

Rapport d'Audit

1/ Qualité de travail	2
a/ Collaboration sur le projet	2
b/ Mise à jour de Symfony	2
c/ Implémentation de tests	3
2/ Qualité utilisateur	4
a/ Corrections d'anomalies	4
b/ Axes d'améliorations	5
3/ Qualité de code	6
a/ Mise en place des bonnes pratiques	6
b/ Analyse du code avec Codacy	6
4/ Qualité de performance	10
5/ Conclusion	12

1/ Qualité de travail

a/ Collaboration sur le projet

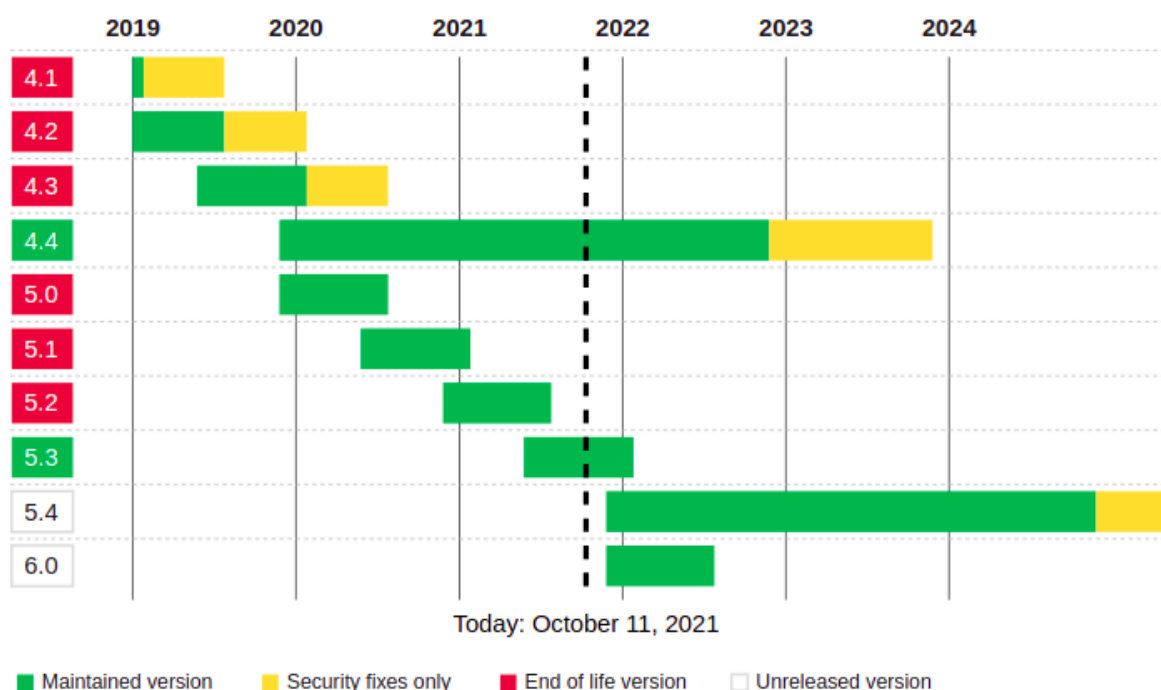
Reportez-vous au document **P8_04_contribution.pdf**, disponibles dans le dossier livrables.

Ce document a été traduit en anglais. Il est disponible à la racine du projet **contribution.md**.

b/ Mise à jour de Symfony

Le projet initial est codé sur la version 3.1 du framework Symfony.

Symfony Releases Calendar



La version 3.1 n'étant plus maintenue, nous avons fait le choix de passer sur une version plus récente : la version 5.3 de Symfony. Elle nous a permis :

- d'apporter de nouvelles fonctionnalités,
- d'améliorer la sécurité,
- d'améliorer la performance.

La maintenabilité de la version 4.4 prendra fin en novembre 2023.

Nous avons préféré travailler sur une version Latest Stable Release 5.3 (plus récente) pour passer, début novembre 2021, sur une version Latest Long-Term Support Release 5.4.

Nous avons procédé à différentes étapes :

- créé un nouveau projet en 5.3,
- configuré le projet,
- copié et collé les fichiers du projet initial.

c/ Implémentation de tests

Les tests unitaires et fonctionnels sont importants à réaliser.

Dans le cadre de notre application ToDo & Co (ou toute application), ils permettent :

- de vérifier que le code fonctionne comme nous l'avons espéré,
- d'éviter que le code ajouté crée des régressions avec le code existant.

Notre but est de mettre en place :

- des tests fonctionnels couvrant le code à 100% dans nos controllers,
- des tests unitaires permettant de couvrir et tester au maximum les cas de l'application grâce aux classes.

Grâce à nos tests, nous avons pu couvrir 96,52% :

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	<div><div></div></div>	96.52%	222 / 230	<div><div></div></div>	87.10%	54 / 62	<div><div></div></div>	53.33%	8 / 15
Controller	<div><div></div></div>	100.00%	64 / 64	<div><div></div></div>	100.00%	11 / 11	<div><div></div></div>	100.00%	4 / 4
DataFixtures	<div><div></div></div>	100.00%	28 / 28	<div><div></div></div>	100.00%	2 / 2	<div><div></div></div>	100.00%	1 / 1
Entity	<div><div></div></div>	96.08%	49 / 51	<div><div></div></div>	93.55%	29 / 31	<div><div></div></div>	50.00%	1 / 2
Form	<div><div></div></div>	100.00%	24 / 24	<div><div></div></div>	100.00%	4 / 4	<div><div></div></div>	100.00%	2 / 2
Listener	<div><div></div></div>	90.00%	9 / 10	<div><div></div></div>	50.00%	1 / 2	<div><div></div></div>	0.00%	0 / 1
Repository	<div><div></div></div>	88.24%	15 / 17	<div><div></div></div>	50.00%	2 / 4	<div><div></div></div>	0.00%	0 / 2
Security	<div><div></div></div>	91.67%	33 / 36	<div><div></div></div>	62.50%	5 / 8	<div><div></div></div>	0.00%	0 / 3

Pour atteindre les 100%, d'autres tests auraient pu être mise en place tel que des tests qui exécutent le code :

- un test créer un formulaire d'affichage utilisateur,
- un test créer un formulaire de redirection utilisateur valide,
- un test créer un utilisateur enregistré dans la base de données,
- etc...

2/ Qualité utilisateur

a/ Corrections d'anomalies

Pour améliorer le côté utilisateur, nous avons procédé à :

- des corrections d'anomalies,
- l'implémentation de nouvelles fonctionnalités.

Une tâche doit être rattachée à un utilisateur

Dans le projet initial, aucune tâche n'était reliée à un utilisateur.

Nous avons donc :

- créé une relation OneToMany entre les entités User et Task,
- affiché, dans les template twig, le nom de l'utilisateur sur chaque tâche avec `{{ task.user.username }}`,
- affiché, dans le template twig, le nom anonyme pour une tâche créé par un utilisateur non connecté grâce à la condition `{% if not task.user %} Anonyme {% else %}`.

Rôle et autorisation

Dans le projet initial, le ROLE_USER a été installé.

Nous avons dû :

- adapter le UserType en rajoutant au ROLE_USER le ROLE_ADMIN afin d'obtenir la possibilité de choisir entre le rôle utilisateur et le rôle administrateur,
- mettre en place des voters pour :
 - * autoriser les utilisateurs à la gestion de leur(s) tâche(s),
 - * autoriser l'administrateur à la gestion des utilisateurs,
 - * autoriser l'administrateur à la suppression des tâches "anonymes".
- modifier les templates twig par rapport aux différentes permissions (boutons "supprimer", boutons "editer", etc...) grâce à `is_granted()`.

Contraintes des entités User et Task

Les contraintes étant trop succinctes, nous avons pu :

- créer les contraintes d'unicités sur username et email grâce à `unique=true`,
- mettre en place des annotations sur les différents attributs des deux entités,
 - * utilisation de `@Assert\NotBlank` pour *username* et *email* et de `@Assert\Email` pour *email* de l'entité User,
 - * utilisation de `@Assert\NotBlank` pour *title* et *content* de l'entité Task,
- mettre en place un `length` à 255 sur le password de l'entité User pour permettre de hasher le mot de passe.

Navigation

Les liens permettant la navigation entre les différentes pages du site ont été revus.

D'autres anomalies ou améliorations ont été réalisées, tel que :

- la suppression d'un utilisateur par un utilisateur ayant le rôle administrateur,
- la date de modification d'une tâche dans l'entité Task,
- etc...

b/ Axes d'améliorations

Avant de mettre l'application en production, nous pouvons aller plus loin dans l'amélioration tant sur le frontend que sur le backend.

Améliorations sur le frontend

Nous pouvons nous pencher sur le design de l'application, ce qui permettrait :

- une charte graphique plus cohérente,
- une mise en page plus homogène,
- de traiter la partie responsive,
- une barre de navigation avec des liens accueil, tâches, utilisateurs, se connecter, se déconnecter, compte, etc...
- au niveau des tâches :
 - * insérer la date de création et de modification dans le template de la liste des tâches,
 - * mettre un code couleur et des labels, par exemple pour la priorité des tâches à traiter (un label urgent en rouge, etc...).

Améliorations sur le backend

Pour faire évoluer l'application côté utilisateur, elle pourrait être agrémenter de nouvelles fonctionnalités tels que :

- paginer la liste des tâches,
- paginer la liste des utilisateurs,
- un utilisateur pourrait modifier et supprimer son propre compte,
- un envoie de confirmation mail à l'utilisateur à la création ou la modification ou la suppression de son compte,
- lors de la création d'un utilisateur, paramétrer les rôles pour qu'un futur utilisateur puisse avoir le rôle utilisateur par défaut. Seul l'administrateur pourra modifier les rôles et donc passer un utilisateur ayant un rôle utilisateur en rôle administrateur.

3/ Qualité de code

a/ Mise en place des bonnes pratiques

Les normes PSR

Lorsque nous travaillons en équipe, il est important de respecter des normes PSR.

Elles ont pour but de proposer un code compréhensif et facilement évolutif :

- **PSR-1 “Basic Coding Standart”** : éléments de codage standard (utf-8, camelCase(), balise <?php, require, use, namespace).
- **PSR-2 “Coding Style Guide”** : ensemble de règles et d'attentes sur la façon de formater le code PHP (au moins 80 caractères par lignes et au maximum 120 caractères, indentation de 4 espaces, la méthode abstract(), etc...)
- **PSR-4 “Autoloading Standart”** : permet de charger automatique ce dont nous avons besoin (utilisation de autolaod.php dans le dossier vendor)
- **PSR-12 “Extended coding Style”** : étend le PSR-2 (héritage, instanceof, trait(), etc...)

Architecture pattern MVC

Le projet initial utilise l'architecture MVC (Model View Controller).

Ce choix a été conservé car il a l'avantage :

- d'obtenir un code clair et propre,
- d'être facile de travailler en équipe,
- de permettre l'évolution de l'application.

Sécurité

Il est indispensable de mettre en place un token pour lutter contre les failles de sécurité.

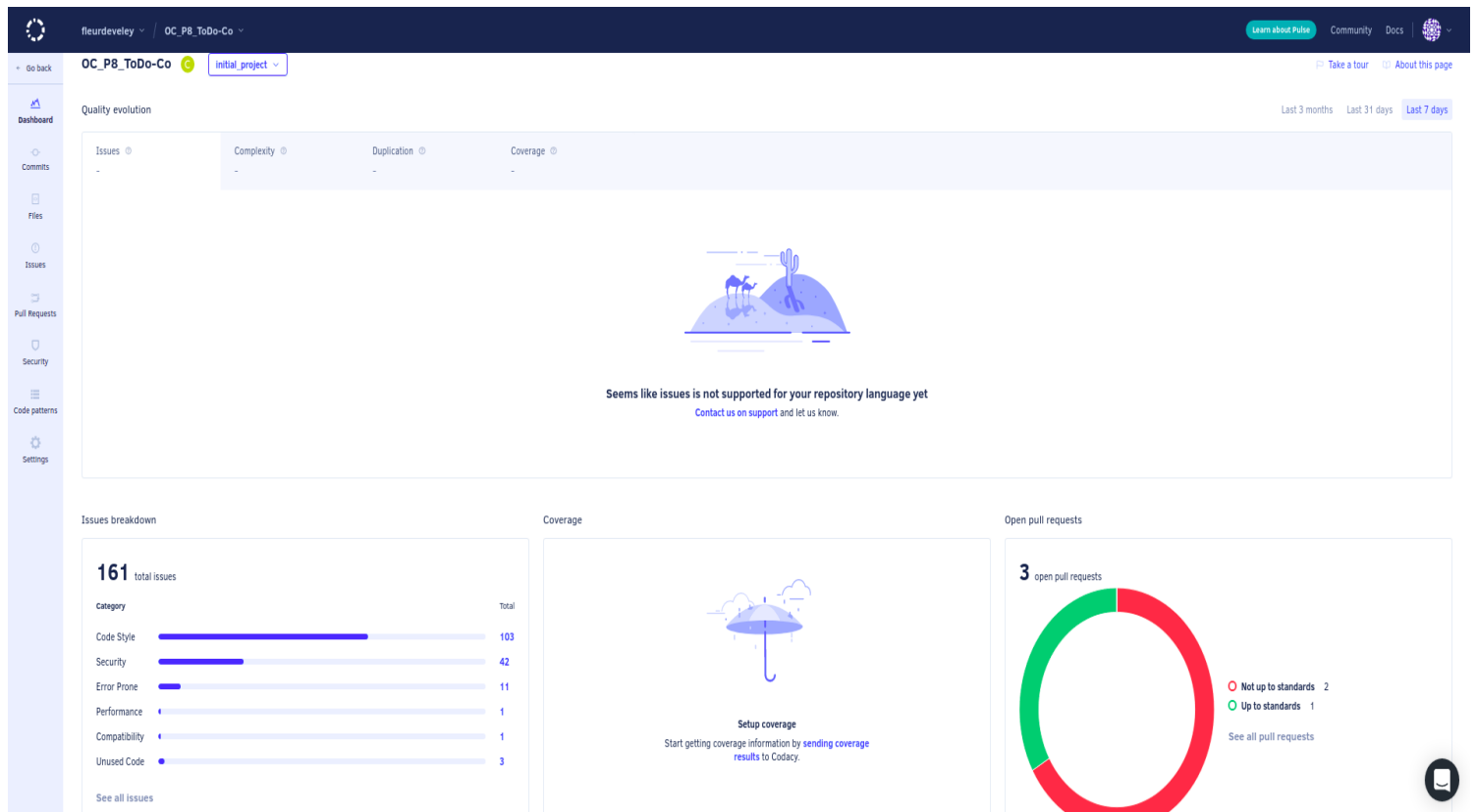
Un système d'authentification permet de vérifier le token pour la connexion d'un utilisateur.

b/ Analyse du code avec Codacy

Afin d'analyser la qualité du code, nous avons choisi Codacy :

- https://app.codacy.com/gh/fleurdeveley/OC_P8_ToDo-Co/dashboard

Note du projet initial : C

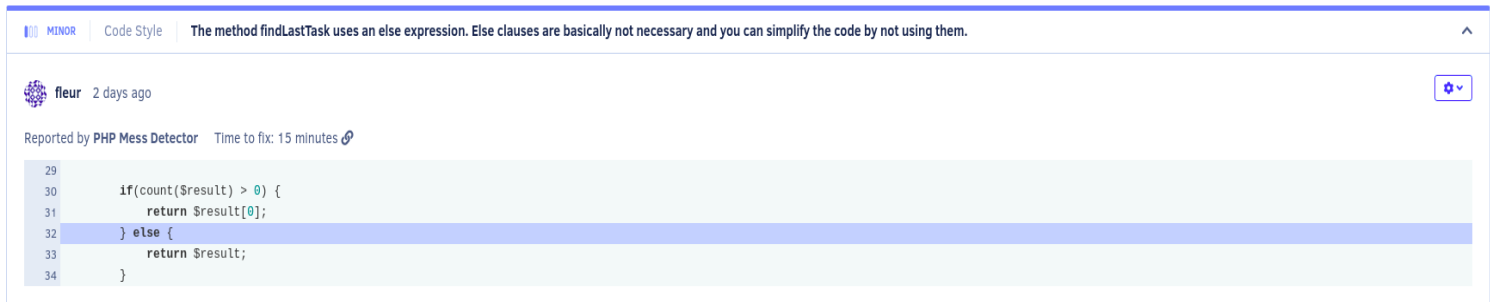


Note du projet retravaillé :

capture d'écran note global du projet retravaillé

L'analyse du code avec Codacy m'a permis de corriger quelques éléments :

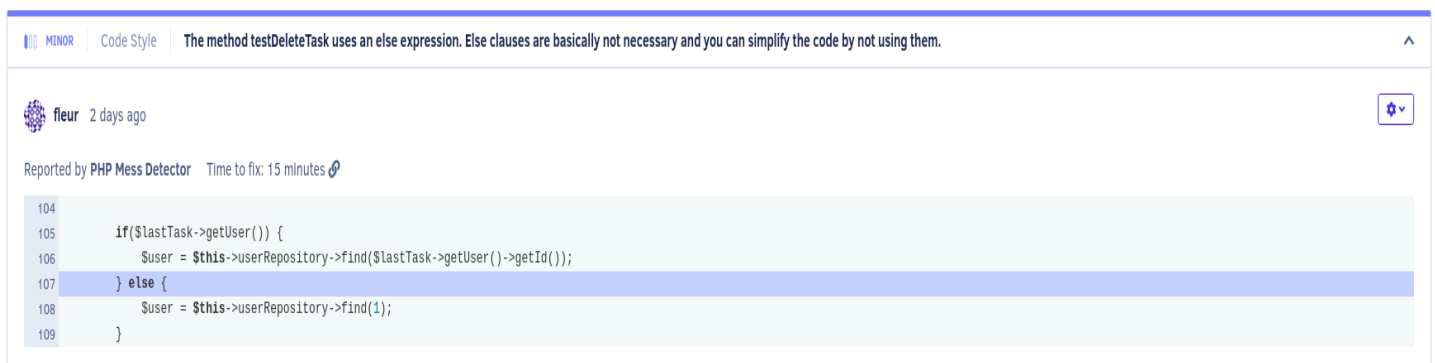
app/html/src/Repository/TaskRepository.php



correction apportée :

```
if(count($result) > 0) { return $result [0]; }  
return $result;
```

app/html/tests/Controller/TaskControllerTest.php



correction apportée :

```
$user = $lastTask->getUser() ?  
$this->userRepository->find($lastTask->getUser()->getId()) :  
$this->userRepository->find(1);
```

app/html/tests/Controller/UserControllerTest.php



correction apportée :

```
$this->client->request('GET', '/users/' . $userDelete->getId() . '/delete');
```

J'ai enlevé \$crawler dans les méthodes où je n'en avais pas besoin.

D'autres anomalies proviennent du code généré par Symfony ou du projet initial. Aucune correction n'a été réalisée. Les fichiers ont été ignorés.

4/ Qualité de performance

Ressource : <https://www.youtube.com/watch?v=aiDo9Ku93ew>

Pour réaliser les tests de performance, nous utilisons Blackfire.

C'est un outil de profilage PHP, développé par SensioLabs.

Il est utile lorsque nous voulons avoir des métriques notamment sur les performances de notre application développée en Symfony.

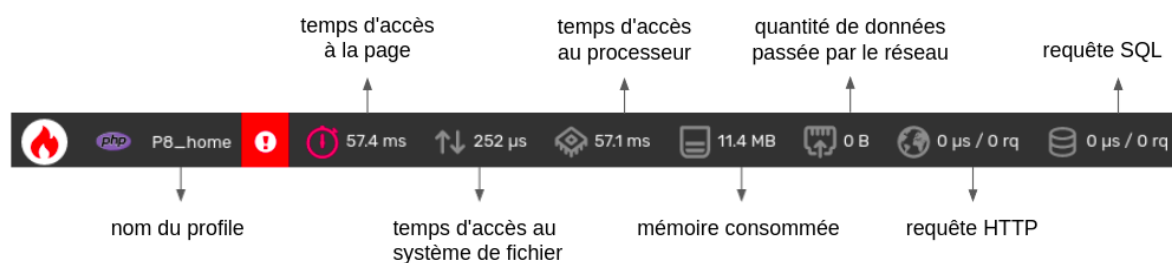
Il va permettre de faire une analyse sur :

- le temps d'accès à la page,
- le temps d'accès au système de fichier,
- le temps d'accès au processeur,
- la mémoire consommée,
- etc...

Ainsi, il nous donne des indications précises.


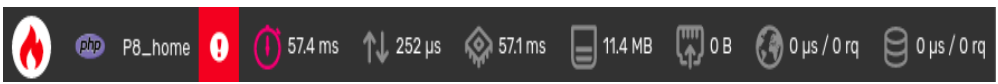
Pour réaliser les tests de performances, la version premium est disponible gratuitement pendant 15 jours.

Grâce au mode premium, voici les différentes métriques permettant l'analyse de nos performances de notre application :



Principales actions à mener pour la performance de l'application

Annexe : P8_06_annexe profils détaillés blackfire

Environnement	Métriques Blackfire de la page d'accueil
Développement	
Production	

Les principales recommandations de blackfire ont été :

1/ Passer les annotations en cache en mode production :

- mise en cache des annotations dans config_prod.yml,
- modification de APP_ENV=dev en APP_ENV=prod dans routes.yml,
- vider le cache avec php bin/console cache:clear.

2/ Le mode production désactive le debug.

5/ Conclusion

Notre travail sur l'application ToDo & Co a permis de :

- corriger les anomalies,
- mettre à jour Symfony,
- mettre en place de nouvelles fonctionnalités,
- mettre en place une documentation de collaboration sur le projet,
- implémenter des tests unitaires et fonctionnels,
- améliorer la performance de l'application.

Le but est d'obtenir les bases d'un code propre, solide et compréhensif.

Ce travail n'est qu'un début.

Des axes d'améliorations ont été proposés, permettant l'évolution de l'application.

Un retour des utilisateurs permettra d'apporter de nouvelles améliorations afin de rendre l'application plus fonctionnelle et performante.