Audit de qualité et performance : *ToDoList*

Date : 2021-01-16

Version : 1.0

Objectif : Effectuer un diagnostic complet de l'application ToDoList tant sur la qualité que la performance de cette dernière.

Sommaire :

1. Cadre du projet

a. Contexte

L'entreprise **ToDo & Co** vient tout juste d'être montée, et l'application **ToDoList** (une application permettant de gérer ses tâches quotidiennes) a dû être développée à toute vitesse pour permettre de montrer à de potentiels investisseurs que le concept est viable (on parle de Minimum Viable Product ou MVP).

Le choix du développeur précédent a été d'utiliser le Framework PHP Symfony. Lorsqu'il a initialisé le projet, la version du projet s'est avérée être la version 3.1.

L'entreprise **ToDo & Co** a enfin réussi à lever des fonds pour permettre le développement de l'entreprise et surtout de l'application.

b. Objectifs

L'objectif de la mission qui m'a été confié se décompose en **5 grandes parties** :

- **1/ La correction de quelques anomalies déjà répertoriées :**

    - **Une tâche doit être attachée à un utilisateur :**

      Actuellement, lorsqu'une tâche est créée, elle n'est pas rattachée à un utilisateur. Il vous est demandé d'apporter les corrections nécessaires afin qu'automatiquement, à la sauvegarde de la tâche, l'utilisateur authentifié soit rattaché à la tâche nouvellement créée.

      Lors de la modification de la tâche, l'auteur ne peut pas être modifié.

      Pour les tâches déjà créées, il faut qu'elles soient rattachées à un utilisateur "anonyme".

    - **Choisir un rôle pour un utilisateur :**

      Lors de la création d'un utilisateur, il doit être possible de choisir un rôle pour celui-ci. Les rôles listés sont les suivants :

      Rôle utilisateur (*ROLE_USER*) ;
      Rôle administrateur (*ROLE_ADMIN*).

Lors de la modification d'un utilisateur, il est également possible de changer le rôle d'un utilisateur.

- **2/ Implémenter de nouvelles fonctionnalités :**

  - **Gestion des autorisations**

    Seuls les utilisateurs ayant le rôle administrateur (*ROLE_ADMIN*) doivent pouvoir accéder aux pages de gestion des utilisateurs.

    Les tâches ne peuvent être supprimées que par les utilisateurs ayant créé les tâches en question.

    Les tâches rattachées à l'utilisateur "anonyme" peuvent être supprimées uniquement par les utilisateurs ayant le rôle administrateur (*ROLE_ADMIN*).

- **3/ Implémentation de tests automatisés**

  - **Tests automatisés**

  Rédiger une batterie de tests unitaires et fonctionnels permettant d'assurer le bon fonctionnement de l'application et faciliter la maintenance de cette dernière.

  Ces tests devront être réalisés avec l'aide de PHPUnit.

  Un jeu de données sera nécessaire pour effectuer les tests. Il faudra donc également prévoir des fixtures au sein du projet.

  Un rapport de couverture de code est également réclamé avec un taux qui devra atteindre à minima 70%.

- **4/ Rédiger des documents techniques**

  - **Fichier README.md**

    Permet d'expliquer comment installer le projet.

  - **Fichier CONTRIBUTING.md**

    Permet d'expliquer comment participer et contribuer à l'évolution de l'application en respectant certaines règles afin de s'assurer de la qualité du code réalisé et aux respects des bonnes pratiques en matière de développement.

  - **Documentation Technique sur l'Authentification**

Ce document a pour objectif d'expliquer à de futurs développeurs comment le système d'authentification fonctionne au sein du projet et doit leur permettre de devenir autonome sur le sujet.

- **5/ Audit de qualité du code et performance de l'application**

L'objectif est de réaliser une étude permettant d'effectuer un état des lieux de la dette technique de l'application. L'idée est d'analyser cette dernière tant sur la partie qualité que sur la partie performance. Cela permettra d'établir un plan d'action pour les évolutions futures de cette application.

2. Application initiale

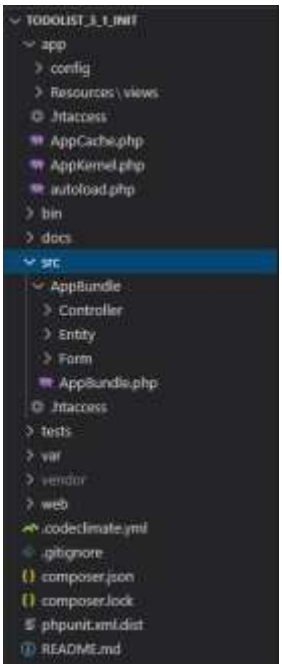
a. Architecture du code, Langage, Framework, BDD :

L'application ToDoList a été développée en PHP utilisant les principes de la **Programmation Orienté Objet** (POO) avec le **Framework Symfony**. La version dans laquelle nous avons initialisé le projet est **la version 3.1 du Framework.**

L'ensemble des vues de l'application utilise le langage Twig.

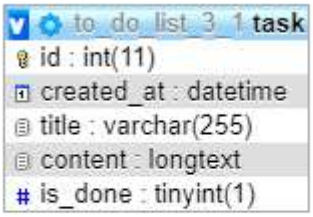
Le **Framework CSS Bootstrap** a été utilisé pour gérer la partie Front de l'application.

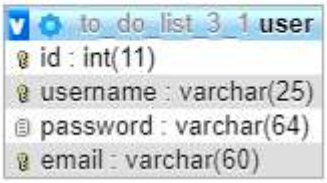
La structure du code respecte une architecture dite **« MVC »** avec néanmoins un bémol à soulever puisque le code métier de l'application a été rédigé directement dans les « Controller ».



La structure de la base de données actuelle est la suivante :

Nous constatons qu'aucune relation existe entre la table « user » et la table « task ». C'est un point sur lequel il faudra remédier dans la version finale de l'application comme stipulé dans les objectifs définis plus haut.

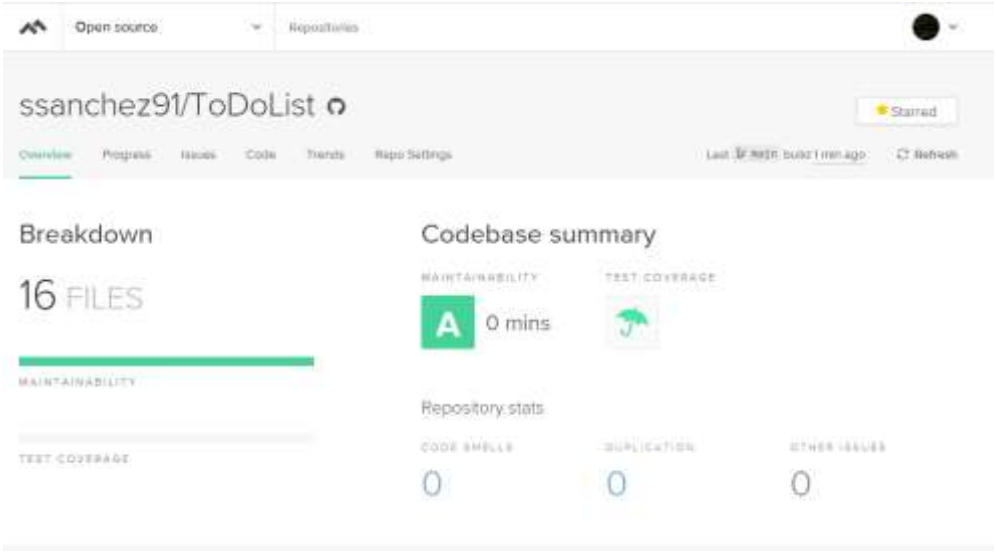




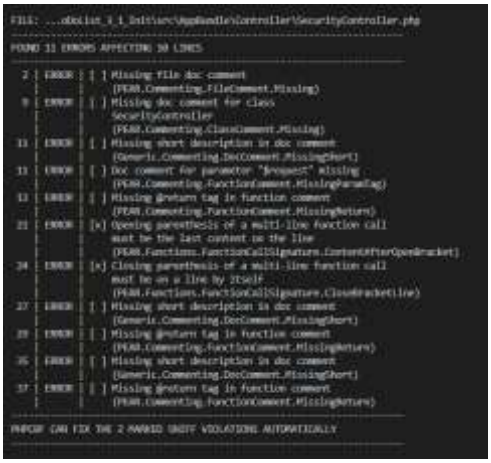
b. Qualité du code

Après avoir cloné le projet, une analyse a été réalisée avec l'aide de CodeClimate.

Le résultat de cette analyse montre que la qualité du code est très bonne puisque nous avons obtenue la note de A. (A noté que seul le code métier a été analysé.)



Afin de contrôler au mieux le respect des règles PSR une analyse PHPCS a été réalisée sur l'environnement local de développement. Nous constatons que de nombreux problèmes sont présents. En voici quelques exemples.



c. Analyse Fonctionnelle

Afin d'orienter au mieux notre analyse j'ai commencé par lister les routes existantes au sein du projet : la commande **php bin/console debug:router** nous a permis de lister l'ensemble des routes existantes.

- En nous rendant sur la page d'accueil **(url : /)** du site nous constatons, une redirection vers la page de login. Arrivé sur cette page il est étrange de voire apparaître le bouton permettant de créer un utilisateur. Il s'agira en effet de corriger cela dans la version finale.

- En allant consulter l'url **/users** force est de constater que n'importe quel visiteur peut consulter la liste des utilisateurs déjà présents.

- Nous constatons également qu'un visiteur peut modifier un utilisateur déjà présent et pire encore il peut à travers la page d'édition modifier son mot passe. Il faudra bien évidemment corriger ce bug dans la version finale.

- En nous rendant sur l'url **/tasks** nous constatons qu'une redirection à lieu vers la page de login, ce qui semble normal si l'on consulte la configuration du firewall actuel :

```
access_control:
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/users, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/, roles: ROLE_USER }
```

- Une fois connecté, nous tentons de nous rendre sur la page **/tasks**. Le résultat est à peu près cohérent nous accédons à la liste des tâches qui nous indique qu'aucune tâche n'est encore présente.

  En revanche il y a deux boutons pour créer une tâche côte à côte ce qui semble inutile.

  La fonctionnalité permettant de créer une tâche semble fonctionner mais n'affecte pas la tâche à l'utilisateur courant.

La suppression de la tâche fonctionne tout comme l'option d'indiquer qu'une tâche a été réalisée.

Dernier point à soulever concernant les tâches, lorsque nous nous sommes connectés, nous avons constaté deux boutons : un pour être redirigé vers la liste des tâches à effectuer et un autre pour lister les tâches déjà traitées.

Il semble que le second bouton ne fonctionne pas. De plus lorsque nous cliquons sur le premier bouton nous avons accès à une liste exhaustive des tâches et non à la liste des tâches dites « à réaliser » comme le laisse présager l'intitulé du bouton.

- De manière générale les menus et les formulaires dont à revoir afin d'obtenir un résultat plus pertinent et davantage fonctionnel pour l'utilisateur.

- Le bouton Se déconnecter fonctionne correctement.

- Dernier point constaté : un problème lié au chargement du fichier jquery.js sur l'ensemble des pages du site (Identifié en ouvrant la console du navigateur).

d. Tests unitaires et fonctionnels

En exécutant la commande suivante : **php bin/phpunit --coverage-html public/test** nous exécutons les tests déjà présents dans l'application et nous générons un rapport de couverture de test du code.

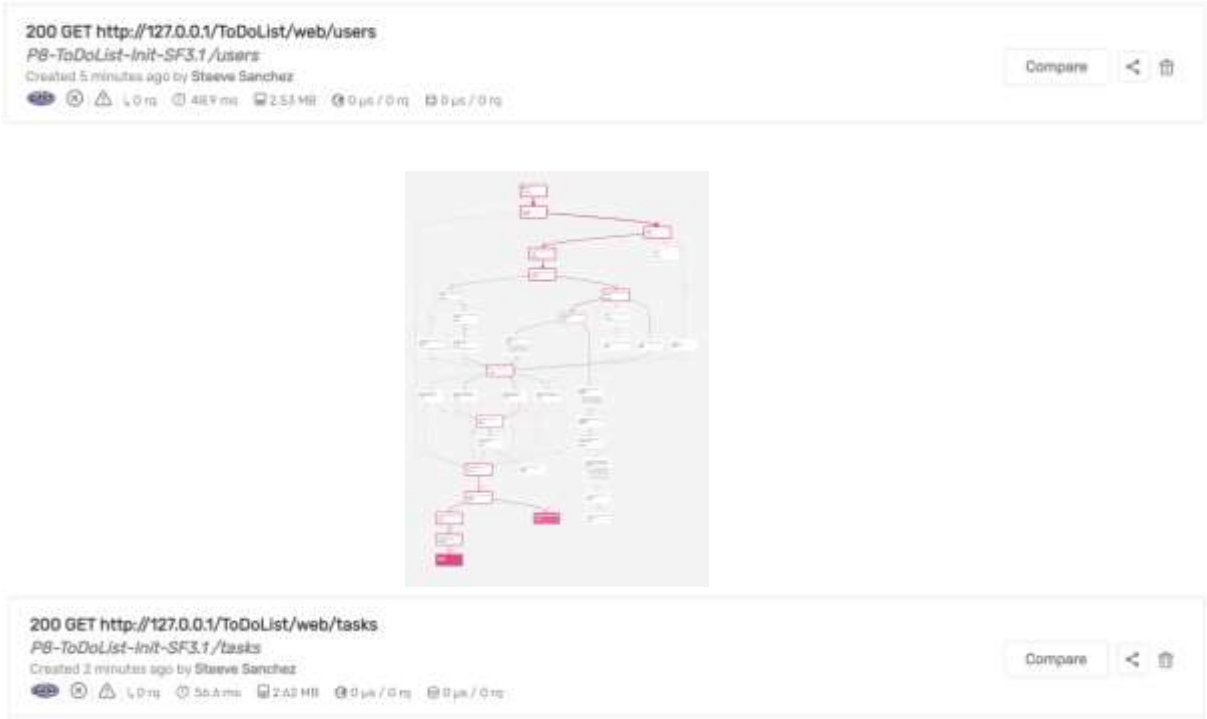
A ce stade nous pouvons en déduire **qu'aucun test n'est présent** dans l'application. Il s'agira donc bien évidemment de pallier à cela comme évoqué dans les objectifs de notre mission.

e. Tests de performance

Des tests de performances ont été réalisés à l'aide de l'outil Blackfire.

J'ai pris la décision d'analyser deux routes :

- /tasks
- /users



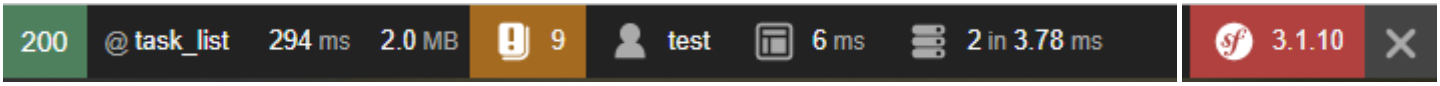
A notre niveau, Blackfire nous sera utile pour étudier deux critères sur le projet : La consommation de mémoire et le temps d'accès aux pages. Les graphiques nous permettent également d'aller chercher en profondeur les classes et méthodes qui consomme le plus de ressources afin d'optimiser les performances.

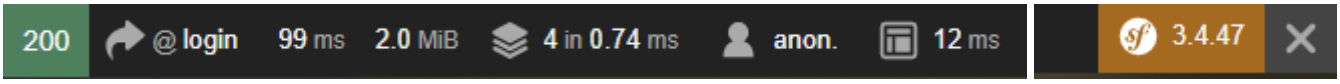
A ce stade nous constatons que les temps de réponses sont très convenables et que la consommation de mémoire est très faible. L'idée est donc de conserver cette stabilité et ce niveau de performance une fois l'application migrée et modifiée en fonction des objectifs demandés.

3. Application Finale

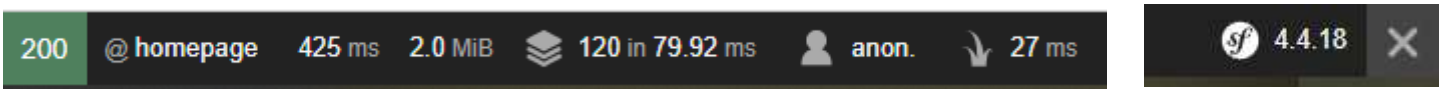
a. Architecture du code, Langage, Framework, BDD :

L'application a subi plusieurs modifications : la première a été de migrer l'application d'une version 3.1 du Framework Symfony à une version 3.4. Pour se faire, l'ensemble des déprécations présentes à l'origine du projet ont été corrigées :





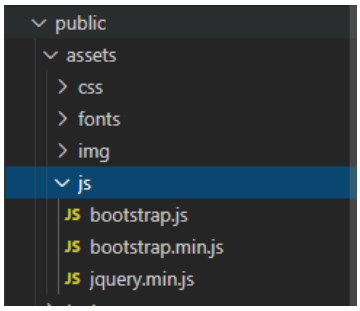
Dans un second temps nous avons migré l'application vers la dernière version LTS connu à jour dans un soucis de maintenanbilité optimal. En effet il est plus sûr de poursuivre les développements sur la dernière version stable connue qui sera maintenue le plus longtemps.



En plus d'effectuer la migration vers la version 4.4 du Framework il a fallu également basculer sous Symfony Flex pour bénéficier de la nouvelle architechture proposée par Symfony.

La nouvelle arborescence du code sous Symfony Flex respect également le principe MVC. En voici le détail ci-dessous :

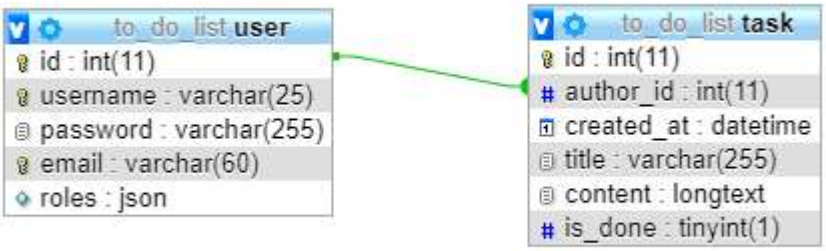
Lors de notre analyse initiale nous avions noté un problème lié à JQuery, ce dernier a été résolu en mettant en place le fichier manquant :



Côté base de données plusieurs évolutions à noter :

La première et non des moindres, est qu'une relation de type un à plusieurs à été mise en place entre la table **« user »** et la table **« task »** pour répondre aux besoins suivants :

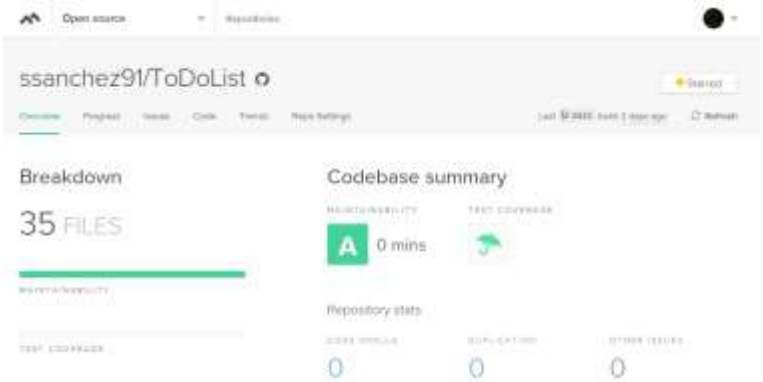
Une tâche nouvellement créée doit être affectée à l'utilisateur courant.



Une seconde évolution est apparue : l'ajout d'un champ « **roles** » afin d'y stocker les rôles d'un utilisateur et ce pour répondre aux besoins suivants :

- Voir besoins répertoriés dans la section objectifs plus haut.
  (1. Cadre du projet => b. **Objectifs** => 2/ ***implémenter de nouvelles fonctionnalités***)

b. Qualité du code



Nous pouvons constater qu'après avoir apporté l'ensemble des corrections demandées et après avoir ajouté les nouvelles fonctionnalités, nous avons réussi à conserver une note de A sur l'Analyse CodeClimate, et ce malgré un nombre de fichier a testé bien supérieur à l'état initial du projet.

A l'état initial du projet nous avions relevé un grand nombre d'erreur de warning à l'aide de PHPCS. L'idée était de réduire au maximum la dette technique sur les contraintes PSR. C'est chose faite puisque qu'après modification de l'ensemble du code, il ne reste plus qu'un seul warning :

```
PS C:\wamp64\www\ToDoList> .\vendor\squizlabs\php_codesniffer\bin\phpcs -s .\src\

FILE: src\Security\AppAuthenticator.php
----------------------------------------------------------------------
FOUND 0 ERRORS AND 1 WARNING AFFECTING 1 LINE
----------------------------------------------------------------------
 34 | WARNING | Line exceeds 120 characters; contains 199
    |         | characters (Generic.Files.LineLength.TooLong)
----------------------------------------------------------------------

Time: 168ms; Memory: 2MB
```

Autre point : le nom des méthodes dans les « Controller » a été changé afin de respecter les conventions. (Suppression du mot Action dans le nom des méthodes).

Afin d'éviter trop de code dans les « Controller », un **eventListener** a été créé pour gérer la partie encodage du mot de passe d'un utilisateur. Cette partie était réalisée directement dans le **UserController** à l'état initial du projet elle est maintenant gérée sur deux évènements :

« **prePersit** » et « **preUpdate** »

c. Analyse Fonctionnelle

```
------------ ---------- -------- ------ --------------------
 Name         Method     Scheme   Host   Path
------------ ---------- -------- ------ --------------------
 homepage     ANY        ANY      ANY    /
 login        ANY        ANY      ANY    /login
 logout       ANY        ANY      ANY    /logout
 task_list    ANY        ANY      ANY    /tasks
 task_create  ANY        ANY      ANY    /tasks/create
 task_edit    GET|POST   ANY      ANY    /tasks/{id}/edit
 task_toggle  ANY        ANY      ANY    /tasks/{id}/toggle
 task_delete  ANY        ANY      ANY    /tasks/{id}/delete
 user_list    ANY        ANY      ANY    /users
 user_create  ANY        ANY      ANY    /users/create
 user_edit    ANY        ANY      ANY    /users/{id}/edit
------------ ---------- -------- ------ --------------------
```

Côté fonctionnel, je me suis intéressé aux mêmes routes qu'à l'état initial pour tenter de corriger au maximum les incohérences :

En me rendant sur l'url « **/** » j'arrive bien sur la HomePage, et je ne suis plus redirigé vers la page de login comme dans le projet initial.

En allant consulter l'url **/users** une redirection s'opère vers la page de login. Une fois connecté avec **un simple utilisateur** nous tombons sur une page 403 FORBIDDEN « Access Denied » comme demandé dans les évolutions à mettre en place.

En nous connectant avec un **compte Administrateur** nous accédons correctement au listing des utilisateurs.

En effet nous avons respecter la demande et mis en place un contrôle d'accès sur la partie users à l'aide du firewall :

```
access_control:
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/users, roles: ROLE_ADMIN }
    - { path: ^/tasks, roles: ROLE_USER }
    - { path: ^/, roles: IS_AUTHENTICATED_ANONYMOUSLY }
```

L'édition d'un utilisateur devient donc également uniquement accessible à un administrateur comme demandé.

La gestion des rôles a été ajoutée et est disponible dans le formulaire d'ajout d'utilisateur ainsi que sur le formulaire d'édition.

L'utilisation d'un DataTransformer côté UserType a été mise en place.

Côté gestion **des tâches** de nombreuses choses ont été réalisées :

La partie dédiée à la gestion des tâches est toujours sous protection du firewall mais de nouvelle règles et fonctionnalités ont été ajoutées :

Tout d'abord l'ajout d'une tâche rattache automatique cette dernière à l'utilisateur courant (Une relation a été mise en place entre l'entité Task et l'entité User).

Un voter a été déployé pour gérer les autorisations de suppression d'une tâche. En effet désormais un utilisateur ayant le rôle ROLE_USER ne peut supprimer que ses tâches et se verra retourné une 403 FORBIDDEN en cas de tentatives sur une tâche ne lui appartenant pas. Autre point géré par le voter, une tâche déjà existante et affichée comme rattachée à un utilisateur anonyme ne peut être supprimée que par un utilisateur possédant le rôle ROLE_ADMIN.

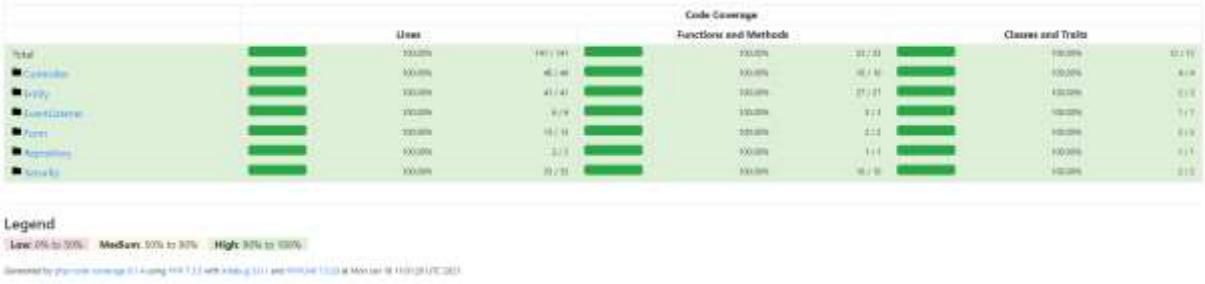
Le bouton proposant de lister les tâches déjà réalisées a été supprimé. Le bouton proposant de lister les tâches en cours a été renommé.

Cela n'est clairement pas suffisant mais dans l'état actuel et par soucis de suivre les consignes demandées j'ai pris la décision de gérer cela ainsi. Nous en reparlerons après l'analyse finale dans le plan d'amélioration qui sera établi.

d. Tests unitaires et fonctionnels

Afin de répondre aux mieux aux exigences de qualité de la société ToDo & Co et par souci d'optimiser les futures évolutions de code de l'application (maintenance), l'ensemble des corrections ainsi que les nouvelles fonctionnalités demandées ont été codées en suivant le mode TDD (Test Drive Développement). Cela consiste à coder dans un premier temps un test qui permettra de valider le code métier à exécuter. Ainsi lors de futures modifications il suffira de rejouer la batterie de tests déjà présentes pour s'assurer que le code existant n'a subit aucune anomalie suite aux nouveau code rédigé.

Cela m'a permis d'arriver aux résultats suivants :



L'application actuelle n'étant pas très grosse j'ai pu établir une couverture du code par des tests à 100%. L'idée n'est évidemment pas de rester à 100% surtout si l'application devient beaucoup plus importante mais de garder une bonne couverture sur les parties importantes du code métier afin de faciliter la maintenance de l'application.

La mise en place d'un outil d'intégration Continu nous permettra par la suite de nous assurer qu'il n'y aura pas de problème lors des futures évolutions du code.

En effet un développeur décidé à contribuer à l'application ne pourra pas effectuer de merge sur la branche principal tant que la batterie de test n'aura pas été validé par Travis.

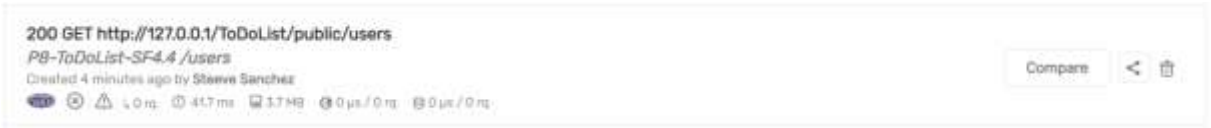
Lors d'une « Pull Request » le développeur attendra le résultat suivant :

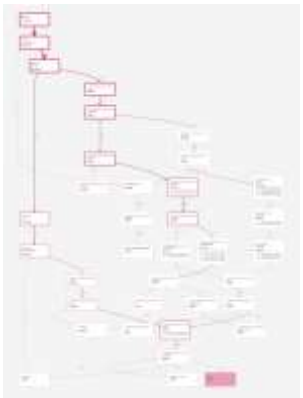
e. Tests de performance

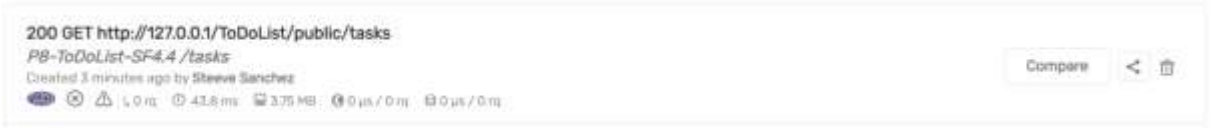
Une analyse similaire a été réalisée à l'aide de BlackFire pour nous assurer que les performances de l'applications n'ont été dégradées suites aux différentes modifications apportées.

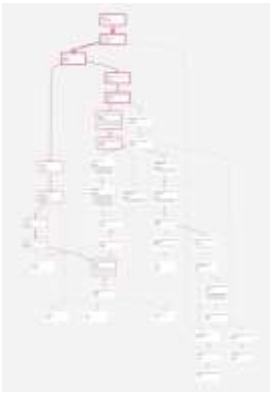
Les deux mêmes routes ont été testées :

- /tasks
- /users







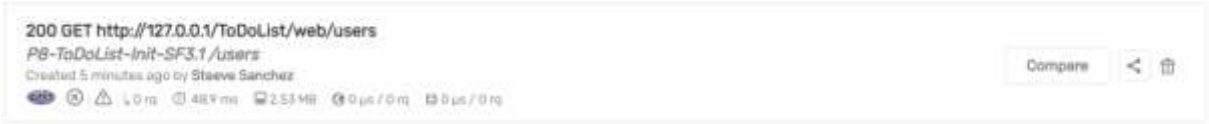


4. Analyse, Comparaison

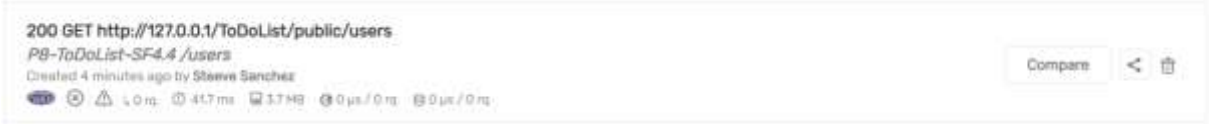
De nombreuses modifications ont été apportées que se soit d'un point de vue fonctionnelle ou architecture. Il faut donc nous assurer que ces dernières n'ont pas mis en péril les performances de notre application.

Pour rappel la première des routes analysée était /users

Version initiale du projet : avec la version ***Symfony 3.1***



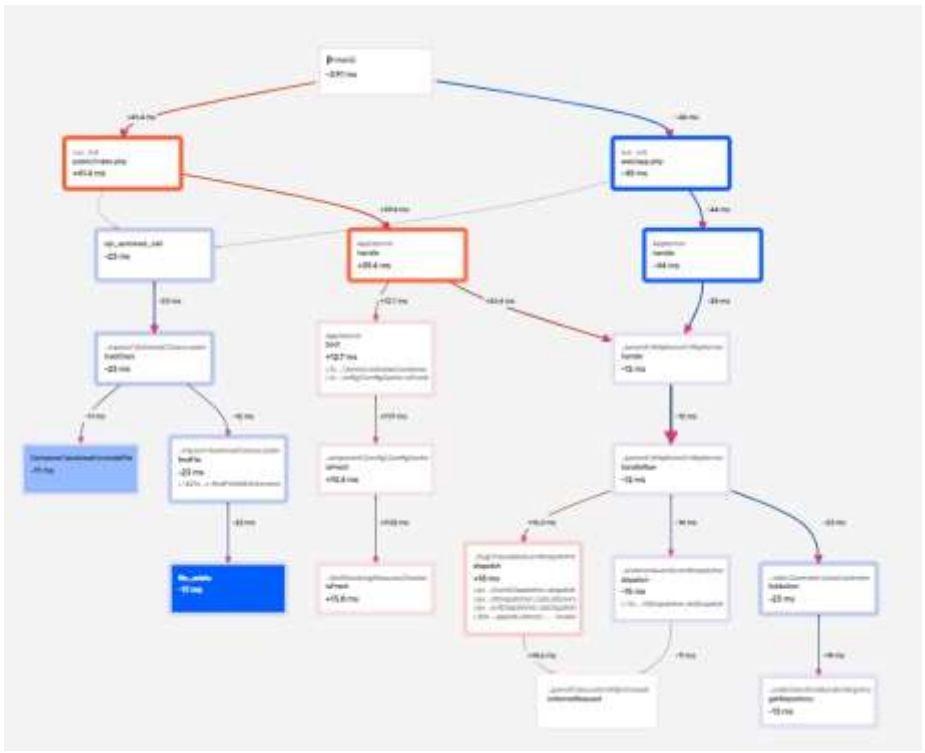
Version finale après modifications : avec la version ***Symfony 3.4***



Nous avions évoqué plus haut deux critères importants à contrôler :

- Le temps d'exécution et la mémoire utilisée.

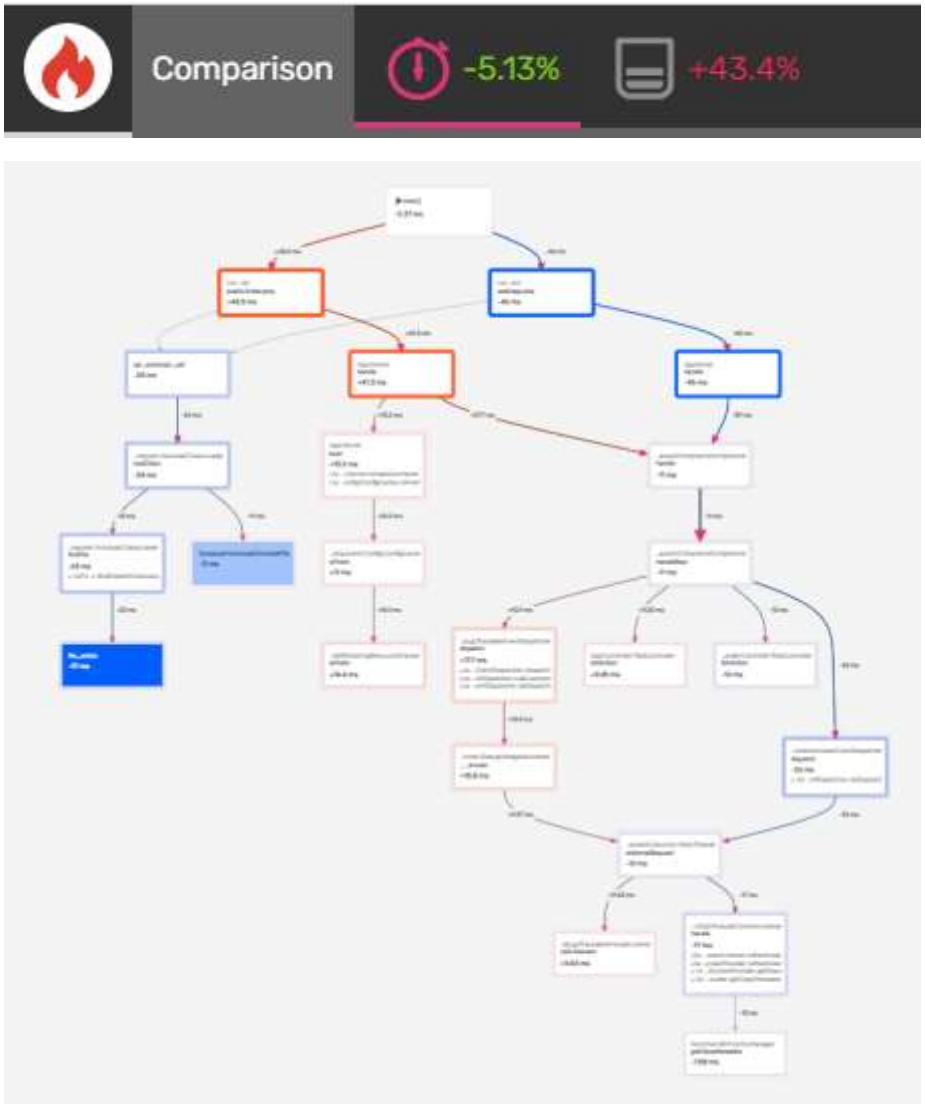
Voici le résultat d'un comparatif entre nos deux versions :



Nous constatons donc un gain de plus de 8% côté temps d'exécution, ce qui prouve que l'application reste de bonne qualité puisque nous avons même réussi à améliorer les temps de réponses. En revanche une forte augmentation côté consommation mémoire est a noté puisqu'elle a quasiment doublé.

Cela reste tout de même très positif dans l'ensemble puisque malgré le fait que la consommation de mémoire a augmenté elle reste tout de même très faible < 4 Mo et le temps d'affichage est excellent puisqu'il est < 42 ms.

En faisant le même exercice avec la route /tasks nous confirmons notre analyse réalisée ci-dessus avec les résultats suivants :



Il sera important de continuer à analyser notre application aux furs et à mesure des évolutions futures apportées à cette dernière, afin de conserver un niveau de qualité et de performance optimale.

De nombreuses évolutions dors et déjà identifiées pourront permettre d'optimiser le code de l'application, l'aspect fonctionnel et voire même gagner encore en performance.

Je terminerai dans la dernière section de cette analyse par lister les possibles améliorations afin de diminuer au maximum la dette technique de l'application.

5. Conclusion : plan d'amélioration

Pour conclure, cette analyse, voici une première liste non exhaustive des évolutions qui permettraient d'améliorer encore l'application ToDoList.

- Afin de minimiser au maximum le code dans les « Controller » et pour respecter au mieux les principes SOLID il serait bon de créer des Managers et ou des Services en fonction des besoins pour gérer le code métier.

- Un ExceptionListener pourrait être créé pour gérer l'affichage des différentes exceptions. Actuellement la page de Symfony par défaut s'affiche.

- Mettre en place un menu de navigation cohérent : actuellement lorsque nous arrivons sur la page d'accueil nous avons accès aux liens suivants :
    - Créer un utilisateur
    - Consulter la liste des utilisateurs
    - Créer une tâche
    - Consulter la liste des tâches

  Ces boutons ne devraient plus être présent et s'afficher en suivant les règles suivantes :

  Connecté en tant qu'utilisateur (ROLE_USER)
    - Afficher le bouton Créer une tâche
    - Afficher le bouton Lister les tâches

  Connecté en tant qu'administrateur (ROLE_ADMIN)

    - Afficher les 4 boutons

- Sur le formulaire de création d'utilisateur des contraintes de validations devraient être ajoutées pour sécuriser davantage l'application.
- Ce même formulaire en mode édition ne devrait pas demander de saisir de nouveau le mot de passe.

- Prévoir une méthode pour pouvoir supprimer un utilisateur.

- Coté gestion des tâches une pagination devrait être mise en place pour lister les tâches.

- Ajouter la possibilité de filtrer sur :

    - Les tâches déjà réalisées
    - Les tâches à réaliser

- De manière générale standardiser l'ensemble des boutons pour la navigation :
    - Bouton de retour
    - Bouton d'annulation
    - Bouton de validation

- Mettre en place un système de confirmation avant la suppression d'une ressource quelle qu'elle soit.

Cette liste bien que non exhaustive est déjà une première approche qui pourrait permettre de continuer à améliorer l'application.

Il sera important de continuer de rédiger les tests associés à ces nouvelles fonctionnalités afin de garder un niveau de couverture de code optimal et s'assurer que tout reste fonctionnel.

Pour rappel pour contribuer au projet vous pouvez consulter le fichier CONTRIBUTING.md pour vous aider à contribuer au projet.