TODO & CO

PLAN TO BE PRODUCTIVE

Audit de qualité du code et performance

Par Clément Thuet

Sommaire

Introduction

Ce document présente les différences d'un point de vu de la qualité du code et de la performance de l'application entre la version initiale de Todo&Co's to-do list (le MVP) et la version actuelle.

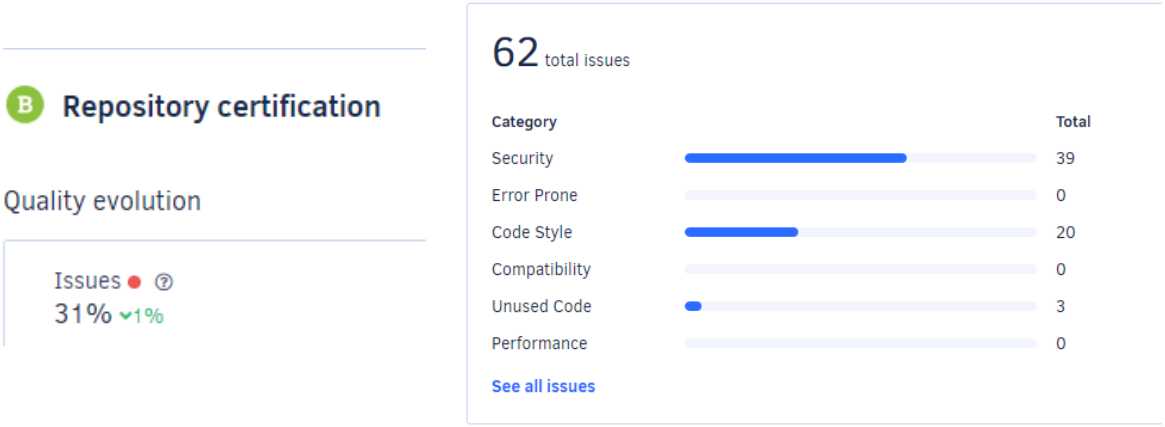
L'objectif lors de l'évolution du projet est de maintenir un niveau de qualité et de performance au moins égal à ce qui se fait actuellement.

Codacy est utilisé pour l'analyse des erreurs de style dans le code, les portions inutilisées, les incompatibilités et les problèmes de sécurité.

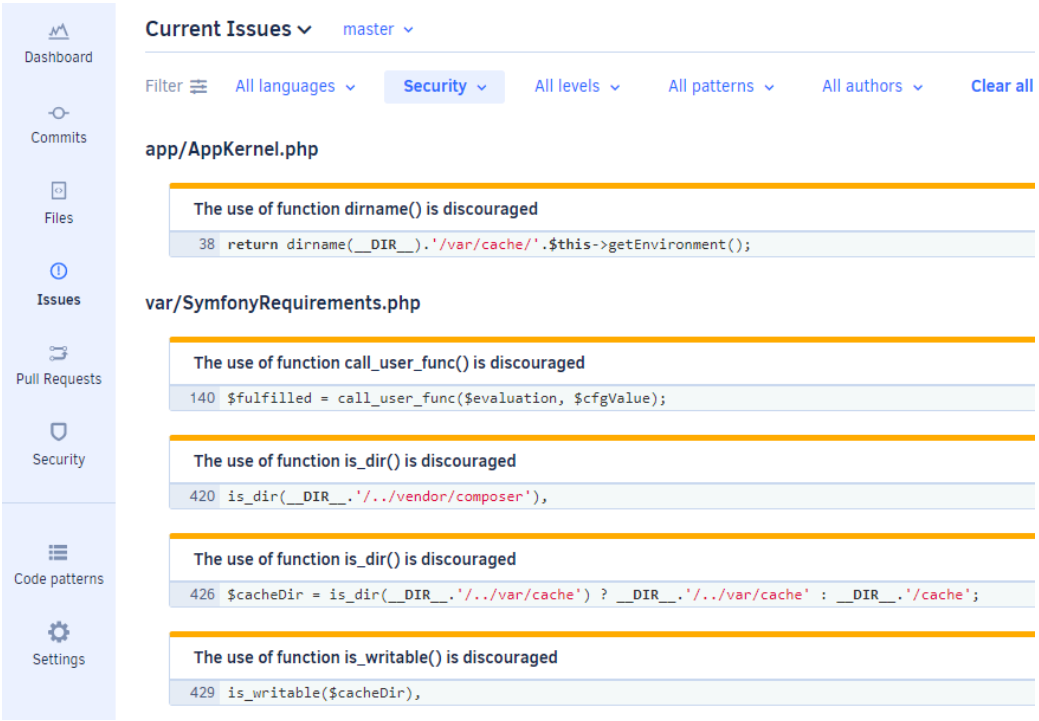
Pour étudier la performance l'outil Blackfire a été mis en place, il permet notamment d'analyser le temps d'exécution de chaque tâche et la mémoire utilisée.

I - Qualité du code

Version initiale :

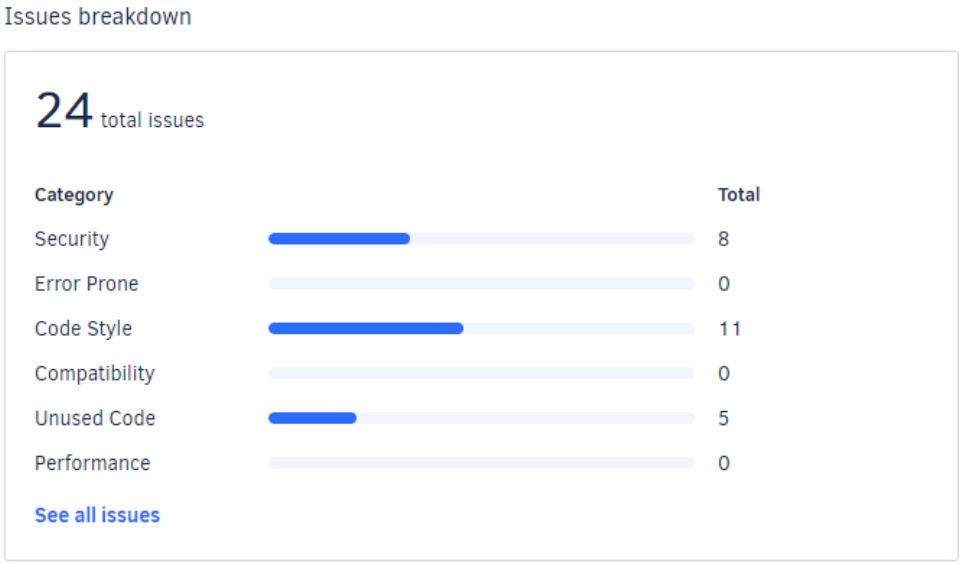


L'application obtenait la note respectable de B avec 62 problèmes détectés, principalement des erreurs de sécurités liées à l'usage de fonction dont l'utilisation est décourageé dans les fichiers de Symfony.



Rien de gênant n'était à déplorer, l'objectif était donc de ne pas introduire de nouvelles erreurs tout en supprimant celles déjà existantes si possible.

Après refonte :



L'application obtient toujours la note de B mais le nombre d'erreur a été pratiquement divisé par 3. Les problèmes restants proviennent majoritairement de point syntaxique discutable comme des noms de variables tels que « $id » ou « $em » jugés trop court alors qu'ils sont compréhensibles.

Les différentes analyses sont accessibles à cette adresse : *https://app.codacy.com/manual/ClementThuet/ToDo-Co/dashboard?bid=17117717*

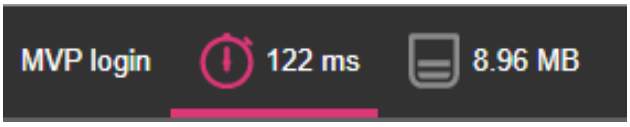
II - Performance

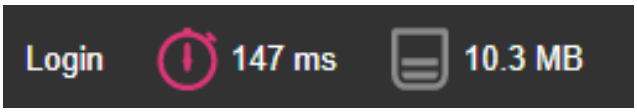
Afin de comparer l'impact des modifications sur le projet et particulièrement le changement de version de Symfony, j'ai utilisé l'outil Blackfire pour analyser différentes actions.

Le nombre de gauche correspond au temps d'exécutions en milliseconde et le nombre de droite à la mémoire utilisée pour effectuer l'opération.

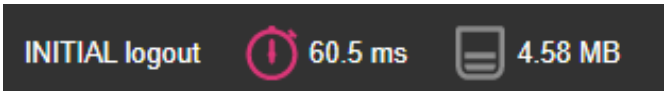
Voici le comparatif avec à gauche l'ancienne version (MVP) et à la droite la nouvelle :

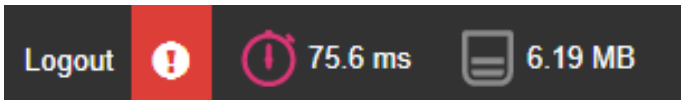
La connexion (-20,4%)



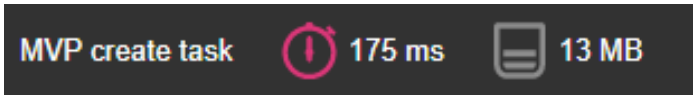


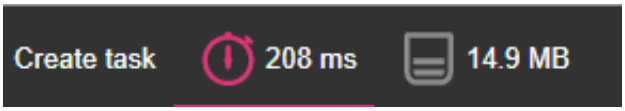
La déconnexion (-24,9 %)



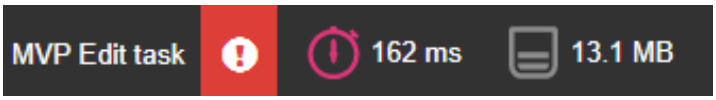


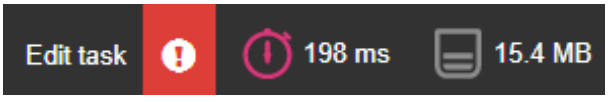
Création d'une tâche (-18,8%)



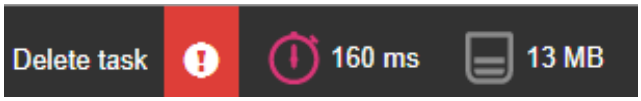


Edition d'une tâche (-22,2%)

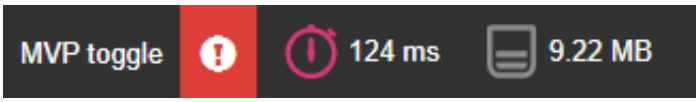


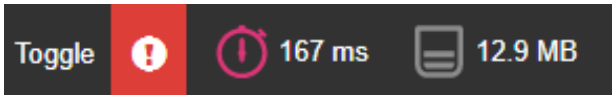


Suppression d'une tâche (-18,5%)

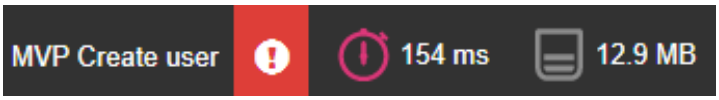


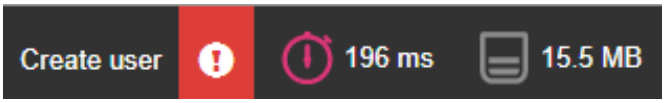
Marquer une tâche comme (non) terminée (-24,6%)



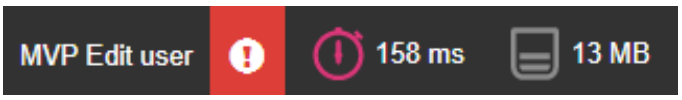


Création d'un utilisateur (-27,2%)





Edition d'un utilisateur (-29,1%)



On se rend compte d'une baisse de performance moyenne en terme de temps d'éxecution de 23,21% et l'augmentation de la mémoire utilisée est également proportionnelle.

D'après les analyses Blackfire cette diminution provient principalement de l'Event Dispatcher, le passage à Symfony 4.4 semble donc en être responsable mais de nombreuses fonctionnalité comme les voters ont été rajouté qui rajoutent aussi de la charge de travail au chargement de chaque page.
L'ajout de calcul afin de déterminer si l'utilisateur connecté a le droit de faire tel ou tel action afin d'afficher, ou non, les bouttons de navigation semble peser également au moment du rendu du template twig.

Ces performances seront donc à surveiller attentivement lors des prochaines mises à jour afin de vérifier qu'elles ne déteriorent pas sensiblement les performances.

III – Couverture de code

Dans l'objectif d'avoir une intégration le plus continue possible et d'éviter l'introduction de bug ce qui conduirait à une récession technique de l'application, il est important d'avoir de nombreux tests unitaires et fonctionnelles qui seront exécutés avant et après chaque mise à jour de l'application pour vérifier que tout fonctionne comme il est attendu.

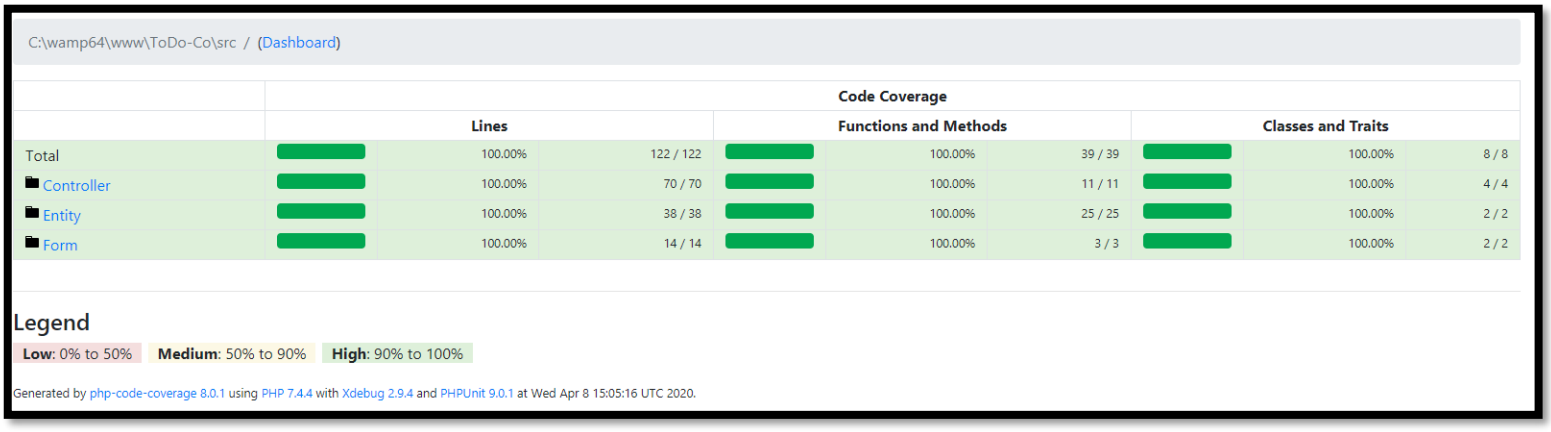
Le fonctionnement de ces tests est décrit dans le document récensant les bonnes pratiques pour contribuer au projet (disponible à cette adresse : *https://github.com/ClementThuet/ToDo-Co/blob/master/docs/contribute.md*)

Afin de s'assurer que le maximum de fonctionnalité fonctionnent correctement j'ai mis en place une analyse du code testé afin de déceler quelles parties du code n'étaient pas testé et corriger ces lacunes.

On utilise pour cela toujours PhpUnit et plus précisément la commande :
phpunit --coverage-html cover

Celle-ci génère un rapport de couverture de code dans le dossier « cover » à la racine du projet. (Disponible ici : *https://github.com/ClementThuet/ToDo-Co/blob/master/cover/index.html*)

Voici un aperçu du tableau de bord une fois les corrections effectuées :



On voit ensuite au survol de chaque fichier à quel endroit chaque ligne de code est testé.


```
24          */
25      public function listDoneAction()
26      {
27          $this->denyAccessUnlessGranted('seeTask');
28          $tasksDone = $this->getDoctrine()->getRepository('App:Task')->findBy(['isDone'=>true]);
29          return $this->render('task/list.html.twig', ['tasks' => $tasksDone]);
30      }
31
32      /**
33       * @Route("/tasks/create", name="task_create")
34       */
40          $form->handleRequest($request);
41
42          if ($form->isSubmitted() && $form->isValid()) {
43              $em = $this->getDoctrine()->getManager();
44              $user = $this->getUser();
45              $task->setUser($user);
46              $em->persist($task);
47              $em->flush();
48              $this->addFlash('success', 'La tâche a été bien été ajoutée.');
49
50              return $this->redirectToRoute('task_list');
51          }
52          return $this->render('task/create.html.twig', ['form' => $form->createView()]);
```

Le code coverage est passé d'aucun test écrit à 100% de couverture, il est primordiale de conserver un haut taux de couverture (>70%) afin de s'assurer de la pérénité de l'application.

IV – Amélioration effectuées

Afin de réduire la dette technique de l'application le principal changement a été le passage de la version 3.1.9 à la version 4.4.5 de Symfony. Pour cela il a fallu revoir totalement le chargement des différents modules ainsi que l'architecture des fichiers et la configuration.

Par exemple les fichiers des vues *.html.twig ont été déplacés de */app/Resources/views/* à */templates/* et les Controller de *src/AppBundle/Controller* à *src/Controller* afin de respecter l'architecture de Symfony 4.

Cela facilitera l'évolution future de l'application notamment au moment du passage à la version 5 lorsque cela sera opportun.

De nouvelles fonctionnalités ont été implantés. Il est désormais possible de choisir à la création d'un utilisateur les rôles qui lui sont attribués.

Quand un utilisateur crée une tâche celle-ci lui est associé ce qui n'étais pas le cas auparavant.

L'utilisateur connecté ne voit désormais plus que ses tâches, auparavant il avait accès aux tâches de tous les utilisateurs.

Une gestion plus fine des droits d'accès a été mise en place grâce aux voters, dorénavant seuls les utilisateurs avec le rôle admin peuvent traiter les tâches crées par l'utilisateur anonyme (concrètement, les tâches déjà existantes qui n'étaient liées à aucun utilisateur).

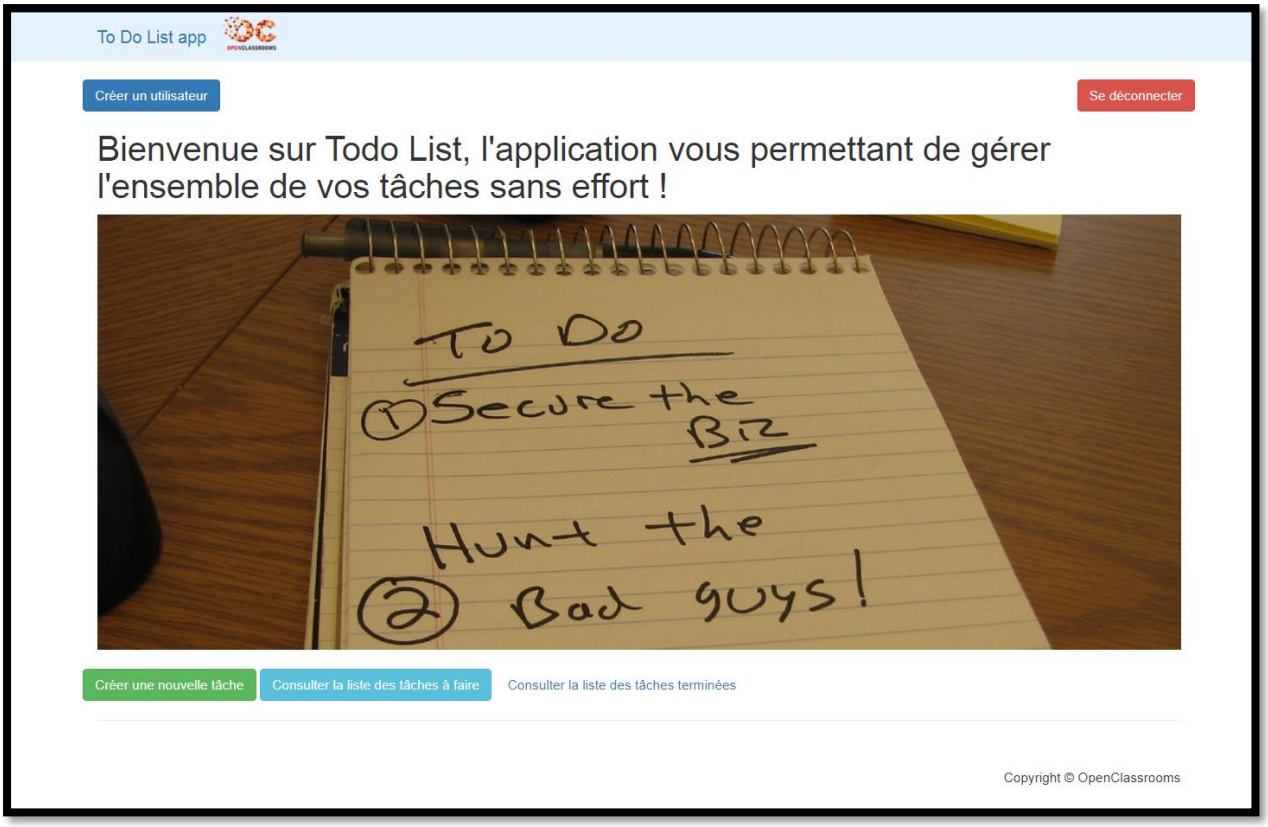
En termes d'interface les menus ont été aérés afin de rendre la navigation plus esthétique, l'image du menu principale a été changée au profit d'une proposition de logo.

Des titres <h1> décrivant le contenu de chaque page ont été mis en place.

Des boutons ont été rajoutés pour revenir en arrière ou au menu principale afin de faciliter la navigation.

L'affichage des liens pour réaliser les différentes actions comme modifier ou supprimer une tâche se fait désormais en respectant les droits de chaque type d'utilisateur.

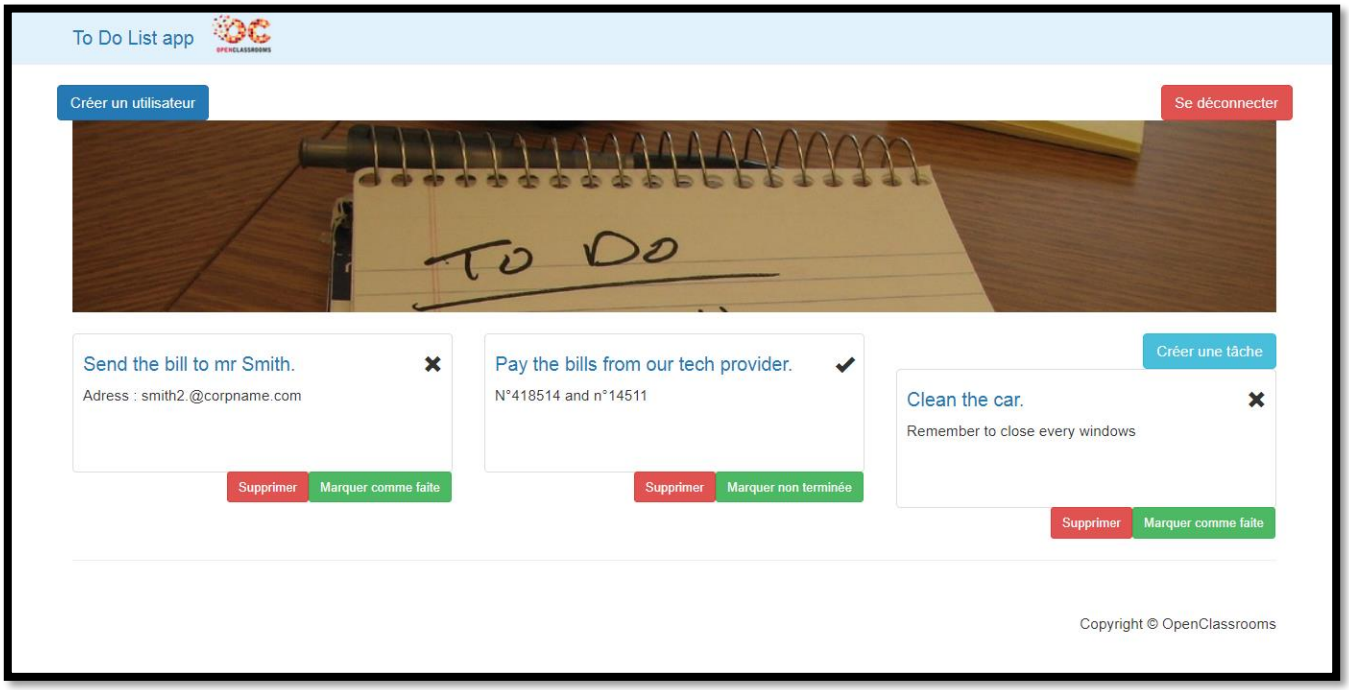
Avant :


To Do List app
Créer un utilisateur
Se déconnecter
Bienvenue sur Todo List, l'application vous permettant de gérer l'ensemble de vos tâches sans effort !
To Do
1 Secure the Biz
2 Hunt the Bad guys!
Créer une nouvelle tâche
Consulter la liste des tâches à faire
Consulter la liste des tâches terminées
Copyright © OpenClassrooms

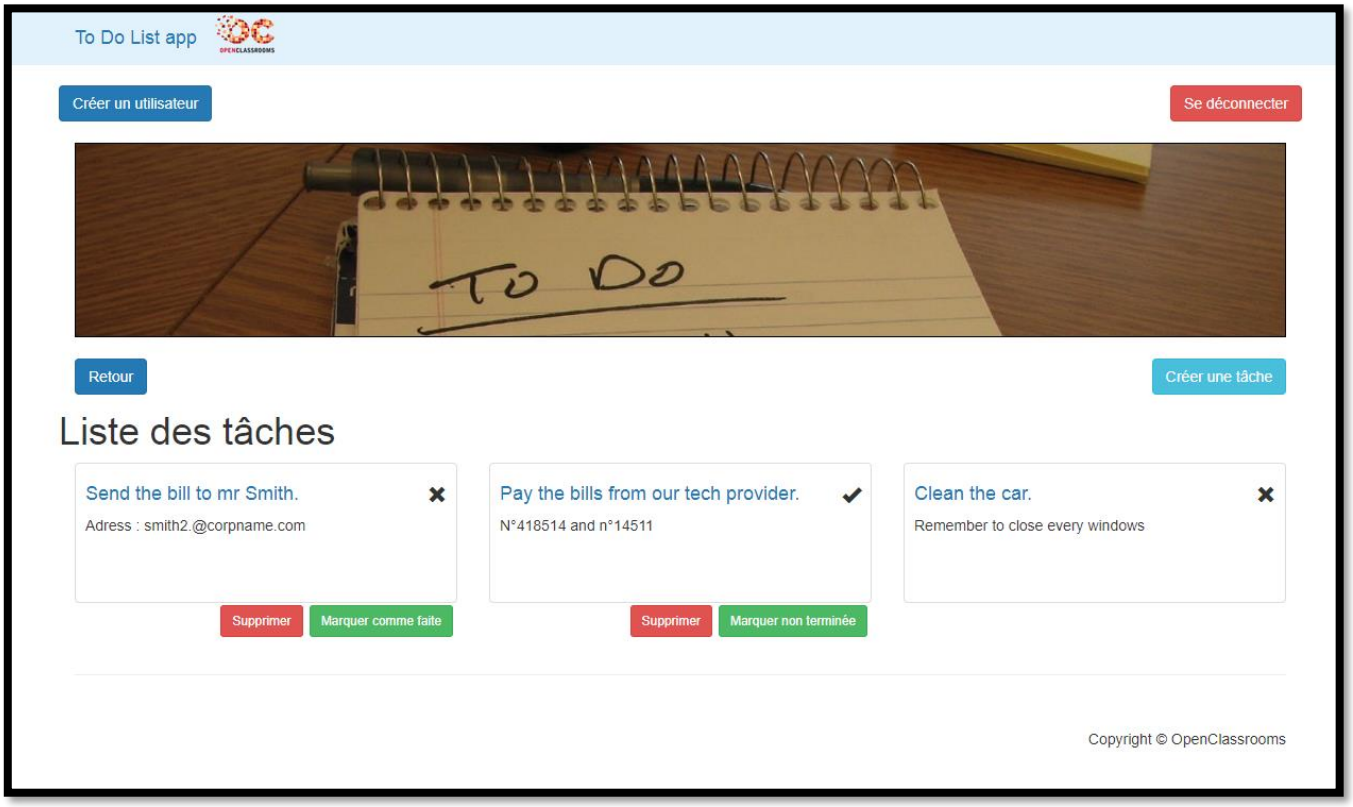
Après :


To Do List app
Créer un utilisateur
Se déconnecter
Bienvenue sur Todo List, l'application vous permettant de gérer l'ensemble de vos tâches sans effort !
T&C
TODO & CO
PLAN TO BE PRODUCTIVE
Créer une nouvelle tâche
Consulter la liste des tâches à faire
Consulter la liste des tâches terminées
Consulter la liste des utilisateurs
Copyright © OpenClassrooms

Avant :


To Do List app
Créer un utilisateur
Se déconnecter
Send the bill to mr Smith.
Adress : smith2.@corpname.com
Supprimer
Marquer comme faite
Pay the bills from our tech provider.
N°418514 and n°14511
Supprimer
Marquer non terminée
Créer une tâche
Clean the car.
Remember to close every windows
Supprimer
Marquer comme faite
Copyright © OpenClassrooms

Après :


To Do List app
Créer un utilisateur
Se déconnecter
Retour
Créer une tâche
Liste des tâches
Send the bill to mr Smith.
Adress : smith2.@corpname.com
Supprimer
Marquer comme faite
Pay the bills from our tech provider.
N°418514 and n°14511
Supprimer
Marquer non terminée
Clean the car.
Remember to close every windows
Copyright © OpenClassrooms

V – Axes d'améliorations

A – Performances

Afin d'améliorer les performances dans le cadre de la production il est possible de réaliser certaines actions :

Rassembler les Service Container en un seul fichier, par défaut Symfony compile les service container dans plusieurs petits fichier, il est possible de les compiler dans un seul fichier en modifiant le fichier services.yaml.

YAML XML PHP

```yaml
# config/services.yaml
parameters:
    # ...
    container.dumper.inline_factories: true
```

Configurer le realpath cache de PHP : Quand un chemin relatif est transformé en chemin absolu PHP le met en cache pour améliorer la performance, Symfony ouvre beaucoup de fichier c'est pourquoi il est recommandé d'utiliser au minimum cette configuration du php.ini.

```ini
; php.ini
; maximum memory allocated to store the results
realpath_cache_size=4096K

; save the results for 10 minutes (600 seconds)
realpath_cache_ttl=600
```

Optimiser l'autoloader de composer : Lors du développement le class loader est optimisé pour trouver les nouvelles classes, en production nous n'en avons pas besoin c'est pour cela qu'il est intéressant de de construire une « class map », un tableau stockant la localisation de toutes les classes et se situant dans `vendor/composer/autoload_classmap.php`
Pour cela il faut exécuter :

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

De manière générale il faut faire attention à utiliser des fichiers minifié lorsque cela est possible, pour le CSS et le JS par exemple, et à utiliser des images les plus légères possible.

B – Fonctionnalités

L'application compte pour le moment assez peu de fonctionnalités, on pourrait imaginer dans le cadre de l'évolution de l'application l'ajout d'options comme la priorisation des tâches ou encore la possibilité de leur assigner des catégories et de les organiser par dossier.

Ou encore mettre en place un calendrier afin d'associer les tâches à une date.

Il pourrait également être intéressant d'intégrer des outils de mise en forme pour les utilisateurs qui souhaitent avoir des listes à puces ou veulent surligner/souligner du texte.

C – Méthodologie

Pour aller plus loin dans la pérennisation de l'application il serait judicieux de mettre en place de l'intégration continue.

Pour cela il faudrait rédiger des tests unitaires et fonctionnelles pour chaque fonctionnalité (et pourquoi pas adopter une approche de Test Driven Development) et vérifier avant chaque merge sur la branche master que les tous les tests sont couronnés des succès.

Il faudrait également s'assurer que Codacy ne détecte pas de nouvelles erreurs, le cas échéant les corriger avant de valider les changements.

On pourrait également réfléchir à la mise en place d'autres outils comme phpDocumentor pour avoir une documentation automatique du projet.