

# Audit Code et performances



---

FABIEN STENNELER

20/12/2019

# SOMMAIRE

## **1. Contexte**

- 1.1. Présentation
- 1.2. Environnement
- 1.3. Volumétrie

## **2. Qualité du code**

- 2.1. Analyses de code
  - 2.1.1. Rapport Symfony Insights
  - 2.1.2. Rapport du profiler de Symfony
  - 2.1.3. Version
  - 2.1.4. Corrections apportées
- 2.2. Sécurité
- 2.3. Respect des standards
- 2.4. Affichage
- 2.5. Fixtures
- 2.6. Tests automatisés

## **3. Performances**

- 3.1. Rapport Blackfire initial
- 3.2. Optimisations

# 1. Contexte

## 1.1. Présentation

La société ToDo & Co est une startup récente. L'application ToDo List a été développée rapidement afin de pouvoir en présenter le concept au plus tôt.

Les fonctionnalités implémentées sont les fonctionnalités de base, l'application fera l'objet de nombreux développements à venir, et son utilisation sera menée à croître rapidement.

## 1.2. Environnement

L'application a été développée sur la base du framework Symfony 3.1.10.

Cette version nécessite à minima la version 5.5.9.

JSON doit être activé.

ctype doit être activé.

Le fichier php.ini doit contenir un réglage date.timezone.

L'application utilise la librairie Doctrine, qui nécessite l'utilisation du driver PDO afin de pouvoir effectuer des requêtes de base de données.

Le système de base de données configuré est MySQL.

Afin de charger les différentes librairies nécessaires au fonctionnement de Symfony, celles-ci sont installées grâce au gestionnaire de dépendances Composer.

## 1.3. Volumétrie

L'application ayant été conçue uniquement dans un but de démonstration, il n'y a donc pas ou peu d'optimisation de performances qui ont été effectuées.

il faudra donc dans les futurs développements surveiller et optimiser l'application sur des montées en charge de volumétrie, à estimer en fonction du trafic prévu.

## 2. Qualité du code

### 2.1. Analyses de code

#### 2.1.1. Rapport Symfony Insights

**Severity**

- 1 Critical
- 5 Major
- 3 Minor
- 5 Info

**Category**

- 1 Architecture
- 4 Bugrisk
- 2 Deadcode
- 3 Performance
- 1 Readability
- 3 Security

**Developer**

- 11 Fabien Stenneler
- 3 Collective

**Stats**

Lines of code: 2,523  
Nb of suggestions: 14

**Last commit**

updated composer by Fabien Stenneler 2 days ago. master

**Issues**

- A Symfony application should be bootable Read doc Ignore all Bugrisk Major
- The composer.json file should be valid Read doc Ignore all Bugrisk Major
- Source code should not contain tasks comments Read doc Ignore all Bugrisk Major
- Symfony applications should not contain a config.php file Read doc Ignore all Security Major
- Commented code should not be committed 2 Read doc Ignore all Deadcode Minor
- Web applications should contain a site.webmanifest file Read doc Ignore all Readability Minor
- The composer.json file should not raise warnings Read doc Ignore all Bugrisk Info
- .htaccess should be avoided 3 Read doc Ignore all Performance Info
- Default favicon should be changed Read doc Ignore all Security Info

L'analyse de code Symfony Insights révèle plusieurs problèmes, dont 1 alerte critique et 4 majeures.

Ces erreurs empêchent en l'état le développement de nouvelles fonctionnalités.

#### Alerte critique

- Des failles de sécurité ont été trouvées dans plusieurs dépendances. Le projet ne pourra pas être déployé en production sans correction.

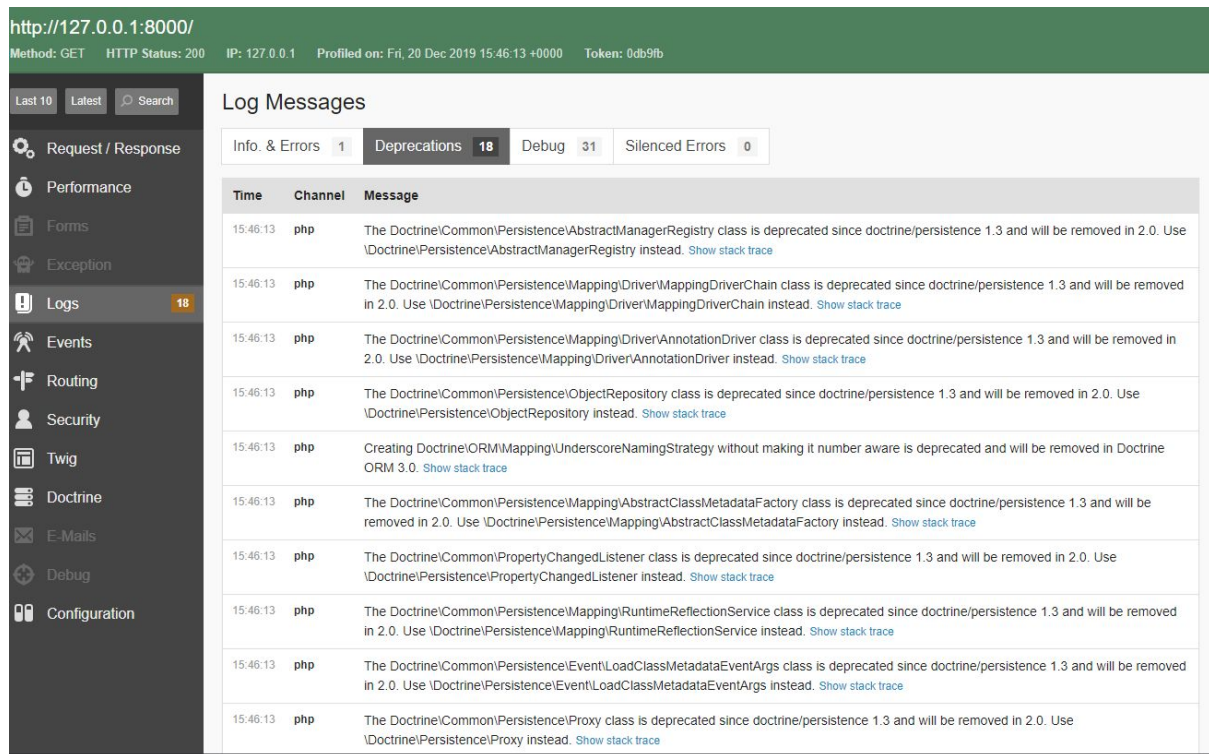
### **Alertes majeures**

- Le projet doit utiliser les migrations de Doctrine. La base données sera difficile à synchroniser lors des futurs développements.
- L'application n'est pas bootable. Les fichiers de configuration doivent être modifiées pour pouvoir lancer la suite des tests de démarrage de Symfony.
- Des commentaires "TODO" ont été laissés dans le code.
- Le fichier config.php doit être supprimé pour passer en production.

### **Alertes mineures**

- Le code commenté doit être supprimé.
- Un fichier .webmanifest doit être créé
- Le fichier composer.json doit être valide
- Le favicon par défaut de Symfony doit être modifié.

## 2.1.2. Rapport du profiler Symfony



The screenshot shows the Symfony Profiler interface. The top bar indicates the URL `http://127.0.0.1:8000/`, method `GET`, status `200`, IP `127.0.0.1`, and profile time `Fri, 20 Dec 2019 15:46:13 +0000`. The left sidebar contains navigation links: Request / Response, Performance, Forms, Exception, Logs (18), Events, Routing, Security, Twig, Doctrine, E-Mails, Debug, and Configuration. The main panel is titled 'Log Messages' and shows a list of messages. The 'Deprecations' tab is selected, showing 18 messages. The messages are all from the 'php' channel and occur at 15:46:13. They list various deprecated classes from Doctrine and Symfony, such as `\Doctrine\Common\Persistence\AbstractManagerRegistry`, `\Doctrine\Common\Persistence\Mapping\Driver\MappingDriverChain`, and `\Doctrine\Common\Persistence\Proxy`, each with a 'Show stack trace' link.

Time	Channel	Message
15:46:13	php	The Doctrine\Common\Persistence\AbstractManagerRegistry class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\AbstractManagerRegistry instead. <a href="#">Show stack trace</a>
15:46:13	php	The Doctrine\Common\Persistence\Mapping\Driver\MappingDriverChain class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\Mapping\Driver\MappingDriverChain instead. <a href="#">Show stack trace</a>
15:46:13	php	The Doctrine\Common\Persistence\Mapping\Driver\AnnotationDriver class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\Mapping\Driver\AnnotationDriver instead. <a href="#">Show stack trace</a>
15:46:13	php	The Doctrine\Common\Persistence\ObjectRepository class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\ObjectRepository instead. <a href="#">Show stack trace</a>
15:46:13	php	Creating Doctrine\ORM\Mapping\UnderscoreNamingStrategy without making it number aware is deprecated and will be removed in Doctrine ORM 3.0. <a href="#">Show stack trace</a>
15:46:13	php	The Doctrine\Common\Persistence\Mapping\AbstractClassMetadataFactory class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\Mapping\AbstractClassMetadataFactory instead. <a href="#">Show stack trace</a>
15:46:13	php	The Doctrine\Common\Persistence\PropertyChangedListener class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\PropertyChangedListener instead. <a href="#">Show stack trace</a>
15:46:13	php	The Doctrine\Common\Persistence\Mapping\RuntimeReflectionService class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\Mapping\RuntimeReflectionService instead. <a href="#">Show stack trace</a>
15:46:13	php	The Doctrine\Common\Persistence\Event\LoadClassMetadataEventArgs class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\Event\LoadClassMetadataEventArgs instead. <a href="#">Show stack trace</a>
15:46:13	php	The Doctrine\Common\Persistence\Proxy class is deprecated since doctrine/persistence 1.3 and will be removed in 2.0. Use \Doctrine\Persistence\Proxy instead. <a href="#">Show stack trace</a>

Il y a 18 erreurs de dépréciations relevées, qui montrent une nécessité de migrer l'application sur une version plus récente de Symfony.

Dans le cas contraire le code pourrait ne plus fonctionner sur une future version.

## 2.1.3. Version

Comme le montrent les rapports d'analyse de code, la priorité absolue est de migrer l'application sur une version majeure récente de Symfony, afin de préparer tous les futurs développements.

La dernière version stable à ce jour est la version 5.0.2.

[Home](#) / [What is Symfony](#) / [Symfony Releases](#)

## Symfony Releases

Symfony releases follow a time-based model: minor versions come out every six months (in May and November); major versions come out every two years. Check out the [release process details](#).

Latest Stable Release	Latest Long-Term Support Release
<b>5.0.2</b>	<b>4.4.2</b>
<ul style="list-style-type: none"><li>• First released in November 2019.</li><li>• <b>Recommended for most users.</b></li><li>• Includes the latest features.</li><li>• It's easier to upgrade to newer versions.</li></ul>	<ul style="list-style-type: none"><li>• First released in November 2019.</li><li>• 3 year support for bugs and security fixes.</li><li>• It doesn't include the latest features.</li><li>• It's harder to upgrade to newer versions.</li></ul>

Les différences structurelles entre la version 3.1 et la version 5.02 nécessitent une adaptation du code de l'application. L'application ne disposant pas de fonctionnalités avancées, il était plus simple de procéder en trois étapes :

- installation d'une nouvelle installation de symfony basée sur le website-skeleton de la version 5.0.2
- copie du contenu du dossier /src de l'application existante vers la nouvelle
- adaptation du code et de la configuration du framework afin de restaurer les fonctionnalités

### 2.1.4. Corrections apportées

Les corrections apportées aux différents alertes indiquées ci-dessus permettent de résoudre la majeure partie des problèmes.



Suggestions	Fixed (14)	Ignored	Stats
► Projects must not depend on dependencies with known security issues			Read doc Security Critical
► Your project should use Doctrine migrations			Read doc Architecture Major
► A Symfony application should be bootable			Read doc Bugrisk Major
► Source code should not contain tasks comments			Read doc Bugrisk Major
► The composer.json file should be valid			Read doc Bugrisk Major
► Symfony applications should not contain a config.php file			Read doc Security Major
► Commented code should not be committed 2			Read doc Deadcode Minor
► Web applications should contain a site.webmanifest file			Read doc Readability Minor
► The composer.json file should not raise warnings			Read doc Bugrisk Info

## 2.2. Sécurité

L'analyse du code a mis en évidence plusieurs problèmes de sécurité.

### 2.2.1. Failles de sécurité majeures

Les failles de sécurité ont été découvertes dans la version 3.1.10 du framework Symfony. La migration de l'application vers la version 5.0.2 permet de corriger en totalité les failles de sécurité révélées.

Cette migration permettra également la mise à jour régulière de l'application sur les dernières versions de Symfony et des dépendances utilisées afin de conserver un niveau de fiabilité important, grâce au gestionnaire de dépendances Composer.

```
$ composer update
```

### 2.2.2. Authentification et gestion des rôles

La gestion des rôles n'a pas été prise en charge lors du développement de l'application. Ceci est un problème bloquant pour les développements futurs, et une réécriture des fonctionnalités d'authentification était nécessaire.

Désormais chaque utilisateur, lors de son authentification, sera connecté à l'application selon le rôle qui lui a été attribué. On pourra ainsi très facilement autoriser ou refuser l'accès à certaines fonctionnalités grâce aux voters de Symfony :

```
// src/Controller/TaskController.php
// ...

class TaskController extends AbstractController
{
    /**
     * @Route("/tasks", name="task_list")
     */
    public function listAction()
    {
        $this->denyAccessUnlessGranted('ROLE_ADMIN');
        // ...
    }
}
```

Deux nouvelles classes ont également été ajoutées au projet afin de pouvoir mieux maîtriser le comportement de l'authentification :

src/security/AppAuthenticator.php

Cette classe étend la classe du composant de sécurité Symfony `AbstractFormLoginAuthenticator`, et contient toutes les méthodes du processus d'authentification, et est appelée tel que défini dans le fichier de configuration `security.yaml` pour gérer le processus d'authentification.

```
// config/packages/security.yaml
// ...
security:
    firewalls:
        // ...
        main:
            pattern: ^/
            provider: app_user_provider
            anonymous: true
            guard:
                authenticators:
                    - App\Security\AppAuthenticator
            logout:
                path: app_logout
                target: homepage
// ...
```

### 2.2.3. Mots de passe

La longueur indiquée pour le stockage du mot de passe risque de tronquer les mots de passe cryptés. On doit donc passer cette valeur à 255 bytes.

Par ailleurs aucune contrainte n'est appliquée sur la longueur du mot de passe. On peut a minima appliquer des validations sur la longueur minimum et la longueur maximum, et dans l'idéal sur la composition du mot de passe (une majuscule, une minuscule et un chiffre).

```
// src/Entity/User.php
// ...

class User implements UserInterface
{
    // ...

    /**
     * @ORM\Column(type="string", length=255)
     * @Assert\Length(
     *     min = 6,
     *     max = 30,
     *     minMessage = "Le mot de passe doit contenir au moins {{
limit }} caractères",
     *     maxMessage = "Le mot de passe ne doit comtenir au
maximum {{ limit }} caractères"
     * )
     * @Assert\Regex(
     *
pattern="/^(?=.*[0-9]) (?=.*[a-z]) (?=.*[A-Z]) ([a-zA-Z0-9]+)$/",
     *     message="Le mot de passe doit contenir au moins un
chiffre, une lettre minuscule et une lettre majuscule."
     * )
     */
    private $password;

    // ...
}
```

#### 2.2.4. Cryptage des mots de passe

Les mots de passe étaient cryptés obligatoirement avec la méthode bcrypt, ce qui n'est pas recommandé par les best practices de Symfony. Le passage de l'encoder en auto permet de résoudre ce problème.

```
// config/packages/security.yaml

security:
  encoders:
    App\Entity\User:
      algorithm: auto
```

#### 2.2.5. Unicité du champ username

La configuration de l'entité User ne précise pas de contrainte d'unicité pour le champ username. Une contrainte a donc été ajoutée.

```
// src/Entity/User.php
// ...

/**
 * @ORM\Table("user")
 * @ORM\Entity
 * @UniqueEntity(
 *     fields={"username"},
 *     message="Le nom d'utilisateur{{ value }} est déjà utilisé."
 * )
 * @UniqueEntity(
 *     fields={"email"},
 *     message="L'adresse email {{ value }} est déjà utilisée."
 * )
 */
class User implements UserInterface
{
    // ...
}
```

### 2.2.6. Limitation des accès

Tous les utilisateurs ont accès à la suppression des tâches, et la gestion des autres utilisateurs.

Nous avons donc mis en place deux limitations majeures :

- Les utilisateurs ne peuvent supprimer que leurs propres tâches
- Seuls les administrateurs peuvent accéder à la gestion des utilisateurs

Pour cela on a réalisé une jointure entre l'entité User et l'entité Task, permettant de relier chaque tâche à un utilisateur précis.

Comme nous ne disposons pas des informations des utilisateurs des premières tâches créées, celles-ci sont considérées comme anonymes et seuls les utilisateurs disposant d'un rôle d'administrateur (ROLE\_ADMIN) peuvent les supprimer.

```
// config/packages/security.yaml
// ...
security:
    // ...
    access_control:
        - { path: ^/login$, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/users, roles: [ ROLE_ADMIN ] }
        - { path: ^/, roles: [ ROLE_USER, ROLE_ADMIN ] }
```

```
// src/Controller/TaskController.php
// ...

/**
 * @Route("/tasks/{id}/delete", name="task_delete")
 */
public function deleteTaskAction(Task $task)
{
    if(
        $task->getUser() === $this->getUser()
        || (
            $task->getUser() === null
            && $this->isGranted('ROLE_ADMIN')
        )
    ) {
        $em = $this->getDoctrine()->getManager();
        $em->remove($task);
        $em->flush();
        // ...
    }
    // ...
}
```



### 2.2.7. Token csrf

Le token csrf (cross-site request forgery) protégeant de l'envoi de formulaires depuis des sites tiers était mis en place et reste actif.



The screenshot shows a web form with a red error message at the top: "The CSRF token is invalid. Please try to resubmit the form." Below the message are two input fields: "Title" with the value "test" and "Content" with the value "test". At the bottom of the form are two buttons: "Retour à la liste des tâches" (blue) and "Ajouter" (green).

### 2.2.8. Failles XSS

Les failles XSS sont protégées au niveau du moteur de templating Twig, la protection reste active.

### 2.2.9. Injections SQL

La protection contre les injections SQL est réalisée au niveau de Doctrine et PDO, et reste active.

### 2.2.10. Protection des dossiers

Le seul dossier accessible depuis un navigateur est le dossier /public contenant les feuilles de styles, les scripts javascript, les images et le fichier index.php, le domaine pointant vers la racine de ce dossier.

## 2.3. Respect des standards

L'analyse Symfony Insights effectuée révèle que les règles du PHP FIG PSR-1, PSR-2 et PSR-4.

On peut néanmoins ajouter que le code n'est pas suffisamment commenté :

- commentaires des classes
- commentaires des méthodes (fonction, paramètres, retour)
- commentaires à l'intérieur des méthodes

On va donc prendre soin de documenter le code. Afin d'améliorer la maintenabilité et la facilité de prise en charge par des futurs développeurs.

```
// src/Form/TaskController.php
// ...

/**
 * @Route("/tasks/{id}/delete", name="task_delete")
 */
public function deleteTaskAction(Task $task)
{
    if(
        $task->getUser() === $this->getUser()
        || (
            $task->getUser() === null
            && $this->isGranted('ROLE_ADMIN')
        )
    ) {
        $em = $this->getDoctrine()->getManager();
        $em->remove($task);
        $em->flush();
        // ...
    }
    // ...
}
```

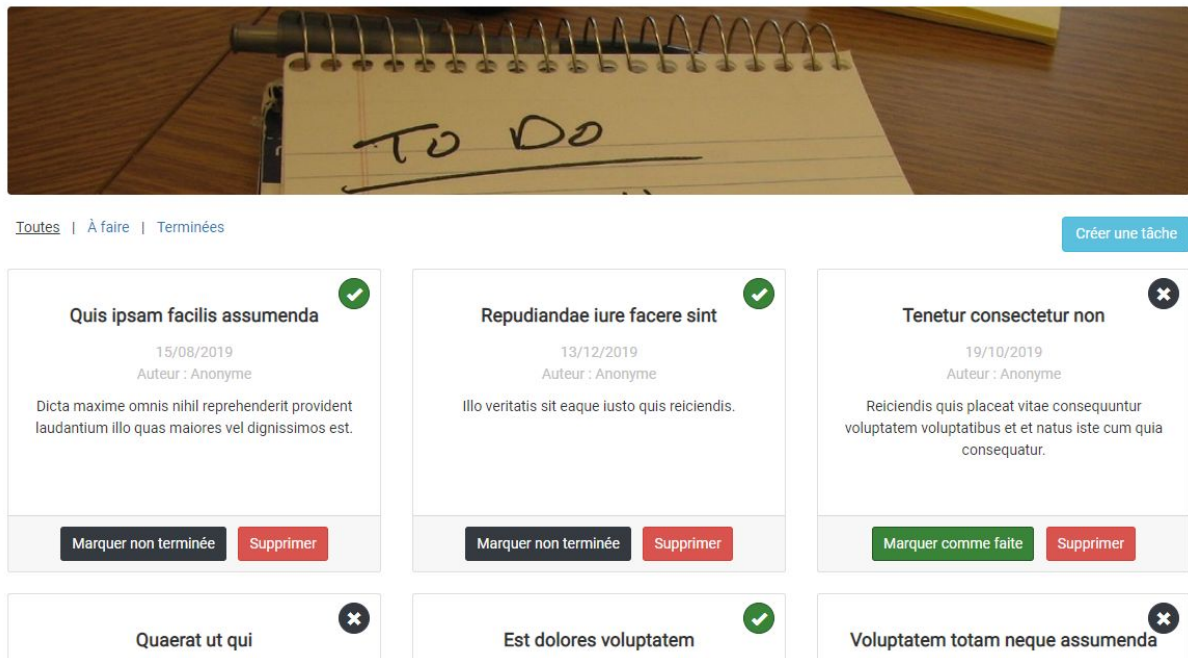
## 2.4. Affichage

Le ressenti de la qualité de l'application passe également par la qualité du rendu visuel et de l'ergonomie.

L'application ayant été développée rapidement, elle n'est pas suffisamment ergonomique et d'un aspect visuel basique.

Plusieurs modifications ont donc été opérées :

- Amélioration du rendu des formulaires :
  - Ajout de "label" dans la définition des champs sur Symfony
  - Mise en page
  - Ajout d'une fonctionnalité "rester connecté"
- Création de headers pour chaque page pour aider à la navigation
- Création d'un menu regroupant les différentes fonctionnalités
- Amélioration des liens de la homepage avec affichage du nombre de tâches à effectuer et terminées
- Prise en charge du filtrage des tâches par statut de la tâche (à effectuer, terminées)
- Amélioration du rendu des tâches et mise en place d'un système de glisser/déposer



## 2.5. Fixtures

L'application ne dispose d'aucune fixture pour initialiser le contenu de la base de données avec des données de démonstration.

Des fixtures doivent donc être créées, la génération de données fake se fera par l'utilisation de la librairie "fzaninotto/faker". Les fixtures sont générées par la librairie Doctrine fixtures, et sont stockées dans le dossier src/DataFixtures.

Le lancement de la création des fixtures se fait grâce à une commande de la console :

```
$ php bin/console doctrine:fixtures:load
```

## 2.6. Tests automatisés

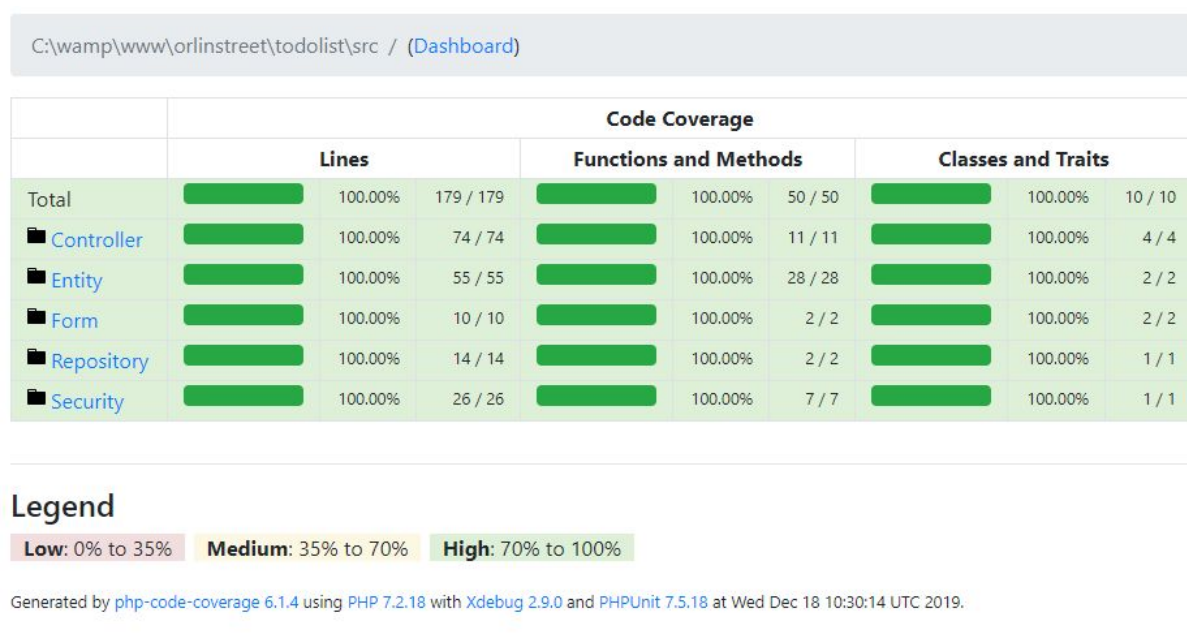
Aucun test n'est réalisé sur l'application. Afin de garantir la fiabilité et le bon fonctionnement de l'application lors de l'ajout de fonctionnalités, il convient de créer tous les tests nécessaires :

- tests unitaires pour vérifier les différentes méthodes
- tests fonctionnels pour tester le comportement des différentes fonctionnalités

Les tests sont stockés dans le dossier /tests.

Une analyse du taux de couverture des tests doit être réalisée afin de garantir que la majeure partie de l'application sera testée.

A la fin de nos développements, nous sommes arrivés à un taux de couverture de 100%.



Source : /test-coverage/index.html

## 2.7. Autres pistes d'amélioration

L'ajout et la modification et la suppression d'une tâche ou d'un utilisateurs sont scindées en deux méthodes dans les contrôleurs et les templates.

Pour plus de simplicité il serait judicieux de réunir les deux actions dans une seul et même méthodes et un seul template.

```
// src/Form/TaskController.php
// ...

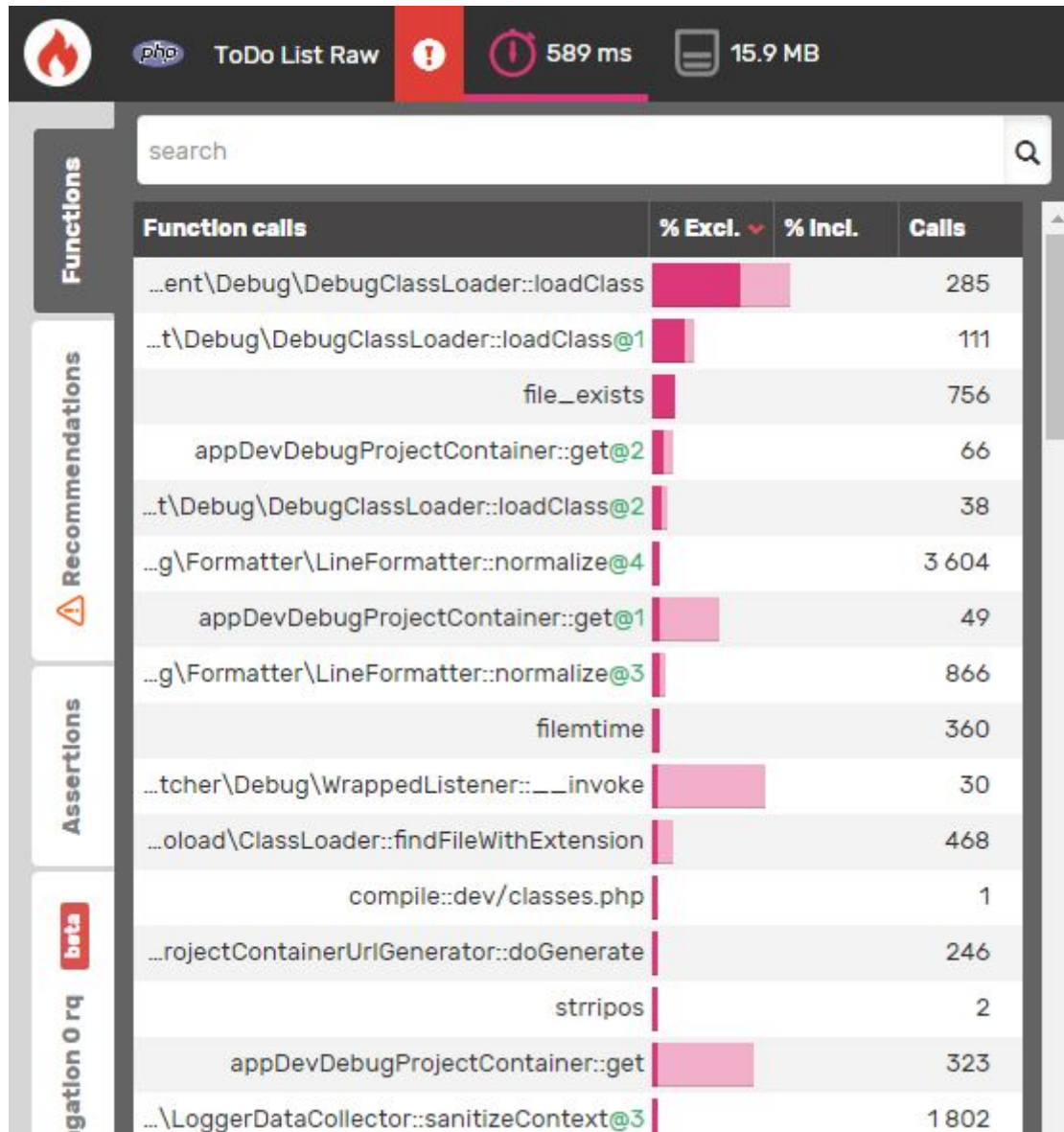
/**
 * @Route(
 *     "/tasks/{action}/{id}",
 *     name="manage_task",
 *     requirements={"action" = "(add|edit|delete)"}
 * )
 */
public function manageTaskAction($action, Task $task)
{
    // ...
    return $this->render('task/manage.html.twig', [
        'form' => $form->createView(),
        'task' => $task,
    ]);
    // ...
}
```

### **3. Performances**

L'application sera amenée à évoluer rapidement, ce qui implique des contraintes d'augmentation du volume de données et du trafic.

Afin de pouvoir au mieux anticiper ces contraintes, les performances de l'application doivent être suivies tout au long du processus de développement afin de pouvoir au mieux les optimiser.

### 3.1. Rapport Blackfire initial







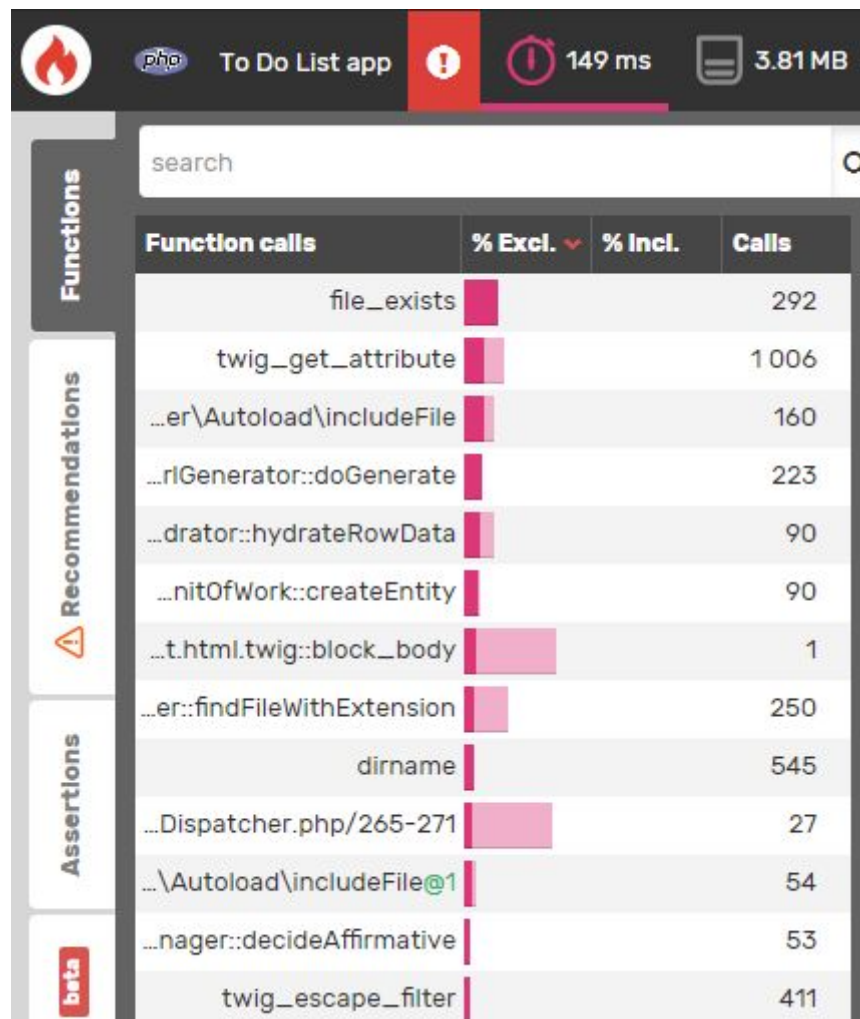
Sur cette analyse on peut voir que les ralentissements sont dûs principalement à une forte consommation du processus d'autoload de classes de Symfony.

Le temps global de chargement est de 589ms.

## 3.2. Optimisations

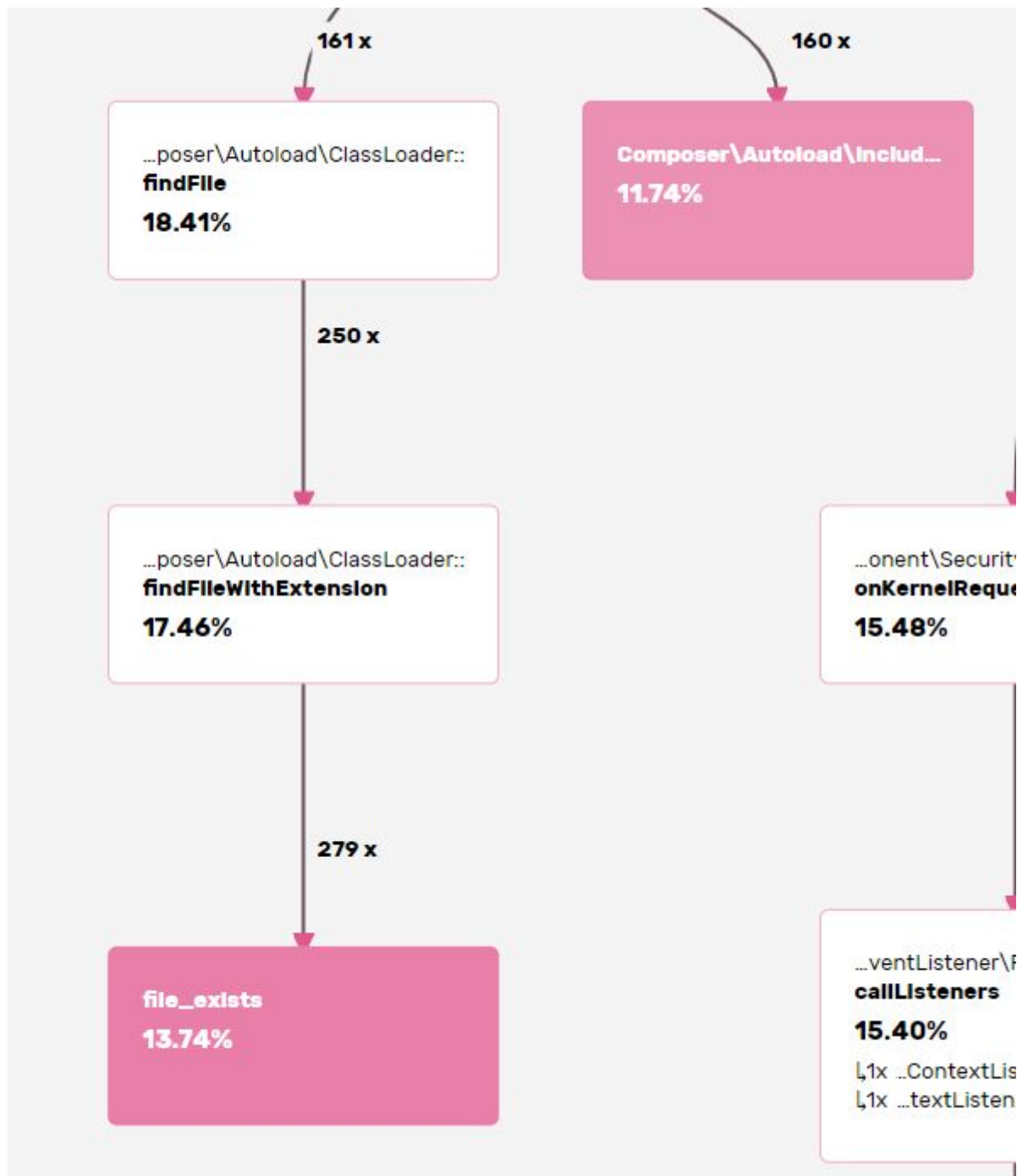
### 3.2.1. Migration sur Symfony 5

Une fois la migration sur Symfony 5 effectuée, nous constatons une amélioration sensible des performances :



### 3.2.2. Class autoloader

Pour chaque classe instanciée, l'autoloader va analyser les différents dossiers du projet, charger le fichier et inclure la classe correspondante. Ce qui va créer un nombre d'appels très important de l'autoloader :

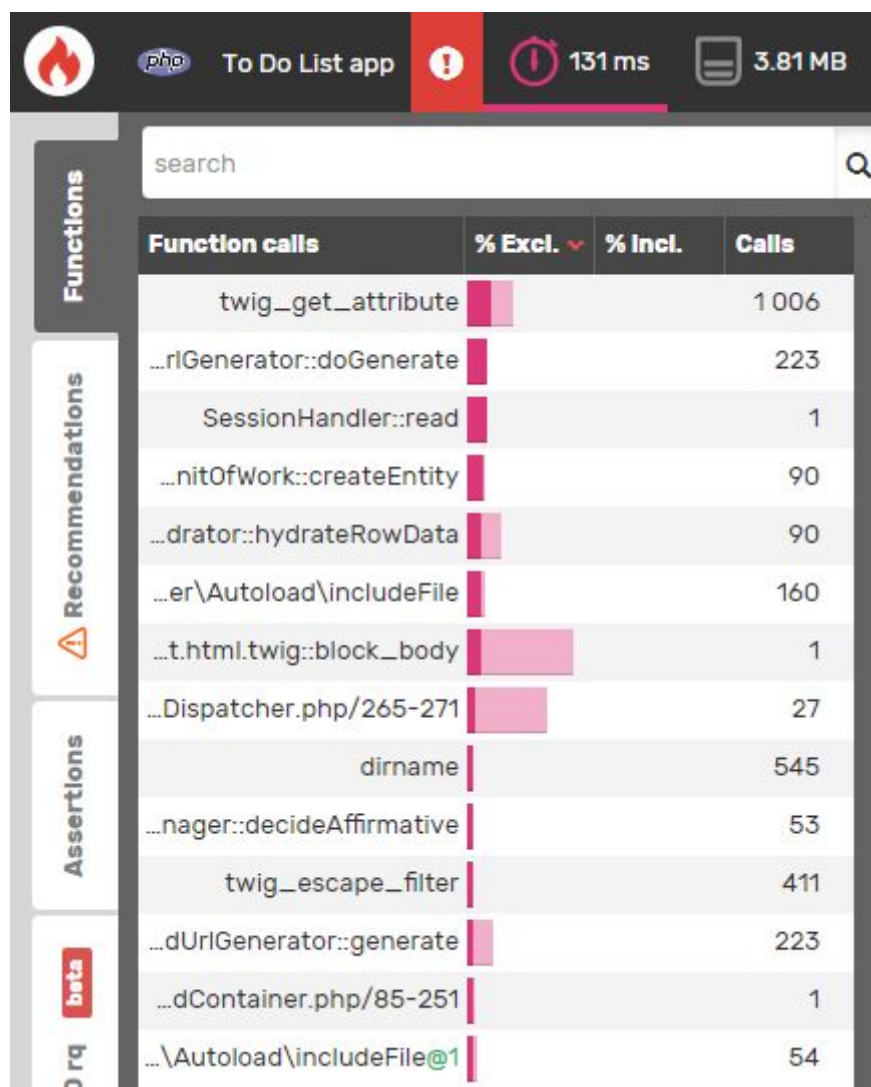


Ce comportement est facilement optimisable en créant, lors de chaque déploiement en production, un fichier statique listant les classes à charger au démarrage de l'application.

Cette liste sera stockée dans le fichier `vendor/composer/autoload_classmap.php` grâce à cette commande effectuée sur la console :

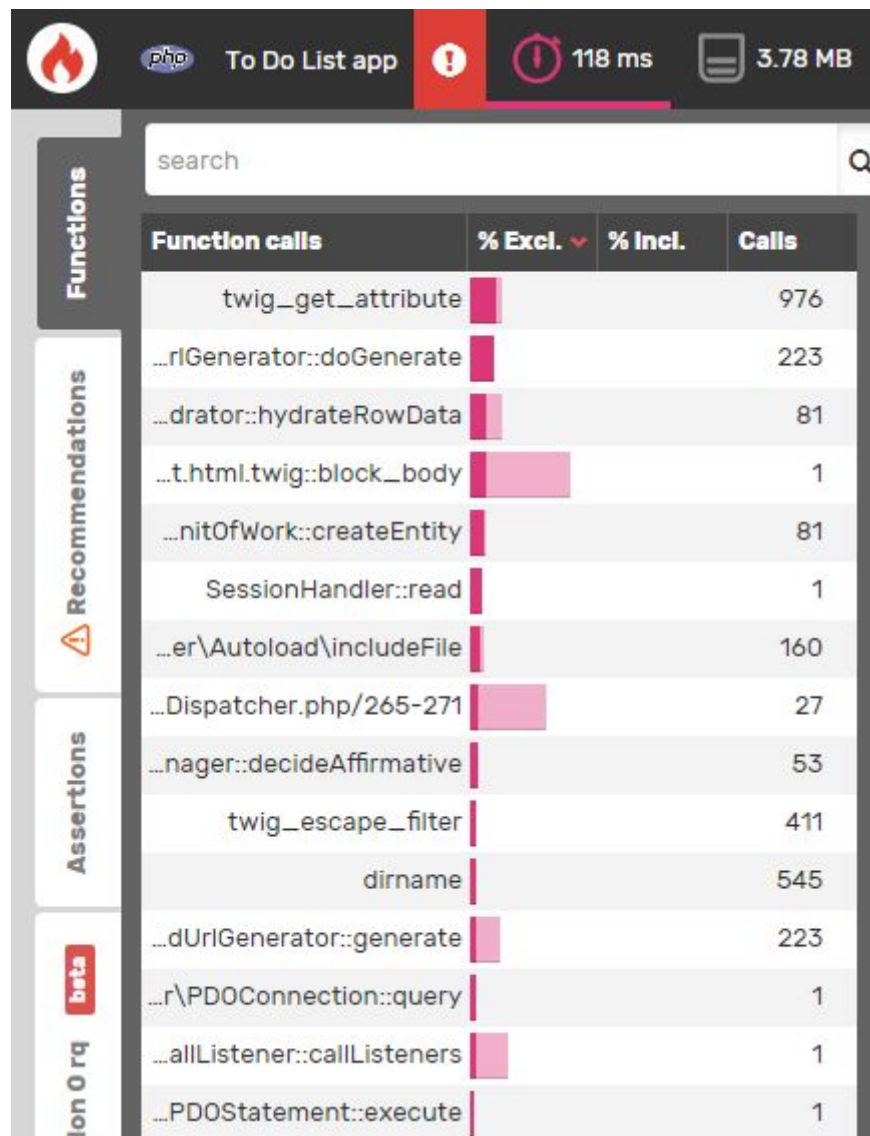
```
$ composer dump-autoload --no-dev -classmap-authoritative
```

Grâce à cette optimisation, on constate directement une amélioration des performances.



### 3.2.3. Activation de OPCache

L'utilisation de l'extension PHP OPCache permet d'améliorer les performance des différentes requêtes en mettant en cache les scripts. Il n'est ainsi plus nécessaire de charger et analyser les script à chaque demande.



Function calls	% Excl.	% Incl.	Calls
twig_get_attribute			976
...rGenerator::doGenerate			223
...drator::hydrateRowData			81
...t.html.twig::block_body			1
...nitOfWork::createEntity			81
SessionHandler::read			1
...er\Autoload\includeFile			160
...Dispatcher.php/265-271			27
...nager::decideAffirmative			53
twig_escape_filter			411
dirname			545
...dUriGenerator::generate			223
...r\PDOConnection::query			1
...allListener::callListeners			1
...PDOStatement::execute			1

Les paramètres ci-dessous ont été modifiés dans le fichier de configuration php.ini :

```
;php.ini

opcache.enable=1
opcache.memory_consumption=256
opcache.max_accelerated_files=20000
opcache.validate_timestamps=0
```

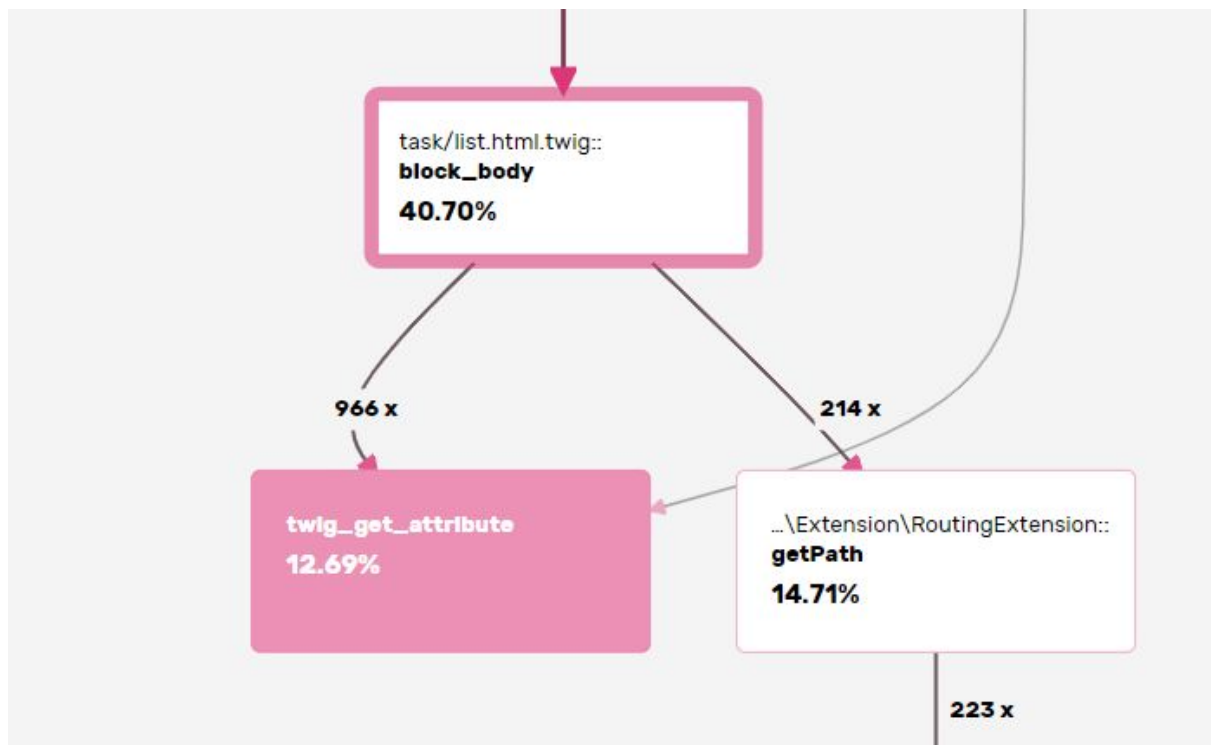
On va également indiquer les classes de Symfony à précharger. Cette liste est contenue dans un fichier créé automatiquement par Symfony.

```
;php.ini

opcache.preload=/path/to/project/var/cache/prod/App_KernelProdContainer.preload.php
```

#### **3.2.4. Réduction des appels de Twig**

Une métrique que l'on retrouve dans les rapports blackfire est la consommation des appels de ressource de Twig.



Un axe d'amélioration serait de créer une méthode dans la classe TaskRepository afin de faire une requête Doctrine sur l'ensemble des champs (y compris les champs des entités possédant des relations), et de les stocker dans un tableau.

Cette solution a été citée par Fabien Potencier comme une méthode pour accélérer le rendu des templates avec Twig.

```
// src/Repository/TaskRepository.php
// ...
class TaskRepository extends ServiceEntityRepository
{
    // ...
    public function findTasks()
    {
        return $this->createQueryBuilder('t')
            ->addSelect('u')
            ->leftJoin('t.user', 'u', 'WITH', null, 'u.id')
            ->orderBy('t.id', 'ASC')
            ->getQuery()
            ->getResult();
    }
}
```

```
// src/Controller/TaskController.php
// ...
class TaskController extends AbstractController
{
    // ...
    public function listAction()
    {
        $tasks = $this->getDoctrine()
            ->getRepository(Task::class)
            ->findTasks();

        return $this->render('task/list.html.twig', [
            'tasks' => $tasks
        ]);
    }
}
```



### 3.2.5. Pagination

Afin d'améliorer sensiblement les performances lors de l'augmentation du volume de données à stocker, il conviendra de mettre en place un système de pagination pour les tâches, à l'aide par exemple du composant Paginator de Symfony.

On pourra ainsi mettre en place soit un système de défilement continu par requêtes Ajax, soit un système standard de pagination.

```
// src/Repository/TaskRepository.php
// ...
class TaskRepository extends ServiceEntityRepository
{
    // ...
    public function findTasks()
    {
        $queryBuilder = $this->createQueryBuilder('t')
            ->addSelect('u')
            ->leftJoin('t.user', 'u', 'WITH', null, 'u.id')
            ->orderBy('t.id', 'ASC')
            ->getQuery()
            ->getResult();
        return new Paginator($queryBuilder);
    }
}
```

```
// src/Repository/TaskRepository.php

use Doctrine\ORM\Tools\Pagination\Paginator;
// ...
class TaskRepository extends ServiceEntityRepository
{
    // ...
    public function findTasks()
    {
        $queryBuilder = $this->createQueryBuilder('t')
            ->addSelect('u')
            ->leftJoin('t.user', 'u', 'WITH', null, 'u.id')
            ->orderBy('t.id', 'ASC')
            ->getQuery()
            ->getResult();
        return new Paginator($queryBuilder);
    }
}
```