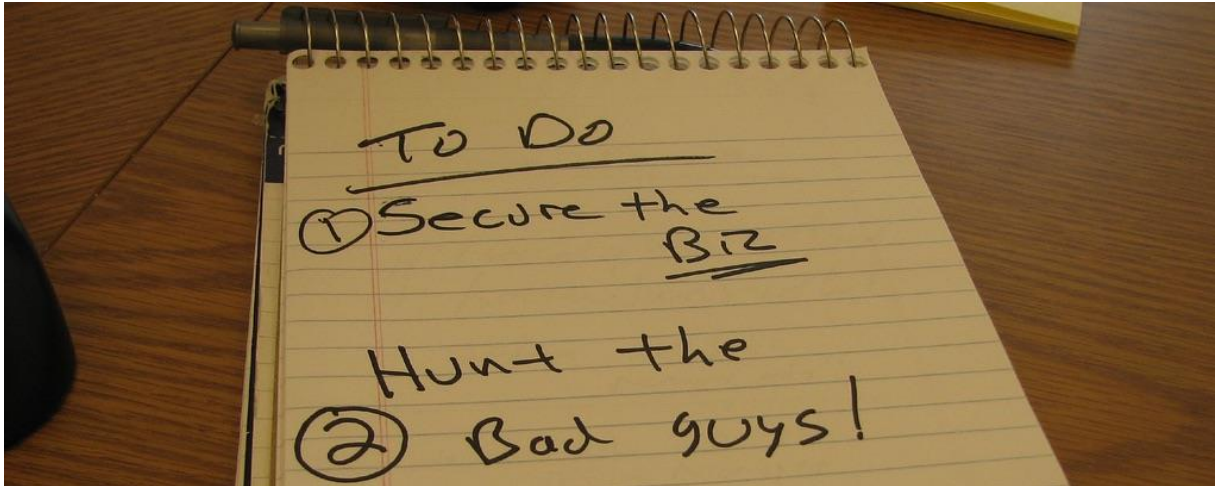


ToDo List



Audit Qualité de Code et Performance

Table des matières

1. Contexte	1
2. Qualité de code	1
2.1 Tests automatisés	1
2.2 Outils de revue de code	2
2.2.1 Codacy	3
2.2.2 Scrutinizer	7
2.2.3 Code Climate	11
2.2.4 SensiolabsInsight	13
2.2.5 Travis CI	16
2.3 Revue manuelle.....	16
3. Performance de l'application	18
3.1 Blackfire.....	18
3.2 Autres axes d'optimisation de la performance	20
3.2.1 Cache HTTP	20
3.2.2 Mises à jour	20
3.2.3 Optimisation des requêtes SQL	21
3.2.4 Les images	21
3.2.5 CSS et JavaScript	21
3.2.6 Lazy Loading	22
4. Conclusion	22

1. Contexte

ToDo & Co est une startup dont le coeur de métier est une application « ToDo List » permettant de gérer ses tâches quotidiennes.

L'entreprise vient tout juste d'être créée, et l'application a dû être développée très rapidement pour permettre de montrer à de potentiels investisseurs que le concept est viable (on parle de Minimum Viable Product ou MVP).

Cette application a été développée avec le framework PHP Symfony.

ToDo & Co ayant réussi à lever des fonds pour permettre le développement de l'entreprise et de l'application, des améliorations ont été apportées telles que :

- l'implémentation de nouvelles fonctionnalités ;
- la correction de quelques anomalies ;
- l'implémentation de tests automatisés.

A ce stade, ToDo & Co a souhaité qu'un audit de qualité de code et de performance de l'application soit réalisé.

2. Qualité de Code

L'application a été initialement développée dans l'environnement suivant :

- Symfony 3.1.*
- PHP =>5.5.9
- Doctrine ORM 2.5
- Bootstrap 3.3.7

L'architecture du projet est du standard de Symfony.

Afin d'étudier la qualité du code, je me suis basée sur le **pourcentage de couverture de code**, sur des **outils de revue de code** et une **revue manuelle**.

2.1 Tests automatisés

Afin d'avoir un code robuste, éprouvé et maintenable il est nécessaire d'effectuer des tests automatisés.

L'un des indicateurs d'un code maintenable est son pourcentage de couverture. Plus il est élevé plus cela signifie que chaque ligne de code a été testée.

Il est donc important d'avoir un pourcentage approchant les 100%.

Mais cela ne signifie pas que chaque éventualité ait été testée, il faut donc veiller à effectuer des tests les plus poussés possibles.

Projet initial

Dans le projet initial de l'application aucun test n'avait été mis en place engendrant donc une **couverture de code à 0%**.

ToDo List

C:\wamp64\www\IP8\IP8_OC_ToDoList\src\AppBundle / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	<div></div>	0.00%	0 / 274	<div></div>	0.00%	0 / 54	<div></div>	0.00%	0 / 11
Command	<div></div>	0.00%	0 / 37	<div></div>	0.00%	0 / 4	<div></div>	0.00%	0 / 2
Controller	<div></div>	0.00%	0 / 96	<div></div>	0.00%	0 / 12	<div></div>	0.00%	0 / 4
Entity	<div></div>	0.00%	0 / 113	<div></div>	0.00%	0 / 36	<div></div>	0.00%	0 / 3
Form	<div></div>	0.00%	0 / 28	<div></div>	0.00%	0 / 2	<div></div>	0.00%	0 / 2
AppBundle.php	<div></div>	n/a	0 / 0	<div></div>	n/a	0 / 0	<div></div>	n/a	0 / 0

Legend

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

Generated by php-code-coverage 6.0.7 using PHP 7.1.9 with Xdebug 2.5.5 and PHPUnit 7.2.4 at Wed Jun 6 6:48:08 UTC 2018.

Actions

Afin de tester l'application, des tests unitaires, fonctionnels et d'intégration ont été développés avec l'outil **PHPUnit**.

Projet final

Le projet final obtient une **couverture de code à 98.99%**.

C:\wamp64\www\IP8\IP8_OC_ToDoList\src\AppBundle / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	<div></div>	98.99%	197 / 199	<div></div>	96.43%	54 / 56	<div></div>	91.67%	11 / 12
Command	<div></div>	100.00%	31 / 31	<div></div>	100.00%	4 / 4	<div></div>	100.00%	2 / 2
Controller	<div></div>	96.92%	63 / 65	<div></div>	83.33%	10 / 12	<div></div>	75.00%	3 / 4
DataFixtures	<div></div>	100.00%	43 / 43	<div></div>	100.00%	6 / 6	<div></div>	100.00%	1 / 1
Entity	<div></div>	100.00%	46 / 46	<div></div>	100.00%	30 / 30	<div></div>	100.00%	2 / 2
Form	<div></div>	100.00%	10 / 10	<div></div>	100.00%	2 / 2	<div></div>	100.00%	2 / 2
Model	<div></div>	100.00%	4 / 4	<div></div>	100.00%	2 / 2	<div></div>	100.00%	1 / 1
Repository	<div></div>	n/a	0 / 0	<div></div>	n/a	0 / 0	<div></div>	n/a	0 / 0
AppBundle.php	<div></div>	n/a	0 / 0	<div></div>	n/a	0 / 0	<div></div>	n/a	0 / 0

Legend

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

Generated by php-code-coverage 6.0.7 using PHP 7.1.9 with Xdebug 2.5.5 and PHPUnit 7.2.7 at Fri Jul 20 12:11:44 UTC 2018.

Les fichiers de couverture de code sont disponibles dans le répertoire **documentation/test-coverage**.

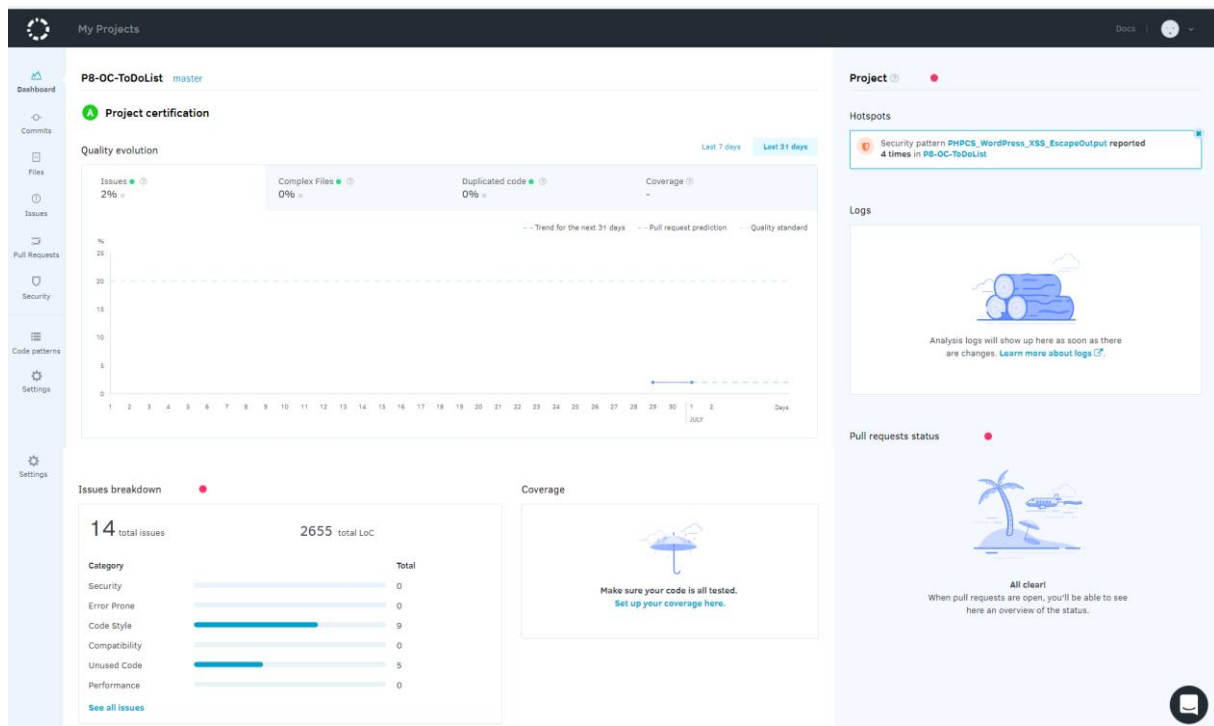
2.2 Outils de revue de code

Afin d'effectuer la revue du code, j'ai utilisé plusieurs outils de revue du code généraux comme Codacy, Scrutinizer, Code Climate et un outil de revue spécifique à Symfony, SensioLabsInsight.

2.2.1 Codacy

Codacy est un outil de revue de code qui vérifie le style de code, la sécurité, la duplication, la complexité et la couverture à chaque changement.

Projet initial



J'ai exclu de l'analyse du code les fichiers de base de Symfony et les fichiers de tests.

Il en ressort que le projet initial obtient :

la certification A, niveau le plus élevé
2% d'Issues, soit **14 issues non résolues**
0% de Complex files, niveau le plus élevé
0% du Duplicated Code, niveau le plus élevé
0% de couverture de code, niveau le plus bas.

Il faut donc apporter notre attention sur les 14 issues non résolues et sur la couverture de code pour se rapprocher des 100%.

Voici les 14 issues non résolues :

1) Prohibit unused arguments

Codacy nous recommande de supprimer les arguments non utilisés dans les méthodes.

Current Issues ▾ master ▾

Language	All ▾	Category	All ▾	Level	All ▾	Pattern	Prohibit unused arguments ▾	Author	All ▾	✕
src/AppBundle/Command/LoadDatasCommand.php										
Avoid unused parameters such as '\$input'.										
35 protected function execute(InputInterface \$input, OutputInterface \$output)										

ToDo List

src/AppBundle/Command/UpdateRoleUserCommand.php

Avoid unused parameters such as '\$input'.

```
26 protected function execute(InputInterface $input, OutputInterface $output)
```

src/AppBundle/Controller/SecurityController.php

Avoid unused parameters such as '\$request'.

```
14 public function loginAction(Request $request)
```

src/AppBundle/Form/TaskType.php

Avoid unused parameters such as '\$options'.

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

src/AppBundle/Form/UserType.php

Avoid unused parameters such as '\$options'.

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

2) Prohibit short variable names

Codacy recommande de ne pas utiliser des normes de variables inférieur à 3 caractères.

Current Issues ▾ master ▾

Language	All ▾	Category	All ▾	Level	All ▾	Pattern	Prohibit short variable names ▾	Author	All ▾	✕
----------	-------	----------	-------	-------	-------	---------	---------------------------------	--------	-------	---

src/AppBundle/Command/UpdateRoleUserCommand.php

Avoid variables with short names like \$em. Configured minimum length is 3.

```
28 $em = $this->getContainer()->get('doctrine.orm.entity_manager');
```

src/AppBundle/Controller/TaskController.php

Avoid variables with short names like \$em. Configured minimum length is 3.

```
36 $em = $this->getDoctrine()->getManager();
```

src/AppBundle/Controller/UserController.php

Avoid variables with short names like \$em. Configured minimum length is 3.

```
37 $em = $this->getDoctrine()->getManager();
```

src/AppBundle/Entity/Task.php

Avoid variables with short names like \$id. Configured minimum length is 3.

```
21 private $id;
```

src/AppBundle/Entity/User.php

Avoid variables with short names like \$id. Configured minimum length is 3.

```
25 private $id;
```

3) Excessive methods

Codacy recommande d'avoir un nombre de méthodes inférieur à 10.

Current Issues ▾ master ▾

Language	All ▾	Category	All ▾	Level	All ▾	Pattern	Excessive methods ▾	Author	All ▾	✕
----------	-------	----------	-------	-------	-------	---------	---------------------	--------	-------	---

tests/AppBundle/Controller/TaskControllerTest.php

The class TaskControllerTest has 26 non-getter- and setter-methods. Consider refactoring TaskControllerTest to keep number of methods under 10.

```
14 class TaskControllerTest extends AppBundleTestCase
```

tests/AppBundle/Controller/UserControllerTest.php

The class UserControllerTest has 13 non-getter- and setter-methods. Consider refactoring UserControllerTest to keep number of methods under 10.

```
13 class UserControllerTest extends AppBundleTestCase
```

4) Avoid else expressions

Codacy recommande de simplifier le code en supprimant les « else » non nécessaires.

Current Issues ▾ master ▾

Language	All ▾	Category	All ▾	Level	All ▾	Pattern	Avoid else expressions ▾	Author	All ▾	✕
----------	-------	----------	-------	-------	-------	---------	--------------------------	--------	-------	---

src/AppBundle/Controller/TaskController.php

The method deleteTaskAction uses an else expression. Else is never necessary and you can simplify the code to work without else.

```
97 else {
```

src/AppBundle/Entity/Task.php

The method getUser uses an else expression. Else is never necessary and you can simplify the code to work without else.

```
123 }else {
```

Actions

Après étude des anomalies remontées par **Codacy**, j'ai effectué les modifications suivantes :

- 1) **Prohibit unused arguments :**
Suppression de l'argument Request \$request dans la fonction Login du SecurityController.
Aucune autre modification apportée pour respecter la structure des autres méthodes.
- 2) **Prohibit short variable names :**
\$id n'a pas été modifié et \$em a été remplacé par \$entityManager.
- 3) **Excessive methods :**
Aucune modification apportée au code.
J'ai exclu de l'analyse les fichiers de tests.
- 4) **Avoid else expressions :**
« else » supprimés car non nécessaires.

ToDo List

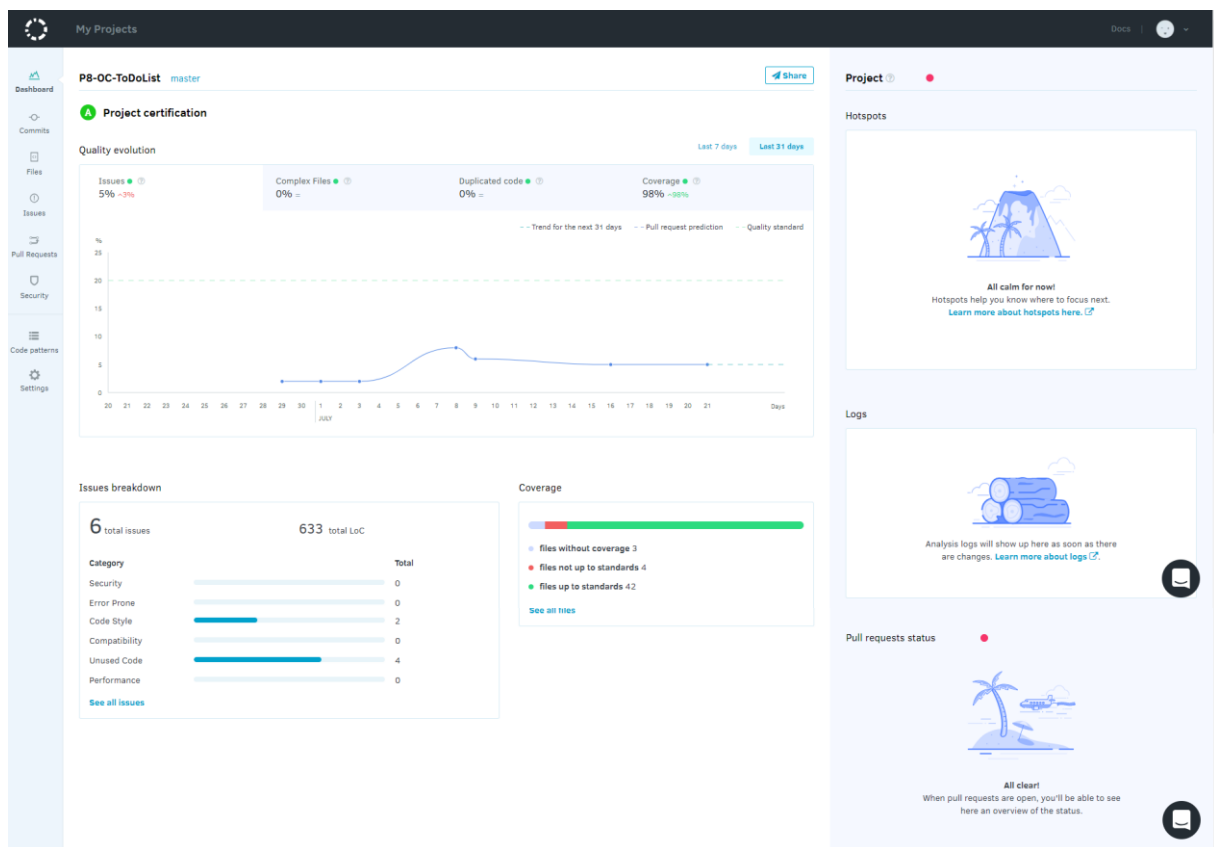
J'ai également supprimé du projet les fichiers liés à l'IDE PHPStorm (répertoire .idea) qui n'ont pas à être sauvegardés dans Github.

Projet final

Après avoir rectifié ces modifications et mis en place des tests automatisés, j'obtiens ces résultats :

la certification A
5% d'Issues, soit 6 issues non résolues
0% de Complex files
0% du Duplicated Code
98% de couverture de code

Le nombre d'issues non résolues a diminué pour n'en avoir que 6 (le pourcentage a augmenté proportionnellement au nombre de fichiers analysés car des fichiers ont été exclu et d'autres supprimés du projet comme le .idea) et la couverture de code est passé à **98%**.



ToDo List

Current Issues ▾ master ▾

Language **All** ▾

Category **Code Style** ▾

Level **All** ▾

Pattern **All** ▾

Author **All** ▾

✕

src/AppBundle/Entity/Task.php

Avoid variables with short names like \$id. Configured minimum length is 3.

30 private \$id;

< Previous Next >

Current Issues ▾ master ▾

Language **All** ▾

Category **Unused Code** ▾

Level **All** ▾

Pattern **All** ▾

Author **All** ▾

✕

src/AppBundle/Command/LoadDatasCommand.php

Avoid unused parameters such as '\$input'.

44 protected function execute(InputInterface \$input, OutputInterface \$output)

src/AppBundle/Command/UpdateRoleUserCommand.php

Avoid unused parameters such as '\$input'.

43 protected function execute(InputInterface \$input, OutputInterface \$output)

src/AppBundle/Form/Type/TaskType.php

Avoid unused parameters such as '\$options'.

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

src/AppBundle/Form/Type/UserType.php

Avoid unused parameters such as '\$options'.

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

< Previous Next >

2.2.2 Scrutinizer

Scrutinizer est également un outil de revue de code axé sur le style de code, la complexité, les bugs et la couverture de code. Il permet également d'effectuer l'intégration continue et de réaliser les tests automatisés.

Projet initial

caroleguardiola / P8-OC-ToDoList

Schedule InspectionUnsubscribe

View Results directly embedded on GitHub

You can view Scrutinizer analysis results and code intelligence data directly on GitHub with our browser extension.

Add to ChromeLearn more

Home

Issues

Code

Search

Files

Insights

Tools

3 days ago

✓

The first inspection on "master" completed

</>

Your code was rated 4.95 (pass).

Learn more about the code rating

🔔

There were 187 issues found.

3 days ago

+

The repository was created. What's next?

•

Run your first inspection (might take a few minutes longer)

•

Change the inspection configuration

•

Change the tracking/notification settings

•

Add collaborators to your repository

10

(very good)

Badges

Scrutinizer

10.00

coverage

98 %

build

passed

code intelligence

available

Latest Alerts

Good job, no alerts here.

caroleguardiola / P8-OC-ToDoList

Schedule InspectionUnsubscribe

Home

Issues

Code

Search

Files

Insights

Tools

Issues (4)

Labels

Deprecated Code

3

Unused Code

1

Severity

Minor

4

Introduced By

caroleguardiola

4

4 files with issues

Count	Path	Last Found
1	app/autoload.php Deprecated Code	30 minutes ago
1	src/AppBundle/Controller/SecurityController.php Unused Code	3 days ago
1	web/app_dev.php Deprecated Code	3 days ago
1	web/app.php Deprecated Code	3 days ago

ToDo List

Scrutinizer soulève les points suivants :

1) La fonction registerLoader est dépréciée

app/autoload.php (1 issue) ← previous file next file →

Labels

Deprecated Code 1

Severity

Minor 1

Introduced By

caroleguardiola 1

```
1 <?php
6 /** @var ClassLoader $loader */
7 $loader = require __DIR__.'/../vendor/autoload.php';
8
9 AnnotationRegistry::registerLoader([$loader, 'loadClass']);
10
11 return $loader;
12
```

Deprecated Code introduced 12 days ago by

The function `Doctrine\Common\Annotations\Registry::registerLoader()` has been deprecated: this method is deprecated and will be removed in doctrine/annotations 2.0 autoloading should be deferred to the globally registered autoloader by then. For now, use `@example AnnotationRegistry::registerLoader('class_exists')` (Ignorable by Annotation)

2) Supprimer le paramètre Request \$request inutilisé

src/AppBundle/Controller/SecurityController.php (1 issue) ← previous file next file →

Labels

Unused Code 1

Severity

Minor 1

Introduced By

caroleguardiola 1

```
1 <?php
11 /**
12  * @Route("/login", name="login")
13  */
14 public function loginAction(Request $request)
15 {
16     $authenticationUtils = $this->get('security.authentication_utils');
17
18     // ...
43
```

Unused Code introduced 3 days ago by

The parameter `$request` is not used and could be removed. (Ignorable by Annotation)

3) La fonction loadClassCache est dépréciée et la classe cache n'est plus nécessaire en PHP 7

web/app_dev.php (1 issue) ← previous file next file →

Labels

Deprecated Code 1

Severity

Minor 1

Introduced By

caroleguardiola 1

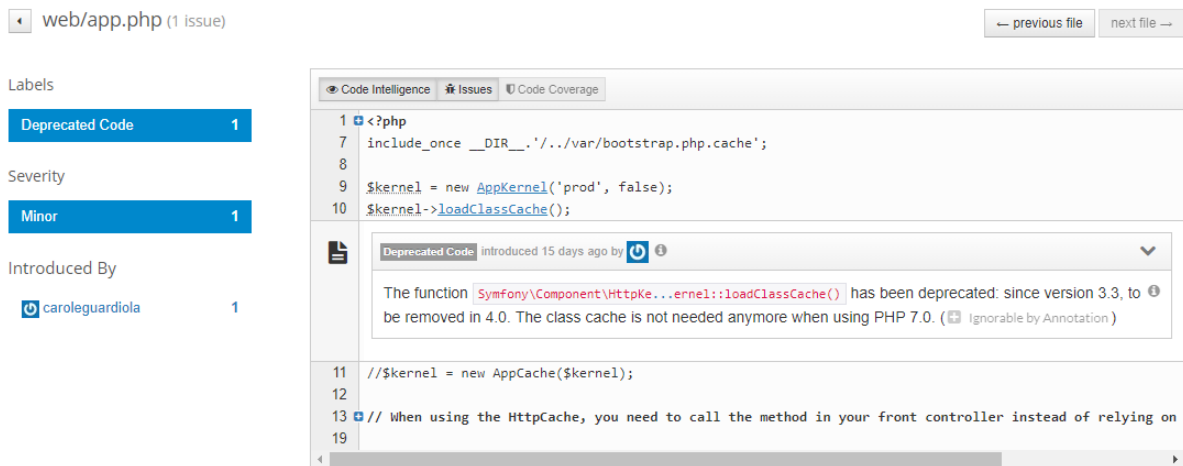
```
1 <?php
23 Debug::enable();
24
25 $kernel = new AppKernel('dev', true);
26 $kernel->loadClassCache();
27
28 $request = Request::createFromGlobals();
29 $response = $kernel->handle($request);
30 $response->send();
31 $kernel->terminate($request, $response);

```

Deprecated Code introduced 15 days ago by

The function `Symfony\Component\HttpKernel::loadClassCache()` has been deprecated: since version 3.3, to be removed in 4.0. The class cache is not needed anymore when using PHP 7.0. (Ignorable by Annotation)

ToDo List



J'ai exclu de l'analyse du code certains fichiers de base de Symfony, les fichiers de tests. Lors de la 1^{ère} analyse les tests avaient déjà été mise en place.

Il en ressort que le projet initial obtient les résultats suivants :

Scrutinizer (revue de code) : 10
Coverage (couverture de code) : 98%

Actions

- 1) Rien n'a été fait à ce stade de l'application.
- 2) J'ai supprimé le paramètre Request \$request non utilisé dans la fonction Login du SecurityController.
- 3) J'ai commenté les lignes `$kernel->loadClassCache()` ; dans les contrôleurs frontaux comme le suggère la note de Symfony (<https://symfony.com/blog/new-in-symfony-3-3-deprecated-the-classloader-component>).

```
// $kernel->loadClassCache();
```

Projet final

Après avoir rectifié ces points j'obtiens ces résultats :

Scrutinizer (revue de code) : 10
Coverage (couverture de code) : 98%

ToDo List

2.2.3 CodeClimate

CodeClimate est également un outil de revue de code axé sur le style de code, la maintenabilité et la couverture de code.

Projet initial

J'ai exclu de l'analyse du code certains fichiers de base de Symfony, les fichiers de tests et les fichiers JS.

Il en ressort que le projet initial obtient les résultats suivants :

ToDo List

Maintenabilité : A – 5hrs
Couverture de code : 0%

Code Climate suggère de ne pas dépasser 250 lignes de code :

The screenshot shows the GitHub repository page for `caroleguardiola/P8-OC-ToDoList`. The repository is starred and has a last commit on the `master` branch 9 minutes ago. The `Issues` tab is selected, showing 1 of 1 total issue. The issue is titled "File `config.php` has 384 lines of code (exceeds 250 allowed). Consider refactoring." and is categorized as "Minor" with a severity of "CAUTION". The issue is open and confirmed. The code snippet shows the beginning of a PHP file with a comment indicating the issue. The filters on the right show that the issue is categorized as "Complexity", is "Open" and "Confirmed", and is sourced from "Code Climate".

Showing 1 of 1 total issue

File `config.php` has 384 lines of code (exceeds 250 allowed). Consider refactoring. [OPEN](#)

```
1 <?php
2
3 /*
4  * ***** CAUTION *****
5  *
```

Found in `web/config.php` - About 5 hrs to fix

SEVERITY

- ☐ Minor

CATEGORY

- ☒ Complexity

STATUS

- ☒ Open
- ☒ Confirmed
- ☐ Invalid
- ☐ Wontfix

SOURCE

- ☒ Code Climate
- [Explore 3rd-party plugins](#)

LANGUAGE

- ☐ PHP

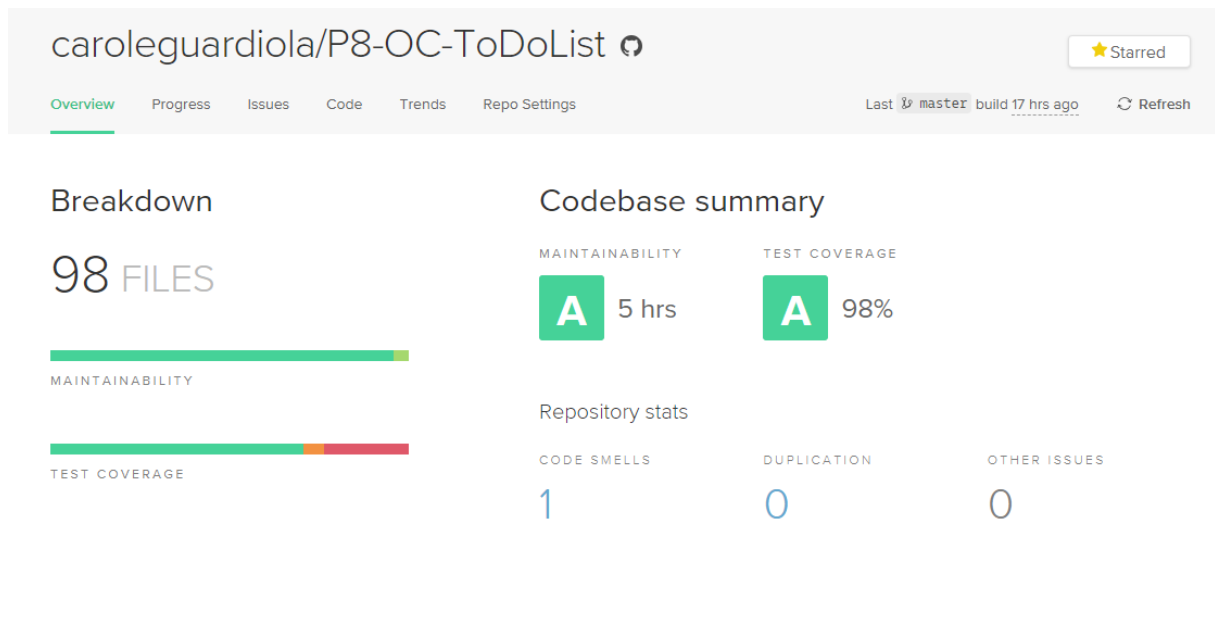
Actions

Aucune modification apportée car le fichier **config.php** est un fichier de base de Symfony.

Projet final

Il en ressort que le projet final obtient les résultats suivants :

Maintenabilité : A – 5hrs
Couverture de code : 98%



2.2.4 SensioLabsInsight

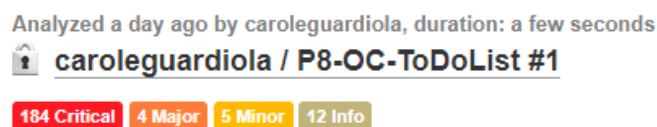
Cet outil a été développé par SensioLabs, créateur de Symfony et est spécialisé dans la revue de code Symfony.

C'est un outil payant mais une seule analyse gratuite peut être effectuée sur chaque projet. J'ai donc réalisé cette analyse.

Projet initial

Cette analyse fait ressortir un certain nombre de points à améliorer notamment dans les fichiers .xml concernant l'IDE PhpStorm. Ces fichiers doivent être ignorés car il ne concerne pas le code de l'application.

SensioLabsInsight répertorie les points suivants :



372856a « XML files should not contain syntax error »
799b61f « Text files should end with a newline character »
d011105 « Text files should end with a newline character »
« XML files should not contain syntax error »
a50867c « The EntityManager should not be flushed within a loop »
c13e29a « Commented code should not be committed »
« Source code should not contain TODO comments »
« The composer.json file should be valid »
« .htaccess should be avoided »
« Default favicon should be changed »
efa53da « Form types should be in Form/Type folders »
« Symfony applications should not contain a config.php file »
93d9c60 « XML files should not contain syntax error »
41f1ca2 « Text files should end with a newline character »
3490d4f « XML files should not contain syntax error »
30f58e1 « XML files should not contain syntax error »
f5b1fe2 « XML files should not contain syntax error »
1850a1f « XML files should not contain syntax error »
d53d32b « The composer.json file should not raise warnings »
6c03400 « XML files should not contain syntax error »

A priori les 184 Critical sont en majorité dus aux fichiers xml et nous ne voyons pas l'ensemble des points remontés.

Mon analyse se portera donc sur les points répertoriés ci-dessus.

Sans tenir compte des fichiers dans le répertoire .idea de PHPStorm, l'analyse du code fait ressortir plusieurs points à améliorer :

- 1) « **Commented code should not be committed** » :
Du code commenté ne doit pas être commité.
- 2) « **Source code should not contain TODO comments** » :
Le code ne doit pas contenir de commentaires TODO.
- 3) « **The composer.json file should be valid** » :
Le fichier composer.json doit comporter une section « description ».
- 4) « **.htaccess should be avoided** » :
SensioLabsInsight recommande d'éviter d'utiliser les fichiers .htaccess.
- 5) « **Default favicon should be changed** » :
Le favicon par défaut de Symfony doit être modifié.
- 6) « **Form types should be in Form/Type folders** » :

ToDo List

Les formulaires doivent être dans un répertoire Form/Type et non dans le répertoire Form directement.

7) « Symfony applications should not contain a config.php file » :

En mode production, Symfony recommande de retirer ce fichier.

Ce fichier est destiné à être utilisé uniquement lors de la configuration initiale d'un projet et doit être supprimé avant la mise en production du projet.

8) « The composer.json file should not raise warnings » :

Définir l'autoload.psr-4 avec un namespace vide n'est pas une bonne idée pour la performance.

Actions

1) « Commented code should not be committed » :

Certains commentaires inutiles ont été supprimés.

2) « Source code should not contain TODO comments » :

Ce message concerne le fichier bootstrap.js, il n'a donc pas été modifié.

3) « The composer.json file should be valid » :

Fichier composer.json modifié avec une section « description ».

4) « .htaccess should be avoided » :

Rien n'a été fait à ce stade. Le fichier ne contient rien de gênant pour notre application puisqu'il renvoie toutes les requêtes à app.php (notre point d'entrée) et Symfony se charge du routing.

5) « Default favicon should be changed » :

Favicon modifié.

6) « Form types should be in Form/Type folders » :

Création du répertoire Type et déplacement des fichiers dans ce répertoire.

7) « Symfony applications should not contain a config.php file » :

Rien n'a été fait à ce stade, il faudra penser à le supprimer en mode production.

8) « The composer.json file should not raise warnings » :

Fichier composer.json modifié.

Projet final

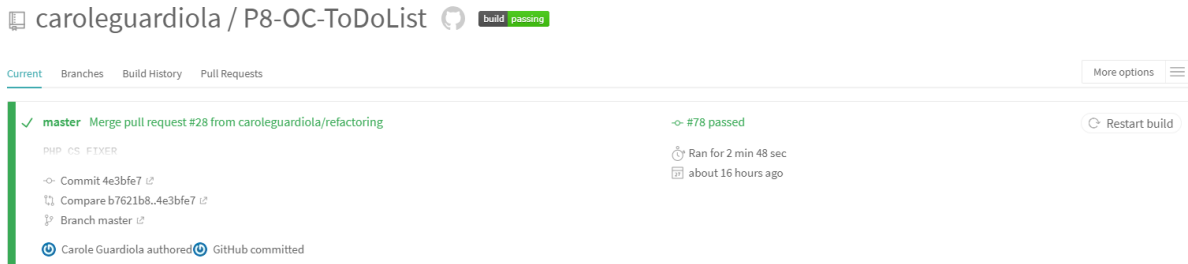
N'étant pas abonnée à SensioLabsInsight, une 2^{ème} analyse n'était pas possible.

Je peux quand même dire que les modifications apportées ont amélioré la maintenabilité de l'application.

2.2.5 Travis CI

L'outil **Travis CI** a également été mis en place pour une intégration continue. Ainsi à chaque Pull Request sur Github, les tests automatisés sont lancés automatiquement et la couverture de code est envoyé directement à Codacy et Code Climate.

Voici le dernier rapport de Travis CI indiquant que tout s'est bien passé :



2.3 Revue manuelle

En plus des outils utilisés, j'ai effectué une revue manuelle.

Projet initial

Il en ressort quelques **anomalies fonctionnelles** :

- 1) Le lien sur le titre du site « **To Do List App** » ne renvoie pas sur la page accueil

To Do List app

- 2) Sur la page d'accueil, le lien sur le bouton « **Consulter la liste des tâches à faire** » renvoie vers la liste de toutes les tâches faites et terminées.

Consulter la liste des tâches à faire

- 3) Sur la page d'accueil, le lien sur le bouton « **Consulter la liste des tâches terminées** » ne renvoie vers rien.

Consulter la liste des tâches terminées

- 4) Il n'y a pas d'accès à la gestion des utilisateurs pour les administrateurs.
- 5) Dans le fichier de vue base.html.twig, il est mentionné un lien vers le fichier jquery.js. Or ce fichier n'existe pas.

Voici également quelques points à améliorer pour une **meilleure navigation pour l'utilisateur** :

- 6) Créer sur toutes les pages hors page d'accueil et login un **bouton pour un retour à l'accueil**.
- 7) Avoir la possibilité de **filtrer les tâches** sur la page de la liste des tâches.
- 8) **Améliorer le thème** pour qu'il soit plus ergonomique et esthétique.
- 9) **Actualiser le logo d'OpenClassrooms**.
- 10) **Personnaliser les pages d'erreurs**.

Voici également quelques points à améliorer pour une **meilleure gestion du site** :

- 11) Il serait préférable de **désactiver les tâches** au lieu de les supprimer en base de données. Ainsi l'utilisateur les verra comme supprimées, mais l'administrateur pourra toujours avoir accès à toutes les tâches et pourra éventuellement si besoin les réactiver.
- 12) Il peut être également intéressant de pouvoir avoir un **historique des actions réalisées sur les tâches** comme par exemple avoir le nom de la personne qui a modifié une tâche ou l'a supprimée, à quelle date etc.

Actions

Anomalies fonctionnelles :

- 1) Activation du lien sur le titre du site « **To Do List App** » vers la page d'accueil.
- 2) et 3) Suppression des 2 boutons et création d'un seul bouton pour consulter la liste de toutes les tâches sur la page d'accueil.



Consulter la liste des tâches

- 3) Création d'un bouton pour accéder à la liste des utilisateurs sur la page d'accueil.



Consulter la liste des utilisateurs

- 5) Activation du lien à jQuery en CDN.


Meilleure navigation pour l'utilisateur :

- 6) Création d'un bouton de retour à l'accueil.



Retour à la page d'accueil

- 7) Création d'un filtre avec Isotope (plugin jQuery) afin de pouvoir sélectionner :
 - toutes les tâches
 - les tâches terminées
 - les tâches à faire



Toutes les tâches Les tâches terminées Les tâches à faire

- 8) Amélioration du design du site le rendant plus actuel et plus ergonomique pour une meilleure expérience utilisateur. Ajout d'icônes de Font Awesome. Changement de police (lien vers Google Fonts). Utilisation des codes couleurs OpenClassrooms.

- 9) Remplacement du logo par le nouveau.



- 10) Personnalisation des pages d'erreurs.

Meilleure gestion du site :

- 11) et 12) Rien n'a été fait à ce stade.

Projet final

Les anomalies fonctionnelles ont été corrigées et des améliorations ont été apportées permettant d'avoir une application fonctionnelle et ergonomique.

Quelques points plus techniques restent en suspens en attendant d'avoir une validation (11) et 12)).

3. Performance de l'application

Mais d'abord, pourquoi la performance est-elle si importante ?

Les utilisateurs n'attendent pas.

« 53% des utilisateurs abandonnent un site qui met plus de 3 secondes à se charger sur mobile. »

« Le temps moyen de chargement d'un site web en 3G est de 19 secondes. »

Source : seosta.com (septembre 2016)

Les utilisateurs attendent de moins en moins.

« La durée moyenne d'attention passe de 12 secondes (en 2000) à 8.25 secondes (en 2015). »

Source : statisticbrain.com

On peut considérer que l'on perd les utilisateurs au bout de 10 secondes.

L'analyse de la performance est essentiellement axée sur 2 points :

- 1) Le temps d'affichage des pages web
- 2) Le temps de chargement des éléments des pages web (images, CSS, JS)

3.1 Blackfire

Pour mon analyse j'ai utilisé l'outil Blackfire, développé par SensioLabs qui nous permet une analyse des performances principalement des fonctions PHP.

Blackfire permet une analyse très poussée, mais pour cet audit je me suis basée sur les métriques suivantes me semblant dans un premier temps les plus parlantes :

ToDo List

1) Le temps en ms :

Il s'agit de la mesure du temps réel qu'il a fallu pour PHP pour exécuter son code : la différence entre le moment où PHP est entré dans la fonction et l'heure à laquelle PHP quitte la fonction.

2) La mémoire en MB :

Il s'agit de la quantité de mémoire consommée par PHP.

3) Les requêtes à la Base de Données :

a. En temps

b. En nombre de requêtes

Projet initial


Voici les données relevées par Blackfire sur chaque requête de l'application :

Routes	URL	Méthode	Temps (ms)	Mémoire (MB)	SQL Queries (ms)	SQL Queries (rq)
homepage	/	GET	152	11,2	0,264	1
login	/login	GET	103	7,25	0	0
login_check	/login_check	POST	574	9,46	0,751	1
logout	/logout	GET	114	9,42	0,255	1
task_list	/tasks	GET	152	11,5	2,38	8
task_create	/tasks/create	GET	241	13,8	0,38	1
		POST	616	13,1	0,834	2
task_edit	/tasks/{id}/edit	GET	223	13,9	1,29	2
		POST	1090	12,5	1,82	3
task_toggle	/tasks/{id}/toggle	GET	178	10,2	1,21	3
task_delete	/tasks/{id}/delete	GET	133	10,1	2,58	3
user_list	/users	GET	164	11,5	0,536	2
user_create	/users/create	GET	259	14,5	0,266	1
		POST	616	13,1	1,57	3
user_edit	/users/{id}/edit	GET	220	14,6	0,537	2
		POST	874	13,2	0,00242	4

Attention : ces mesures ont été réalisées en local et en environnement production, ils sont donc ici à titre d'indication. Il faudrait les réaliser en « vrai » environnement production et les comparer avec des mesures prises après améliorations apportées à l'application.

Blackfire suggère les recommandations suivantes :

1) Les annotations Doctrine doivent être en cache en mode 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 d'appliquer cette configuration 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'.

Actions

A ce stade rien n'a été mis en place. Il faudra appliquer en mode production les recommandations de Blackfire.

3.2 Autres axes d'optimisation de la performance

3.2.1 Cache HTTP

Une analyse plus poussée pourrait être réalisée avec une mise en place du cache http avec le système de Symfony.

Cependant dans cette application nous utilisons la session et de ce fait le cache est différent pour chaque utilisateur et non partageable pour toutes les requêtes.

3.2.2 Mises à jour

Les mises à jour vont très souvent dans le sens de la performance.

Il faut penser à mettre à jour notamment les versions PHP et serveurs.

Il est également important d'utiliser une version de Symfony maintenue.

Actions

La version de Symfony 3.1 utilisée n'est plus maintenue, une mise à jour de Symfony a été effectuée pour passer à une version 3.4 maintenue par Symfony. En l'occurrence il s'agit de la plus récente version de Symfony 3.4, la 3.4.12.

Un passage à la version 4 de Symfony aurait apporté des changements majeurs non nécessaires pour cette application.

3.2.3 Optimisation des requêtes SQL

Ce qui prend le plus de temps c'est de se connecter à la base de données.

Il faut donc éviter de faire trop de requêtes SQL et si possible il est préférable de faire quelques grosses requêtes.

Au regard des données relevées par Blackfire (**SQL Queries**) il peut également être intéressant de mettre du cache au niveau de la base de données.

Actions

Aucune action menée sur ce point.

3.2.4 Les images

Les images peuvent être un élément de lourdeur important sur un site internet ce qui augmente significativement le temps de téléchargement des pages web.

Il faut donc les **compresser** ou les **optimiser** avec des outils tel que **TinyPNG**.

Une solution également est de compresser avec l'outil **GZIP** au niveau du serveur.

Actions

Toutes les images de l'application ont été compressées avec TinyPNG.

Nous obtenons une perte de poids totale de 13%.

3.2.5 CSS et JavaScript

Le chargement des fichiers CSS et JS prend beaucoup de temps dans le rendu d'une page HTML sur le navigateur des visiteurs.

Une des solutions seraient d'utiliser en mode production :

- 1) **La minification des fichiers CSS et JS** : versions compressées qui permet de charger plus rapidement les fichiers.
- 2) **Le CDN (« Content delivery network »)** : réseau de serveurs qui met à disposition des librairies. Il devient ainsi inutile de prévoir ces librairies sur son propre serveur, il suffit de « pointer »

ToDo List

vers eux. Cela permet entre autres une accélération du chargement, l'actualisation automatique des librairies.

Cependant cette solution ne permet pas de personnaliser les librairies et permet de faire transiter des informations sur les visiteurs du site.

L'exécution de JavaScript prend également beaucoup de temps.

Le **Bundle Assetic** permet de regrouper tous les fichiers CSS ainsi que tous les fichiers JS, afin de réduire le nombre de requêtes HTTP que doivent faire les visiteurs pour afficher une page. Il permet également de minifier les fichiers, afin de diminuer leur taille et donc accélérer leur chargement pour les visiteurs.

ATTENTION : *ASSETIC n'est plus recommandé à partir de la version 4.0 de Symfony, il faut utiliser à la place Webpack Encore.*

D'autres outils existent pour l'optimisation du CSS et du JS comme **PURGECSS**.

Actions

J'ai opté pour utiliser le CDN pour Bootstrap en CSS et JS, pour jQuery et Isotope.

Si la transit de données des visiteurs devenait un problème, il faudrait opter pour une autre solution.

A ce stade ASSETIC n'a pas été mis en place, si l'application venait à se développer il peut être intéressant de l'utiliser.

3.2.6 Lazy Loading

Cela consiste à spécifier quels composants d'un programme doivent être chargés lors du démarrage de celui-ci.

Le **lazy loading** est une technique qui retarde le chargement des ressources non critiques au moment du chargement de la page. Au lieu de cela, ces ressources non critiques sont chargées au moment du besoin. En ce qui concerne les images, "non critique" est souvent synonyme de "hors écran".

Actions

Méthode non appliquée à ce stade de l'application.

4. Conclusion

En conclusion, après les modifications apportées à l'application et malgré quelques points à approfondir, nous obtenons une bonne qualité de code et une bonne couverture de code comme le montrent les analyses du projet final de Codacy, Scrutinizer et CodeClimate.

En terme de performance, en appliquant les recommandations de Blackfire l'application devrait avoir une bonne performance.

A noter que quelques points restent en suspens :

ToDo List

- **Qualité de code :**
 - Meilleure gestion du site :
 - Désactiver les tâches au lieu de les supprimer
 - Avoir un historique des actions réalisées sur les tâches
 - En mode production, supprimer le fichier config.php.
- **Performance :**
 - En mode production, appliquer les recommandations de Blackfire.
 - Analyse plus poussée sur la mise en cache (cache http, cache au niveau de la base de données) si nécessaire.
 - Mise en place d'ASSETIC ou autres outils d'optimisation du CSS/JS si nécessaire.
 - Mise en place du lazy loading si nécessaire.