

# Rapport Audit Qualité & Performance

ToDo & Co

Version : 1

Date de la dernière mise à jour :

**Documentation technique ToDo & Co**

**Page 1 | 8**

## SOMMAIRE

<b>1. Expression des besoins .....</b>	<b>3</b>
<b>2. Corrections d'anomalies réalisées .....</b>	<b>3</b>
<b>3. Nouvelles fonctionnalités .....</b>	<b>3</b>
<b>4. Tests automatisés .....</b>	<b>3</b>
<b>5. Audit Qualité .....</b>	<b>4</b>
5.1. issues non résolues.....	5
5.2. Plan d'actions Codacy.....	6
<b>6. Audit Performance.....</b>	<b>6</b>
6.1. Recommandations Blackfire.....	7
6.2. Autres recommandations.....	8

## 1. Expression des besoins

Avant de détailler nos actions mises en place, nous nous permettons de faire un petit rappel sur les besoins exprimés :

- Corrections d'anomalies
- Nouvelles fonctionnalités
- Tests automatisés
- Audit Qualité
- Audit Performance

## 2. Corrections d'anomalies réalisées

<b>ToDo &amp; Co</b>	Utilisateur rattaché à la tâche nouvellement créée.
	Les tâches déjà créées doivent être rattachées à un utilisateur anonyme
	Lors de la modification de la tâche, l'utilisateur ne peut pas être modifié
	Lors de la création d'un utilisateur, il doit être possible de choisir un rôle pour celui-ci (ROLE_USER, ROLE_ADMIN)
	Possibilité de changer le rôle d'un utilisateur lors de la modification

## 3. Nouvelles fonctionnalités

<b>ToDo &amp; Co</b>	La gestion des utilisateurs est accessible uniquement aux utilisateurs avec le ROLE_ADMIN
	Les tâches ne peuvent être supprimées que par les utilisateurs ayant créé les tâches en question.
	Les tâches rattachées à l'utilisateur "anonyme" ne peuvent être supprimées uniquement par les utilisateurs ayant le rôle administrateur (ROLE_ADMIN).





## 4. Tests automatisés



Nous avons mis en place des tests unitaires et fonctionnels permettant d'assurer que le fonctionnement de l'application est bien en adéquation avec les demandes.

Ces tests ont été mis en place à l'aide de **PHPUnit**.

Il est important de réaliser les tests en cas de modification ou d'amélioration du code au niveau de l'application et de s'assurer d'avoir un taux de couverture supérieur à 70%.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	<div><div></div></div>	98.37%	121 / 123	<div><div></div></div>	95.35%	41 / 43	<div><div></div></div>	87.50%	7 / 8
 Controller	<div><div></div></div>	97.10%	67 / 69	<div><div></div></div>	83.33%	10 / 12	<div><div></div></div>	75.00%	3 / 4
 Entity	<div><div></div></div>	100.00%	44 / 44	<div><div></div></div>	100.00%	29 / 29	<div><div></div></div>	100.00%	2 / 2
 Form	<div><div></div></div>	100.00%	10 / 10	<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

## Legend

Low: 0% to 50%    Medium: 50% to 90%    High: 90% to 100%

Generated by php-code-coverage 5.3.2 using PHP 7.0.23 with Xdebug 2.5.5 and PHPUnit 6.5.13 at Mon Sep 24 7:28:12 UTC 2018.

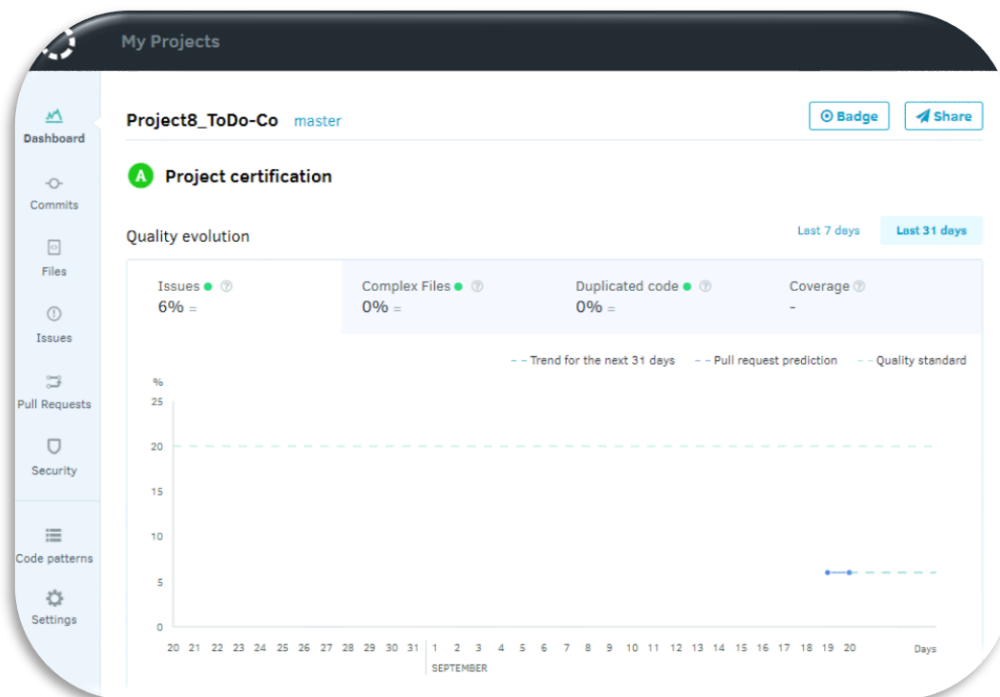
## 5. Audit Qualité



Un audit Qualité du code a été réalisé au niveau de l'application permettant de voir les améliorations et recommandations à apporter afin d'avoir une structure professionnelle et ainsi obtenir une application robuste.

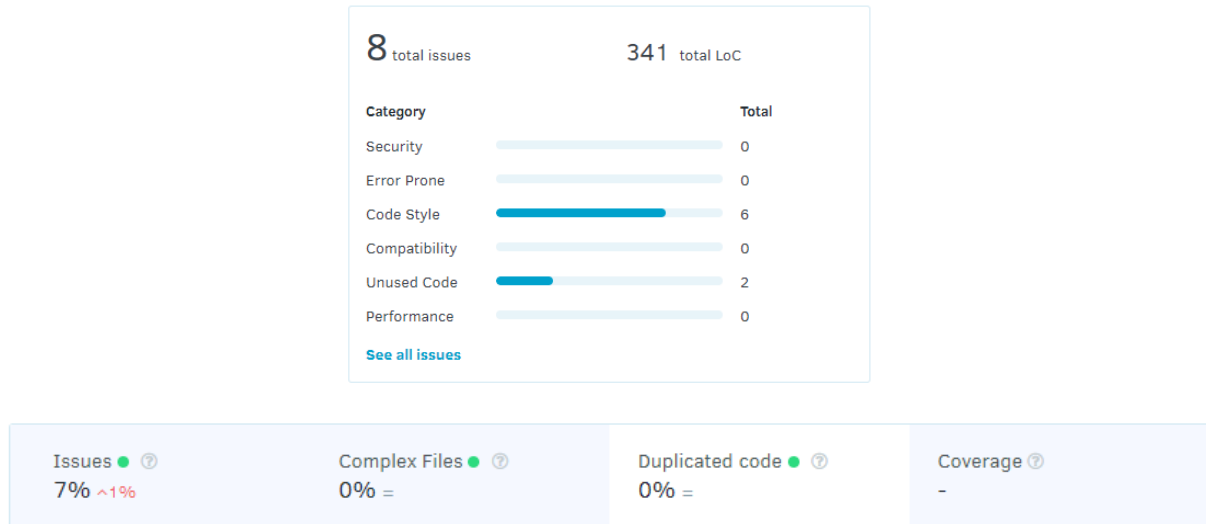
Pour cela, l'audit a été réalisé à l'aide de l'outil **Codacy**. Cet outil permet d'obtenir un code professionnel respectant les recommandations PSR et les bonnes pratiques.

Un niveau de certification est attribué en fonction de la qualité du code. Pour notre cas, nous avons obtenu la certification A.



Lors de l'audit Qualité du code, j'ai exclu les fichiers de tests ainsi que les fichiers de configuration de Symfony.

Ci-dessous le **tableau des issues** à résoudre présenté par Codacy ainsi que le **tableau des évolutions** au niveau qualité :



Nous devons donc nous focaliser sur les 8 issues non résolues représentant 7% des issues et avoir un taux de couverture de code approchant les 100%.

## 5.1. issues non résolues

- 1) Utiliser des variables supérieures ou égales à 3 caractères.

src/AppBundle/Controller/TaskController.php

```
Avoid variables with short names like $em. Configured minimum length is 3.
35 $em = $this->getDoctrine()->getManager();
```

src/AppBundle/Controller/UserController.php

```
Avoid variables with short names like $em. Configured minimum length is 3.
35 $em = $this->getDoctrine()->getManager();
```

src/AppBundle/Entity/Task.php

```
Avoid variables with short names like $id. Configured minimum length is 3.
19 private $id;
```

src/AppBundle/Entity/User.php

```
Avoid variables with short names like $id. Configured minimum length is 3.
22 private $id;
```

- 2) Simplification du code en supprimant les « else »

src / AppBundle / Controller / TaskController.php

```
La méthode deleteTaskAction utilise une expression else. Sinon, il n'est jamais nécessaire et vous pouvez simplifier le code pour travailler sans vous.
103 } sinon {
```

- 3) Suppression des arguments non utilisés dans les méthodes

src / AppBundle / Controller / SecurityController.php

```
Évitez les paramètres inutilisés tels que «$ request».
```

```
14 fonction publique loginAction (Request $ request)
```

src / AppBundle / Form / UserType.php

```
Évitez les paramètres non utilisés tels que «$ options».
```

```
15 fonction publique buildForm (FormBuilderInterface $ builder, tableau $ options)
```

## 5.2. Plan d'actions Codacy

Après avoir analysé la liste des issues proposées par Codacy, voici le plan d'actions qui a été mis en place.

Issue	Action
<b>Issue n°1</b> : Utiliser des variables $\geq 3$ caractères	Remplacement de <code>\$em</code> par <code>\$entityManager</code> . <code>\$id</code> n'a pas été remplacé.
<b>Issue n°2</b> : Suppression des arguments non utilisés dans les méthodes	Suppression de l'argument <code>\$request</code> dans la méthode <code>Login</code> du <code>SecurityController</code> . Suppression de l'argument <code>\$option</code> dans la méthode <code>buildForm</code> de la classe <code>UserType</code> .

## 6. Audit Performance



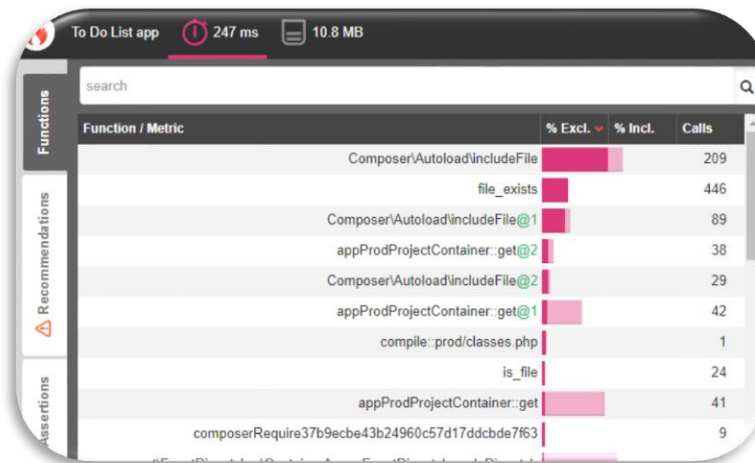
Afin d'analyser la performance de l'application `ToDoList`, j'ai utilisé l'outil `Blackfire` développé par `SensioLabs` et qui permet une analyse dans les détails des performances l'application.

`Blackfire` peut être utilisé à n'importe quelle étape du cycle de vie de l'application : pendant le développement, le test, le transfert et la production, pour profiler, tester, déboguer et optimiser ses performances.

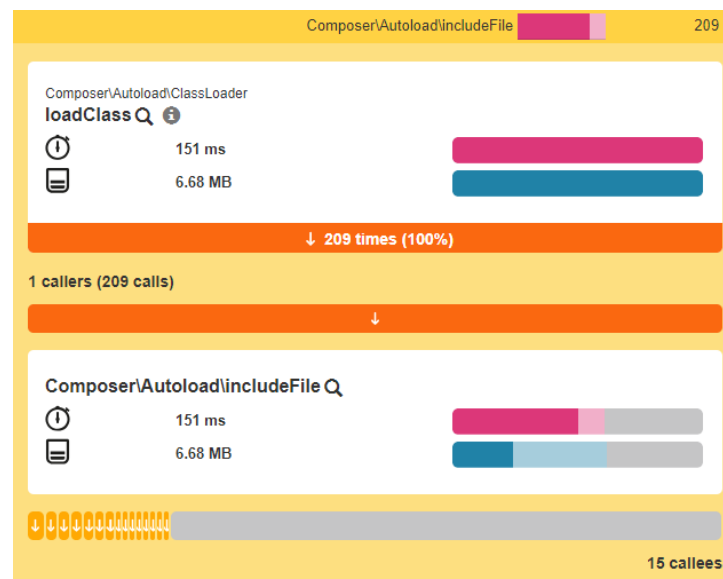
Il a permis de repérer les fonctions et méthodes mettant le plus de temps à s'exécuter.

Nous nous apercevons que suite à l'analyse de notre code via les listes des fonctions/méthodes et les `CallGraph` de l'outil, les fonctions et méthodes occasionnant un temps d'exécution faible sur les pages de notre application sont celles du Framework `Symfony` et notamment l'`Autoload`.

Prenons l'exemple de la page permettant de visualiser la liste des tâches,



Visualisation du détail de la première ligne : **Composer\Autoload\includeFile**



Nous pouvons analyser le temps d'exécution ainsi que la mémoire utilisée.

### 6.1. Recommendations Blackfire

Blackfire suggère les recommandations suivantes :

- 1) Les annotations de doctrine doivent être mises en cache en production

🚨 Doctrine annotations should be cached in production

Doctrine already includes a cache mechanism, so you just need to configure it. For instance, when using the Doctrine ORM inside a Symfony application, add the following configuration to cache the annotations parsing:


```

1 # app/config/config_prod.yml
2 doctrine:
3   orm:
4     metadata_cache_driver: apc

```

Blackfire recommande de mettre en place l'application ci-dessus en mode production.

## 2) La classmap de l'autoloader de Composer doit être vidée en mode production

 The Composer autoloader class map should be dumped in production

Using `composer dump-autoload --optimize` tells Composer to dump an optimized version of the autoloader by generating a **classmap**. Because the classmap can be huge, it's highly recommended to have a PHP opcode cache installed (like Zend OPcache).

Blackfire recommande d'utiliser la commande `composer dump-autoload --optimize` en mode production et d'utiliser un cache PHP comme **Zend OPcache**.

### 6.2. Autres recommandations

Afin d'optimiser également la performance de l'application, il est fortement recommandé de migrer celle-ci vers des versions plus récentes et maintenue de PHP et Symfony.