*

}

Vous pouvez utiliser IS\_AUTHENTICATED\_FULLY n'importe où les rôles sont utilisés: comme access\_controlou dans Twig.

IS\_AUTHENTICATED\_FULLY n'est pas un rôle, mais il agit en quelque sorte comme un rôle, et chaque utilisateur qui s'est connecté l'aura. En fait, il existe des attributs spéciaux comme celui-ci:

* IS\_AUTHENTICATED\_REMEMBERED: Tous les utilisateurs connectés l'ont, même s'ils sont connectés à cause d'un cookie "se souvenir de moi". Même si vous n'utilisez pas la [fonctionnalité Se souvenir de moi](https://symfony.com/doc/current/security/remember_me.html) , vous pouvez l'utiliser pour vérifier si l'utilisateur est connecté.
* IS\_AUTHENTICATED\_FULLY: Ceci est similaire à IS\_AUTHENTICATED\_REMEMBERED, mais plus fort. Les utilisateurs qui se sont connectés uniquement à cause d'un cookie "se souvenir de moi" auront IS\_AUTHENTICATED\_REMEMBERED mais ne l'auront pas IS\_AUTHENTICATED\_FULLY.
* IS\_AUTHENTICATED\_ANONYMOUSLY: Tous les utilisateurs (même anonymes) l'ont - ceci est utile lors de la mise en liste blanche des URL pour garantir l'accès - certains détails sont dans [Comment fonctionne la sécurité access\_control?](https://symfony.com/doc/current/security/access_control.html).
* IS\_ANONYMOUS: Seuls les utilisateurs anonymes sont mis en correspondance par cet attribut.
* IS\_REMEMBERED: Seuls les utilisateurs authentifiés à l'aide de la [fonctionnalité se souvenir de moi](https://symfony.com/doc/current/security/remember_me.html) , (c'est-à-dire un cookie souvenir de moi).
* IS\_IMPERSONATOR: Lorsque l'utilisateur actuel [emprunte l'identité d'](https://symfony.com/doc/current/security/impersonating_user.html) un autre utilisateur dans cette session, cet attribut correspondra.

**Nouveau dans la version 5.1:**Les IS\_ANONYMOUS, IS\_REMEMBEREDet les IS\_IMPERSONATOR attributs ont été introduits dans Symfony 5.1.

## **5a) Récupération de l'objet utilisateur**[**¶**](https://symfony.com/doc/current/security.html#a-fetching-the-user-object)

Après l'authentification, l' User objet de l'utilisateur actuel est accessible via le getUser() raccourci:

public function index()

{

*// usually you'll want to make sure the user is authenticated first*

$this->denyAccessUnlessGranted('IS\_AUTHENTICATED\_FULLY');

*// returns your User object, or null if the user is not authenticated*

*// use inline documentation to tell your editor your exact User class*

*/\*\* @var \App\Entity\User $user \*/*

$user = $this->getUser();

*// Call whatever methods you've added to your User class*

*// For example, if you added a getFirstName() method, you can use that.*

return new Response('Well hi there '.$user->getFirstName());

}

## **5b) Récupérer l'utilisateur d'un service**[**¶**](https://symfony.com/doc/current/security.html#b-fetching-the-user-from-a-service)

Si vous devez obtenir l'utilisateur connecté à partir d'un service, utilisez le [Security](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Security/Core/Security.php) service:

*// src/Service/ExampleService.php*

*// ...*

use Symfony\Component\Security\Core\Security;

class ExampleService

{

private $security;

public function \_\_construct(Security $security)

{

*// Avoid calling getUser() in the constructor: auth may not*

*// be complete yet. Instead, store the entire Security object.*

$this->security = $security;

}

public function someMethod()

{

*// returns User object or null if not authenticated*

$user = $this->security->getUser();

}

}

### Récupérer l'utilisateur dans un modèle [¶](https://symfony.com/doc/current/security.html#fetch-the-user-in-a-template)

Dans un modèle Twig, l'objet utilisateur est disponible via la app.uservariable grâce à la [variable d'application globale Twig](https://symfony.com/doc/current/templates.html#twig-app-variable) :

|  |  |
| --- | --- |
| 1  2  3 | {% if is\_granted('IS\_AUTHENTICATED\_FULLY') %}  <p>Email: {{ app.user.email }}</p>  {% endif %} |

## **Déconnexion**[**¶**](https://symfony.com/doc/current/security.html#logging-out)

Pour activer la déconnexion, activez le logout paramètre config sous votre pare-feu:

* [YAML](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | *# config/packages/security.yaml*  security:  *# ...*  firewalls:  main:  *# ...*  logout:  path: app\_logout  *# where to redirect after logout*  *# target: app\_any\_route* |

* [XML](https://symfony.com/doc/current/security.html)

* [PHP](https://symfony.com/doc/current/security.html)

Ensuite, vous devrez créer un itinéraire pour cette URL (mais pas un contrôleur):

* [Annotations](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | *// src/Controller/SecurityController.php*  namespace App\Controller;  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  use Symfony\Component\Routing\Annotation\Route;  class SecurityController extends AbstractController  {  */\*\**  *\* @Route("/logout", name="app\_logout", methods={"GET"})*  *\*/*  public function logout()  {  *// controller can be blank: it will never be executed!*  throw new \Exception('Don\'t forget to activate logout in security.yaml');  }  } |

* [YAML](https://symfony.com/doc/current/security.html)

* [XML](https://symfony.com/doc/current/security.html)

* [PHP](https://symfony.com/doc/current/security.html)

Et c'est tout! En envoyant un utilisateur sur la app\_logoutroute (c'est-à-dire vers /logout) Symfony désidentifiera l'utilisateur actuel et le redirigera.

### Personnalisation de la déconnexion [¶](https://symfony.com/doc/current/security.html#customizing-logout)

**Nouveau dans la version 5.1:**Le a Log out Eventée introduit dans Symfony 5.1. Avant cette version, vous deviez utiliser un [gestionnaire de réussite de déconnexion](https://symfony.com/doc/current/reference/configuration/security.html#reference-security-logout-success-handler) pour personnaliser la déconnexion.

Dans certains cas, vous devez exécuter une logique supplémentaire lors de la déconnexion (par exemple, invalider certains jetons) ou vouloir personnaliser ce qui se passe après une déconnexion. Lors de la déconnexion, un [LogoutEvent](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Security/Http/Event/LogoutEvent.php) est envoyé. Enregistrez un [écouteur d'événements ou un abonné](https://symfony.com/doc/current/event_dispatcher.html) pour exécuter une logique personnalisée. Les informations suivantes sont disponibles dans la classe d'événements:

Get Token()

Renvoie le jeton de sécurité de la session sur le point de se déconnecter.

getRequest()

Renvoie la demande en cours.

getResponse()

Renvoie une réponse, si elle est déjà définie par un écouteur personnalisé. Utilisez setResponse()pour configurer une réponse de déconnexion personnalisée.

**POINTE**

Chaque pare-feu de sécurité possède son propre répartiteur d'événements ( security. event\_dispatcher. FIREWALLNAME). L'événement de déconnexion est distribué à la fois sur le répartiteur global et sur le pare-feu. Vous pouvez vous inscrire sur le répartiteur de pare-feu si vous souhaitez que votre écouteur soit exécuté uniquement pour un pare-feu spécifique. Par exemple, si vous avez un pare api - main feu et, utilisez cette configuration pour vous inscrire uniquement sur l'événement de déconnexion dans le main pare - feu:

* [YAML](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *# config/services.yaml*  services:  *# ...*  App\EventListener\CustomLogoutSubscriber:  tags:  - name: kernel.event\_subscriber  dispatcher: security.event\_dispatcher.main |

* [XML](https://symfony.com/doc/current/security.html)

* [PHP](https://symfony.com/doc/current/security.html)

## **Roles hierarchies**[**¶**](https://symfony.com/doc/current/security.html#hierarchical-roles)

Au lieu de donner plusieurs rôles à chaque utilisateur, vous pouvez définir des règles d'héritage de rôle en créant une hiérarchie de rôles:

* [YAML](https://symfony.com/doc/current/security.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | *# config/packages/security.yaml*  security:  *# ...*  role\_hierarchy:  ROLE\_ADMIN: ROLE\_USER  ROLE\_SUPER\_ADMIN: [ROLE\_ADMIN, ROLE\_ALLOWED\_TO\_SWITCH] |

* [XML](https://symfony.com/doc/current/security.html)

* [PHP](https://symfony.com/doc/current/security.html)

Les utilisateurs avec le ROLE\_ADMIN rôle auront également le ROLE\_USER rôle. Et les utilisateurs avec ROLE\_SUPER\_ADMIN, auront automatiquement ROLE\_ADMIN, ROLE\_ALLOWED\_TO\_SWITCHet ROLE\_USER(héritée de ROLE\_ADMIN).

Pour que la hiérarchie des rôles fonctionne, n'essayez pas d'appeler $user->getRoles()manuellement. Par exemple, dans un contrôleur s'étendant du [contrôleur de base](https://symfony.com/doc/current/controller.html#the-base-controller-class-services) :

*// BAD - $user->getRoles() will not know about the role hierarchy*

$hasAccess = in\_array('ROLE\_ADMIN', $user->getRoles());

*// GOOD - use of the normal security methods*

$hasAccess = $this->isGranted('ROLE\_ADMIN');

$this->denyAccessUnlessGranted('ROLE\_ADMIN');

**REMARQUE**

Les role\_hierarchy valeurs sont statiques - vous ne pouvez pas, par exemple, stocker la hiérarchie des rôles dans une base de données. Si vous en avez besoin, créez un voteur de [sécurité](https://symfony.com/doc/current/security/voters.html) personnalisé qui recherche les rôles d'utilisateur dans la base de données.

## **Foire aux questions**[**¶**](https://symfony.com/doc/current/security.html#frequently-asked-questions)

**Puis-je avoir plusieurs pare-feu?**

Oui! Mais ce n'est généralement pas nécessaire. Chaque pare-feu est comme un système de sécurité distinct. Et donc, à moins que vous n'ayez des besoins d'authentification très différents, un pare-feu fonctionne généralement bien. Avec l' [authentification Guard](https://symfony.com/doc/current/security/guard_authentication.html) , vous pouvez créer diverses manières d'autoriser l'authentification (par exemple, la connexion au formulaire, l'authentification par clé API et LDAP) sous le même pare-feu.

**Puis-je partager l'authentification entre des pare-feu?**

Oui, mais seulement avec une certaine configuration. Si vous utilisez plusieurs pare-feu et que vous vous authentifiez auprès d'un seul pare-feu, vous ne serez pas authentifié automatiquement contre les autres pare-feu. Différents pare-feu sont comme différents systèmes de sécurité. Pour ce faire, vous devez spécifier explicitement le même [contexte de pare-feu](https://symfony.com/doc/current/reference/configuration/security.html#reference-security-firewall-context) pour différents pare-feu. Mais généralement pour la plupart des applications, un seul pare-feu principal suffit.

**La sécurité ne semble pas fonctionner sur mes pages d'erreur**

Comme le routage est effectué avant la sécurité, 404 pages d'erreur ne sont couvertes par aucun pare-feu. Cela signifie que vous ne pouvez pas vérifier la sécurité ou même accéder à l'objet utilisateur sur ces pages. Voir [Comment personnaliser les pages d'erreur](https://symfony.com/doc/current/controller/error_pages.html) pour plus de détails.

**Mon authentification ne semble pas fonctionner: aucune erreur, mais je ne suis jamais connecté**

Parfois, l'authentification peut réussir, mais après la redirection, vous êtes déconnecté immédiatement en raison d'un problème de chargement Userde la session. Pour voir s'il s'agit d'un problème, consultez votre fichier journal ( var/log/dev.log) pour le message de journal:

**Impossible d'actualiser le jeton car l'utilisateur a changé**

Si vous voyez cela, il y a deux causes possibles. Tout d'abord, il peut y avoir un problème lors du chargement de votre utilisateur à partir de la session. Voir [Comprendre comment les utilisateurs sont actualisés à partir de la session](https://symfony.com/doc/current/security/user_provider.html#user-session-refresh) . Deuxièmement, si certaines informations utilisateur ont été modifiées dans la base de données depuis la dernière actualisation de la page, Symfony déconnectera volontairement l'utilisateur pour des raisons de sécurité.

Un modèle est le meilleur moyen d'organiser et de rendre le HTML depuis l'intérieur de votre application, que vous ayez besoin de rendre le HTML à partir d'un [contrôleur](https://symfony.com/doc/current/controller.html) ou de générer le [contenu d'un e-mail](https://symfony.com/doc/current/mailer.html) . Les modèles dans Symfony sont créés avec Twig: un moteur de modèle flexible, rapide et sécurisé.

## **Twig Templating Langage**[**¶**](https://symfony.com/doc/current/templates.html#twig-templating-language)

Le langage de [création de](https://twig.symfony.com/) modèles [Twig](https://twig.symfony.com/) vous permet d'écrire des modèles concis et lisibles qui sont plus conviviaux pour les concepteurs Web et, de plusieurs manières, plus puissants que les modèles PHP. Jetez un œil à l'exemple de modèle Twig suivant. Même si c'est la première fois que vous voyez Twig, vous en comprenez probablement la plupart:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <!DOCTYPE html>  <html>  <head>  <title>Welcome to Symfony!</title>  </head>  <body>  <h1>{{ page\_title }}</h1>  {% if user.isLoggedIn %}  Hello {{ user.name }}!  {% endif %}  *{# ... #}*  </body>  </html> |

La syntaxe de Twig est basée sur ces trois constructions:

* {{ ... }}, utilisé pour afficher le contenu d'une variable ou le résultat de l'évaluation d'une expression;
* {% ... %}, utilisé pour exécuter une logique, telle qu'une conditionnelle ou une boucle;
* {# ... #}, utilisé pour ajouter des commentaires au modèle (contrairement aux commentaires HTML, ces commentaires ne sont pas inclus dans la page rendue).

Vous ne pouvez pas exécuter de code PHP dans les modèles Twig, mais Twig fournit des utilitaires pour exécuter une logique dans les modèles. Par exemple, les **filtres** modifient le contenu avant d'être rendus, comme le upper filtre pour le contenu en majuscules:

|  |  |
| --- | --- |
| 1 | {{ title|upper }} |

Twig est livré avec une longue liste de [balises](https://twig.symfony.com/doc/2.x/tags/index.html) , [filtres](https://twig.symfony.com/doc/2.x/filters/index.html) et [fonctions](https://twig.symfony.com/doc/2.x/functions/index.html) disponibles par défaut. Dans les applications Symfony, vous pouvez également utiliser ces [filtres et fonctions Twig définis par Symfony](https://symfony.com/doc/current/reference/twig_reference.html) et vous pouvez [créer vos propres filtres et fonctions Twig](https://symfony.com/doc/current/templating/twig_extension.html) .

Twig est rapide dans l' prod [environnement](https://symfony.com/doc/current/configuration.html#configuration-environments) (car les modèles sont compilés en PHP et mis en cache automatiquement), mais pratique à utiliser dans l' dev environnement (car les modèles sont recompilés automatiquement lorsque vous les modifiez).

### Configuration des rameaux [¶](https://symfony.com/doc/current/templates.html#twig-configuration)

Twig a plusieurs options de configuration pour définir des choses comme le format utilisé pour afficher les nombres et les dates, la mise en cache du modèle, etc. Lisez la [référence de configuration Twig](https://symfony.com/doc/current/reference/configuration/twig.html) pour en savoir plus.

## **Création de modèles**[**¶**](https://symfony.com/doc/current/templates.html#creating-templates)

Avant d'expliquer en détail comment créer et rendre des modèles, regardez l'exemple suivant pour un aperçu rapide de l'ensemble du processus. Tout d'abord, vous devez créer un nouveau fichier dans le templates/répertoire pour stocker le contenu du modèle:

|  |  |
| --- | --- |
| 1  2  3 | *{# templates/user/notifications.html.twig #}*  <h1>Hello {{ user\_first\_name }}!</h1>  <p>You have {{ notifications|length }} new notifications.</p> |

Ensuite, créez un [contrôleur](https://symfony.com/doc/current/controller.html) qui restitue ce modèle et lui transmet les variables nécessaires:

*// src/Controller/UserController.php*

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class UserController extends AbstractController

{

*// ...*

public function notifications()

{

*// get the user information and notifications somehow*

$userFirstName = '...';

$userNotifications = ['...', '...'];

*// the template path is the relative file path from `templates/`*

return $this->render('user/notifications.html.twig', [

*// this array defines the variables passed to the template,*

*// where the key is the variable name and the value is the variable value*

*// (Twig recommends using snake\_case variable names: 'foo\_bar' instead of 'fooBar')*

'user\_first\_name' => $userFirstName,

'notifications' => $userNotifications,

]);

}

}

### Attribution d'un nom au modèle [¶](https://symfony.com/doc/current/templates.html#template-naming)

Symfony recommande ce qui suit pour les noms de modèle:

* Utilisez [de cas de serpent](https://en.wikipedia.org/wiki/Snake_case) pour les noms de fichiers et des répertoires (par exemple blog\_posts.twig, admin/default\_theme/blog/index.twigetc.);
* Définissez deux extensions pour les noms de fichiers (par exemple , index.html.twigou blog\_posts.xml.twig) étant la première extension ( html, xml, etc.) , le format final que le modèle va générer.

Bien que les modèles génèrent généralement du contenu HTML, ils peuvent générer n'importe quel format texte. C'est pourquoi la convention à deux extensions simplifie la façon dont les modèles sont créés et rendus pour plusieurs formats.

### Emplacement du modèle [¶](https://symfony.com/doc/current/templates.html#template-location)

Les modèles sont stockés par défaut dans le templates/répertoire. Lorsqu'un service ou un contrôleur rend le product/index.html.twigmodèle, ils font en fait référence au <your-project>/templates/product/index.html.twigfichier.

Le répertoire de modèles par défaut est configurable avec l' option [twig.default\_path](https://symfony.com/doc/current/reference/configuration/twig.html#config-twig-default-path) et vous pouvez ajouter d'autres répertoires de modèles [comme expliqué plus loin](https://symfony.com/doc/current/templates.html#templates-namespaces) dans cet article.

### Variables de modèle [¶](https://symfony.com/doc/current/templates.html#template-variables)

Un besoin courant de modèles est d'imprimer les valeurs stockées dans les modèles transmis par le contrôleur ou le service. Les variables stockent généralement des objets et des tableaux au lieu de chaînes, de nombres et de valeurs booléennes. C'est pourquoi Twig fournit un accès rapide aux variables PHP complexes. Considérez le modèle suivant:

|  |  |
| --- | --- |
| 1 | <p>{{ user.name }} added this comment on {{ comment.publishedAt|date }}</p> |

La user.namenotation signifie que vous souhaitez afficher certaines informations ( name) stockées dans une variable ( user). Est-ce userun tableau ou un objet? Est-ce nameune propriété ou une méthode? Dans Twig, cela n'a pas d'importance.

Lors de l'utilisation de la foo.barnotation, Twig essaie d'obtenir la valeur de la variable dans l'ordre suivant:

1. $foo['bar'] (tableau et élément);
2. $foo->bar (objet et propriété publique);
3. $foo->bar() (objet et méthode publique);
4. $foo->getBar()( méthode objet et getter );
5. $foo->isBar()( méthode objet et émetteur );
6. $foo->hasBar()( méthode objet et hasser );
7. Si aucun des éléments ci-dessus n'existe, utilisez null.

Cela permet de faire évoluer le code de votre application sans avoir à changer le code du modèle (vous pouvez commencer avec des variables de tableau pour la preuve de concept de l'application, puis passer à des objets avec des méthodes, etc.)

### Lien vers des pages [¶](https://symfony.com/doc/current/templates.html#linking-to-pages)

Au lieu d'écrire les URL des liens à la main, utilisez la path()fonction pour générer des URL en fonction de la [configuration de routage](https://symfony.com/doc/current/routing.html#routing-creating-routes) .

Plus tard, si vous souhaitez modifier l'URL d'une page particulière, il vous suffit de changer la configuration de routage: les modèles généreront automatiquement la nouvelle URL.

Considérez la configuration de routage suivante:

* [Annotations](https://symfony.com/doc/current/templates.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | *// src/Controller/BlogController.php*  namespace App\Controller;  *// ...*  use Symfony\Component\Routing\Annotation\Route;  class BlogController extends AbstractController  {  */\*\**  *\* @Route("/", name="blog\_index")*  *\*/*  public function index()  {  *// ...*  }  */\*\**  *\* @Route("/article/{slug}", name="blog\_post")*  *\*/*  public function show(string $slug)  {  *// ...*  }  } |

* [YAML](https://symfony.com/doc/current/templates.html)

* [XML](https://symfony.com/doc/current/templates.html)

* [PHP](https://symfony.com/doc/current/templates.html)

Utilisez la path()fonction Twig pour créer un lien vers ces pages et passez le nom de la route comme premier argument et les paramètres de la route comme deuxième argument facultatif:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <a href="{{ path('blog\_index') }}">Homepage</a>  *{# ... #}*  {% for post in blog\_posts %}  <h1>  <a href="{{ path('blog\_post', {slug: post.slug}) }}">{{ post.title }}</a>  </h1>  <p>{{ post.excerpt }}</p>  {% endfor %} |

La path()fonction génère des URL relatives. Si vous devez générer des URL absolues (par exemple lors du rendu des modèles d'e-mails ou de flux RSS), utilisez la url()fonction, qui prend les mêmes arguments que path() (par exemple ).<a href="{{ url('blog\_index') }}"> ... </a>

### Lien vers les ressources CSS, JavaScript et image [¶](https://symfony.com/doc/current/templates.html#linking-to-css-javascript-and-image-assets)

Si un modèle doit être lié à un élément statique (par exemple une image), Symfony fournit une asset()fonction Twig pour aider à générer cette URL. Tout d'abord, installez le assetpackage:

|  |
| --- |
| composer require symfony/asset |

Vous pouvez maintenant utiliser la asset()fonction:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *{# the image lives at "public/images/logo.png" #}*  <img src="{{ asset('images/logo.png') }}" alt="Symfony!"/>  *{# the CSS file lives at "public/css/blog.css" #}*  <link href="{{ asset('css/blog.css') }}" rel="stylesheet"/>  *{# the JS file lives at "public/bundles/acme/js/loader.js" #}*  <script src="{{ asset('bundles/acme/js/loader.js') }}"></script> |

Le asset()but principal de la fonction est de rendre votre application plus portable. Si votre application vit à la racine de votre hôte (par exemple https://example.com), alors le chemin rendu devrait être /images/logo.png. Mais si votre application réside dans un sous-répertoire (par exemple https://example.com/my\_app), chaque chemin d'accès aux ressources doit être rendu avec le sous-répertoire (par exemple /my\_app/images/logo.png). La asset()fonction s'en charge en déterminant comment votre application est utilisée et en générant les bons chemins en conséquence.

**POINTE**

La asset()fonction prend en charge diverses techniques de contournement du cache via les options de configuration [version](https://symfony.com/doc/current/reference/configuration/framework.html#reference-framework-assets-version) , [version\_format](https://symfony.com/doc/current/reference/configuration/framework.html#reference-assets-version-format) et [json\_manifest\_path](https://symfony.com/doc/current/reference/configuration/framework.html#reference-assets-json-manifest-path) .

**POINTE**

Si vous souhaitez aider à empaqueter, versionner et minimiser vos ressources JavaScript et CSS de manière moderne, lisez [le Webpack Encore de Symfony](https://symfony.com/doc/current/frontend.html) .

Si vous avez besoin d'URL absolues pour les actifs, utilisez la absolute\_url()fonction Twig comme suit:

|  |  |
| --- | --- |
| 1  2  3 | <img src="{{ absolute\_url(asset('images/logo.png')) }}" alt="Symfony!"/>  <link rel="shortcut icon" href="{{ absolute\_url('favicon.png') }}"> |

### La variable globale de l'application [¶](https://symfony.com/doc/current/templates.html#the-app-global-variable)

Symfony crée un objet contextuel qui est injecté automatiquement dans chaque modèle Twig sous la forme d'une variable appelée app. Il donne accès à certaines informations d'application:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <p>Username: {{ app.user.username ?? 'Anonymous user' }}</p>  {% if app.debug %}  <p>Request method: {{ app.request.method }}</p>  <p>Application Environment: {{ app.environment }}</p>  {% endif %} |

La appariable (qui est une instance de [AppVariable](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Bridge/Twig/AppVariable.php)) vous donne accès à ces variables:

App. User

L' [objet utilisateur actuel](https://symfony.com/doc/current/security.html#create-user-class) ou null si l'utilisateur n'est pas authentifié.

app.request

L' [Request](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/HttpFoundation/Request.php)objet qui stocke les [données de demande en](https://symfony.com/doc/current/components/http_foundation.html#accessing-request-data) cours (selon votre application, il peut s'agir d'une [sous-demande](https://symfony.com/doc/current/components/http_kernel.html#http-kernel-sub-requests) ou d'une demande régulière).

app.session

L' [Session](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/HttpFoundation/Session/Session.php)objet qui représente le courant [de la session de l' utilisateur](https://symfony.com/doc/current/session.html) ou nuls' il n'y en a pas.

app.flashes

Un tableau de tous les [messages flash](https://symfony.com/doc/current/controller.html#flash-messages) stockés dans la session. Vous pouvez également obtenir uniquement les messages d'un certain type (par exemple app.flashes('notice')).

app.environment

Le nom du courant [environnement de configuration](https://symfony.com/doc/current/configuration.html#configuration-environments) ( dev, prod, etc.).

app.debug

Vrai si en [mode débogage](https://symfony.com/doc/current/configuration/front_controllers_and_kernel.html#debug-mode) . Faux sinon.

app.token

Un [TokenInterface](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Component/Security/Core/Authentication/Token/TokenInterface.php) objet représentant le jeton de sécurité.

En plus de la appariable globale injectée par Symfony, vous pouvez également [injecter des variables automatiquement dans tous les modèles Twig](https://symfony.com/doc/current/templating/global_variables.html) .

## **Modèles de rendu**[**¶**](https://symfony.com/doc/current/templates.html#rendering-templates)

### Rendu d'un modèle dans les contrôleurs [¶](https://symfony.com/doc/current/templates.html#rendering-a-template-in-controllers)

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

*// src/Controller/ProductController.php*

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Response;

class ProductController extends AbstractController

{

public function index()

{

*// ...*

*// the `render()` method returns a `Response` object with the*

*// contents created by the template*

return $this->render('product/index.html.twig', [

'category' => '...',

'promotions' => ['...', '...'],

]);

*// the `renderView()` method only returns the contents created by the*

*// template, so you can use those contents later in a `Response` object*

$contents = $this->renderView('product/index.html.twig', [

'category' => '...',

'promotions' => ['...', '...'],

]);

return new Response($contents);

}

}

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 render()méthode du twigservice.

### Rendre un modèle dans les services [¶](https://symfony.com/doc/current/templates.html#rendering-a-template-in-services)

Injectez le twig service Symfony dans vos propres services et utilisez sa render() méthode.  Lorsque vous utilisez le [câblage automatique de service,](https://symfony.com/doc/current/service_container/autowiring.html) vous devez uniquement ajouter un argument dans le constructeur de service et le taper avec la [Environment](https://github.com/symfony/symfony/blob/5.1/src/Twig/Environment.php) classe:

*// src/Service/SomeService.php*

namespace App\Service;

use Twig\Environment;

class SomeService

{

private $twig;

public function \_\_construct(Environment $twig)

{

$this->twig = $twig;

}

public function someMethod()

{

*// ...*

$htmlContents = $this->twig->render('product/index.html.twig', [

'category' => '...',

'promotions' => ['...', '...'],

]);

}

}

### Rendre un modèle dans les e-mails [¶](https://symfony.com/doc/current/templates.html#rendering-a-template-in-emails)

Lisez les documents sur l' [intégration de mailer et Twig](https://symfony.com/doc/current/mailer.html#mailer-twig) .

### Rendre un modèle directement à partir d'une route [¶](https://symfony.com/doc/current/templates.html#rendering-a-template-directly-from-a-route)

Bien que les modèles soient généralement rendus dans les contrôleurs et les services, vous pouvez rendre des pages statiques qui n'ont pas besoin de variables directement à partir de la définition de l'itinéraire. Utilisez le spécial [TemplateController](https://github.com/symfony/symfony/blob/5.1/src/Symfony/Bundle/FrameworkBundle/Controller/TemplateController.php) fourni par Symphony:

* [YAML](https://symfony.com/doc/current/templates.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | *# config/routes.yaml*  acme\_privacy:  path: /privacy  controller: Symfony\Bundle\FrameworkBundle\Controller\TemplateController  defaults:  *# the path of the template to render*  template: 'static/privacy.html.twig'  *# special options defined by Symfony to set the page cache*  maxAge: 86400  sharedAge: 86400  *# whether or not caching should apply for client caches only*  private: true  *# optionally you can define some arguments passed to the template*  context:  site\_name: 'ACME'  theme: 'dark' |

* [XML](https://symfony.com/doc/current/templates.html)

* [PHP](https://symfony.com/doc/current/templates.html)

**Nouveau dans la version 5.1:**L' contextoption a été introduite dans Symfony 5.1.

### Vérifier si un modèle existe [¶](https://symfony.com/doc/current/templates.html#checking-if-a-template-exists)

Les modèles sont chargés dans l'application à l'aide d'un [chargeur de modèles Twig](https://twig.symfony.com/doc/2.x/api.html#loaders) , qui fournit également une méthode pour vérifier l'existence de modèles. Tout d'abord, récupérez le chargeur:

*// in a controller extending from AbstractController*

$loader = $this->get('twig')->getLoader();

*// in a service using autowiring*

use Twig\Environment;

public function \_\_construct(Environment $twig)

{

$loader = $twig->getLoader();

}

Ensuite, passez le chemin du modèle Twig à la exists()méthode du chargeur:

if ($loader->exists('theme/layout\_responsive.html.twig')) {

*// the template exists, do something*

*// ...*

}

## **Modèles de débogage**[**¶**](https://symfony.com/doc/current/templates.html#debugging-templates)

Symfony fournit plusieurs utilitaires pour vous aider à déboguer les problèmes dans vos modèles.

### Modèles de brindilles pelucheuses [¶](https://symfony.com/doc/current/templates.html#linting-twig-templates)

La lint: twig commande vérifie que vos modèles Twig n'ont pas d'erreurs de syntaxe. Il est utile de l'exécuter avant de déployer votre application en production (par exemple dans votre serveur d'intégration continue):

|  |
| --- |
| php bin/console lint:twig  php bin/console lint:twig templates/email/  php bin/console lint:twig templates/article/recent\_list.html.twig  php bin/console lint:twig --show-deprecations templates/email/ |

### Inspection des informations sur les rameaux [¶](https://symfony.com/doc/current/templates.html#inspecting-twig-information)

La debug: twig commande répertorie toutes les informations disponibles sur Twig (fonctions, filtres, variables globales, etc.). Il est utile de vérifier si vos [extensions Twig personnalisées](https://symfony.com/doc/current/templating/twig_extension.html) fonctionnent correctement et également de vérifier les fonctionnalités Twig ajoutées lors de l' [installation des packages](https://symfony.com/doc/current/setup.html#symfony-flex) :

|  |
| --- |
| php bin/console debug:twig  php bin/console debug:twig --filter=date  php bin/console debug:twig @Twig/Exception/error.html.twig |

### Les utilitaires Dump Twig [¶](https://symfony.com/doc/current/templates.html#the-dump-twig-utilities)

Symfony fournit une fonction [dump ()](https://symfony.com/doc/current/components/var_dumper.html#components-var-dumper-dump) comme alternative améliorée à la var\_dump() fonction PHP . Cette fonction est utile pour inspecter le contenu de n'importe quelle variable et vous pouvez également l'utiliser dans les modèles Twig.

Tout d'abord, assurez-vous que le composant VarDumper est installé dans l'application:

|  |
| --- |
| composer require symfony/var-dumper |

Utilisez ensuite la balise ou la fonction selon vos besoins: {% dump %} {{ dump() }}

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | *{# templates/article/recent\_list.html.twig #}*  *{# the contents of this variable are sent to the Web Debug Toolbar*  *instead of dumping them inside the page contents #}*  {% dump articles %}  {% for article in articles %}  *{# the contents of this variable are dumped inside the page contents*  *and they are visible on the web page #}*  {{ dump(article) }}  <a href="/article/{{ article.slug }}">  {{ article.title }}  </a>  {% endfor %} |

Pour éviter les fuites d'informations sensibles, la dump()fonction / balise n'est disponible que dans les [environnements de configuration](https://symfony.com/doc/current/configuration.html#configuration-environments)dev et . Si vous essayez de l'utiliser dans l' environnement, vous verrez une erreur PHP.test prod

## **Réutilisation du contenu du modèle**[**¶**](https://symfony.com/doc/current/templates.html#reusing-template-contents)

### Inclusion de modèles [¶](https://symfony.com/doc/current/templates.html#including-templates)

Si certains codes Twig sont répétés dans plusieurs modèles, vous pouvez les extraire dans un seul « fragment de modèle » et les inclure dans d'autres modèles. Imaginez que le code suivant pour afficher les informations utilisateur se répète à plusieurs endroits:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | *{# templates/blog/index.html.twig #}*  *{# ... #}*  <div class="user-profile">  <img src="{{ user.profileImageUrl }}" alt="{{ user.fullName }}"/>  <p>{{ user.fullName }} - {{ user.email }}</p>  </div> |

Tout d'abord, créez un nouveau modèle Twig appelé blog/\_user\_profile.html.twig(le \_préfixe est facultatif, mais c'est une convention utilisée pour mieux différencier les modèles complets et les fragments de modèle).

Ensuite, supprimez ce contenu du blog/index.html.twigmodèle d' origine et ajoutez ce qui suit pour inclure le fragment de modèle:

|  |  |
| --- | --- |
| 1  2  3  4 | *{# templates/blog/index.html.twig #}*  *{# ... #}*  {{ include('blog/\_user\_profile.html.twig') }} |

La include()fonction Twig prend comme argument le chemin du modèle à inclure. Le modèle inclus a accès à toutes les variables du modèle qui l'inclut (utilisez l' option [with\_context](https://twig.symfony.com/doc/2.x/functions/include.html) pour contrôler cela).

Vous pouvez également transmettre des variables au modèle inclus. Ceci est utile par exemple pour renommer des variables. Imaginez que votre modèle stocke les informations utilisateur dans une variable appelée à la blog\_post.authorplace de la uservariable attendue par le fragment de modèle. Utilisez ce qui suit pour renommer la variable:

|  |  |
| --- | --- |
| 1  2  3  4 | *{# templates/blog/index.html.twig #}*  *{# ... #}*  {{ include('blog/\_user\_profile.html.twig', {user: blog\_post.author}) }} |

### Incorporation de contrôleurs [¶](https://symfony.com/doc/current/templates.html#embedding-controllers)

[L'inclusion de fragments de modèle](https://symfony.com/doc/current/templates.html#templates-include) est utile pour réutiliser le même contenu sur plusieurs pages. Cependant, cette technique n'est pas la meilleure solution dans certains cas.

Imaginez que le fragment de modèle affiche les trois articles de blog les plus récents. Pour ce faire, il doit effectuer une requête de base de données pour obtenir ces articles. Lorsque vous utilisez la include()fonction, vous devez effectuer la même requête de base de données dans chaque page qui inclut le fragment. Ce n'est pas très pratique.

Une meilleure alternative consiste à **intégrer le résultat de l'exécution de certains contrôleurs** avec les fonctions render()et controller()Twig.

Tout d'abord, créez le contrôleur qui rend un certain nombre d'articles récents:

*// src/Controller/BlogController.php*

namespace App\Controller;

*// ...*

class BlogController extends AbstractController

{

public function recentArticles($max = 3)

{

*// get the recent articles somehow (e.g. making a database query)*

$articles = ['...', '...', '...'];

return $this->render('blog/\_recent\_articles.html.twig', [

'articles' => $articles

]);

}

}

Ensuite, créez le blog/\_recent\_articles.html.twigfragment de modèle (le \_préfixe dans le nom du modèle est facultatif, mais c'est une convention utilisée pour mieux différencier les modèles complets et les fragments de modèle):

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | *{# templates/blog/\_recent\_articles.html.twig #}*  {% for article in articles %}  <a href="{{ path('blog\_show', {slug: article.slug}) }}">  {{ article.title }}  </a>  {% endfor %} |

Vous pouvez maintenant appeler ce contrôleur à partir de n'importe quel modèle pour intégrer son résultat:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | *{# templates/base.html.twig #}*  *{# ... #}*  <div id="sidebar">  *{# if the controller is associated with a route, use the path() or url() functions #}*  {{ render(path('latest\_articles', {max: 3})) }}  {{ render(url('latest\_articles', {max: 3})) }}  *{# if you don't want to expose the controller with a public URL,*  *use the controller() function to define the controller to execute #}*  {{ render(controller(  'App\\Controller\\BlogController::recentArticles', {max: 3}  )) }}  </div> |

Lors de l'utilisation de la controller()fonction, les contrôleurs ne sont pas accessibles via une route Symfony régulière mais via une URL spéciale utilisée exclusivement pour servir ces fragments de modèle. Configurez cette URL spéciale dans l' fragmentsoption:

* [YAML](https://symfony.com/doc/current/templates.html)

|  |  |
| --- | --- |
| 1  2  3  4 | *# config/packages/framework.yaml*  framework:  *# ...*  fragments: { path: /\_fragment } |

* [XML](https://symfony.com/doc/current/templates.html)

* [PHP](https://symfony.com/doc/current/templates.html)

**MISE EN GARDE**

L'incorporation de contrôleurs nécessite de faire des requêtes à ces contrôleurs et de rendre certains modèles en conséquence. Cela peut avoir un impact significatif sur les performances de l'application si vous intégrez de nombreux contrôleurs. Si possible, [mettez en cache le fragment de modèle](https://symfony.com/doc/current/http_cache/esi.html) .

**VOIR ÉGALEMENT**

Les modèles peuvent également [incorporer du contenu de manière asynchrone](https://symfony.com/doc/current/templating/hinclude.html) avec la hinclude.jsbibliothèque JavaScript.

### Héritage et dispositions de modèle [¶](https://symfony.com/doc/current/templates.html#template-inheritance-and-layouts)

Au fur et à mesure que votre application se développe, vous trouverez de plus en plus d'éléments répétés entre les pages, tels que les en-têtes, les pieds de page, les barres latérales, etc. L' [inclusion de modèles](https://symfony.com/doc/current/templates.html#templates-include) et l' [intégration de contrôleurs](https://symfony.com/doc/current/templates.html#templates-embed-controllers) peuvent aider, mais lorsque les pages partagent une structure commune, il est préférable d'utiliser l' **héritage** .

Le concept d' [héritage de modèle Twig](https://twig.symfony.com/doc/2.x/tags/extends.html) est similaire à l'héritage de classe PHP. Vous définissez un modèle parent à partir duquel d'autres modèles peuvent s'étendre et les modèles enfants peuvent remplacer des parties du modèle parent.

Symfony recommande l'héritage de modèle à trois niveaux suivant pour les applications moyennes et complexes:

* templates/ base.html.twig, Définit les éléments communs à tous les modèles d'application, tels que <head>, <header>, <footer>, etc .;
* templates/layout.html.twig, s'étend à partir de base.html.twiget définit la structure de contenu utilisée dans toutes ou la plupart des pages, comme une mise en page de contenu + barre latérale à deux colonnes. Certaines sections de l'application peuvent définir leurs propres dispositions (par exemple templates/blog/layout.html.twig);
* templates/\*.html.twig, les pages d'application qui s'étendent à partir du layout.html.twigmodèle principal ou de toute autre disposition de section.

En pratique, le base.html.twigmodèle ressemblerait à ceci:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | *{# templates/base.html.twig #}*  <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>{% block title %}My Application{% endblock %}</title>  {% block stylesheets %}  <link rel="stylesheet" type="text/css" href="/css/base.css"/>  {% endblock %}  </head>  <body>  {% block body %}  <div id="sidebar">  {% block sidebar %}  <ul>  <li><a href="{{ path('homepage') }}">Home</a></li>  <li><a href="{{ path('blog\_index') }}">Blog</a></li>  </ul>  {% endblock %}  </div>  <div id="content">  {% block content %}{% endblock %}  </div>  {% endblock %}  </body>  </html> |

La [balise de bloc Twig](https://twig.symfony.com/doc/2.x/tags/block.html) définit les sections de page qui peuvent être remplacées dans les modèles enfants. Ils peuvent être vides, comme le contentbloc ou définir un contenu par défaut, comme le titlebloc, qui s'affiche lorsque les modèles enfants ne les remplacent pas.

Le blog/layout.html.twigmodèle pourrait ressembler à ceci:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *{# templates/blog/layout.html.twig #}*  {% extends 'base.html.twig' %}  {% block content %}  <h1>Blog</h1>  {% block page\_contents %}{% endblock %}  {% endblock %} |

Le modèle s'étend depuis base.html.twiget définit uniquement le contenu du content bloc. Le reste des blocs de modèles parents affichera leur contenu par défaut. Cependant, ils peuvent être remplacés par le modèle d'héritage de troisième niveau, tel que blog/index.html.twig, qui affiche l'index du blog:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | *{# templates/blog/index.html.twig #}*  {% extends 'blog/layout.html.twig' %}  {% block title %}Blog Index{% endblock %}  {% block page\_contents %}  {% for article in articles %}  <h2>{{ article.title }}</h2>  <p>{{ article.body }}</p>  {% endfor %}  {% endblock %} |

Ce modèle s'étend à partir du modèle de deuxième niveau ( blog/layout.html.twig) mais remplace les blocs de différents modèles parents: page\_contentsde blog/layout.html.twiget titlede base.html.twig.

Lorsque vous effectuez le rendu du blog/index.html.twigmodèle, Symfony utilise trois modèles différents pour créer le contenu final. Ce mécanisme d'héritage augmente votre productivité car chaque modèle inclut uniquement son contenu unique et laisse le contenu répété et la structure HTML à certains modèles parents.

**MISE EN GARDE**

Lors de l'utilisation extends, un modèle enfant est interdit de définir des parties de modèle en dehors d'un bloc. Le code suivant renvoie un SyntaxError:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *{# app/Resources/views/blog/index.html.twig #}*  {% extends 'base.html.twig' %}  *{# the line below is not captured by a "block" tag #}*  <div class="alert">Some Alert</div>  *{# the following is valid #}*  {% block content %}My cool blog posts{% endblock %} |

Lisez les documents d' [héritage de modèle Twig](https://twig.symfony.com/doc/2.x/tags/extends.html) pour en savoir plus sur la façon de réutiliser le contenu du bloc parent lors de la substitution de modèles et d'autres fonctionnalités avancées.

## **Échappement de sortie**[**¶**](https://symfony.com/doc/current/templates.html#output-escaping)

Imaginez que votre modèle inclut le code pour afficher le nom d'utilisateur. Si un utilisateur malveillant définit son nom et que vous affichez cette valeur inchangée, l'application affichera une fenêtre contextuelle JavaScript.Hello {{ Name }}<script>alert('hello!')</script>

Il s'agit d'une attaque XSS ( [Cross-Site Scripting](https://en.wikipedia.org/wiki/Cross-site_scripting) ). Et tandis que l'exemple précédent semble inoffensif, l'attaquant pourrait écrire du code JavaScript plus avancé pour effectuer des actions malveillantes.

Pour éviter cette attaque, utilisez «sortie d'échappement» pour transformer les caractères qui ont une signification spéciale (par exemple, remplacer <par l' &lt;entité HTML). Les applications Symfony sont sécurisées par défaut car elles effectuent un échappement de sortie automatique grâce à l' [option Twig autoescape](https://symfony.com/doc/current/reference/configuration/twig.html#config-twig-autoescape) :

|  |  |
| --- | --- |
| 1  2  3 | <p>Hello {{ name }}</p>  *{# if 'name' is '<script>alert('hello!')</script>', Twig will output this:*  *'<p>Hello &lt;script&gt;alert(&#39;hello!&#39;)&lt;/script&gt;</p>' #}* |

Si vous rendez une variable approuvée et contenant du contenu HTML, utilisez le [filtre brut Twig](https://twig.symfony.com/doc/2.x/filters/raw.html) pour désactiver l'échappement de sortie pour cette variable:

|  |  |
| --- | --- |
| 1  2  3 | <h1>{{ product.title|raw }}</h1>  *{# if 'product.title' is 'Lorem <strong>Ipsum</strong>', Twig will output*  *exactly that instead of 'Lorem &lt;strong&gt;Ipsum&lt;/strong&gt;' #}* |

Lisez les [documents d'échappement de sortie de Twig](https://twig.symfony.com/doc/2.x/api.html#escaper-extension) pour en savoir plus sur la façon de désactiver l'échappement de sortie pour un bloc ou même un modèle entier.

## **Espaces de noms de modèles**[**¶**](https://symfony.com/doc/current/templates.html#template-namespaces)

Bien que la plupart des applications stockent leurs modèles dans le templates/ répertoire par défaut , vous devrez peut-être stocker tout ou partie d'entre eux dans des répertoires différents. Utilisez l' twig.pathsoption pour configurer ces répertoires supplémentaires. Chaque chemin est défini comme une paire où est le répertoire du modèle et l' espace de noms Twig, ce qui est expliqué plus loin:key: valuekeyvalue

* [YAML](https://symfony.com/doc/current/templates.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | *# config/packages/twig.yaml*  twig:  *# ...*  paths:  *# directories are relative to the project root dir (but you*  *# can also use absolute directories)*  'email/default/templates': ~  'backend/templates': ~ |

* [XML](https://symfony.com/doc/current/templates.html)

* [PHP](https://symfony.com/doc/current/templates.html)

Lors du rendu d'un modèle, Symfony le recherche d'abord dans les twig.paths répertoires qui ne définissent pas d'espace de noms, puis revient au répertoire de modèles par défaut (généralement templates/).

En utilisant la configuration ci-dessus, si votre application affiche par exemple le layout.html.twigmodèle, Symfony recherchera d'abord email/default/templates/layout.html.twiget backend/templates/layout.html.twig. Si l'un de ces modèles existe, Symfony l'utilisera au lieu d'utiliser templates/layout.html.twig, qui est probablement le modèle que vous vouliez utiliser.

Twig résout ce problème avec les **espaces** de **noms** , qui regroupent plusieurs modèles sous un nom logique sans rapport avec leur emplacement réel. Mettez à jour la configuration précédente pour définir un espace de noms pour chaque répertoire de modèles:

* [YAML](https://symfony.com/doc/current/templates.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | *# config/packages/twig.yaml*  twig:  *# ...*  paths:  'email/default/templates': 'email'  'backend/templates': 'admin' |

* [XML](https://symfony.com/doc/current/templates.html)

* [PHP](https://symfony.com/doc/current/templates.html)

Maintenant, si vous rendez le layout.html.twigmodèle, Symfony rendra le templates/layout.html.twigfichier. Utilisez la syntaxe spéciale @+ espace de noms pour faire référence aux autres modèles d'espaces de noms (par exemple @email/layout.html.twiget @admin/layout.html.twig).

**REMARQUE**

Un seul espace de noms Twig peut être associé à plusieurs répertoires de modèles. Dans ce cas, l'ordre dans lequel les chemins sont ajoutés est important car Twig commencera à rechercher des modèles à partir du premier chemin défini.

### Modèles d'ensembles [¶](https://symfony.com/doc/current/templates.html#bundle-templates)

Si vous [installez des packages / bundles](https://symfony.com/doc/current/setup.html#symfony-flex) dans votre application, ils peuvent inclure leurs propres modèles Twig (dans le Resources/views/répertoire de chaque bundle). Pour éviter de jouer avec vos propres modèles, Symfony ajoute des modèles de bundle sous un espace de noms automatique créé après le nom du bundle.

Par exemple, les modèles d'un bundle appelé AcmeFooBundlesont disponibles sous l' AcmeFooe space de noms. Si cet ensemble comprend le modèle <your-project>/vendor/acmefoo-bundle/Resources/views/user/profile.html.twig, vous pouvez vous y référer en tant que @AcmeFoo/user/profile.html.twig.