

ToDo

AUDIT DE CODE

Performance et qualité

## Table des matières

1 – CONTEXTE	2
2 – OBJECTIF	2
1 – VERSION DES PRINCIPAUX COMPOSANTS DE L'APPLICATION	3
2 – ANALYSE DE LA QUALITE DU CODE	3
A – ANALYSE CODE CLIMATE	3
3 – ANALYSE DE PERFORMANCE	3
1 – VERS QUELLE VERSION DE SYMFONY SE TOURNER ?	4
2 – L'UPGRADE	4
1 – LES AMÉLIORATIONS DEMANDÉES	5
A – CORRECTIONS D'ANOMALIES	5
1 – UNE TÂCHE DOIT ÊTRE ATTACHÉE À UN UTILISATEUR	5
2 – CHOISIR UN RÔLE POUR UN UTILISATEUR	5
B – IMPLÉMENTATION DE NOUVELLES FONCTIONNALITÉS	7
1 – AUTORISATION	7
2 – IMPLÉMENTATION DE TESTS AUTOMATISÉS	8
3 – LES AMÉLIORATIONS APPORTÉES	10
A – BACK-END	10
B– FRONT-END	13
C– CONTRAINTES SUR LES ATTRIBUTS	13
D– AMÉLIORATIONS POSSIBLES	15
1 – VERSION DES PRINCIPAUX COMPOSANTS DE L'APPLICATION	16
2 – ANALYSE DE CODE	16
3 – ANALYSE DE PERFORMANCE	16
1 – ANALYSE DE CODE	17
2 – ANALYSE DE PERFORMANCE	17
3 – AMÉLIORATIONS POSSIBLES	17

## 1 – CONTEXTE

La startup ToDo & Co a réussi une levée de fond pour le développement de l'application « To Do List » permettant de gérer ses tâches quotidiennes.

Un « Minimum Viable Product » a été développé pour permettre de présenter le concept à de potentiels investisseurs.

Le projet a été réalisé avec le framework Symfony.

## 2 – OBJECTIF

Les principaux objectifs sont :

- L'implémentation de nouvelles fonctionnalités
- La correction d'anomalies
- L'implémentation de tests automatisés
- L'analyse du projet sous forme d'audit

Il est également possible de corriger les points remontés par l'audit de qualité.

Ce document analysera donc le projet et explicitera les améliorations apportées.

### 1 – VERSION DES PRINCIPAUX COMPOSANTS DE L'APPLICATION

PHP	7.2.10 (environnement de développement)
Symfony	3.1.6
Bootstrap	3.3.7

### 2 – ANALYSE DE LA QUALITE DU CODE

#### A – ANALYSE CODE CLIMATE

En ignorant les fichiers propres à Symfony, le projet obtient un A en termes de qualité de code.

## Test for quality



Code climate : <https://codeclimate.com/github/Monsieur76/S8>

### 3 – ANALYSE DE PERFORMANCE

L'analyse de performance a été réalisée avec BlackFire sur trois pages différentes.

Page	Temps d'exécution	Mémoire	Blackfire
/	381 ms	11.4 MB	<a href="https://cutt.ly/veu0QOX">https://cutt.ly/veu0QOX</a>
/login	355 ms	11.4 MB	<a href="https://cutt.ly/VeigVEC">https://cutt.ly/VeigVEC</a>
/tasks/	507 ms	13.9 MB	<a href="https://cutt.ly/Yeig2UH">https://cutt.ly/Yeig2UH</a>

Cette valeur nous servira de référence afin de comparer les performances de la nouvelle version de l'application.

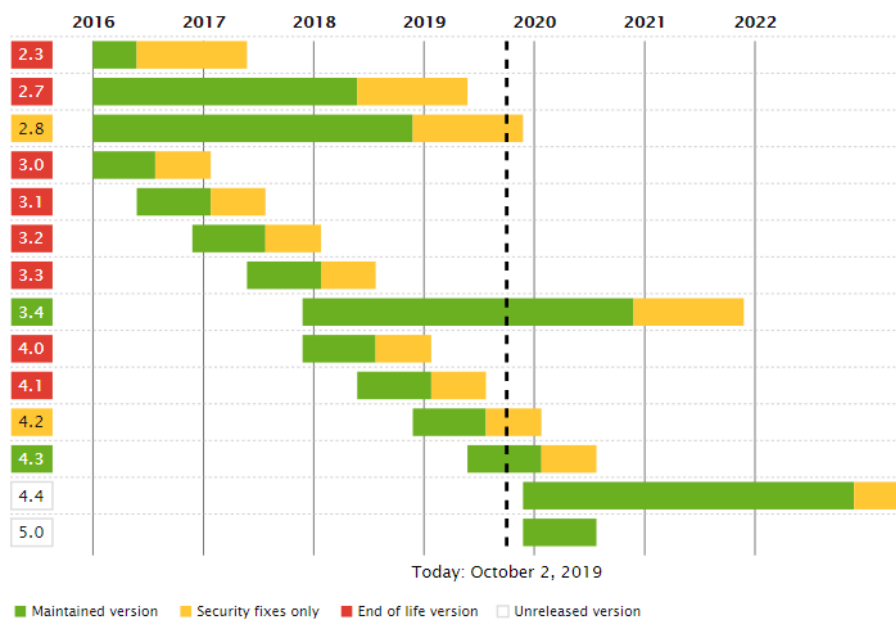
Référence : <https://blackfire.io/docs/introduction>

---

### III – SYMFONY

---

#### 1 – VERS QUELLE VERSION DE SYMFONY SE TOURNER ?



A la vue de la « road map » de Symfony, la solution choisie a été de passer à la version 3.4.

La version 3.4 est maintenue jusqu'à fin 2020.

La mise à niveau de version de symfony sera donc nécessaire plus tard mais pas de suite.

Référence : <https://symfony.com/roadmap>

#### 2 – L'UPGRADE

La mise à niveau comprend la mise à jour de plugin et de nouveau service pour cause de dépréciations.

Cela permet un gros gain de temps par rapport aux anciennes versions qui permette d'optimiser certaine fonction.

Tout ce qui concerne l'authentification sera ici explicité succinctement, les détails concernant la mise en place et le fonctionnement de l'authentification sont disponibles dans le fichier PDF dédié à l'authentification.

### 1 – LES AMÉLIORATIONS DEMANDÉES

#### A – CORRECTIONS D'ANOMALIES

##### 1 – UNE TÂCHE DOIT ÊTRE ATTACHÉE À UN UTILISATEUR

*src/AppBundle/Controller/TaskController.php*

```
$task->setUser($this->getUser());  
if ($this->getUser() === null) {  
    $task->setAuthor( author: 'Anonyme' );  
}
```

On définit ici la clef étrangère ( `$task->setUser()` ) de la tâche à celle de l'utilisateur courant ( `$this->getUser()` ).

La 2<sup>ème</sup> ligne signifie que si il n'y a pas d'utilisateur courant ( `$this->getUser() === null` ) L'auteur de la tâche en cour est anonyme.

Cela est juste par précaution.

##### 2 – CHOISIR UN RÔLE POUR UN UTILISATEUR

*App/config /security.yaml*

```
role_hierarchy:
  ROLE_USER: ROLE_USER
  ROLE_ADMIN: ROLE_ADMIN
```

En premier lieu, des rôles avec une hiérarchie ont été créés dans le security.yaml.

Le rôle du « Super Admin » sera explicité plus tard.

Références :

- <https://symfony.com/doc/current/security.html#denying-access-roles-and-other-authorization>  
(Rôles)

- <https://symfony.com/doc/current/security.html#security-role-hierarchy> (Hiérarchie)

La possibilité de choisir entre les deux rôles demandés a été ajouté au formulaire :

*src/Form/ UserType.php*

```
->add( child: 'roles', type: ChoiceType::class, [
    'multiple' => true,
    'expanded' => false,
    'choices' => [
        'Admin' => 'ROLE_ADMIN',
        'Utilisateur' => 'ROLE_USER',
    ],
],
```

Et enfin le choix est disponible dans la vue (création et édition) :

Roles

Admin

Utilisateur

Modifier

Référence :

- <https://symfony.com/doc/current/forms.html> (Formulaire)

- <https://symfony.com/doc/current/reference/forms/types/choice.html>

(Choice Type)

## B – IMPLÉMENTATION DE NOUVELLES FONCTIONNALITÉS

### 1 – AUTORISATION

Grâce aux « access\_control » (Contrôle d'accès), nous pouvons restreindre l'accès à certaines parties du site.

*config/packages/security.yaml*

```
access_control:
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/, roles: IS_AUTHENTICATED_ANONYMOUSLY }
```

Ici, toute la partie concernant les utilisateurs (/) est accessible à tous les utilisateurs anonyme.

Références :

- <https://symfony.com/doc/current/security.html#security-authorization-access-control>

- [https://symfony.com/doc/current/security/access\\_control.html](https://symfony.com/doc/current/security/access_control.html)

Ceux-ci a été complété avec les méthodes de symfony is\_garented (annotation).

Références :

<https://symfony.com/doc/3.4/security.html>

Un voter a été créé afin que les tâches puissent être supprimées uniquement par les utilisateurs les ayant créés et afin que les tâches de l'utilisateur « anonyme » puissent être supprimées par les



utilisateurs ayant le rôle administrateur.

*src/AppBundle/Security/UserVoter.php*

```
protected function voteOnAttribute($attribute, $task, TokenInterface $token)
{
    $user = $token->getUser();
    if (!$user instanceof UserInterface) {
        return false;
    }
    if ($task->getUser()->getId() == null && $user->getRoles() == ['ROLE_ADMIN']) {
        return true;
    }
    switch ($attribute) {
        case 'EDIT' || 'DELETE' && $task->getUser()->getId() == $user->getId():
            return $task->getUser()->getId() == $user->getId();
            break;
    }
    return false;
}
```

Si la tâche a pour clef étrangère null (donc pas d'utilisateur créé) et que l'utilisateur courant a pour rôle admin alors il return true.

La 2eme condition return la même valeur true avec le

```
return $task->getUser()->getId() == $user->getId();
```

Cette action du voter a également été utilisée dans la vue de la liste des tâches :

*App/Ressources/Views/task/list.html.twig*

```
</form>
{% if is_granted('EDIT',task)%}
    <form action="{ path('task_edit', {'id' : task.id }) }">
        <button class="btn btn-warning btn-sm pull-right">Modifier</button>
    </form>
```

Le bouton « Modifier » s'affiche désormais en fonction des permissions créées dans le voter.

## 2 – IMPLÉMENTATION DE TESTS AUTOMATISÉS





Les tests unitaires et fonctionnels ont été faits grâce à PHPUnit 6.5.0.

Les fichiers de tests se trouvent à la racine du projet dans le fichier tests/.

Les fichiers de couverture de code se trouvent à la racine du fichier.

La couverture de code est de 92% :

*Couverturecode.html*

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	<div><div></div></div>	92.31%	132 / 143	<div><div></div></div>	85.71%	36 / 42	<div><div></div></div>	62.50%	5 / 8
 Controller	<div><div></div></div>	92.68%	76 / 82	<div><div></div></div>	69.23%	9 / 13	<div><div></div></div>	50.00%	2 / 4
 Entity	<div><div></div></div>	88.37%	38 / 43	<div><div></div></div>	92.59%	25 / 27	<div><div></div></div>	50.00%	1 / 2
 Form	<div><div></div></div>	100.00%	18 / 18	<div><div></div></div>	100.00%	2 / 2	<div><div></div></div>	100.00%	2 / 2
 AppBundle.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

Une couverture de code de 92% n'est pas significative, des tests supplémentaires ont donc été ajoutés.

```
C:\wamp\www\minichat\S8\S8>vendor\bin\phpunit
PHPUnit 6.5.0 by Sebastian Bergmann and contributors.

.....                                     26 / 26 (100%)

Time: 20 seconds, Memory: 32.00MB

OK (26 tests, 57 assertions)

C:\wamp\www\minichat\S8\S8>
```

Les tests peuvent être lancés grâce à la commande « php bin/phpunit » ; pour filtrer et ne tester qu'une fonction en particulier on peut utiliser la commande suivante :

« php bin/phpunit --filter=nomDeLaFonction »

Les tests ont été effectués avec les nouvelles méthodes d'assertions (raccourcie) de la version 4.3 de Symfony ([https://symfony.com/doc/current/testing/functional\\_tests\\_assertions.html](https://symfony.com/doc/current/testing/functional_tests_assertions.html))

De nombreuses améliorations sont possibles concernant les tests, entre autres :

- Utilisation du bundle AliceBundle afin de gérer la base de données de test plus facilement

(<https://github.com/hautelook/AliceBundle>)

- Utilisation de Panther, un vrai navigateur web pour les tests, permettant entre autres de tester le CSS et le JavaScript. Panther est développé par l'équipe de Symfony.

(<https://symfony.com/blog/introducing-symfony-panther-a-browser-testing-and-web-scraping-library-for-php> | <https://les-tilleuls.coop/fr/blog/article/panther-php-symfony> |

<https://github.com/symfony/panther>)

- La version de PHPUnit fourni avec Symfony est la 6.5.0 mais des versions plus récentes sont disponibles (<https://phpunit.de/getting-started/phpunit-8.html>).

Références :

- <https://symfony.com/doc/current/testing.html>

- [https://symfony.com/doc/current/components/dom\\_crawler.html](https://symfony.com/doc/current/components/dom_crawler.html)

- <https://phpunit.de/getting-started/phpunit-6.html>

## 3 – LES AMÉLIORATIONS APPORTÉES

### A – BACK-END

Les méthodes ont eu des améliorations d'autowiring

```
public function listAction(TaskRepository $taskRepository)
```

Installation du bundle Faker

```
"incenteev/composer-parameter-handler": "^2.0",  
"fzaninotto/faker": "^1.8",  
"friendsofphp/php-cs-fixer": "^2.15"
```

Ceux lui ci permet de généré aléatoirement des noms, adresse, ect...

Référence :

<https://github.com/fzaninotto/Faker>

Installation du plugin de fixture.

```
"doctrine/doctrine-fixtures-bundle": "^3.2"
```

Ce plugin permet de généré une base de donnée aléatoire avec des donné factice.

```
protected $faker;
protected $token;
protected $encoder;
protected $manager;
protected $count = 21;
protected $maxRandom = 20;

public function __construct(
    UserPasswordEncoderInterface $encoder,
    TokenGeneratorInterface $token,
    ObjectManager $manager
) {
    $this->faker = Faker\Factory::create( locale: 'fr_FR');
    $this->encoder = $encoder;
    $this->token = $token;
    $this->manager = $manager;
}

public function randomRole()
{
    $random = rand(0, 1);
    if ($random === 0) {
        $roles = ['ROLE_ADMIN'];
    } else {
        $roles = ['ROLE_USER'];
    }
    return $roles;
}

protected function createMany($reference, string $className, int $count, callable $factory)
{
    for ($i = 1; $i < $count; $i++) {
        $entity = new $className();
        $factory($entity, $i);
        $this->manager->persist($entity);
        $this->addReference( name: $reference.$i, $entity);
        $this->manager->flush();
    }
}
```

Cette classe est une classe parent qui va permettre aux autre classe de généré leur fixture en fonction des entités qui gère.

Référence :

<https://symfony.com/doc/master/bundles/DoctrineFixturesBundle/index.html>

Et voici comment une classe étend de la classe baseController et fonctionne pour remplir notre base de données.

```
public function load(ObjectManager $manager)
{
    $this->createMany('task', Task::class, $this->count, function (Task
```

```

$task) use ($manager) {
    $user = $this->getReference('user'.rand(1, $this->maxRandom));
    $task->setTitle($this->faker->title);
    $task->setUser($user);
    $task->toggle($this->faker->boolean);
    $task->setAuthor($this->faker->name);
    $task->setContent($this->faker->text);
});
}

```

Les classes ne sont pas compliquées à mettre en place.

*Src/AppBundle/Controller/TaskController*

```

/**
 * @Route("/complete", name="task_list_done")
 * @param TaskRepository $taskRepository
 * @return Response
 */
public function listActionDone(TaskRepository $taskRepository)
{
    return $this->render('task/list.html.twig', ['tasks' =>
    $taskRepository->findBy(['isDone' =>
        true]]]);
}

```

Une méthode a été créée pour pouvoir avoir une séparation des tâches accomplies et non accomplies.

*Src/AppBundle/Controller*

```

/**
 * Class UserController
 * @package AppBundle\Controller
 * @Route("/users")
 */

```

Toutes les classes controller ont une route par défaut qui va s'étendre de cette annotation.

Référence :

<https://symfony.com/doc/3.4/routing.html>

Une clé étrangère a été créée pour les tâches ceux qui permettent de relier une tâche à un utilisateur.

*Src/AppBundle/Entity/Task.php*

```

/**
 * @ORM\ManyToOne(targetEntity="AppBundle\Entity\User", inversedBy="task")

```

```
* @ORM\OrderBy({"order" = "DESC", "id" = "DESC"})
* @ORM\JoinColumn(nullable=true)
*/
private $users;
```

## B– FRONT-END

Grâce au voter les boutons suppression et modification sont que visible à l'utilisateur a qui appartient la tâche.

*App/Ressources/Views/task*

```
{% if is_granted('EDIT',task)%}
    <form action="{{ path('task_edit', {'id' : task.id }) }}">
        <button class="btn btn-warning btn-sm pull-right">Modifier</button>
    </form>
{% endif %}
{% if is_granted('DELETE',task)%}
    <form action="{{ path('task_delete', {'id' : task.id }) }}">
        <button class="btn btn-danger btn-sm pull-right">Supprimer</button>
    </form>
```

Des path ont été rajouté pour lieur le bouton ToDo au la page d'accueille

```
<a class="navbar-brand" href="{{ path('homepage') }}">To Do List app</a>
```

Un autre path a été rajouté à la page

*App/Ressources/Views/default/index.html*

```
<a href="{{ path('task_list') }}" class="btn btn-info">Consulter la liste
des tâches à faire</a>
<a href="{{ path('task_list_done') }}" class="btn btn-secondary">Consulter
la liste des tâches terminées</a>
```

Une suppression d'une duplication du bouton créé une tâche a été effectuée.

Et les Modals ont été retravaillé pour avoir une meilleure vision de l'action par l'utilisateur.

## C– CONTRAINTES SUR LES ATTRIBUTS

Des contraintes ont été ajoutées sur les attributs des entités Utilisateurs et Tâches.

Sur tous les attributs de type « string », ces contraintes concernent :

- La longueur afin éviter un nom d'utilisateur, un titre ou un contenu de tâche d'une seule lettre.

- La longueur maximale pour éviter des erreurs de base de données type SQLSTATE et afficher un message d'erreur compréhensible pour l'utilisateur.

- Une contrainte sur un contenu vide avec Assert\NotBlank

Exemple :

*src/Entity/Task.php*

```
/**
 * @ORM\Column(type="text")
 * @Assert\NotBlank(message="Vous devez saisir du contenu.")
 * * @Assert\Length(
 *     min="3",
 *     max="255"
 *     minMessage="Le champ contenu doit au moins {{limit}} caractères de
Long",
 *     maxMessage="Le champ contenu ne peut pas contenir plus de
{{limit}} caractères"
 */
private $content;
```

Référence :

<https://symfony.com/doc/3.4/reference/constraints/Length.html>

<https://symfony.com/doc/3.4/reference/constraints/NotBlank.html>

Pour l'entité Utilisateur, une contrainte d'unicité a été ajoutée sur l'attribut « email ».

*src/AppBundle/Entity/User.php*

```
/**
 * @ORM\Table("user")
 * @ORM\Entity
 * @UniqueEntity("email")
 */
```

Pour l'entité User, une contrainte d'unicité a été ajoutée sur l'attribut « email ».

*src/AppBundle/Entity/User.php*

```
/**
 * @ORM\Column(type="string", length=60, unique=true)
 * @Assert\NotBlank(message="Vous devez saisir une adresse email.")
 * @Assert\Email(message="Le format de l'adresse n'est pas correcte.")
 */
private $email;
```

Ceux qui permet de ne pas enregistrer une adresse email qui n'est pas valide.

Référence :

<https://symfony.com/doc/3.4/reference/constraints/Email.html>

## D– AMÉLIORATIONS POSSIBLES

Voici une liste de potentielles améliorations pour l'application :

- Une page « compte » pour les utilisateurs afin qu'ils puissent voir leurs informations et les modifier.
- La possibilité de s'inscrire en fonction de l'utilisation future de l'application.
- Une fonctionnalité en cas de mot de passe oublié.
- En imaginant une application participative, la possibilité d'ajouter un champ « participants » aux tâches, et la possibilité de « s'ajouter » à une tâche pour les utilisateurs afin de voir qui travaille sur quoi et combien de personnes sont déjà en train d'effectuer une tâche.
- Ajouter des notifications, par exemple lorsqu'une tâche est « marquée comme faite » par un utilisateur autre que le créateur, le créateur pourrait en être averti.



## 1 – VERSION DES PRINCIPAUX COMPOSANTS DE L'APPLICATION

PHP	7.2.10 (environnement de développement)
Symfony	3,4,*

## 2 – ANALYSE DE CODE

Le projet obtient un A en termes de qualité de code.

# Test for quality



Code climate : <https://codeclimate.com/github/Monsieur76/S8>

## 3 – ANALYSE DE PERFORMANCE

Les performances ont été améliorées grâce à l'optimisation de « l'autoloader ».

Référence : <https://symfony.com/doc/current/performance.html#optimize-composer-autoloader>

L'analyse de performance a été réalisée avec BlackFire sur une page.

Page	Temps d'exécution	Mémoire	Blackfire
/	329ms	12.6mb	<a href="https://cutt.ly/reu0Wpl">https://cutt.ly/reu0Wpl</a>
/login	331ms	12.7mb	<a href="https://cutt.ly/meig3O8">https://cutt.ly/meig3O8</a>
/tasks/	481 ms	15.1mb	<a href="https://cutt.ly/keig6ID">https://cutt.ly/keig6ID</a>

---

## VI – COMPARATIF

---

### 1 – ANALYSE DE CODE

Les deux analyses de code obtiennent une note de A en termes de qualité de code.

La qualité de code n'a donc pas régressé avec cette nouvelle version de l'application.

### 2 – ANALYSE DE PERFORMANCE

Comparatif des performances réalisées grâce à Blackfire.

Page	Temps d'exécution	Mémoire	Blackfire
/	-8.45%	+10.7%	<a href="https://cutt.ly/Neu0WYK">https://cutt.ly/Neu0WYK</a>
/login	-6.73%	+10.8%	<a href="https://cutt.ly/7eigNOk">https://cutt.ly/7eigNOk</a>
/tasks/	-5.17%	+8.93%	<a href="https://cutt.ly/Aeig6K2">https://cutt.ly/Aeig6K2</a>

Le temps d'exécution des pages a diminué en ayant une application à jour, plus robuste, plus sécurisée et avec quelques fonctionnalités supplémentaires, c'est plutôt positif.

### 3 – AMÉLIORATIONS POSSIBLES

En termes de performances, des améliorations sont envisageables :

- Mettre en place une pagination pour les listes de tâches et des utilisateurs.
- Optimiser l'application en production comme avec le « dump autoload » :

<https://symfony.com/doc/current/performance.html>

- Les images et les scripts peuvent être mis en local pour un gain de temps lorsqu'ils devront être recherchés
- Les images peuvent être compressées (avec [compressor.io](https://compressor.io) par exemple) pour gagner du temps