Audit de qualité du code et de

performance

Projet ToDo & Co

1.Contexte du projet

Une application a été développée en version démo ou “minimum viable product” par une startup afin de pouvoir lever des fonds pour démarrer.
Une fois fait, cette ébauche d’application doit être remaniée afin d’offrir un produit stabilisé, propre, sûr, et à jour.

Des anomalies ont été détectées et corrigées; quelques modifications ont été opérées afin de restaurer une certaine cohérence et des tests unitaires et fonctionnels ont été effectués afin de vérifier le bon comportement de l'application.
Enfin, un comparatif de performances entre l’ancienne et la nouvelle version a été réalisé afin de garantir l’amélioration effective des performances de l’application après sa mise à jour et son amélioration.

2. Mise à jour : migration Symfony 3.1 à Symfony 6.3

1) Dossiers et structure:

    a) Le répertoire `app/` a été remplacé par `config/`, `src/`, `var/` et `public/` dans Symfony 4.
    b) Les bundles ne sont plus autant mis en avant. Dans Symfony 4+, les applications sont simplifiées et tout le code réside généralement dans le répertoire `src/`.

2) Services:

Dans Symfony 4+, les services sont privés par défaut, ce qui signifie qu’on ne peut plus les récupérer directement à partir du conteneur avec `$container->get('service_name')`.

3) Routes:

Passage des annotations aux attributs.

4) Templates:

   Ils ont dû être déplacés depuis app\Resources\views\ vers \templates\

5) Doctrine:

   Mise à jour Doctrine vers une version compatible avec Symfony 6.

6) Dépendances et composants:

De nombreux composants ont été introduits, dépréciés ou supprimés. Mise à jour ou remplacement des paquets nécessaires.

7) Dépréciations:

   Tout au long de la mise à niveau, correction de toutes les dépréciations.

8) API Platform et autres bundles tiers:

   Mise à jour des bundles tiers vers des versions compatibles avec Symfony 6.

9) PHP, Mysql… :
   a) Mise à jour des versions PhP 5.6.40 vers la 8.2
   b) Mise à jour des versions MySQL 5.5 vers 8.0.30

3. Tests et améliorations de code

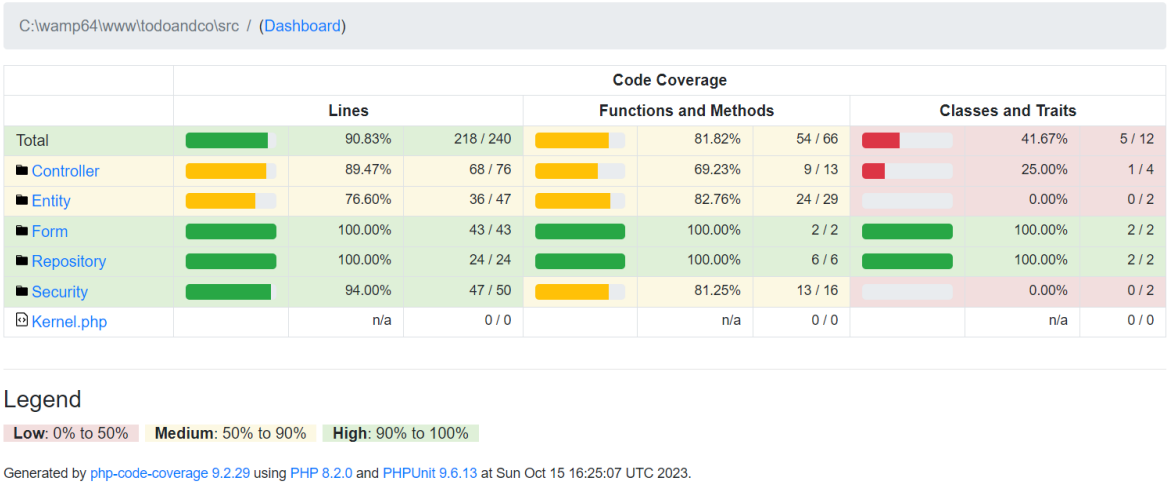
a) Certains bugs et incohérences ont dû être corrigés (boutons qui ne fonctionnaient pas, ou pas de la façon attendue, etc).
b) Refactoring et mise au propre du code selon les bonnes pratiques : controllers, services, entités, templates, tests
c) Utilisation d'outils d'amélioration de qualité du code :
   ○ Php CS Fixer
   ○ Phpunit
   ○ Php Stan
   ○ Symfony profiler
   ○ Codacy

4. Mise en place des tests

Des tests unitaires ont été mis en place via phpunit, de façon la plus étendue possible. Les erreurs remontées ont été corrigées.

```
C:\wamp64\www\todoandco>php ./vendor/bin/phpunit --coverage-html public/phpunit
PHPUnit 9.6.13 by Sebastian Bergmann and contributors.

Testing
.............Veuillez saisir un email valide
invalid-email
..This value should not be blank.

Username:
.....................                              40 / 40 (100%)

Time: 00:05.036, Memory: 66.00 MB

OK (40 tests, 82 assertions)
```

Couverture des tests:



C:\wamp64\www\todoandco\src / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	90.83%	218 / 240		81.82%	54 / 66		41.67%	5 / 12	
Controller	89.47%	68 / 76		69.23%	9 / 13		25.00%	1 / 4	
Entity	76.60%	36 / 47		82.76%	24 / 29		0.00%	0 / 2	
Form	100.00%	43 / 43		100.00%	2 / 2		100.00%	2 / 2	
Repository	100.00%	24 / 24		100.00%	6 / 6		100.00%	2 / 2	
Security	94.00%	47 / 50		81.25%	13 / 16		0.00%	0 / 2	
Kernel.php		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0

Legend

Low: 0% to 50% **Medium**: 50% to 90% **High**: 90% to 100%

Generated by php-code-coverage 9.2.29 using PHP 8.2.0 and PHPUnit 9.6.13 at Sun Oct 15 16:25:07 UTC 2023.

5. Audit de qualité

En plus de re-développer en suivant les bonnes pratiques et les normes, d'optimiser la qualité du code par le nettoyage et la refactorisation du code, du développement des tests unitaires et fonctionnels, les
outils suivants ont été utilisé afin d'accroître à la plus haute qualité et de réduire à 0 la dette technique de l'application :

- Php CS Fixer
- Php Stan
- Codacy

Cela contribue à assurer d'autant plus que les évolutions et diverses modifications actuelles ou ultérieures
n'entraînent aucune régression de l'application.

Rapport Qualité de code Codacy - Projet final SF 6.3

5. Comparatif audit de performances

Réalisé simplement via le profiler Symfony.
Voici les données renvoyées par l'outil:

1) **Informations sur le contrôleur** :

Indique quel contrôleur a été appelé pour traiter la requête actuelle et combien de temps il a pris.

2) **Détails des événements** :

Symfony fonctionne sur la base d'un système d'événements (par exemple, avant et après que la requête soit traitée, avant et après que la réponse soit envoyée, etc.). Cette section montre combien de temps chaque écouteur d'événement a pris.

3) **Mémoire utilisée** :

Il donne une indication de la quantité de mémoire utilisée pendant la requête.

4) **Bases de données** :

Si l'application interagit avec une base de données, cet onglet montrera souvent combien de temps les requêtes de la base de données ont pris et combien de requêtes ont été exécutées pendant la requête actuelle.

Au fil des versions, le code source de Symfony a été régulièrement optimisé. Des structures de données plus efficaces, une meilleure gestion de la mémoire et une réduction des appels inutiles ont contribué à améliorer la rapidité du framework.

Les différences sont remarquables.

Compilation et Conteneur :

> Le conteneur de services de Symfony est l'une des pièces maîtresses de son architecture. Entre Symfony 3 et 6, le conteneur a été optimisé, notamment avec une compilation plus efficace et une résolution plus rapide des services.

Autowiring :

> Introduit après Symfony 3, l'autowiring a réduit la nécessité d'une configuration manuelle étendue pour les services. Bien que cela puisse sembler être une commodité pour les développeurs, cela a également conduit à une configuration de service plus optimale et standardisée, réduisant le surcoût de la configuration mal optimisée.

Nouveaux composants :

> De nouveaux composants ont été introduits, chacun étant optimisé pour une meilleure performance par rapport aux solutions précédentes. Par exemple, le composant `HttpClient` introduit dans Symfony 4.3 est très performant pour faire des requêtes HTTP.

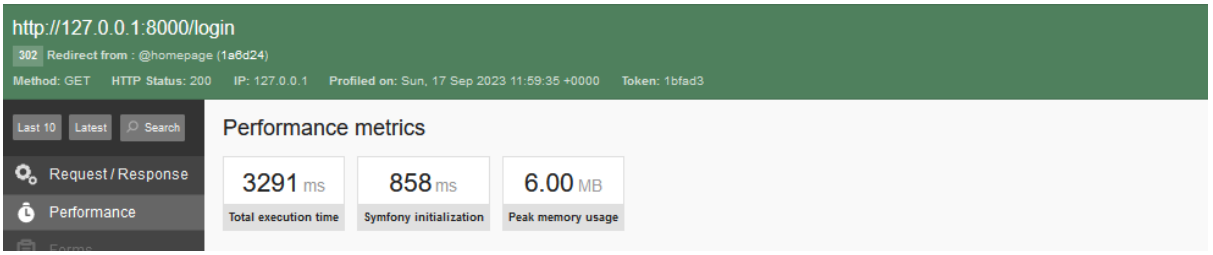
Améliorations du Profiler :

> Le profiler de Symfony, bien qu'il ne soit généralement pas utilisé en production, a aidé les développeurs à identifier et corriger les goulots d'étranglement plus efficacement, ce qui a indirectement contribué à améliorer les performances.

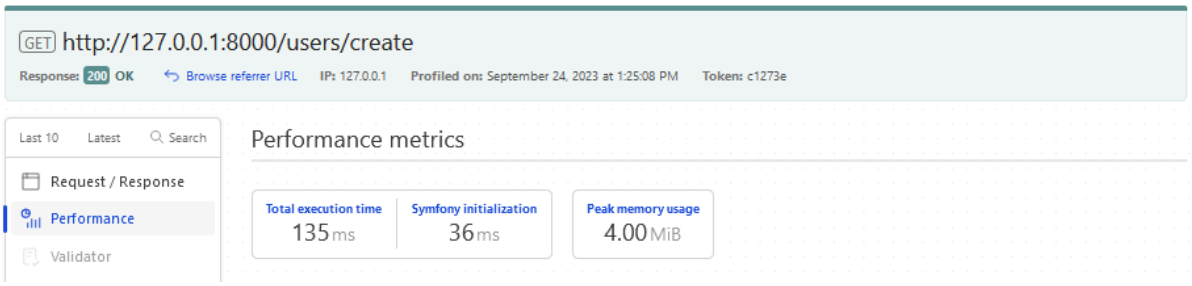
Version initiale: Symfony 3.1
Version finale: Symfony 6.3

1. Page d’accueil -> Redirection vers le login

old

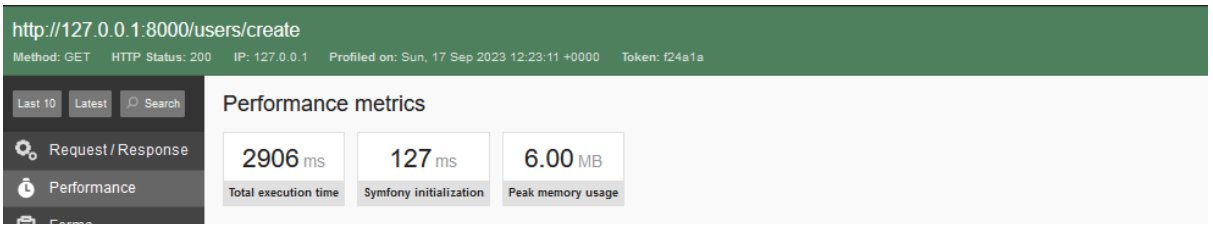


new

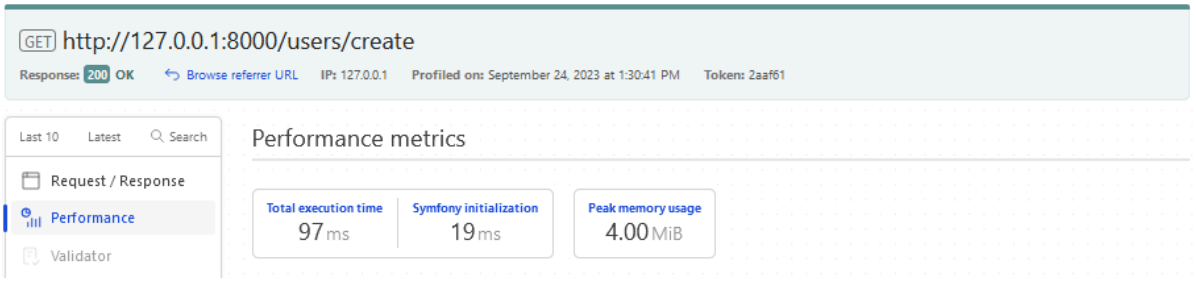


2. Création utilisateur

old

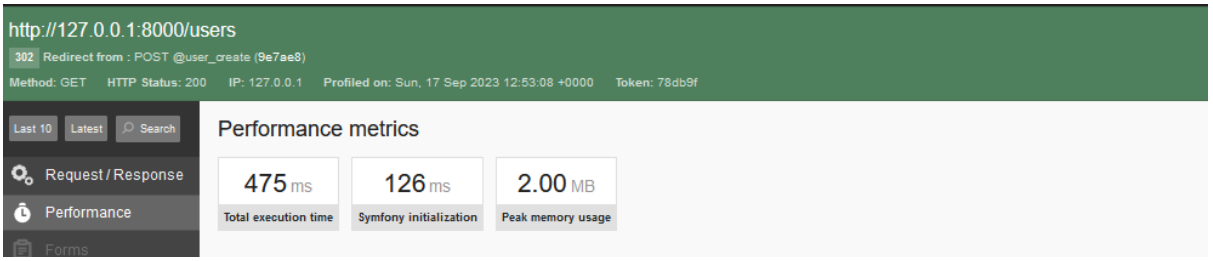


new

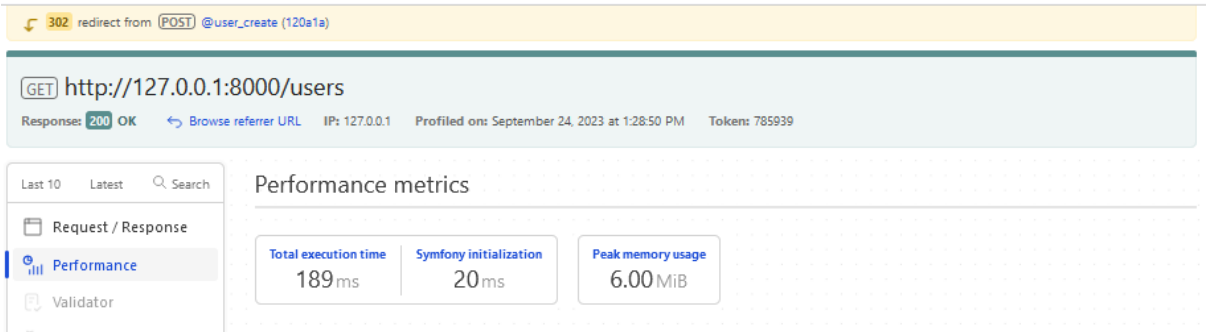


3. Liste desutilisateurs

old

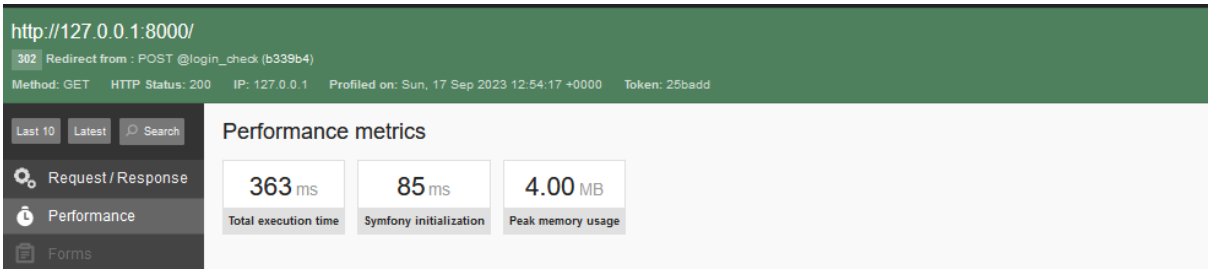


new

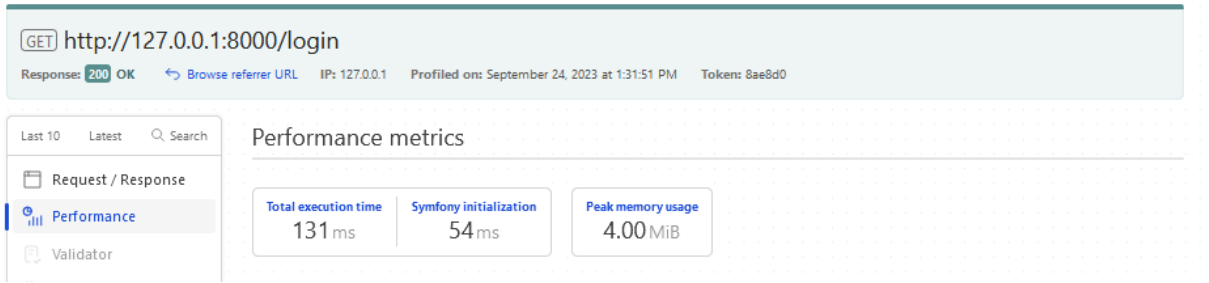


4. Connexion

old

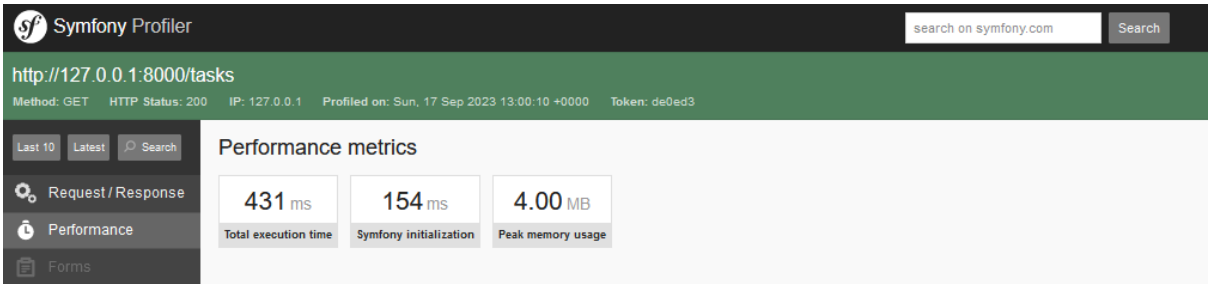


new

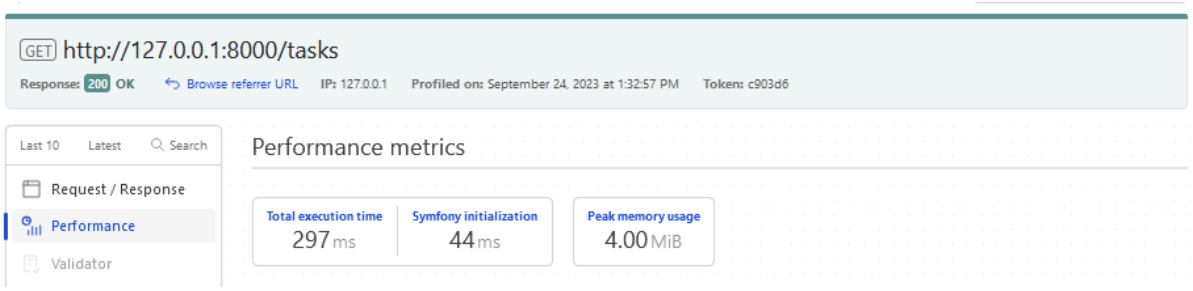


5. Consultation des tâches à faire

old

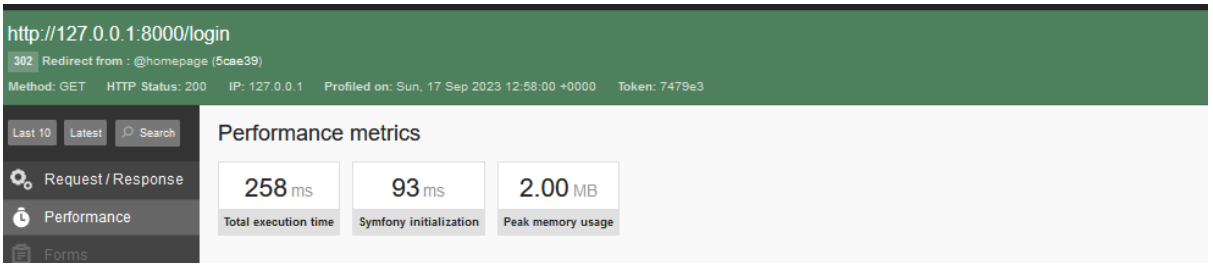


new

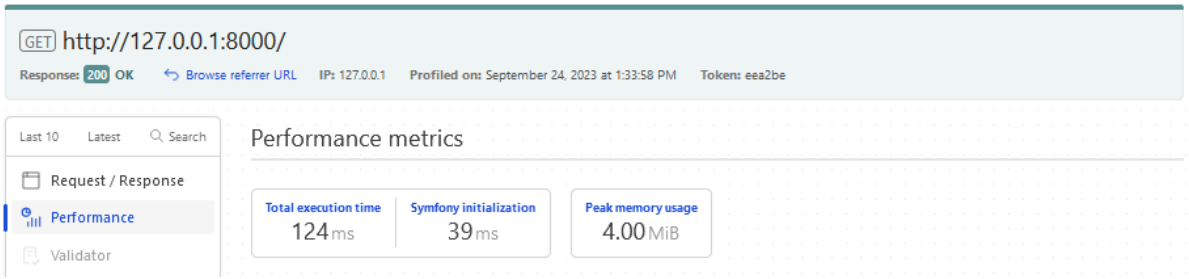


6. Consulter tâches terminées

old

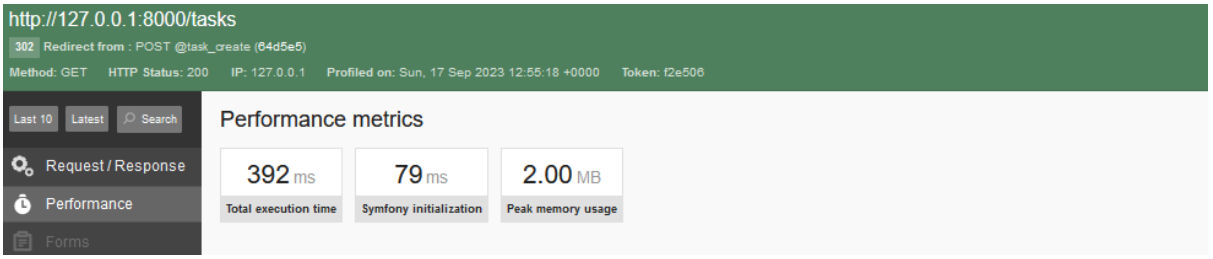


new

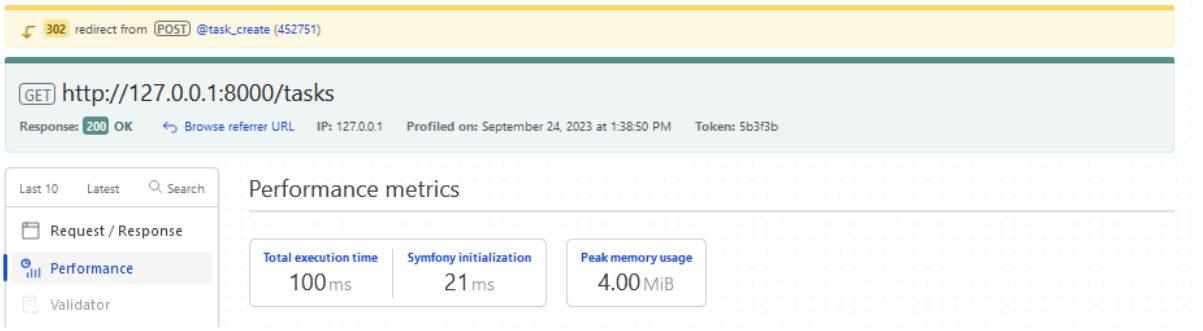


7. Creation tâche

old

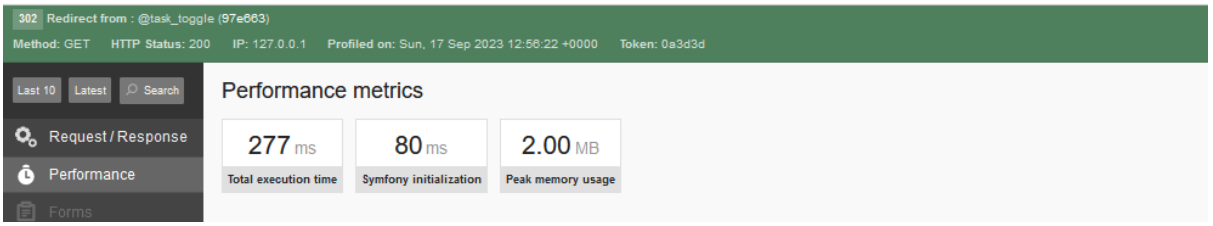


new

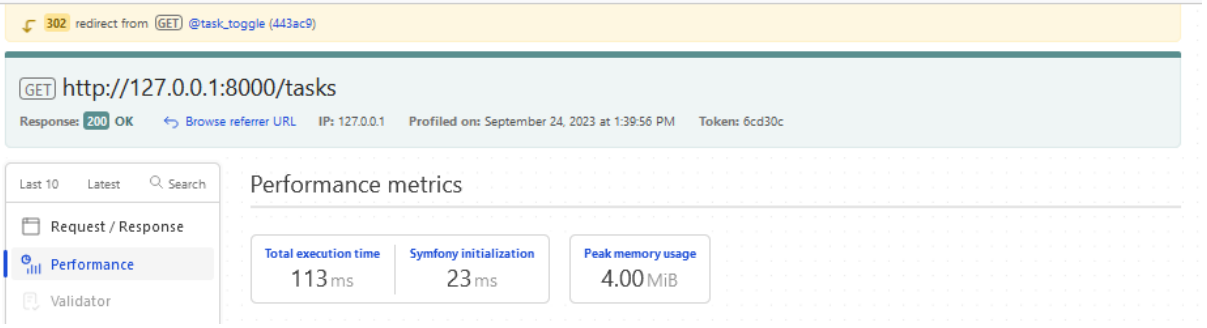


8. Marquer une tâche comme faite

old

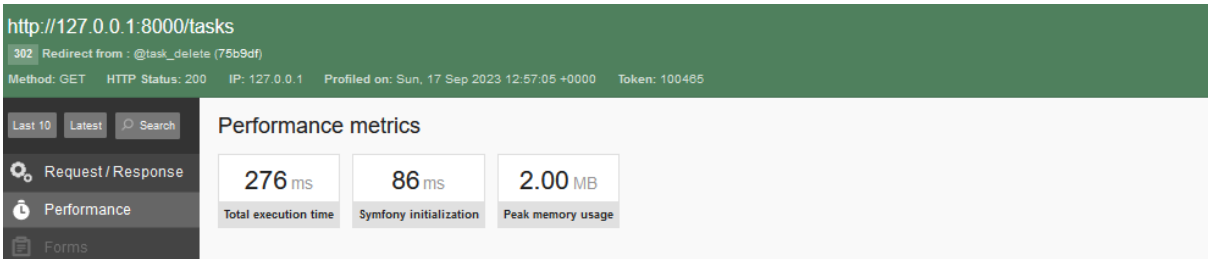


new

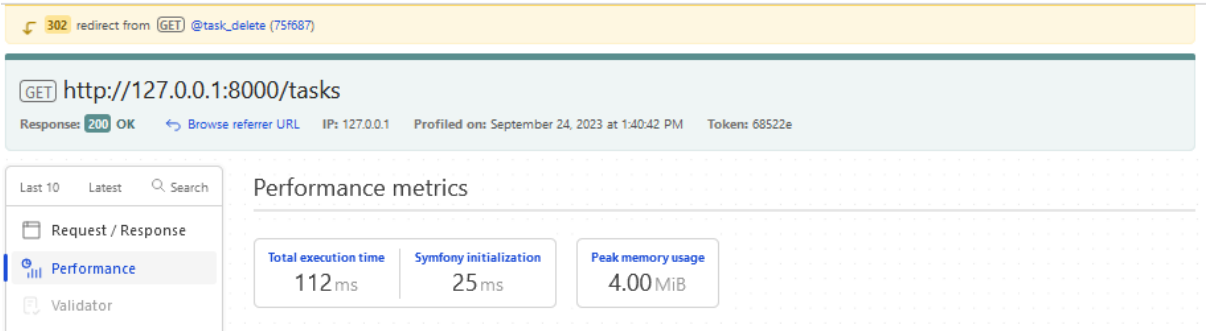


9. Suppression tâche

old

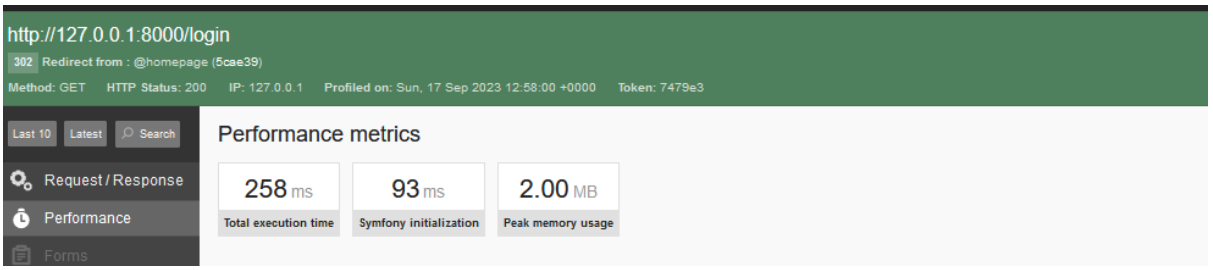


new

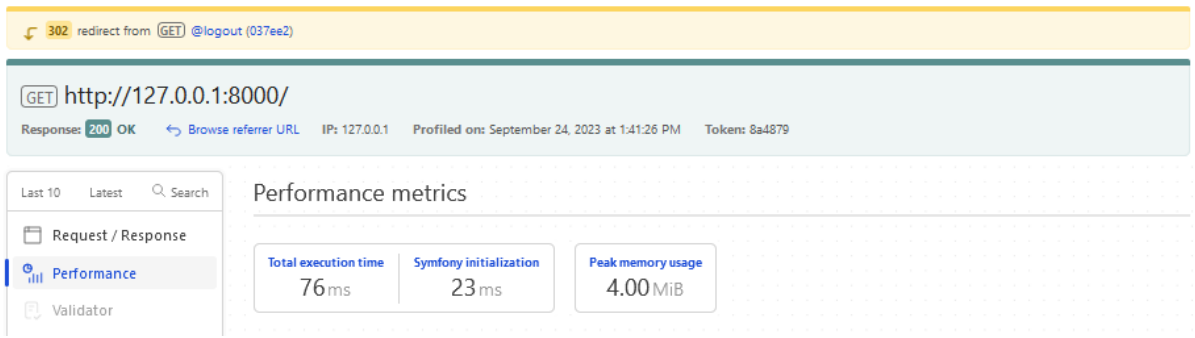


10. Déconnexion

old



new



En analysant les résultats obtenus, on peut constater aisément, et bien qu'utilisant un outil très peu optimal comparé à Blackfire, utilisé pour le profilage du projet initial, on obtient pour notre application finale de hauts gains de performances.

L’analyse démontre en effet une réduction systématique de 70 à 80% de l’utilisation maximale de mémoire.