



---

# Analyse comparative de qualité et de performance

---

V.1 vs V.2

Date de la dernière mise à jour : 17/06/2024

# sommaire

## sommaire

|   |           |
|---|-----------|
|   | <b>2</b>  |
| <b>1. Introduction</b>  | <b>3</b>  |
| 1.1 Contexte de l'analyse comparative                                     | 3         |
| 1.1.1 Objectifs   | 3         |
| 1.1.2 Description des deux versions comparées                             | 3         |
| 1.2 Méthodologie  | 4         |
| 1.1.1 Description des outils utilisés                                     | 4         |
| 1.1.2 Critères d'évaluation et seuils de qualité                          | 5         |
| 1.1.3 Période d'analyse, versions du projet analysé et taux de couverture | 5         |
| <b>2. Résultats de l'analyse comparative</b>                              | <b>7</b>  |
| 2.1 Analyse de qualité avec Codacy  | 7         |
| 2.2.1 - Version initiale  | 7         |
| 2.2.2 - V.2   | 8         |
| 2.2.3 - Comparaison   | 9         |
| 2.2 Analyse de qualité avec Symfony Insight                               | 11        |
| 2.2.1 - Version initiale  | 11        |
| 2.2.2 - V.2   | 12        |
| 2.2.3 - Comparaison   | 13        |
| 2.2 Analyse de performance avec Blackfire                                 | 14        |
| 2.2.1 - Métriques   | 14        |
| 2.2.2 - Etat des lieux  | 15        |
| 2.2.3 - Optimisation  | 17        |
| 2.2.4 - Synthèses des différences   | 19        |
| <b>3. Synthèse des résultats</b>  | <b>20</b> |
| 3.1 Comparaison globale des deux versions                                 | 20        |
| 3.1.1 - Qualité du code   | 20        |
| 3.1.2 - Performance   | 21        |
| <b>4. Conclusion</b>  | <b>22</b> |
| 3.1 Plan d'action pour les versions futures                               | 22        |
| 3.1.1 - Recommandations globale pour la maintenance                       | 22        |
| 3.1.2 - Stratégie d'amélioration continue                                 | 23        |

# 1. Introduction

## 1.1 Contexte de l'analyse comparative

### 1.1.1 Objectifs

Dans le contexte d'un changement de version d'une application datant de 8 ans. Les fondateurs demandent un audit de qualité et de performance de leur application. Pour cela nous allons dans un premier temps comparer les différentes analyses des deux versions grâce à plusieurs outils qui seront présentés dans la section suivante.

L'objectif principal de cette étude étant de déterminer la dette technique initiale et son amélioration.

### 1.1.2 Description des deux versions comparées

|                        | Version initiale  | V.2  |
|------------------------|---|--|
| Date de création       | novembre 2018   | juin 2024  |
| Technologies utilisées | <ul style="list-style-type: none"><li>- symfony/symfony v3.1.6</li><li>- PHP v5.5.9</li><li>- doctrine/orm v2.5.5</li></ul> | <ul style="list-style-type: none"><li>- symfony/framework-bundle v7.0.6</li><li>- PHP v8.2</li><li>- doctrine/orm v3.1.2</li></ul> |

## 1.2 Méthodologie

### 1.1.1 Description des outils utilisés

Nous utiliserons 5 outils pour l'analyse et qui serviront dans les futurs développements:

#### **Symfony Insight**

Symfony Insight est un service d'analyse de la qualité du code conçu spécifiquement pour les projets Symfony. Il offre une évaluation détaillée du code en vérifiant les bonnes pratiques de développement, la sécurité, les performances, et la maintenabilité. Symfony Insight fournit des rapports clairs et des recommandations pour améliorer la qualité du code Symfony.

#### **Codacy**

Codacy est une plateforme de revue de code automatisée qui permet de détecter les problèmes de qualité, de sécurité et de style de code. Elle analyse le code source en continu pour identifier les erreurs, les vulnérabilités et les violations des normes de codage.

#### **Php CS Fixer**

PHP CS Fixer est un outil puissant pour maintenir la qualité et la cohérence du code PHP en appliquant automatiquement des normes de codage définies. Il permet une correction automatique des violations des normes de style de code dans les fichiers PHP.

#### **Blackfire**

Blackfire est un outil de profilage et d'analyse des performances pour les applications PHP. Il permet de mesurer et d'optimiser les performances du code en identifiant les goulots d'étranglement et les points d'amélioration potentiels. Blackfire fournit des profils détaillés des performances de l'application, incluant l'utilisation de la mémoire, le temps d'exécution, et les requêtes SQL, afin d'aider à améliorer l'efficacité du code.

#### **GitHub Action**

GitHub Actions est une plateforme d'intégration et de livraison continues (CI/CD) qui permet d'automatiser la pipeline de création, de test et de déploiement. Elle permet de tester chaque demande d'extraction vers le référentiel, ou déployer des demandes d'extraction fusionnées en production.

## 1.1.2 Critères d'évaluation et seuils de qualité

### Critères d'évaluation

- ❖ les erreurs rencontrées
- ❖ la sécurité
- ❖ le style de code
- ❖ la performance
- ❖ la maintenabilité
- ❖ la documentation

Codacy effectue en plus une évaluation sur :

- ❖ la complexité
- ❖ la duplication de code
- ❖ le code smells (mauvaise pratique)

### Les seuils

Symfony Insight : médaille d'argent.

Codacy : note de A

PHPUnit : seuil de couverture 70% minimum, actuellement

## 1.1.3 Période d'analyse, versions du projet analysé et taux de couverture

### Période d'analyse

L'analyse a été effectuée courant juin 2024 sur une période de deux jours dans un intervalle d'une semaine. Les différents tests ont montré de légère différence entre chaque tests, nous prendrons donc une analyse reflétant la moyenne de celles effectuées.

### Versions du projet analysées

L'analyse porte sur la v.1 de 2018 et la v.2 de 2024.

La v.2 comporte une montée en version de Symfony et Php notamment, ainsi que l'ajout de fonctionnalités :

- ❖ l'autorisation avec la notion de rôle,
- ❖ l'implémentation de tests unitaires et fonctionnels automatisés,



















la correction d'anomalie :

- ❖ tâche rattachée à son auteur,
- ❖ choix d'un rôle pour l'utilisateur.

## Taux de couverture des tests avec PHPUnit

Le taux de couverture du code par les tests est actuellement de 94,63% avec 36 tests et 112 assertions. Les dataFixtures et les entités ne sont pas prises en compte dans la couverture.

C:\Users\laurel\Desktop\OCR\projet todo\projet8-TodoList\src / [\(Dashboard\)](#)

|                | Code Coverage   |         |                       |   |         |                    |   |               |
|----------------|---|---------|-----------------------|---|---------|--------------------|---|---------------|
|                | Lines   |         | Functions and Methods |   |         | Classes and Traits |   |               |
| Total          |   | 94.63%  | 282 / 298             |   | 79.55%  | 35 / 44            |   | 41.67% 5 / 12 |
| ■ Controller   |  | 94.05%  | 79 / 84               |  | 78.57%  | 11 / 14            |  | 25.00% 1 / 4  |
| ■ DataFixtures |   | n/a     | 0 / 0                 |   | n/a     | 0 / 0              |   | n/a 0 / 0     |
| ■ Entity       |   | n/a     | 0 / 0                 |   | n/a     | 0 / 0              |   | n/a 0 / 0     |
| ■ EnumTodo     |   | n/a     | 0 / 0                 |   | n/a     | 0 / 0              |   | n/a 0 / 0     |
| ■ Form         |  | 100.00% | 91 / 91               |  | 100.00% | 4 / 4              |  | 100.00% 2 / 2 |
| ■ Manager      |  | 96.88%  | 31 / 32               |  | 90.00%  | 9 / 10             |  | 50.00% 1 / 2  |
| ■ Repository   |  | 83.93%  | 47 / 56               |  | 63.64%  | 7 / 11             |  | 0.00% 0 / 2   |
| ■ Security     |  | 97.14%  | 34 / 35               |  | 80.00%  | 4 / 5              |  | 50.00% 1 / 2  |
| Kernel.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 9.2.31 using PHP 8.2.8 and PHPUnit 9.6.19 at Tue Jun 11 23:08:10 UTC 2024.

```

PHPUnit 9.6.19 by Sebastian Bergmann and contributors.

Testing
.....                                     36 / 36 (100%)

Time: 00:08.726, Memory: 76.00 MB

OK (36 tests, 112 assertions)

```

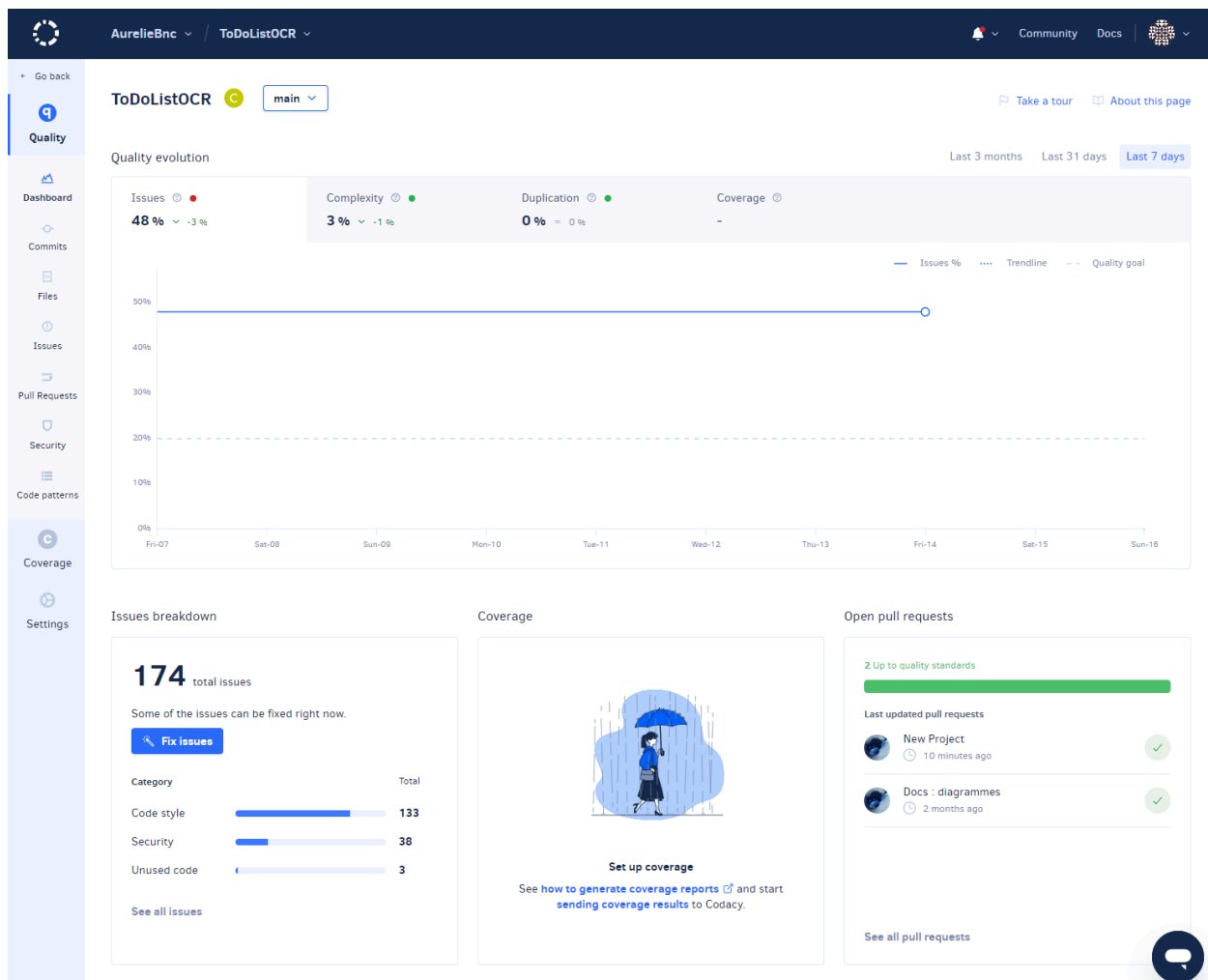
## 2. Résultats de l'analyse comparative

### 2.1 Analyse de qualité avec Codacy

#### 2.2.1 - Version initiale

##### ■ Note globale et répartition des issues

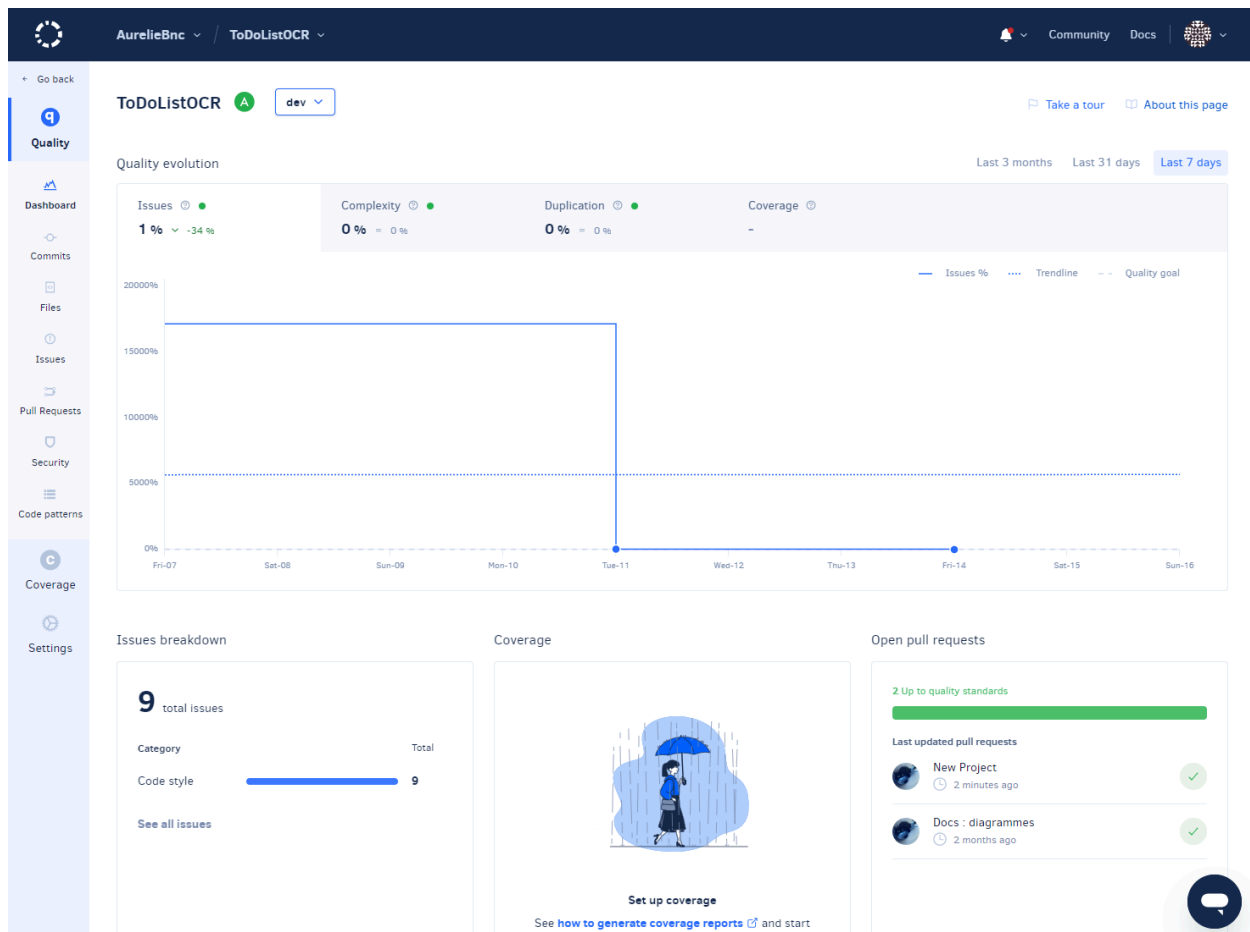
|                 |                   |   |
|-----------------|-------------------|---|
| Note : <b>C</b> | <b>174</b> issues | <ul style="list-style-type: none"><li>- <b>133</b> code style</li><li>- <b>38</b> security</li><li>- <b>3</b> unused code</li></ul> |
|-----------------|-------------------|---|



## 2.2.2 - V.2

## ■ Note globale et répartition des issues

|                 |                 |                       |
|-----------------|-----------------|-----------------------|
| Note : <b>A</b> | <b>9</b> Issues | - <b>9</b> code style |
|-----------------|-----------------|-----------------------|







## ■ Détails des principales issues et leur impact

**Code Style :** Les problèmes de style de code ne bloquent généralement pas l'exécution du code, mais ils ont plusieurs impacts négatifs:

- ❖ la lisibilité, rend le code plus difficile à lire et à comprendre.
- ❖ la maintenance, les incohérences peuvent augmenter la complexité lors des modifications et des corrections de bugs.
- ❖ le non suivi des conventions d'équipe qui apporte de la frictions et une baisse générale de la qualité du projet.

**Security :** Les problèmes de sécurité sont critiques et peuvent avoir des impacts graves:

- ❖ vulnérabilité, les failles de sécurité peuvent être exploitées par des attaques pour accéder aux données sensibles, compromettre l'intégrité du système ou causer des interruptions de service
- ❖ réputation, une faille de sécurité peut nuire à la réputation du projet.
- ❖ coût, une correction de failles de sécurité coûte généralement plus cher que de les prévenir dès le départ.

**Unused Code :** Le code inutilisé peut sembler inoffensif, mais il peut apporter plusieurs inconvénients :

- ❖ confusion, peut induire en erreur la lecture du code
- ❖ maintenance, garder du code inutilisé alourdit le projet et rend la maintenance plus difficile inutilement.
- ❖ performance, même inutilisé le code peut alourdir le projet, allongeant les temps d'analyse, de compilation et augmentant les ressources.

### 2.2.3 - Comparaison

#### ■ Synthèse des différences

On constate une nette amélioration du nombre d'issue entre les deux versions. Il n'y a plus de code non utilisé, ni de failles de sécurité. Le nombre d'issues sur le style de code est anecdotique et aucune régression n'est constatée.

## 2.2 Analyse de qualité avec Symfony Insight

### 2.2.1 - Version initiale

#### ■ Note globale et répartition des issues

|                      |                         |  |
|----------------------|-------------------------|--|
| Note : <b>27/100</b> | - <b>16</b> suggestions | - <b>7</b> critiques<br>- <b>2</b> majeures<br>- <b>6</b> mineures<br>- <b>1</b> infos |
|----------------------|-------------------------|--|

SymfonyHub

Sf "Symfony: The Fast Track", a new book to learn Symfony 6

Aurélien BENINCA

shil /  
ToDoListOCR #1

Analyzed a month ago by shil, duration: a minute

Edit configuration

Analyze

27/100

5.3 days  
to get the Platinum  
Medal

Search

**Severity**  
7 Critical  
2 Major  
6 Minor  
1 Info

**Risk**  
1 Data leak  
4 Productivity  
1 Reliability  
5 Reputation  
3 Security  
2 Uninsured

**Developer**  
10 Soro0h  
6 Collective

**Stats**  
Lines of code: 2,523  
Nb of suggestions: 16

**Last commit**  
Added login CRUD on  
tasks and users by  
Soro0h 8 years ago.  
main

Sf  
SymfonyInsight  
No medal

Changes: +16 suggestions 7 critical 2 major 6 minor 1 info

Suggestions (16) Fixed Ignored

Stats

► The dependencies of your project could not be installed

Read doc Ignore all Uninsured Critical

► Your application failed to start

Read doc Ignore all Uninsured Critical

► Your project must not rely on dependencies with known security issues 3

Read doc Ignore all Security Critical

► Your project must not expose sensitive infrastructure configuration

Read doc Ignore all Data leak Critical

► Your project must use a custom favicon instead of the default one

Read doc Ignore all Reputation Critical

► Your project should use Doctrine migrations

Read doc Ignore all Reliability Major

► Your project should not contain "FIXME" comments

Read doc Ignore all Productivity Major

► Your project should not contain commented code 2

Read doc Ignore all Productivity Minor

► Your project should not use an .htaccess file 3

Read doc Ignore all Reputation Minor

► Web applications should contain a site.webmanifest file

Read doc Ignore all Reputation Minor

## 2.2.2 - V.2

### ■ Note globale et répartition des issues

|                               |                        |   |
|-------------------------------|------------------------|---|
| Note : <b>75/100 (argent)</b> | - <b>9</b> suggestions | - <b>3</b> mineures<br>- <b>6</b> infos |
|-------------------------------|------------------------|---|

The screenshot displays the SymfonyInsight interface for a project named 'ToDoListOCR #113'. The top navigation bar includes 'SymfonyHub', the 'sf' logo, a description of the tool, and a user profile 'Aurélien BENINCA'. The main header shows the project name, a star rating of 75/100, and a '4 hours' badge. The left sidebar contains a search bar, a severity filter (3 Minor, 6 Info), a risk filter (7 Productivity, 2 Reputation), and a developer filter (6 Collective, 3 Shil). The main content area shows a list of suggestions with details on how to read the documentation, ignore all, and the associated risk level (Minor or Info). The suggestions are: 'Your project should not contain commented code' (Productivity, Minor), 'Your project should not use an .htaccess file' (Reputation, Minor), 'Web applications should contain a site.webmanifest file' (Reputation, Minor), and 'Text files should end with a valid new line character.' (Productivity, Info). The bottom of the page prompts the user to share the report by adding collaborators.

SymfonyInsight helps you protect your projects against security issues and technical debt.

Aurélien BENINCA

SymfonyInsight

Dashboard Builds Docs What we cover Pricing Account

shil / **ToDoListOCR #113** Analyzed 8 minutes ago, duration: a minute Edit configuration Analyze

Changes: no new suggestions!

Suggestions (9) Fixed Ignored Stats

► Your project should not contain commented code Read doc Ignore all Productivity Minor

► Your project should not use an .htaccess file Read doc Ignore all Reputation Minor

► Web applications should contain a site.webmanifest file Read doc Ignore all Reputation Minor

► Text files should end with a valid new line character. 8 Read doc Ignore all Productivity Info

Want to share this report?  
[Add Collaborators](#) in the Project Edit page

4 hours to get the Platinum Medal

Search

Severity  
3 Minor  
6 Info

Risk  
7 Productivity  
2 Reputation

Developer  
6 Collective  
3 Shil

Stats  
Lines of code: 11,062  
Nb of suggestions: 9

Last commit  
`codacy: fix style` by Shil 9 minutes ago.

SymfonyInsight No medal



## ■ Détails des principales issues et leur impact

**Critical :** Les issues critiques sont les plus graves et doivent être traitées rapidement. Elles impactent :

- ❖ la sécurité, impliquant différents problèmes de vulnérabilités.
- ❖ la stabilité, elles peuvent provoquer des comportements imprévisibles, compromettent l'intégrité du projet, voire provoquer un arrêt du programme.

**Major :** Les issues majeures sont graves mais moins urgentes. Elles impactent :

- ❖ la performance, dégradation de la performance et la qualité de l'expérience utilisateur.
- ❖ la maintenance, les problèmes de conception ou de structure de code peuvent rendre la maintenance difficile et coûteuse.
- ❖ fonctionnalité, elles peuvent affecter des fonctionnalités importantes de l'application.

**Minor :** Les issues mineurs sont moins graves, mais l'accumulation peut causer des problèmes à long terme :

- ❖ lisibilité, peut rendre le code plus difficile à lire et à comprendre.
- ❖ cohérence, le manque de cohérence peut entraîner des erreurs et des bugs dans le futur.
- ❖ expériences utilisateurs, elles peuvent affecter des aspects non critiques de l'expérience utilisateur.

**Info :** Ce ne sont pas des problèmes, mais elles fournissent des informations utiles pour améliorer le code selon les normes en cours.

### 2.2.3 - Comparaison

## ■ Synthèse des différences

Tout comme pour Codacy, nous constatons une nette amélioration de la qualité du code et du nombre d'issues, avec notamment une suppression totale des critiques et majeures.

## 2.2 Analyse de performance avec Blackfire

### 2.2.1 - Métriques

L'outil Blackfire.io développé par SensioLabs permet d'analyser la performance de l'application.

Il s'agit d'un profileur de performance d'application PHP. Il permet de mesurer l'utilisation des ressources matérielles mobilisées pour son exécution.

Pour chaque profilage Blackfire permet d'identifier différentes métriques :

- ❖ Le temps de génération de la page, découpé en temps d'accès au processeur et temps d'accès au système de fichiers,
- ❖ "I/O Wait" fait référence au temps que le programme passe à attendre l'achèvement des opérations d'entrée/sortie (I/O). Cela inclut principalement les opérations telles que la lecture ou l'écriture de fichiers, les accès au réseau, les requêtes à une base de données, etc,
- ❖ La mémoire utilisée,
- ❖ L'utilisation du réseau.

Les mesures ci-après ont été réalisées en environnement de production sur le serveur local de développement.

## 2.2.2 - Etat des lieux

On constate que l'autoload est un des principaux goulots d'étranglement de l'application.

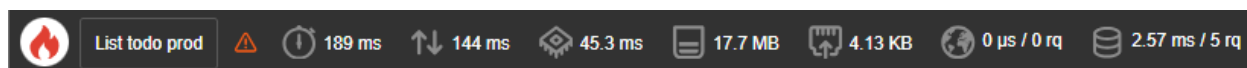
### ■ Environnement de Développement



|                                    |  |     |
|------------------------------------|--|-----|
| Composer\Autoload\includeFile      |  | 399 |
| ...poser\Autoload\includeFile@1    |  | 190 |
| ...poser\Autoload\includeFile@2    |  | 81  |
| ...sLoader::findFileWithExtension  |  | 733 |
| ...poser\Autoload\includeFile@3    |  | 40  |
| ...::vendor/autoload_runtime.php   |  | 1   |
| ...\Autoload\ClassLoader::findFile |  | 736 |
| ...:composer/autoload_static.php   |  | 1   |
| ...utoload\ClassLoader::loadClass  |  | 407 |
| ...a47d826a9d9c2bd2::getLoader     |  | 1   |
| ...load\ClassLoader::loadClass@1   |  | 190 |
| ...load\ClassLoader::loadClass@2   |  | 81  |
| run_init::vendor/autoload.php      |  | 1   |
| ...load\ClassLoader::loadClass@3   |  | 40  |

## ■ Environnement de Production

Nous pouvons constater une nette amélioration sur tous les indicateurs lorsque l'on passe en environnement de production qui optimise notamment l'autoload et permet un meilleur temps de chargement de la page de liste des tâches.



|   | Function calls                     | % Excl. ▾   | % Incl. | Calls |
|---|------------------------------------|-------------|---------|-------|
|   | Composer\Autoload\includeFile      | <div></div> |         | 291   |
|   | ...poser\Autoload\includeFile@1    | <div></div> |         | 164   |
|   | ...poser\Autoload\includeFile@2    | <div></div> |         | 71    |
| ■ | ...sLoader::findFileWithExtension  | <div></div> |         | 584   |
|   | ...poser\Autoload\includeFile@3    | <div></div> |         | 37    |
|   | ...\Autoload\ClassLoader::findFile | <div></div> |         | 587   |
|   | ...load\ClassLoader::loadClass@4   | <div></div> |         | 14    |
| ■ | ...:composer/autoload_static.php   | <div></div> |         | 1     |
|   | ...utoload\ClassLoader::loadClass  | <div></div> |         | 297   |
|   | ...a47d826a9d9c2bd2::getLoader     | <div></div> |         | 1     |
|   | ...load\ClassLoader::loadClass@1   | <div></div> |         | 164   |
| ■ | ...:vendor/autoload_runtime.php    | <div></div> |         | 1     |
|   | ...load\ClassLoader::loadClass@2   | <div></div> |         | 71    |
| ■ | run_init::vendor/autoload.php      | <div></div> |         | 1     |
|   | ...load\ClassLoader::loadClass@3   | <div></div> |         | 37    |

BlackFire nous donne des suggestions d'amélioration de l'application.

The screenshot shows the BlackFire interface with a sidebar on the left containing navigation links: Recommendations (11), Assertions, Timeline, Callgraph, Propagation (0 rq), SQL (2.57 ms / 5 rq), HTTP (0 µs / 0 rq), and Cache. The main content area is titled 'Recommendations' and includes a link to 'Read more about recommendations'. A text box explains that recommendations are best practices to improve application performance. A dashed box highlights that 8 more recommendations were found for this profile, consisting of 5 Quality and 3 Security recommendations, with a link to 'Subscribe to the add-ons'. Below this, three specific recommendations are listed, each with a 'How to fix' button:

- .env configuration should not be parsed in production**  
metrics.symfony.dotenv.parse.count 2 == 0
- Doctrine DQL statements should be cached in production**  
metrics.doctrine.dql.parsed.count 0 - metrics.doctrine.paginator.fetch\_join\_collection.count 1 == 0
- The Composer autoloader class map should be dumped in production**  
metrics.composer.autoload.find\_file.count 584 <= 50

### 2.2.3 - Optimisation

#### ■ Autoload : chargement des classes

Composer gère le chargement automatique des classes en générant une classe de chargement automatique. Chaque fois que Php charge une classe, Composer doit rechercher le fichier correspondant sur le système de fichiers, ce qui est lent.

Nous devons utiliser la commande :

```
composer dump-autoload --optimize
```

#### Amélioration des performances

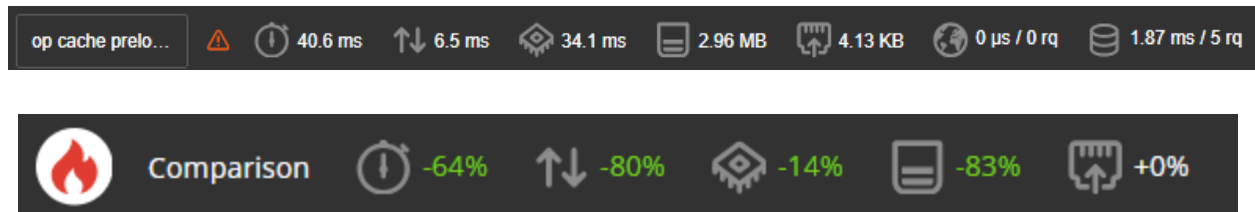
La commande génère un autoload statique qui est plus rapide que l'autoload dynamique par défaut.

L'autoload statique permet de réduire le temps de recherche et de chargement des classes, ce qui améliore les performances d'exécution de l'application.



## Réduction de la surcharge

En utilisant un autoloading statique, Composer évite de parcourir les chemins de fichiers et de dossiers à chaque requête, ce qui réduit la surcharge et le temps d'exécution.



On constate une amélioration importante de tous les métriques.

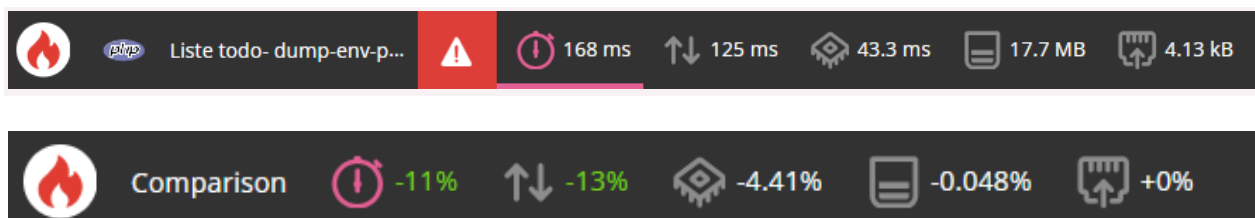
La classmap pouvant être volumineuse, il est fortement recommandé d'utiliser l'extension OpCache de php et de configurer son preload dans le fichier php.ini du serveur de production.

### ■ Configuration du fichier .env

Sur le serveur de production, il est nécessaire de définir les variables d'environnement réelles au lieu de les charger à partir du fichier .env, afin d'économiser des ressources lors de l'analyse du fichier pour chaque requête HTTP.

L'exécution de la commande suivante est requise :

```
composer dump-env prod
```

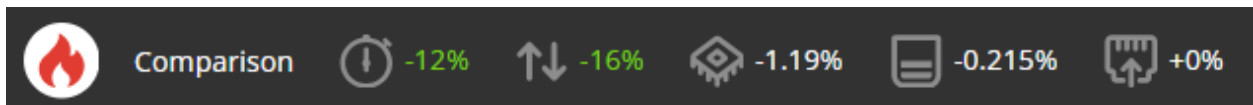


On constate une amélioration globale des métriques.

## ■ Doctrine DQL statement

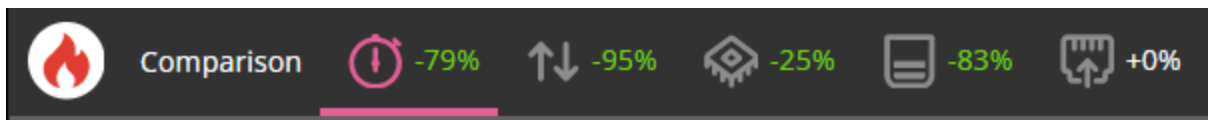
La transformation des requêtes DQL en SQL est un processus lourd, Doctrine utilise un mécanisme de cache pour sauvegarder le résultat des traitements qui doit être configuré avec par exemple l'extension php apcu.

```
doctrine:
    orm:
        query_cache_driver: apc
```



On constate une amélioration globale des métriques.

## 2.2.4 - Synthèses des différences



L'utilisation de BlackFire a permis de réduire :

- ❖ le temps de chargement de 189 ms à 40,6 ms,
- ❖ I/O Wait de 144 ms à 40,6 ms,
- ❖ CPU de 45,3 ms à 34,1 ms.
- ❖ Peak memory de 17,7 mb à 2,96 mb.

## 3. Synthèse des résultats

### 3.1 Comparaison globale des deux versions

#### 3.1.1 - Qualité du code

**Réduction significative des issues :** La nouvelle version du projet montre une amélioration notable par rapport à la précédente, avec une diminution de 163 issues. Cette baisse significative est un indicateur clé de l'optimisation et de la stabilisation du code. Elle a permis de renforcer la fiabilité globale du logiciel.

**Élimination du code non utilisé :** L'audit du code a révélé qu'il n'y a plus de segments de code inutilisés. Cela contribue non seulement à la clarté et à la maintenabilité du projet, mais réduit également la complexité de la base de code. Cela permet de mieux comprendre et maintenir le code, et facilite les futures mises à jour et corrections.

**Correction des failles de sécurité :** La nouvelle version a réussi à combler toutes les failles de sécurité détectées. Ce point est crucial, car il assure la protection des données et la fiabilité des fonctionnalités offertes par le logiciel. La sécurité est un aspect essentiel pour gagner la confiance des utilisateurs et prévenir des vulnérabilités potentiellement exploitables.

**Diminution des issues liées au style de code :** Les problèmes liés au style de code sont désormais anecdotiques. Un effort a été réalisé afin de respecter les normes de codage et les bonnes pratiques. Un code uniformisé et bien stylisé facilite la collaboration entre les développeurs et améliore la lisibilité du code.

**Absence de régressions :** Aucune régression n'a été constatée dans la nouvelle version. L'absence de régression est un signe de la qualité des tests effectués et de l'efficacité du processus de développement. Cela signifie que les nouvelles fonctionnalités et corrections n'ont pas introduit de nouveaux problèmes, ce qui témoigne de la robustesse et de la fiabilité du logiciel.

En résumé, les améliorations apportées à la nouvelle version du projet sont significatives. Cela démontre un progrès notable en termes de qualité, de sécurité et de maintenabilité. Ces efforts contribueront à la satisfaction des utilisateurs et à la pérennité du projet.

### 3.1.2 - Performance

**Amélioration de l'Autoload avec Composer:** Utilisation de `composer dump-autoload --optimize` pour générer un autoload statique plus rapide, réduisant le temps de chargement des classes et la surcharge globale.

**Utilisation d'OpCache PHP:** Configuration recommandée dans `php.ini` pour OpCache afin de précharger les fichiers en mémoire et optimiser le chargement des classes en production.

**Configuration du fichier `.env` pour la production:** Utilisation de `composer dump-env prod` pour générer un fichier `.env` adapté à l'environnement de production, améliorant l'efficacité en évitant la lecture répétée du fichier.

**Optimisation des requêtes Doctrine DQL:** Utilisation d'un cache comme APCu pour stocker les résultats des requêtes DQL et réduire le temps de traitement SQL, améliorant les performances globales.

**Amélioration des métriques avec Blackfire:** Observations de réductions significatives : temps de chargement, I/O Wait, utilisation CPU et pic de mémoire, confirmant l'efficacité des optimisations appliquées.

Cette approche garantit des performances optimisées tout en minimisant les ressources nécessaires pour l'exécution et l'analyse des requêtes dans l'application.

## 4. Conclusion

### 3.1 Plan d'action pour les versions futures

#### 3.1.1 - Recommandations globale pour la maintenance

##### Qualité du code

Il serait intéressant d'intégrer à la CI du projet les tests de PHPUnit, afin de maintenir la qualité du code en lançant les tests automatisés à chaque nouvel ajout de code. Cela permettrait notamment :

- de détecter rapidement les erreurs,
- éviter les régressions de code,
- contribuer à l'amélioration de la fiabilité des déploiements (fréquences plus soutenues, moins de risques),
- amélioration de la qualité du code.

L'utilisation de Php Cs Fixer ou de l'extension de Codacy lors du développement permet également d'éviter les régressions et maintenir une bonne qualité de code.

La création de Guidelines permettrait également d'assurer la consistance et la qualité du code.

##### Performance

Il est important de maintenir à jour Symfony, ses composants et ses différentes dépendances, afin de bénéficier des améliorations de performance et des correctifs de sécurité.

Continuer à suivre les bonnes pratiques comme :

- la revue de code,
- réaliser des tests de performance de manière régulière,
- documenter les optimisations et les configurations pour faciliter la maintenance.

L'utilisation d'outils de profiling et monitoring (comme BlackFire) permet d'identifier les goulots d'étranglement, les performances des requêtes et des templates. La version payante permet également de programmer des builds périodiques.

### 3.1.2 - Stratégie d'amélioration continue

#### Qualité

- Intégration de PHPUnit dans la CI GitHub Action - <https://github.com/php-actions/phpunit>
- Maintien de la qualité du code avec Php Cs Fixer et Codacy : examiner régulièrement les rapports de Codacy pour identifier et corriger les problèmes de qualité avant de pousser les PR sur la branche principale.
- Maintenir la couverture de test au-delà de 70% en établissant les tests nécessaires pour chaque nouvelles fonctionnalités.
- Établir des Guidelines à suivre par les développeurs.

#### Performance

- Mise à jour régulière des dépendances du projet.
- Construire une veille technologique pour se tenir informer des avancées et éventuellement faire une refactorisation du projet.
- Suivi et planification de tests réguliers.
- Utilisation des outils mis en place pour identifier les goulots d'étranglement.

#### Générale

- Revue de code, attendre la validation des PR par ses pairs avant de merger.
- Documentation, établir des documentations du code, des configurations spécifiques et des fonctionnalités mises en place.
- Suivre le guide de contribution établis.