Audit de qualité du code & performance de l'application

I. Table des matières

II. Contexte

Mon rôle pour ToDo & Co est d'améliorer la qualité de l'application et de définir les contours de la contribution de futurs développeurs.

L'application de ToDo & Co consiste en la création d'une liste de tâches que les utilisateurs peuvent créer, éditer, supprimer et marquer comme terminées.

III. Migration de la version de Symfony

Le projet est initialement en version 5.5.9 de PHP et 3.1 de Symfony. L'objectif est évidemment de mettre à jour le projet de manière à garantir sa sécurité et de le proposer sur la version de Symfony 7.0.3 avec PHP 8.2.9.

Maintenant que le projet est sur une version récente de Symfony, l'objectif est de le maintenir à jour.

Lors d'un changement de version majeur de Symfony, il faut passer l'ensemble du projet à la dernière version mineure de la version précédente de manière à voir l'ensemble des dépréciations de cette version pour les corriger avant de passer « sereinement » à la version supérieure. De cette manière nous n'avons pas de changements majeurs à apporter à notre code car une version majeure est quasiment équivalente à la version mineure précédente en ayant enlevé les dépréciations toujours présentes de la version mineure.

IV. Anomalies…

a) …à corriger

A la demande initiale, les anomalies repérées et à corriger sont :

- Une tâche doit être rattachée à un utilisateur à sa sauvegarde. (Les tâches déjà présentes doivent être rattachées à un utilisateur « anonyme ».
- Choisir un rôle pour un utilisateur à sa création. A la modification d'un utilisateur (par un admin) il doit être possible de changer le rôle d'un utilisateur.

b) …découvertes et corrigées

Outre les anomalies précédentes corrigées, j'ai également mis en place la création de l'utilisateur par lui-même directement.

Des problèmes de redirections ont également été réglé.

L'utilisateur connecté peut modifier/supprimer ses propres tâches uniquement. Les tâches anonymisées peuvent, elles, uniquement être supprimées par un administrateur de l'application.

L'utilisateur connecté pouvait consulter sur un même dashboard l'ensemble des tâches ouvertes et fermées simultanément. Comme dit précédemment, il n'a désormais accès qu'à ses propres tâches. De plus, il peut consulter séparément ses tâches ouvertes et celles terminées (et passer de l'un à l'autre le statut d'une de ses tâches) pour plus de visibilité.

V. Fonctionnalités ajoutées

De manière à assurer l'ensemble de ces fonctionnalités de manière pérenne, un système de sécurité et d'authentification a été mis en place (cf. la documentation technique).

Il est donc impossible pour un utilisateur non connecté de réaliser certaines actions, impossible également pour un utilisateur d'interagir avec les tâches d'autres utilisateurs, de modifier le rôle d'un utilisateur sans être administrateur.

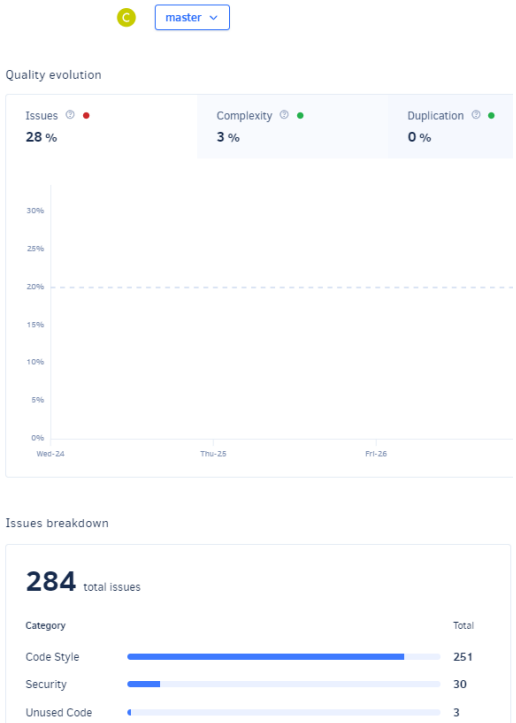
VI. Tests (unitaires et fonctionnels)

Une batterie de tests unitaires et fonctionnels a été implémentée de manière à garantir le bon fonctionnement du projet. Ces tests ont été mis en place avec PHP/Unit et l'extension XDebug. Nous avons également utilisé le bundle Dama de manière à reset la base de données à chaque lancement de tests. Sauf exception, il vous est également demandé d'utiliser ce bundle.

Lors de l'implémentation d'une nouvelle fonctionnalité il est également demandé au développeur d'implémenter les tests correspondants. Il est également demandé d'avoir une couverture de code d'au moins 70%.

Total	90.96%	322 / 354
Controller	88.10%	111 / 126
Entity	74.47%	35 / 47
Form	100.00%	143 / 143
Security	86.84%	33 / 38

VII. Qualité du code



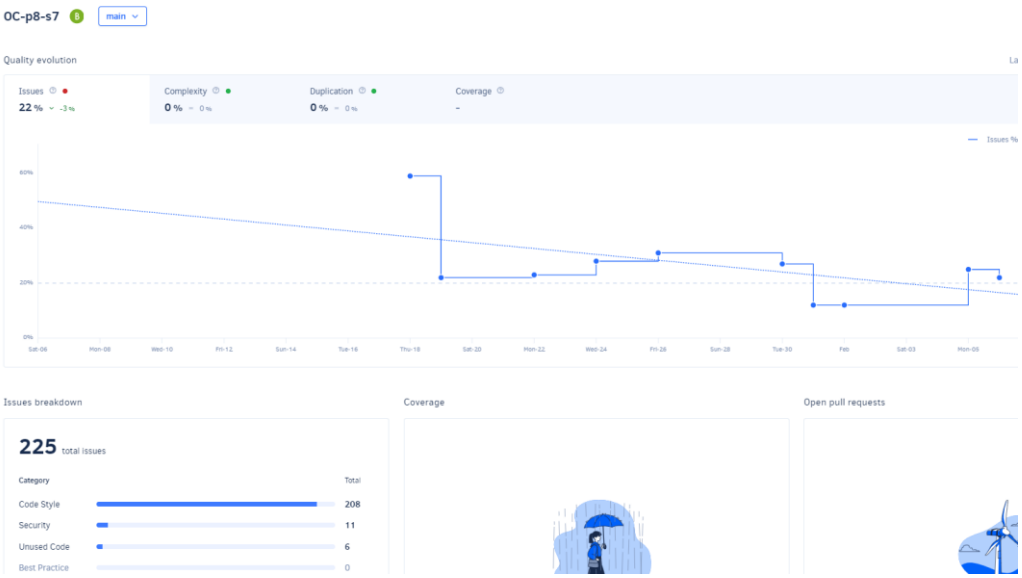
En ce qui concerne la qualité du code, nous nous efforçons de maintenir un code propre qui respecte les PSR. Nous utilisons également des outils comme Codacy dans cet objectif.

Codacy attribue une note pour la qualité globale du code de notre application. Plus cette note est proche de A, meilleure est la qualité de code.

On voit ici que le projet est initialement noté C ce qui est relativement peu satisfaisant. On voit aussi un certain nombre d'issues liées à la sécurité et au style de code. Le nombre d'issues liées au code inutilisé est plutôt satisfaisant. Beaucoup de ces éléments sont liés aux fichiers générés par Symfony. Cependant parmi les éléments de sécurité reportés ici, nombre d'entres sont dû à des utilisations non recommandées de variables super-globales.

L'objectif est donc de réduire ces erreurs critiques et de limiter celles implémentées dans nos fichiers une fois Symfony à jour.

Finalement nous avons actuellement un score de B mais surtout une complexité nulle et comme prévu une baisse conséquente des issues liées à la sécurité. Le code inutilisé de son côté augmente en lien avec les fichiers générés par Symfony (paramètres inutilisés sur des méthodes de formulaires et de fichiers de migrations).

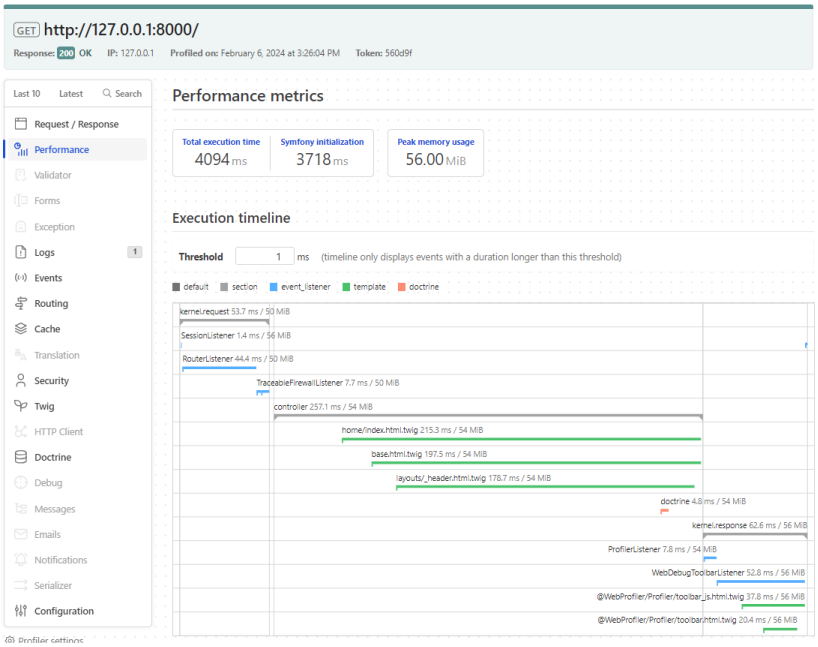


Outre ces éléments de qualité de code et comme nous l'avons vu dans la section *anomalies* le projet comportait initialement des problèmes importants de sécurité et l'absence de tests fonctionnels ne permet pas de corriger des éléments simples (redirections, comportements imprévisibles…). Pour ces raisons et pour l'ampleur du travail à effectuer (avant d'être en mesure de réaliser les migrations) sur un projet d'une taille modeste, j'ai jugé préférable de recréer le projet.

VIII. Performance

Le WebProfiler de Symfony nous permet de mettre en évidence certains éléments de performance.

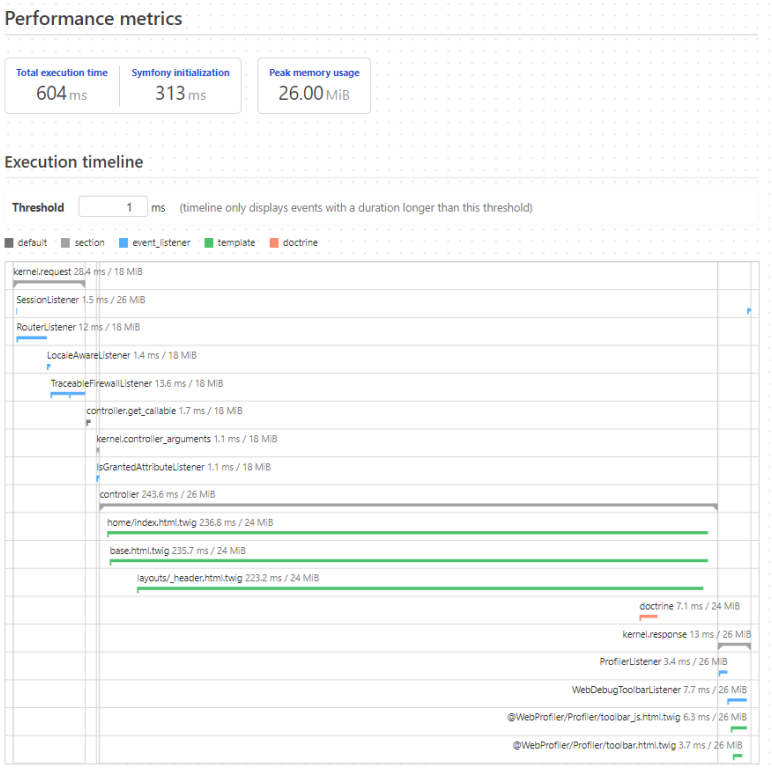
Ainsi, avec un cache vidé, l'affichage de la page d'accueil prend un temps conséquent et nécessite beaucoup de mémoire :



Une fois le cache mis en place, l'affichage de la page d'accueil est nettement plus rapide et nécessite deux fois moins de mémoire.

Cependant il est encore possible d'améliorer ces performances en passant le projet en production et en optimisant l'autoload de composer avec la commande :

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



--no-dev permet d'exclure les classes nécessaires seulement en développement.

--classmap-authoritative permet de créer un mappage des classes et d'empêcher de rechercher les classes introuvables.

Dans l'objectif d'améliorer continuellement la performance de l'application et donc l'expérience utilisateur, nous pouvons envisager certaines opérations à réaliser en production :

- Mettre en place l'OP cache afin de compiler nos fichiers en OP code et ainsi gagner en temps d'exécution.
- Mettre en place l'APCu afin d'avoir un système de cache en mémoire et pas au niveau du disque pour gagner énormément en rapidité. (Même si cela prend plus de place)
- Utiliser le cache HTML Varnish sur un serveur à part situé entre le client et le serveur, ainsi dans notre projet par exemple, si un utilisateur demande à accéder à la liste de ses tâches ouvertes, Varnish met en cache les informations utilisateurs. Il fait de même si un autre utilisateur souhaite accéder à la page de ses tâches et lorsqu'un utilisateur avec les informations en cache revient, il transmet directement la page HTML si celle si n'est pas « périmée ». Cette optimisation est importante surtout si le nombre d'utilisateurs (connectés ou non) devient conséquent.