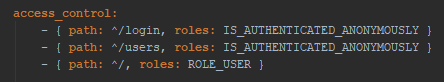
**Documentation technique TodoList**

1. **Authentification**

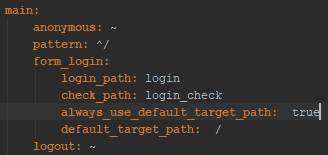
Le fonctionnement du processus d’authentification se trouve dans le fichier security.yml.

Toute les routes autre que « /login » et « /users » nécessite d’être authentifié.

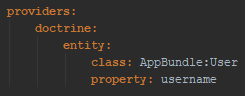


Si un utilisateur tente d’accéder à n’importe quel autre page il est redirigé vers la page d’authentification « /login ». Si l’authentification réussit l’utilisateur est redirigé vers la homepage (« / »).

Les routes rattachées à l’authentification sont gérées par le SecurityController.



Pour s’authentifier il faut fournir au provider une entité User en insérant son nom d’utilisateur et son mot de passe. Ses informations sont enregistrées en base de données.



1. **Au*to*risation/Gestion des droits d’accès**

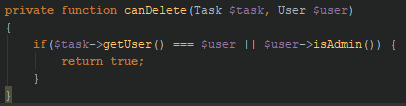
Il y a deux niveaux d’authentification : User et Admin



Seuls les admins peuvent supprimer les tâches qu’ils n’ont pas créés.

La gestion des droits se fait avec un Voter. Le fichier se trouve dans AppBundle/Security/TaskVoter.

Le Voter détermine si, en fonction de son rôle (user/admin), l’utilisateur peut ou non accéder à la ressource.

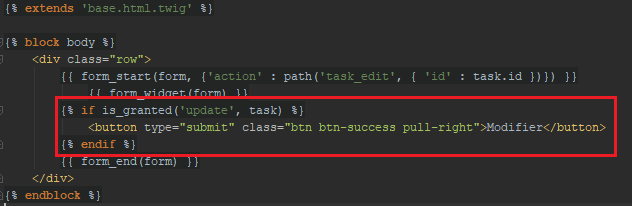


Cet élément du Voter vérifie si l’utilisateur peux supprimer la tâche (s’il est admin ou s’il est l’auteur).

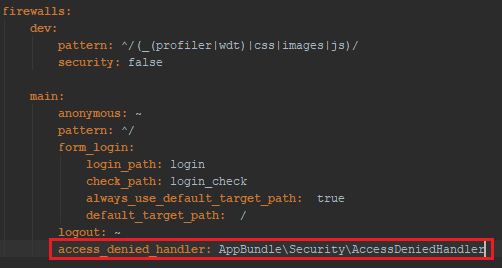
La modification suit la même logique.

Le Voter permet d’utiliser la fonction « is\_granted » suivi de deux arguments : l’action et l’entité.

Cette fonction est dans notre cas utilisée dans les templates.



Si un utilisateur cherche à accéder à une ressource dont il n’a pas les droits via l’url, l’erreur est gérée par un « access\_denied\_handler ». Le fichier se trouve, tout comme le Voter, dans le dossier Security du Bundle.



1. **Qualité de code et dette technique**
2. **Version de Symfony**

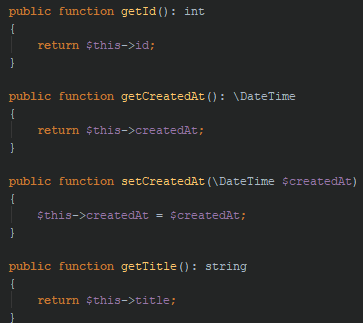
L’application a évolué de Symfony 3.3 à 3.4. L’intérêt de ce changement est de passer sur une version stable de Symfony, une LTS (Long Term Support).

<https://symfony.com/roadmap>

La version sera 3.4 sera maintenu jusqu’en 2021 lorsque la version 3.3 ne l’ai déjà plus.

1. **Type hint**

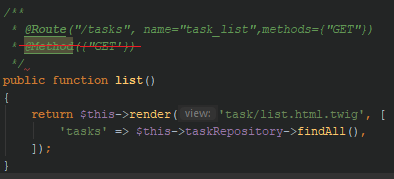
Le type hinting a été mis en place pour chaque entité.

****

1. **Adaptation de la syntaxe**

Dans Symfony 3.4 les méthodes de controller n’ont plus besoin d’être préfixée de « action ».

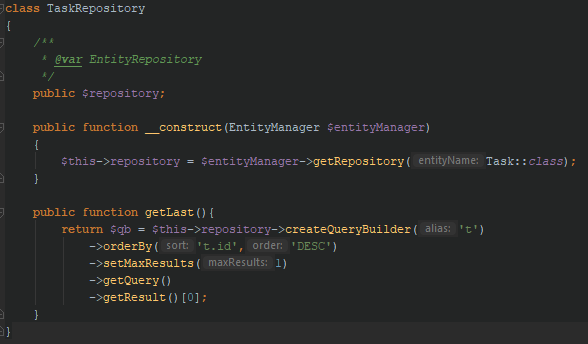
Aussi, les commentaires déterminant les routes dans les commentaire peuvent prendre la méthode en paramètre (Get, Post, Delete).



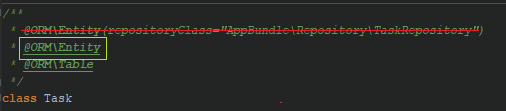
**Gestion des repositories**

Les repositories sont gérés sous le principe de « composition over inheritance ».

Cette architecture (inspiré d’un tuto de Toma Vostruba) permet de déclarer les repositories comme des services.

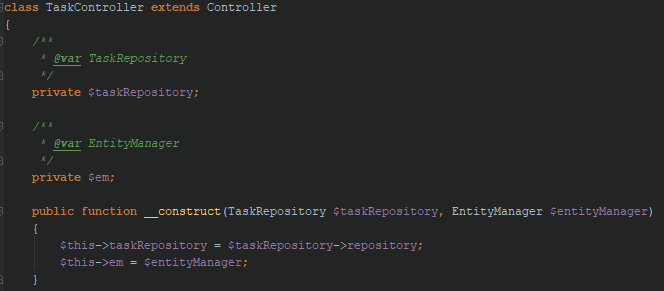


Les entités sont désormais indépendantes de tout repository.



Ce principe a plusieurs avantages :

* Possibilité d’utiliser le type hinting dans les repositories
* Possibilité d’utiliser l’autowiring dans les controllers pour appeler les repositories.



**e) Mis en place de PSR**

Utilisation de php-cs-fixer pour mettre le code aux normes PSR.

1. **Fonctionnement des tests**

Il y a 3 classes de tests respectives aux 3 principaux controller :

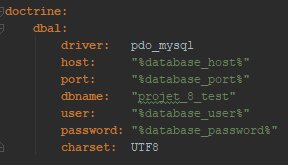
* DefaultControllr
* TaskController
* UserController

Toutes les classes de tests héritent de la classe Boot qui utilise les fonction setUp et tearDown de PHPUnit.

La classe Boot contient également une méthode pour s’authentifier en tant que user ou admin ainsi qu’une méthode pour créer une tâche.

Cette classe permet la création en bdd de deux utilisateurs (user et admin) et la suppression des données pour chaque fin de test.

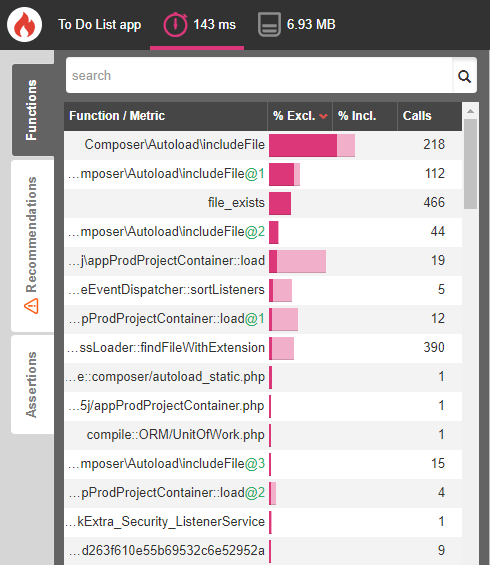
Les tests utilisent une base de données différente de l’application, configuré dans le fichier config\_test.yml.



1. **Performance**

Pour la performance l’outil Blackfire nous fournit des indicateurs pertinents.

Cependant, pour notre application, les éléments demandant le plus de performances sont des éléments de Symfony.



Cet indicateur de Blackfire nous liste les fonctions les plus exigentes pour la page d’accueil.

On remarque que l’autoloader de composer est le plus exigeant en terme de performance.

Pour améliorer les performances des applications composer a prévu plusieurs solutions listées ici :

<https://getcomposer.org/doc/articles/autoloader-optimization.md>

Pour notre application le premier niveau a été mis en place.