Audit de qualité du code et performance de l'application

Introduction :

L'audit de qualité du code et de performance est une étape essentielle dans le processus de développement d'une application. Il permet d'évaluer la qualité du code source, d'identifier les problèmes de performance et de proposer des améliorations. Dans le cadre de cet audit, nous utiliserons l'outil **Codacy** pour évaluer la qualité du code et le **profiler de Symfony** pour analyser les performances de l'application.

Importance de l'audit de qualité du code :

L'audit de qualité du code est important pour plusieurs raisons :

- Assurer la maintenabilité : Un code de qualité facilite la maintenance et les évolutions de l'application. Il est plus facile à comprendre, à tester et à modifier.
- Réduire la dette technique : En identifiant les problèmes de code, tels que les duplications, les mauvaises pratiques, les vulnérabilités, les erreurs de syntaxe, l'audit permet de réduire la dette technique et d'améliorer la stabilité de l'application.
- Favoriser la collaboration : Un code propre et bien structuré facilite la collaboration entre les membres de l'équipe de développement. Il permet de mieux comprendre et de travailler ensemble de manière efficace.
- Améliorer la sécurité : L'audit de code permet de détecter les vulnérabilités et les failles de sécurité, ce qui contribue à renforcer la sécurité de l'application.
- Utilisation de **Codacy** pour l'audit de qualité du code : **Codacy** est un outil d'analyse statique du code qui permet d'évaluer la qualité du code source en se basant sur des règles prédéfinies et des bonnes pratiques. Il fournit des métriques et des rapports détaillés sur la qualité du code. L'utilisation de **Codacy** dans l'audit de qualité du code permet de :

  - Identifier les duplications de code.

  - Détecter les erreurs de syntaxe et les problèmes de style.

  - Vérifier la conformité aux bonnes pratiques de programmation.

  - Évaluer la complexité du code.

  - Détecter les vulnérabilités et les failles de sécurité.

Importance de l'audit de performance :

L'audit de performance est crucial pour garantir une application réactive et performante. Il permet d'identifier les goulots d'étranglement, les requêtes lentes, les problèmes de mémoire et de proposer des optimisations pour améliorer les performances globales de l'application. Les avantages de l'audit de performance sont les suivants :

- Améliorer l'expérience utilisateur : Une application performante offre une meilleure expérience utilisateur en réduisant les temps de chargement et en assurant une navigation fluide.
- Optimiser l'utilisation des ressources : L'audit de performance permet d'identifier les parties du code qui consomment le plus de ressources (CPU, mémoire, requêtes réseau) et de les optimiser, ce qui réduit les coûts d'infrastructure.
- Prévenir les problèmes à grande échelle : En détectant les problèmes de performance à un stade précoce, l'audit permet de prévenir les problèmes à grande échelle et les pannes de l'application.
- Utilisation du profiler de Symfony pour l'audit de performance : **Le profiler de Symfony** est un outil puissant intégré au Framework Symfony qui permet d'analyser les performances de l'application. Il fournit des informations détaillées sur les requêtes, les temps d'exécution, les appels de base de données, les requêtes réseau, etc. En utilisant le profiler de Symfony dans l'audit de performance, nous pouvons :

  - Identifier les requêtes lentes et les problèmes de performances.

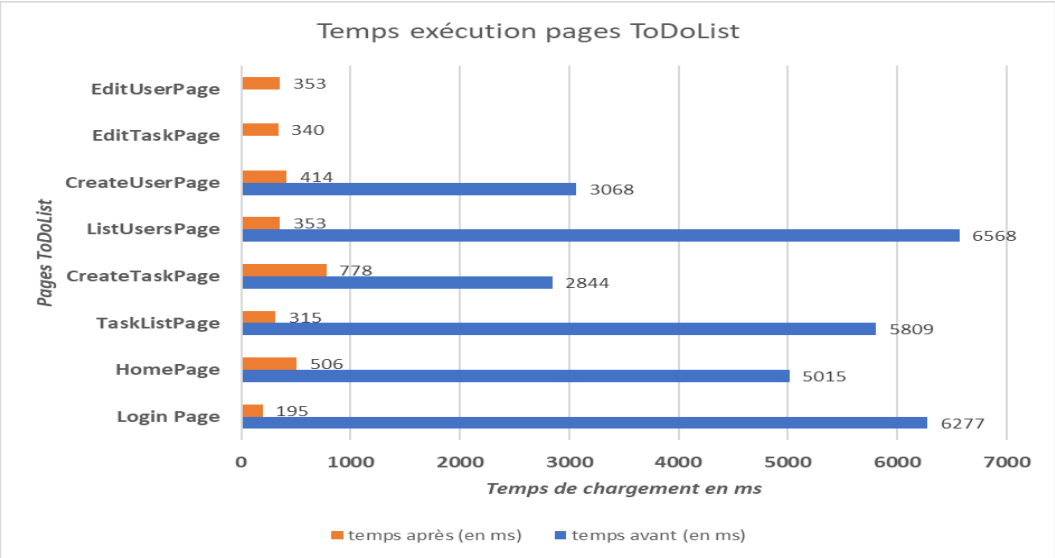
  - Analyser l'utilisation de la mémoire et les fuites de mémoire potentielles.

  - Optimiser les requêtes de base de données.

  - Mesurer les temps d'exécution des différentes parties de l'application.

  - Identifier les problèmes de cache et proposer des optimisations.

Grâce au **profiler de Symfony**, nous avons pu mesurer la vitesse de chargement des différentes pages de l'application. Voici les résultats :

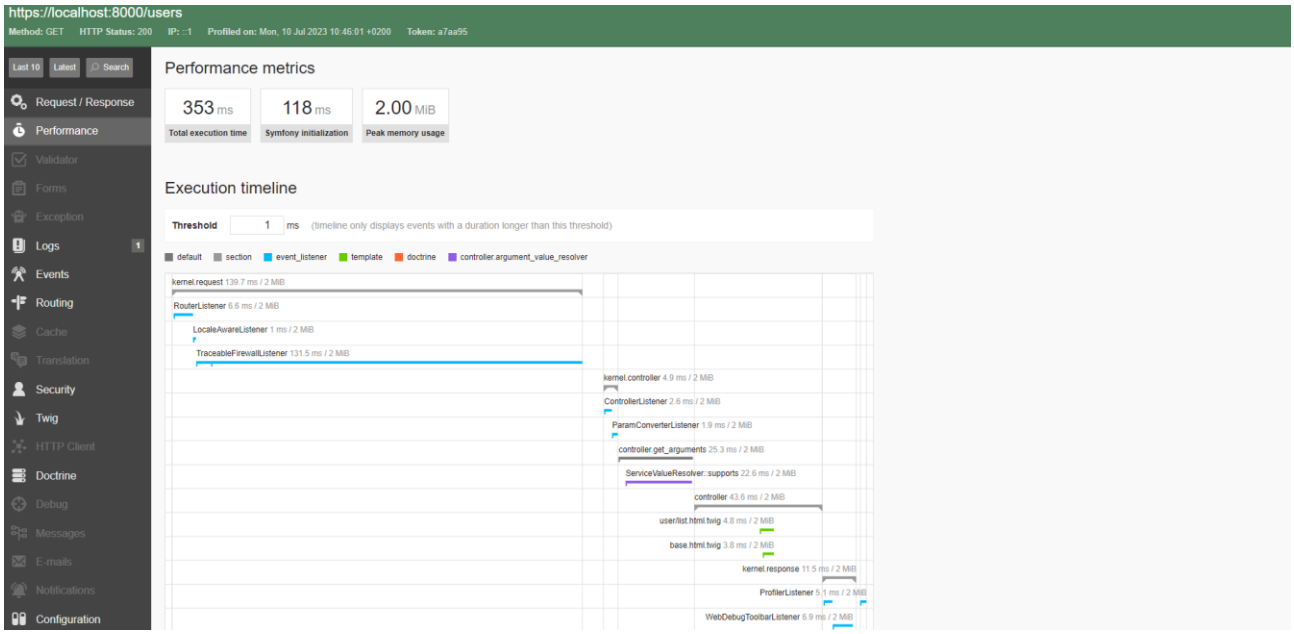


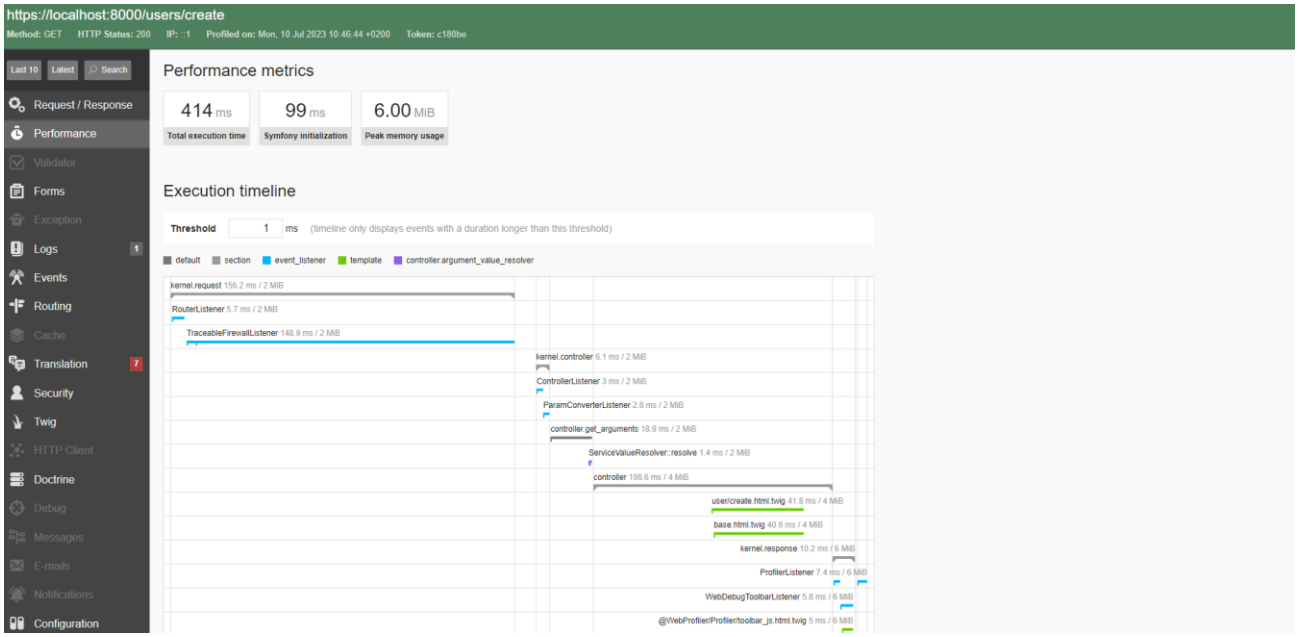
Nous remarquons que les pages avant les améliorations apportées mettent trop de temps à se charger (rappelons que plus de la moitié des utilisateurs changent de page si elle se charge en plus de 3 secondes).
Cependant les résultats sont faussés car plusieurs pages étaient ouvertes en même temps que l'application ce qui a pu augmenter le temps d'exécution des pages de l'application.

Le Profiler met également en évidence un point commun intéressant entre les différentes pages, concernant la vitesse de chargement et ce qui pourrait être résolu.

Regardons cela ensemble :

Voici différentes photographies de pages de l'application





On remarque deux lignes dont le traitement de la requête prend beaucoup de temps et peut affecter les performances globales de l'application : **“kernel.request”** et **“TraceableFirewallListener”**. Voyons les définitions :

"kernel.request" : Cette ligne représente le temps écoulé depuis le début du traitement de la requête par le kernel de Symfony. Cela inclut le processus de gestion de la requête entrante, le routage, la résolution des contrôleurs et d'autres opérations préliminaires nécessaires avant d'exécuter le contrôleur et de générer une réponse. Un temps élevé pour cette étape peut indiquer un problème de performance au niveau du kernel de Symfony lui-même ou dans les étapes de préparation de la requête.

"TraceableFirewallListener" : Cette ligne représente le temps écoulé lors de l'exécution du TraceableFirewallListener, qui est un écouteur d'événements du composant de sécurité de Symfony. Cet écouteur gère les tâches liées à l'authentification, l'autorisation et la gestion des pare-feux dans votre application. Un temps élevé pour cette étape peut indiquer que le processus d'authentification ou d'autorisation prend beaucoup de temps, ou qu'il y a des ralentissements dans les opérations de pare-feu.

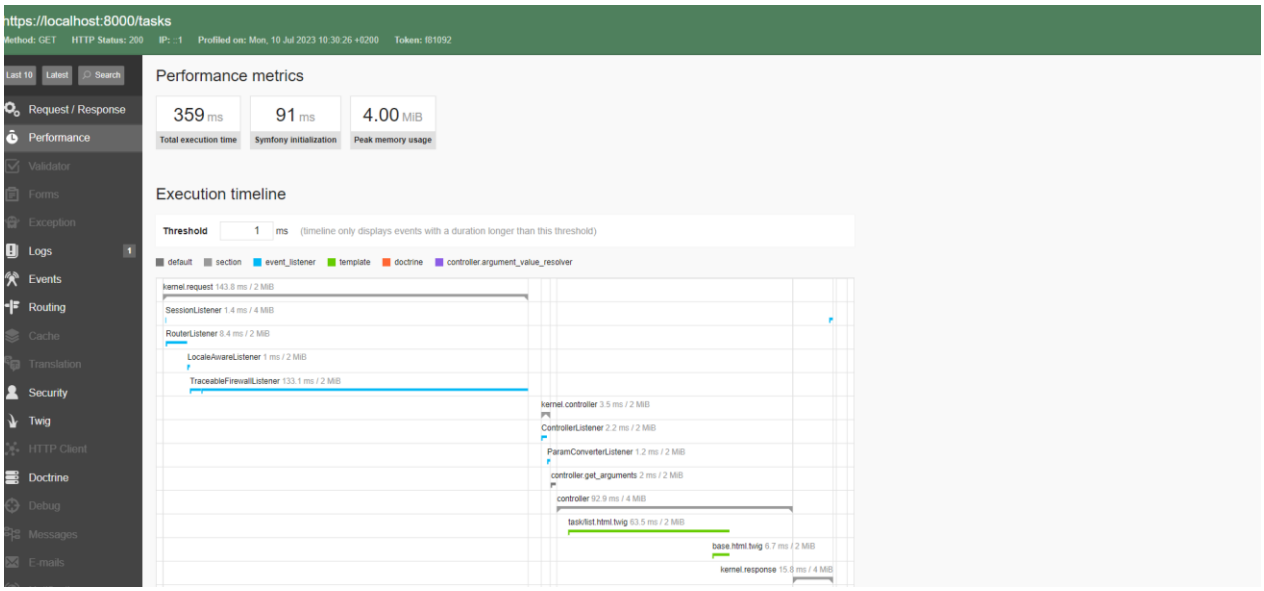
Propositions d'améliorations pour l'application :

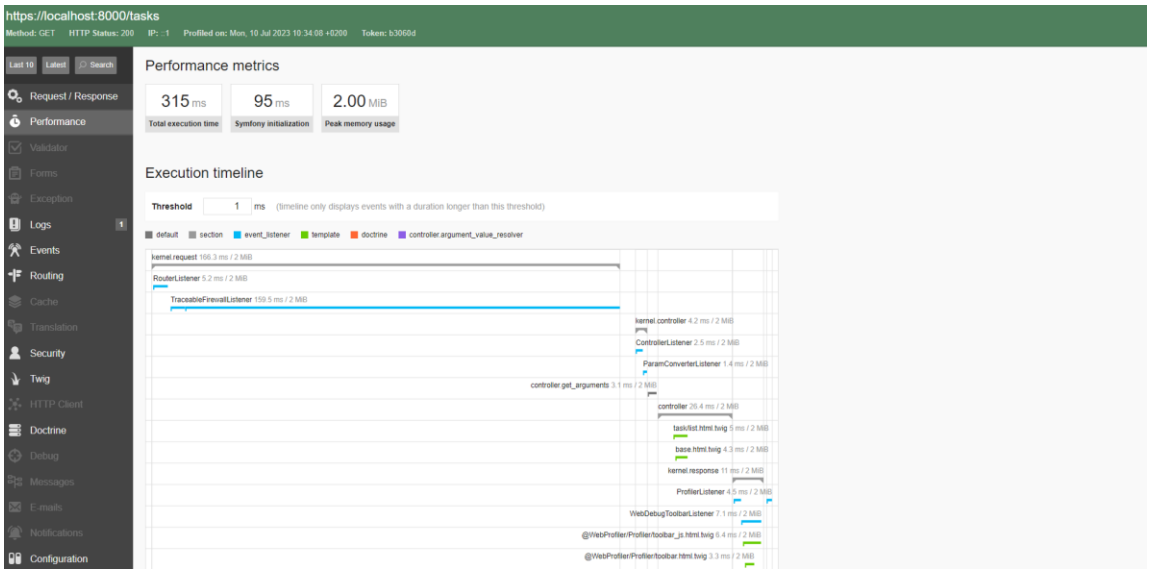
1. Mettre à jour Bootstrap pour une version récente : La mise à jour de Bootstrap vers une version récente peut améliorer la performance de l'application en bénéficiant des améliorations de performances et des optimisations introduites dans les nouvelles versions.
2. Optimiser les images et utiliser le lazy-loading : L'optimisation des images en réduisant leur taille et en utilisant des formats adaptés peut considérablement améliorer les performances de l'application. De plus, en utilisant des techniques de lazy-loading, telles que charger les images au fur et à mesure de leur affichage, on réduit le temps de chargement initial de la page.

3. Intégrer le cache de Symfony : L'utilisation du cache de Symfony permet de stocker en mémoire certaines parties de l'application, réduisant ainsi le temps de traitement des requêtes. En utilisant des mécanismes de mise en cache tels que **Redis** ou **Memcached**, on peut améliorer significativement les performances en évitant de répéter des opérations coûteuses.
4. Implémenter un bouton "load-more" pour charger les tâches au fur et à mesure : Si l'application gère un grand nombre de tâches, il est recommandé d'implémenter une fonctionnalité de chargement progressif, où seules les tâches nécessaires sont chargées à la fois. Par exemple, au lieu de charger toutes les 100 tâches en une seule fois, on peut les charger par lots de 10 ou 20 à mesure que l'utilisateur fait défiler la page. Cela réduit la charge initiale de l'application et améliore la réactivité.
5. Le HTML et le CSS peut être largement améliorés : cela permettra de réduire la dette technique du site.
6. Utilisation du Javascript et AJAX pour mettre à jour une partie d'une page web sans avoir à recharger toute la page. Cela permet de réduire le temps de chargement global de l'application, car seules les parties de la page qui doivent être mises à jour sont récupérées du serveur. Cela réduit la quantité de données à transférer sur le réseau et diminue le temps d'attente pour les utilisateurs.
7. Adopter des bonnes pratiques de sécurité : il faut s’assurer que l'application suit les bonnes pratiques de sécurité, telles que l'authentification et l'autorisation appropriées, la protection contre les attaques par injection SQL, les attaques XSS, etc. Une application sécurisée est moins susceptible de subir des attaques qui pourraient affecter ses performances.
8. Optimiser les requêtes de base de données : Analysez les requêtes SQL utilisées par l'application et recherchez des opportunités d'optimisation, telles que l'ajout d'index appropriés, l'utilisation de jointures efficaces (Custom Queries), la limitation des résultats retournés, etc. L'objectif est de réduire la charge sur la base de données et d'améliorer les temps de réponse.
9. Utiliser des requêtes asynchrones : Si l'application effectue de nombreuses requêtes HTTP vers des API externes, envisagez d'utiliser des requêtes asynchrones pour éviter les temps d'attente inutiles et améliorer la réactivité de l'application. Cela peut être réalisé en utilisant des bibliothèques telles que Guzzle pour effectuer des appels asynchrones.
10. On peut refactoriser le code (avec Rector php par exemple). En réécrivant des parties du code de manière plus efficace, cela permettra de réduire le temps d'exécution et d’améliorer les performances globales de l'application.
11. On peut également mettre en place une date limite pour chaque tâche pour éviter un encombrement inutile. Par exemple : Mettre une dead-line d’un mois et envoyer un rappel par mail à l’utilisateur concerné toutes les semaines pour le prévenir que la tâche est en attente et sera bientôt supprimée.

Pour améliorer le temps de chargement, j'ai compressé les images du site et les ai converti en format "nouvelle génération" (ici Webp). Cela a réduit la taille des images et donc diminué le temps de chargement des pages. J'ai également mis un 'defer' dans le script.js de la page.

Voyons cela sur cet exemple de la page :





Les images sont floues mais on peut noter une légère amélioration concernant les fichiers twig et le kernel.reponse sans être sûr qu'il y a bien un lien entre les deux.

Mais voici le type d'amélioration que l'on peut apporter pour tenter d'optimiser les performances de notre application.

Conclusion :

L'audit de qualité du code et de performance est une étape cruciale pour assurer la pérennité et l'amélioration continue d'une application. L'utilisation d'outils tels que **Codacy** et le **profiler de Symfony** permet d'obtenir des métriques précises, d'identifier les problèmes et de proposer des solutions d'optimisation. En mettant en œuvre les améliorations suggérées, telles que la mise à jour de Bootstrap, l'optimisation des images, l'utilisation du cache de Symfony et l'implémentation d'un chargement progressif, vous pourrez améliorer significativement la qualité, la stabilité et les performances de votre application.

On peut également utiliser des outils plus performants tels que Blackfire.io (payant) par exemple pour obtenir des données plus précises.