



AUDIT DE QUALITE DU CODE

TO-DO-LIST

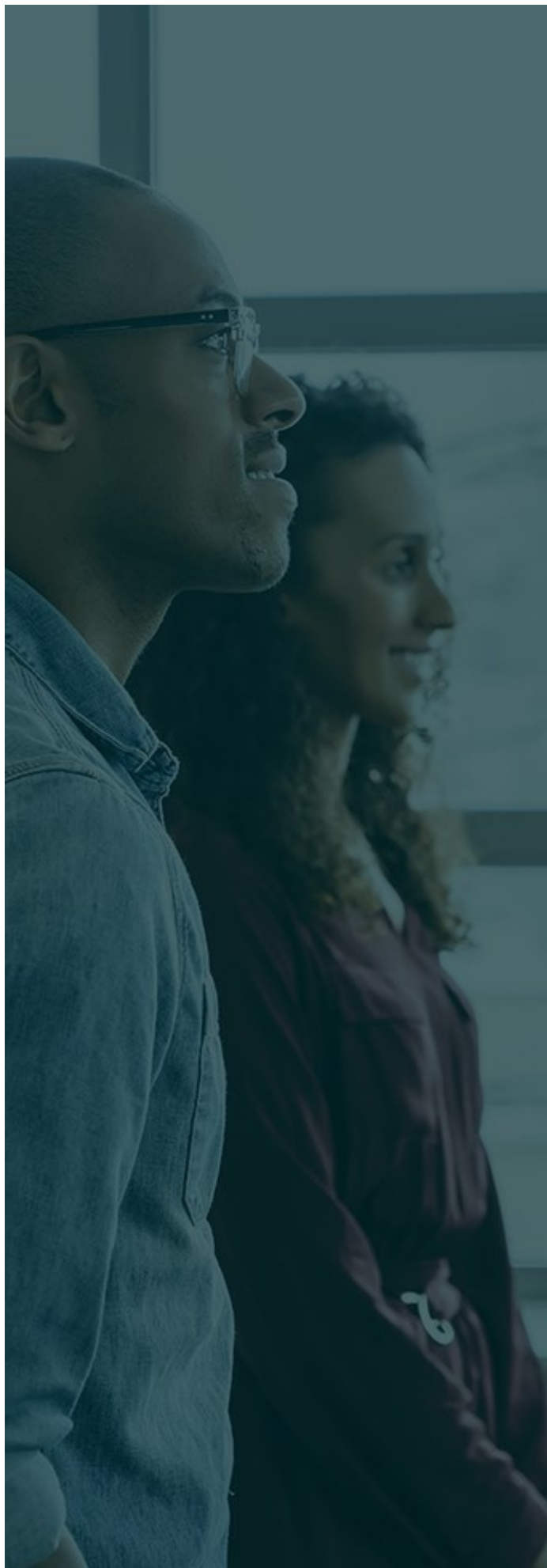


TABLE DES MATIÈRES

Introduction.....	3
1. Rapport de synthèse.....	4
2. Présentation de l'entreprise	6
3. Description de l'activité	Erreur ! Signet non défini.
4. Analyse de marché ..	Erreur ! Signet non défini.
5. Plan d'exploitation ...	Erreur ! Signet non défini.
6. Plan marketing et ventes .	Erreur ! Signet non défini.
7. Plan financier	Erreur ! Signet non défini.
Annexe	Erreur ! Signet non défini.
Instructions pour commencer à estimer les coûts de démarrage .. Erreur ! Signet non défini.	
Instructions pour commencer sur les prévisions de résultat Erreur ! Signet non défini.	

INTRODUCTION

Dans le cadre de l'amélioration continue de notre application,

Nous allons faire une évaluation de son état actuel afin d'identifier les domaines d'amélioration potentiels.

Pour ce faire, nous allons suivre une démarche structurée en trois étapes :

- Tout d'abord, nous réaliserons un état des lieux de la dette technique de l'application, permettant ainsi d'identifier les problèmes techniques et les vulnérabilités qui pourraient impacter sa performance et sa sécurité.
- Ensuite, nous effectuerons un audit de code approfondi sur deux axes principaux : la qualité de code et la performance.
 - Pour évaluer la qualité de code, nous utiliserons des outils tels que Codacy qui nous permet de détecter les erreurs, les complexités et les problèmes de lisibilité.
 - En ce qui concerne la performance, nous utiliserons des outils de profiling tels que Symfony, pour identifier les goulots d'étranglement et les opportunités d'amélioration.
- Enfin, après avoir apporté les modifications nécessaires, nous réaliserons un nouvel état des lieux de l'application pour évaluer l'impact des améliorations apportées et identifier les domaines qui nécessitent encore des améliorations.

Cette démarche nous permet de garantir que notre application est robuste, performante et répond aux besoins des utilisateurs.

1. RAPPORT DE SYNTHÈSE

Le projet ToDoList présente une dette technique significative qui peut impacter sa performance et sa sécurité.

Dans ce rapport, nous allons présenter les résultats de notre analyse et identifier les problèmes techniques et les vulnérabilités qui nécessitent une attention particulière.

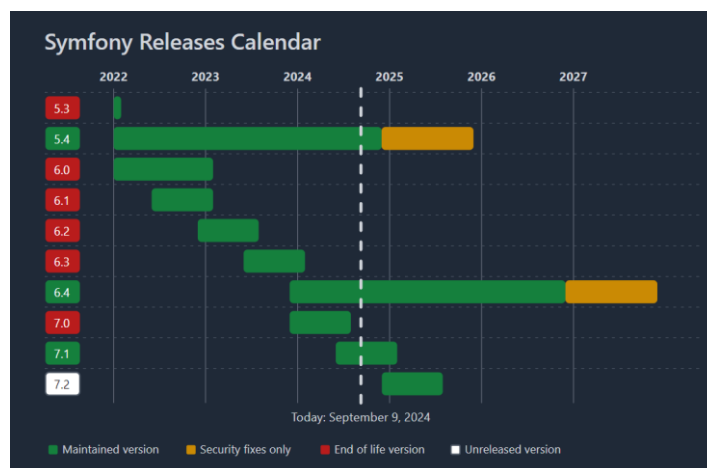
Lorsque nous analysons les fichiers existants nous pouvons voir les points suivant.

README.md

- le fichier README.md n'est pas à jour et contient aucune information, notamment les instructions d'installation et d'utilisation

Composer.json

- La version de PHP requise est $\geq 5.5.9$ ce qui est une ancienne version aujourd'hui nous sommes à PHP 8.1
- La version de Symfony est 3.1.* ce qui est également une ancienne version aujourd'hui nous sommes à Symfony 6.4 LTS. Ce qui entraîne une perte en fonctionnalités et un danger d'un point de vue sécurité.



Resources/views

- Les fichiers de vues semblent être organisés de manière logique, avec des dossiers pour les différentes entités (security, task, user).
- Cependant il y a des problèmes de logique comme celle de laisser le bouton pour la création de l'utilisateur
- Le bouton « consulter la liste des tâches terminées » n'affiche rien même lorsqu'il existe des tâches terminées.
- La librairie Swiftmail est installée mais n'est pas utilisée dans le code.
- Cliquer sur « afficher la liste des tâches à faire » affiche toutes les tâches

- L'utilisation de « AppBundle » dans les namespaces et du a l'ancienne version de Symfony. Maintenant on n'utilisera « App ».
- Les contrôleurs étendent la classe Controller. On utilisera plutôt la classe AbstractController disponible depuis la version 3.3.
- La méthode LoginAction utilise l'AuthenticationUtils dans le container. On utilisera plutôt l'injection de dépendance.
- Problème de gestion des droits d'accès et de modification des comptes. Il est nécessaire de mettre en place un système d'accès et de gestion des rôles.
- Il manque le token CSRF ce qui peut créer une faille de sécurité.

config/security.yml

- La partie access_control n'est pas mis en place

Controller/SecurityController.php, TaskController.php et UserController.php

- Il manque des exceptions pour gérer les erreurs qui pourraient se produire lors de la navigation.
- les méthodes ne sont pas sécurisées par rapport à l'utilisateur.

Entity/User.php, Entity/Task.php

- Il manque quelque méthode.

test/AppBundle/Controller

- Le test existant n'est pas un vrai test pour cette application car il vérifie uniquement si on atteint la page '/' avec un code 200 et que ce dernier contient une balise h1 avec le text 'Welcome to Symfony'
- Il manque tous les tests pour vérifier que l'application fonctionne correctement.

2. AUDIT DE CODE APPROFONDI

TO-DO-LIST (Initial)

Résultats de l'analyse Codacy :

- **Qualité globale** 29% (note C)

La note est basse, indiquant que le code a des problèmes de qualité qui nécessitent une attention particulière

- **Évolution** 29% :

La complexité du code est élevée, ce qui peut rendre difficile sa maintenance et sa compréhension.

- **Problèmes** 285 :

Il y a un total de 285 problèmes détectés dans le code, dont :

- **Code style** 252 :

Les problèmes de style de code sont nombreux et peuvent impacter la lisibilité du code.

- **Sécurité** 30 :

Les problèmes de sécurité sont présents et peuvent impacter la sécurité du code.

- **Code inutilisé** 3 :

Il y a du code inutilisé qui peut être supprimé pour améliorer la maintenance du code.

Vulnérabilités

- **Sécurité** : Les 30 problèmes de sécurité détectés peuvent impacter la sécurité du code et doivent être résolus en priorité.

- **Complexité** : La complexité élevée du code peut rendre difficile sa maintenance et sa compréhension, ce qui peut impacter la performance de l'application.

Résultats de l'analyse Profiler Symfony :

3. AMELIORATION DU CODE

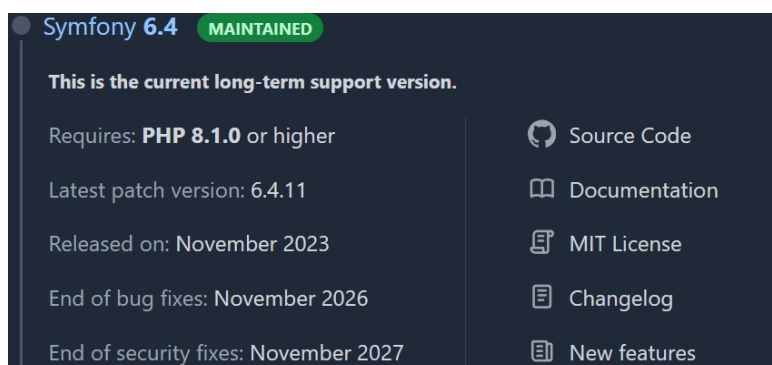
Migration vers Symfony 6.4 LTS

L'application étant sous une version de Symfony 3.1 (plus supportée).

Elle ne peut donc pas bénéficier des améliorations du framework. Pour résoudre les problèmes d'obsolescence de certains composants, fonction et méthodes il est crucial de passer à une version récente du framework.

Lorsque nous avons initialisé ce projet, la version la plus récente était la version 6.4. Nous avons donc migré vers cette dernière version.

Cependant, malgré qu'on soit sur une version LTS cette dernière n'aura plus de mise à jour fonctionnelle à partir de Novembre 2026 et n'aura plus de mise à jour sécuritaire en Novembre 2027.



Symfony 6.4 **MAINTAINED**

This is the current long-term support version.

Requires: **PHP 8.1.0** or higher

Latest patch version: 6.4.11

Released on: November 2023

End of bug fixes: November 2026

End of security fixes: November 2027

[Source Code](#)

[Documentation](#)

[MIT License](#)

[Changelog](#)

[New features](#)

Il peut être intéressant de l'améliorer lors de la sortie de Symfony 7.4 LTS.

La roadmap de symfony ne mentionne pas encore le date de lancement de cette release.

Voici quelque modification mis au goût du jour :

Controllers

Les Controllers « extends » désormais l'AbstractController en lieu et place de la classe Controller.

```
10 class SecurityController extends AbstractController
```

Nous avons également mis en place les écritures appelées "attribut" en PHP utilisé depuis PHP 8.0:


```
#[IsGranted('IS_AUTHENTICATED')]
#[Route('/new-task', name: 'new.task')]
```

Entity

En ce qui concerne les entités :

Les Task ont été considérées comme une collection qui peut être liée à plusieurs ou un User et un User peut avoir une ou plusieurs Task. Ce champ est de type relation avec l'entité

User.

```
#[ORM\OneToMany(targetEntity: Task::class, mappedBy: 'user', orphanRemoval: true)]
private Collection $tasks;
```

```
#[ORM\ManyToOne(inversedBy: 'tasks')]
#[ORM\JoinColumn(nullable: true)]
private ?User $user = null;
```

Ainsi lors de la création d'une Task qui peut se faire uniquement si l'utilisateur est connecté à un contrôleur qui gère l'assignation à la Task.

```
if ($form->isSubmitted() && $form->isValid()) {
    ... $task->setCreatedAt(new DateTime());
    ... $task->setUser($user);
    ... $entityManager->persist($task);
    ... $entityManager->flush();

    ... $this->addFlash('success', 'TaskOld Enregistré correctement');
    ... return $this->redirectToRoute('list.task', [], Response::HTTP_SEE_OTHER);
}
```

Pour les anciennes Tasks n'ayant pas d'utilisateur, une requête SQL via une migration a été créée pour chercher et mettre en anonyme toutes les Task n'ayant pas d'id.

```
public function up(Schema $schema): void
{
    $this->addSql('ALTER TABLE task ADD COLUMN user_id INT DEFAULT NULL');
    $this->addSql('UPDATE task SET user_id = (SELECT id FROM user WHERE username = \'anonyme\') WHERE user_id IS NULL OR user_id = 0');
    $this->addSql('ALTER TABLE task ADD CONSTRAINT FK_527EDB25A76ED395 FOREIGN KEY (user_id) REFERENCES user (id)');
}
```

Mise en place l'unicité sur le champs email de l'entité User.

```
18 #[UniqueEntity(fields: ['email'], message: 'Il y a déjà un compte avec cette email')]
```

Sécurité

L'accès aux parties sensibles du site a été restreint aux utilisateurs authentifiés. Seul la page de login est accessible de manière anonyme.

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

Un système d'autorisation a été mis en place avec

- ROLE USER
- ROLE ADMIN

Ces rôles sont choisis à la création du compte.

L'administration des comptes est disponible uniquement aux utilisateurs possédant le rôle ROLE ADMIN.

J'ai également ajouté une couche de sécurité en utilisant les attributs avec IsGranted afin de vérifier que ce dernier possède bien le rôle approprié pour aller sur la section demandé

En plus de ce point une logique a été mis en place limitant la suppression et modification du Task uniquement par l'auteur du Task et la suppression d'un Task anonyme

Uniquement par les utilisateurs ayant le ROLE ADMIN.

```

#[IsGranted('IS_AUTHENTICATED')]
#[Route("/{id}/deleted', name: 'deleted.task', methods: ['POST'])]
public function delete(Request $request, #[CurrentUser()] ?User $user, Task $task, EntityManagerInterface
$entityManager): Response
{
    if (in_array("ROLE_ADMIN", $user->getRoles()) && $task->getUser()->getUsername() === "anonyme" || $user ===
    $task->getUser()) {
        $this->addFlash('success', 'Task supprimé correctement');
        if ($this->isCsrfTokenValid('delete' . $task->getId(), $request->getPayload()->getString('_token'))) {
            $entityManager->remove($task);
            $entityManager->flush();

            $this->addFlash('success', 'Task supprimé correctement');
            return $this->redirectToRoute('list.task', [], Response::HTTP_SEE_OTHER);
        }
    }
    if ($user !== $task->getUser()) {
        $this->addFlash('error', 'Vous n'êtes pas le créateur de cette tâches');
        return $this->redirectToRoute('list.task', [], Response::HTTP_SEE_OTHER);
    }

    You, le mois dernier via PR #12 • feat(Anomalie):TDLWT-11-Correcting-Anomalies. ...
    return $this->redirectToRoute('list.task', [], Response::HTTP_SEE_OTHER);
}

```

Les formes de Symfony (FormType) intègrent directement un CSRF (Cross-Site Request Forgery) pour protéger contre les attaques de type CSRF.

Mais pour le login ayant fait la page sans FormType car je voulais que ce soit security.yaml qui gère la connexion j'ai donc ajouté la ligne suivante pour assurer la protection :

```

<input type="hidden" name="_csrf_token" value="{% csrf_token('authenticate') %}">

```

4. AUDIT DE CODE APPROFONDI (2)

Afin d'assurer le suivi qualité du code, une série de contrôles à été mises en place de façon à réduire la dette technique de

- Implémentation de nouveaux tests unitaires et fonctionnels
- L'utilisation constante d'un outil de revue du code avant déploiement (codacy)
- Implémentation d'outils à savoir: Profiler Symfony.

TestUnitaire et Fonctionnel

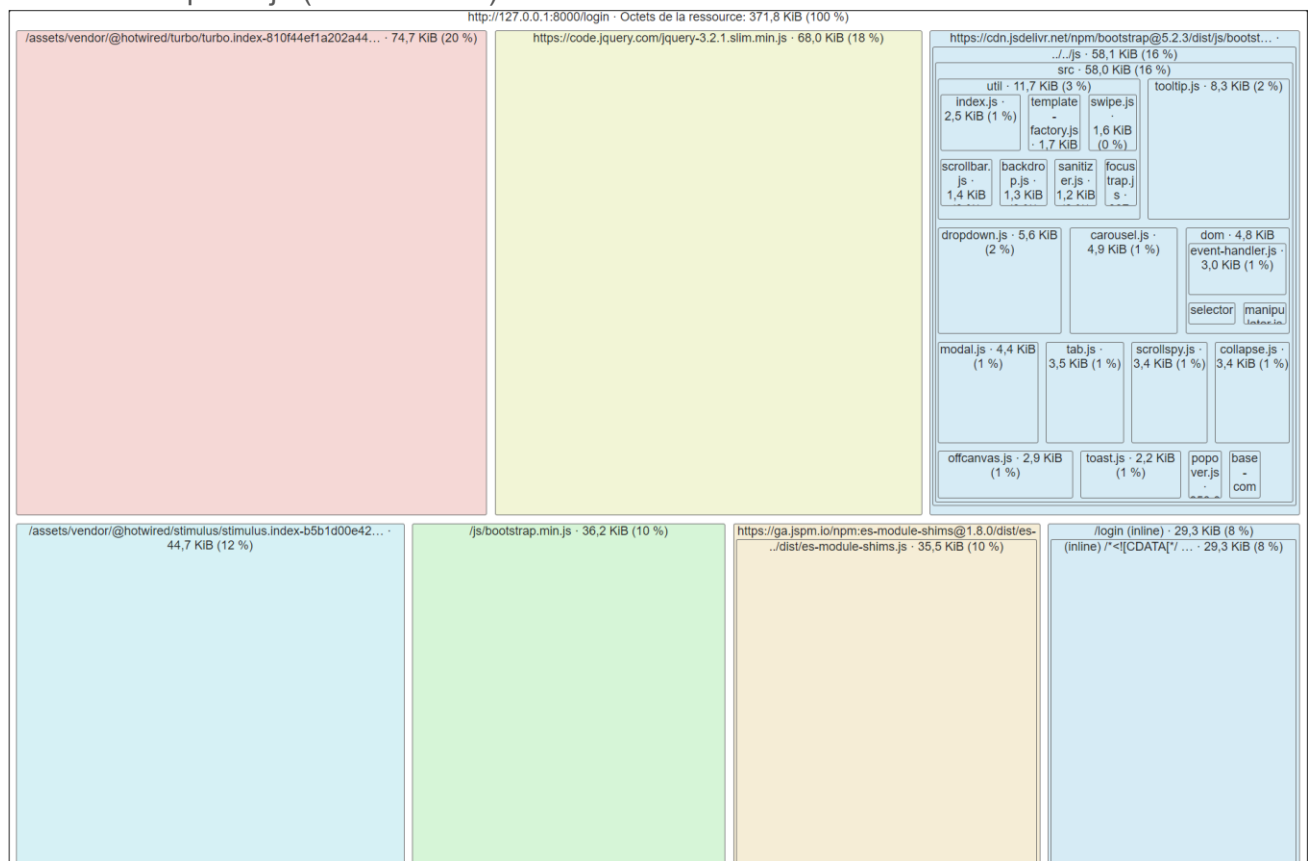
Nous avons réalisé des tests unitaires et fonctionnels avec PHP-unit.

Le taux de couverture est de 92,82% pour les lignes

Packer Optionnel :

Cdn Poppers.js (19,6 KiB)

Cdn Bootstrap.min.js (59 + 36.2KiB)



Résultats de l'analyse Codacy :

• Qualité globale

- To-Do-List : 29% (note C)
- To-Do-List-With-Test : 11% (note B)

La qualité globale du code a augmenté, passant d'une note C à une note B. Cela signifie que le code a été amélioré et que les problèmes de qualité sont moins graves.

Nombre de problèmes

- To-Do-List : 285 problèmes
- To-Do-List-With-Test : 725 problèmes

Le nombre de problèmes a augmenté, mais cela peut être dû à la présence de tests qui ont révélé des problèmes supplémentaires.

Répartition des problèmes

- To-Do-List :
 - Code style : 252 problèmes
 - Sécurité : 30 problèmes
 - Code inutilisé : 3 problèmes
- To-Do-List-With-Test :
 - Code style : 700 problèmes
 - Sécurité : 14 problèmes
 - Code inutilisé : 11 problèmes

Les problèmes de code style et de code inutilisé sont plus nombreux dans le deuxième projet, mais les problèmes de sécurité sont moins nombreux.

Vulnérabilités

- To-Do-List : Les problèmes de sécurité et de complexité sont des vulnérabilités majeures.
- To-Do-List-With-Test : Les problèmes de sécurité sont toujours une vulnérabilité, mais moindre avec une possibilité d'améliorer facilement les codes inutilisés

En résumé :

Le deuxième projet a une qualité globale meilleure que le premier, mais il y a plus de problèmes à résoudre. Les problèmes de sécurité sont moins nombreux, mais les problèmes de code style et de code inutilisé sont plus nombreux.

Résultats de l'analyse Profiler Symfony :

Information Utile :

Tous d'abord il faut savoir comment lire les données pour cela il est important de savoir qu'il n'y a pas de normes universelles pour déterminer ce qui est lent ou rapide en termes de temps d'exécution d'une page web.

Cependant, il existe des lignes directrices et des recommandations qui peuvent aider à évaluer la performance d'une page web.

Voici quelques-unes des lignes directrices couramment utilisées :

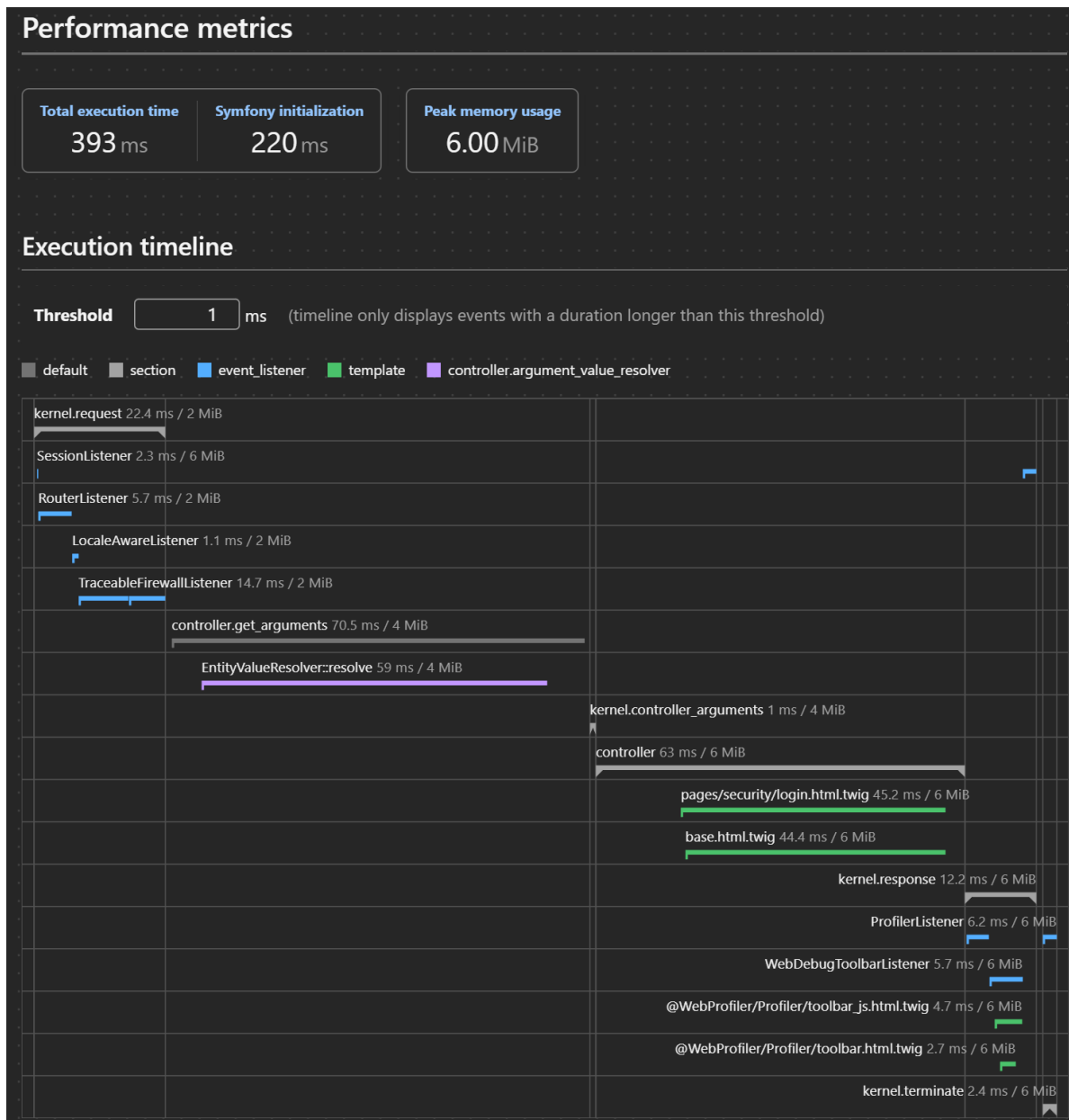
- **Google** : recommande que les pages web soient chargées en moins de 3 secondes pour une expérience utilisateur optimale.
- **W3C** : recommande que les pages web soient chargées en moins de 2 secondes pour une expérience utilisateur satisfaisante.
- **Amazon** : a constaté que chaque seconde de retard dans le chargement d'une page peut entraîner une perte de 1% des ventes.
- **Yahoo!** : a constaté que chaque seconde de retard dans le chargement d'une page peut entraîner une perte de 2,5% des ventes.

En termes de temps d'exécution, voici quelques lignes directrices générales :

- **Moins de 100 ms** : Très rapide, idéal pour les applications en temps réel.
- **100 ms à 500 ms** : Rapide, acceptable pour la plupart des applications web.
- **500 ms à 1 seconde** : Moyen, peut être acceptable pour certaines applications web, mais peut causer des problèmes d'expérience utilisateur.
- **1 seconde à 2 secondes** : Lent, peut causer des problèmes d'expérience utilisateur et de conversion.
- **Plus de 2 secondes** : Très lent, peut entraîner des pertes de ventes et une expérience utilisateur médiocre.

Il est important de noter que ces lignes directrices sont générales et peuvent varier en fonction du type d'application, de la complexité de la page et des attentes des utilisateurs.

De plus le Profiler Symfony crée une perturbation en terme de calcul et prend donc également de la ressource (Collecte de données, Analyse des données, Stockage des données, Affichage des données)



Page de login :

Temps d'exécution total : 393 ms, ce qui est relativement rapide.

Initialisation de Symfony : 220 ms, ce qui représente environ 56% du temps d'exécution total.

Utilisation de mémoire : 6,00 MiB, ce qui est une quantité raisonnable pour une page de login.

Événements les plus coûteux :

- controller.get_arguments : 70,5 ms / 4 MiB,
- EntityValueResolver::resolve : 59 ms / 4 MiB
- pages/security/login.html.twig : 45,2 ms / 6 MiB,

Ce qui représente environ 18%, 15% et 12% du temps d'exécution total.



Page de d'accueil :

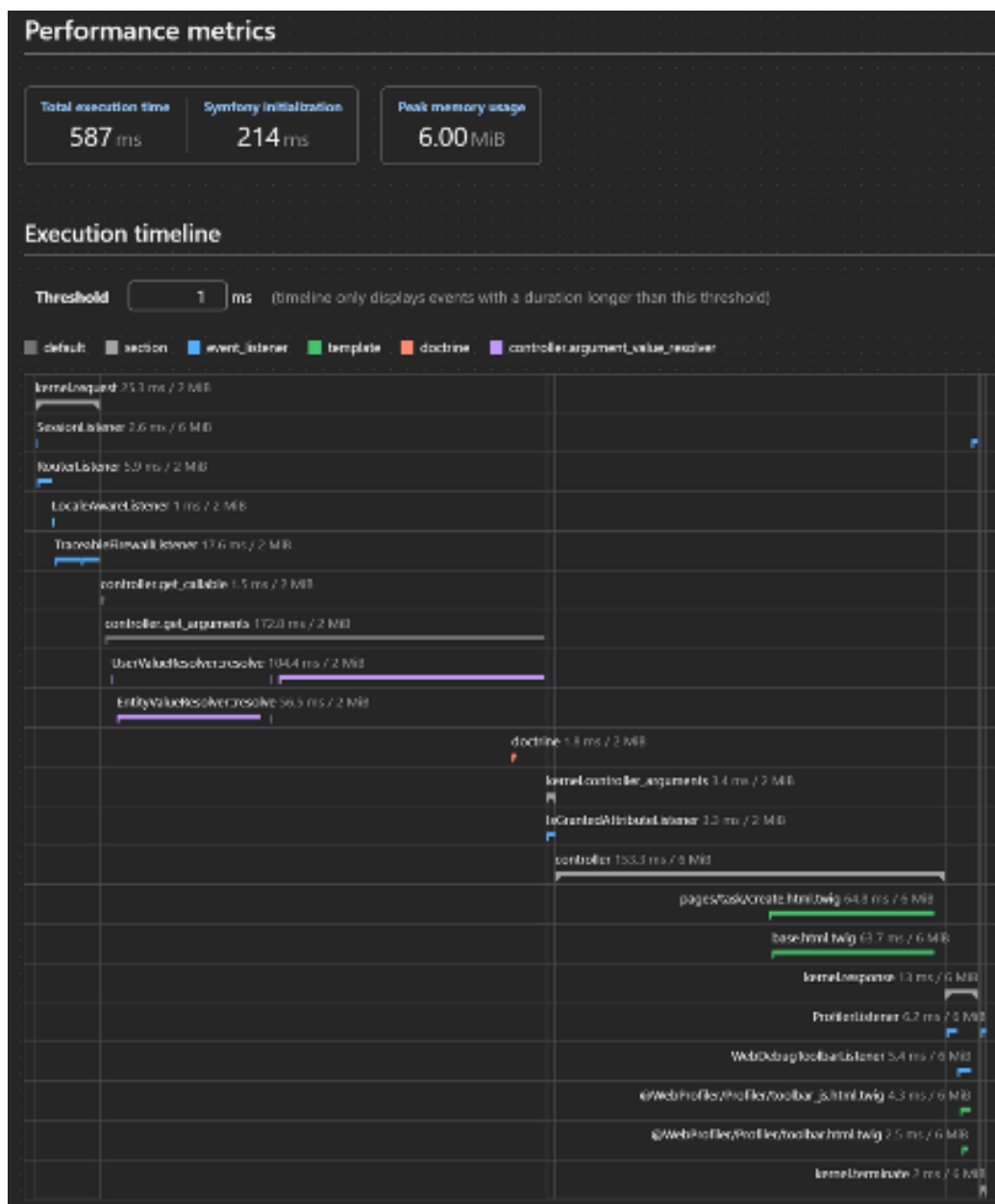
Temps d'exécution total : 484 ms, ce qui est relativement rapide.

Initialisation de Symfony : 228 ms, ce qui représente environ 47% du temps d'exécution total.

Utilisation de mémoire : 2,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- kernel.controller_arguments: 167,1 ms / 2 MiB,
- IsGrantedAttributeListener : 167 ms / 2 MiB
- controller: 47,4 ms / 2 MiB,
- Ce qui représente environ 34%, 34% et 10% du temps d'exécution total.



Page création d'une Task :

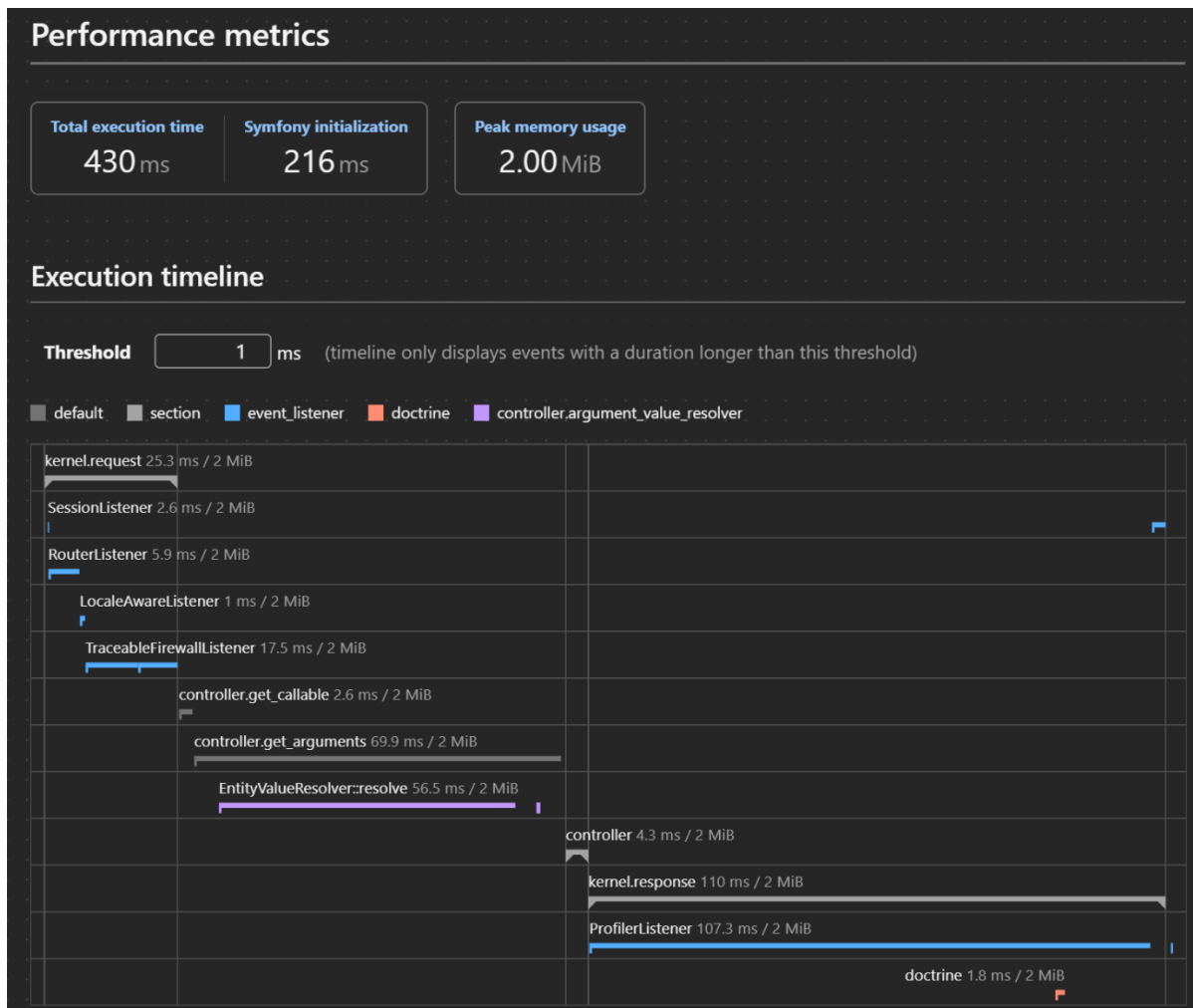
Temps d'exécution total : 587 ms, ce qui est relativement rapide.

Initialisation de Symfony : 214 ms, ce qui représente environ 36% du temps d'exécution total.

Utilisation de mémoire : 6,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- controller.get_arguments: 172,8 ms / 2 MiB
 - UserValueResolver::resolve: 104,4 ms / 2 MiB,
 - EntityValueResolver::resolve: 56,5 ms / 2 MiB,
 - controller : 153,3 ms / 6 MiB
-
- Ce qui représente environ 29%, 18%, 10% et 26% du temps d'exécution total.



Création d'une Task Valide :

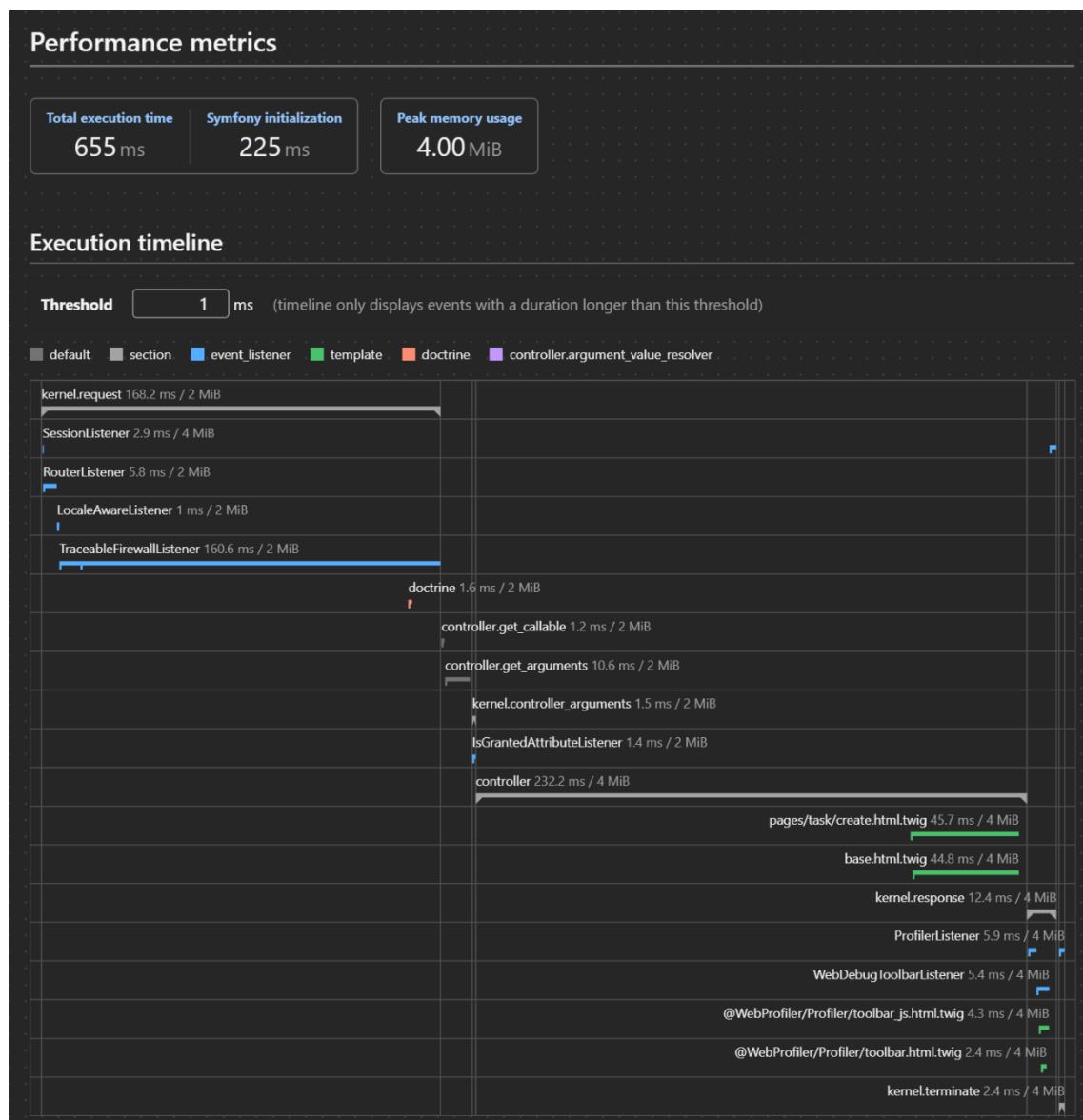
Temps d'exécution total : 430 ms, ce qui est relativement rapide.

Initialisation de Symfony : 216 ms, ce qui représente environ 50% du temps d'exécution total.

Utilisation de mémoire : 2,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- kernel.response: 110 ms / 2 MiB
- ProfilerListener : 107,3 ms / 2 MiB,
- controller.get_arguments : 69,9 ms / 2 MiB
- EntityValueResolver::resolve : 56,5 ms / 2 MiB
- Ce qui représente environ 26%, 25%, 16% et 13% du temps d'exécution total.



Création d'une Task Non Valide :

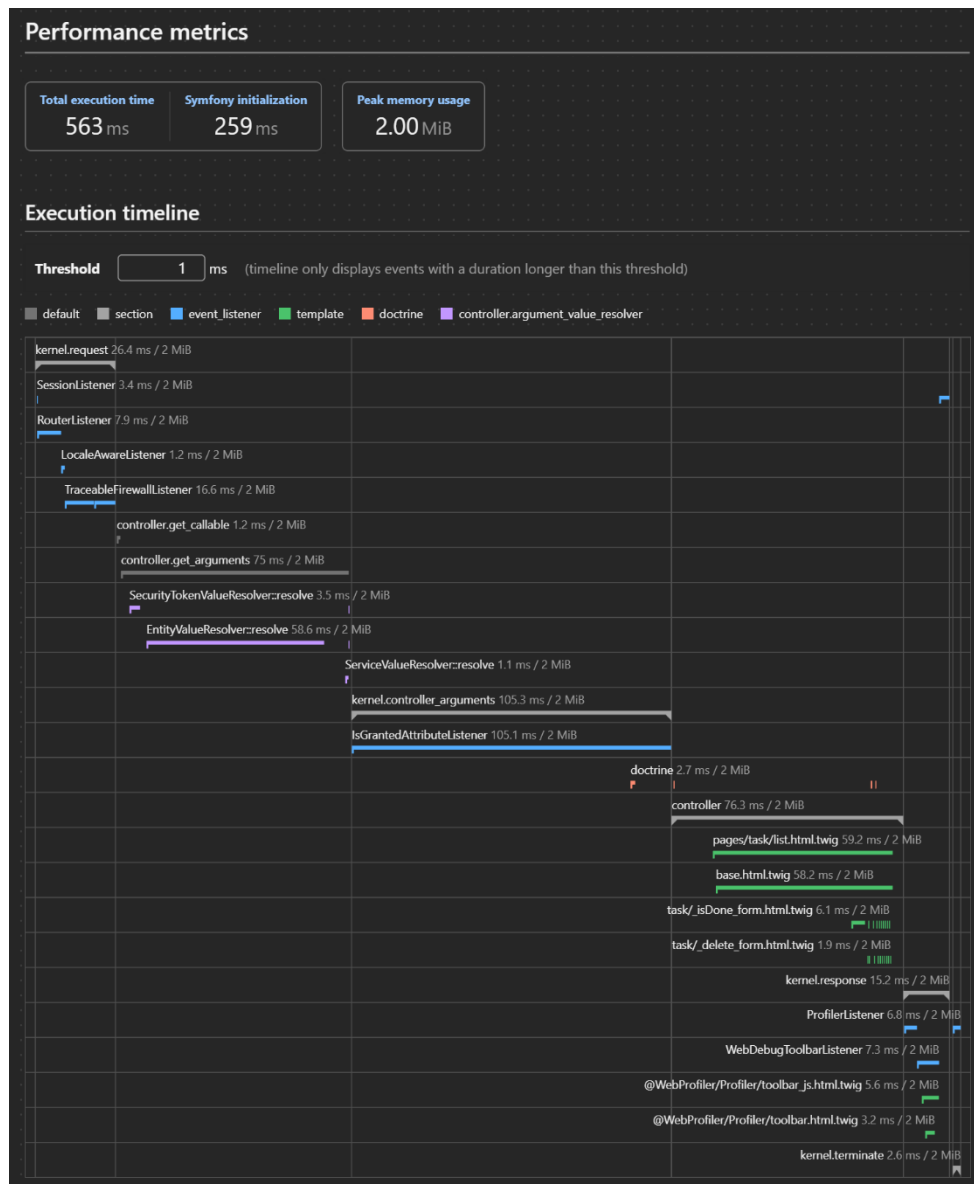
Temps d'exécution total : 655 ms, ce qui reste relativement rapide malgré la recherche de résolution d'une information qui n'y est pas alors qu'elle a été donnée obligatoire.

Initialisation de Symfony : 225 ms, ce qui représente environ 34% du temps d'exécution total.

Utilisation de mémoire : 4,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- kernel.request : 168,2 ms / 2 MiB,
- TraceableFirewallListener : 160,6 ms / 2 MiB,
- controller : 232,2 ms / 4 MiB,
- Ce qui représente environ 26%, 24%, et 35% du temps d'exécution total.



Page List des Task all:

Temps d'exécution total : 563 ms, ce qui est relativement rapide.

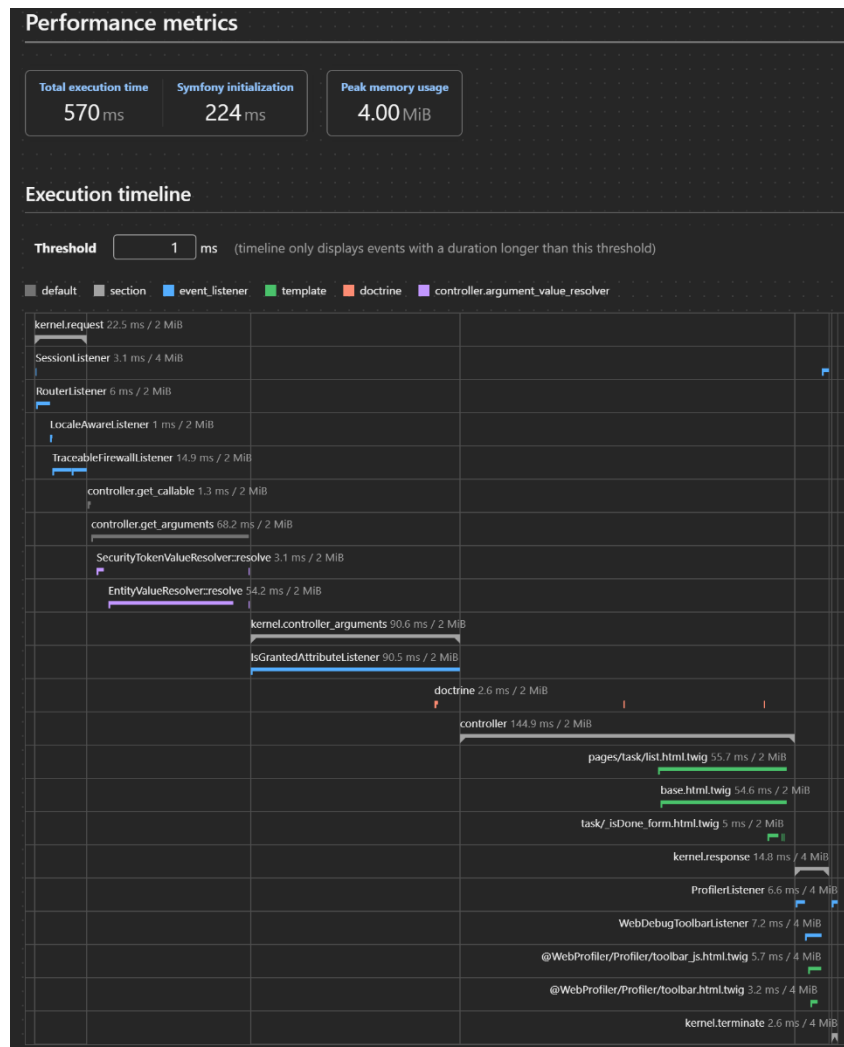
Initialisation de Symfony : 259 ms, ce qui représente environ 46% du temps d'exécution total.

Utilisation de mémoire : 2,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- kernel.controller_arguments : 105,3 ms / 2 MiB
- IsGrantedAttributeListener : 105,1 ms / 2 MiB,
- EntityValueResolver::resolve : 58,6 ms / 2 MiB
- controller.get_arguments : 75 ms / 2 MiB

- Ce qui représente environ 19%, 19%, 10%, et 13% du temps d'exécution total.



Page List des Task avec une request q=notDone:

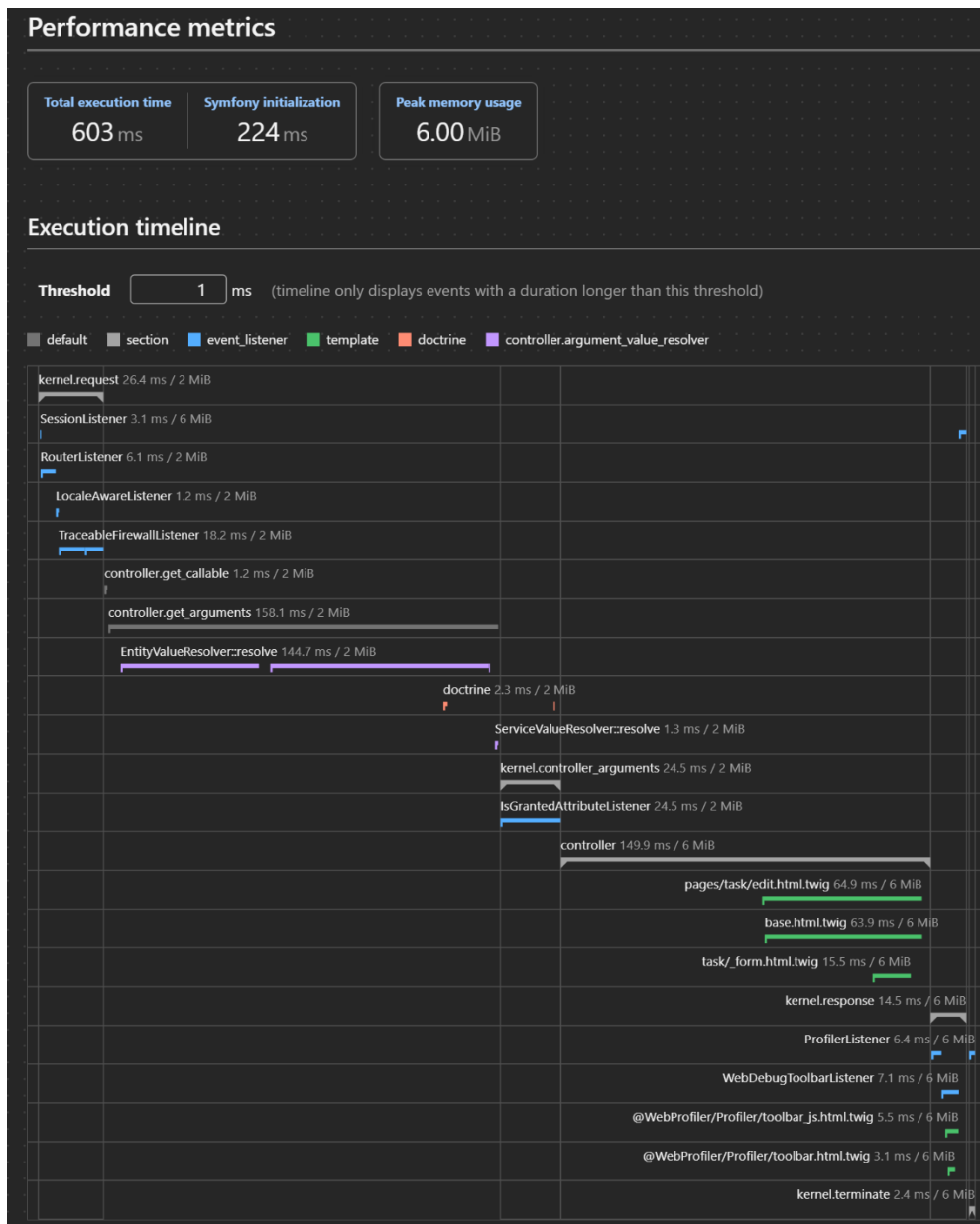
Temps d'exécution total : 570 ms, ce qui est relativement rapide.

Initialisation de Symfony : 224 ms, ce qui représente environ 39% du temps d'exécution total.

Utilisation de mémoire : 4,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- kernel.controller_arguments : 90,6 ms / 2 MiB,
- IsGrantedAttributeListener : 90,5 ms / 2 MiB
- controller : 144,9 ms / 2 MiB
- EntityValueResolver::resolve : 54,2 ms / 2 MiB
- Ce qui représente environ 16%, 16%, 25% et 10% du temps d'exécution total.



Page Edit Task :

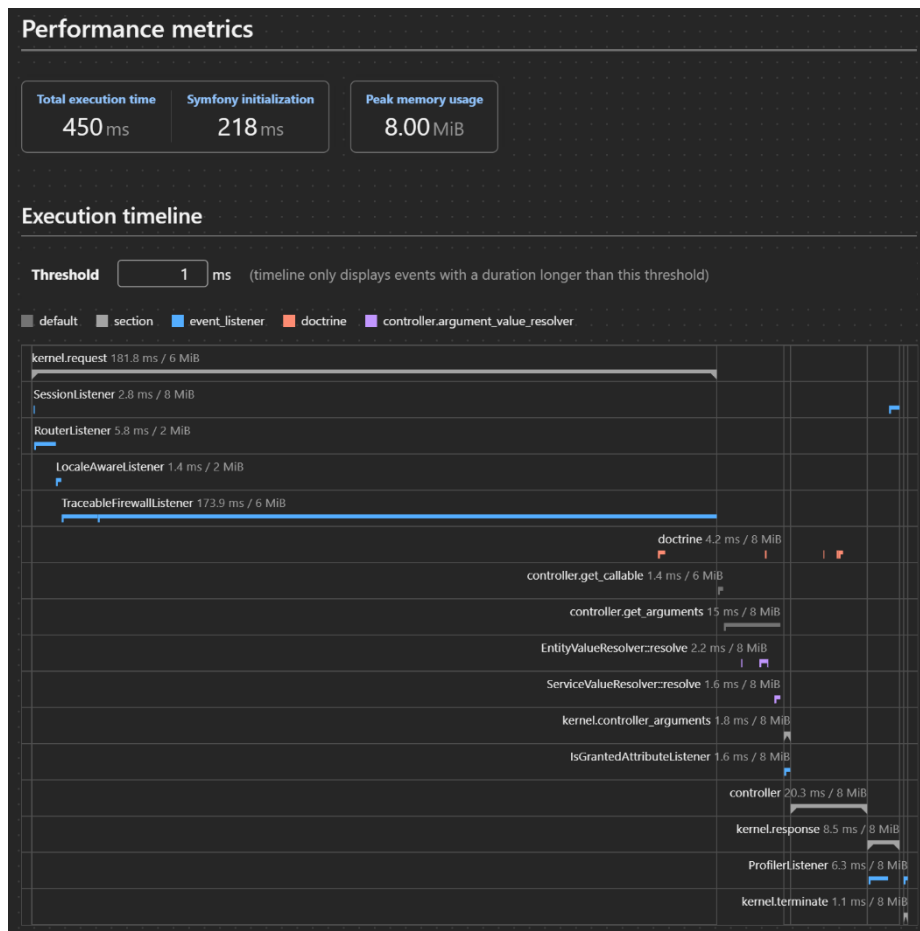
Temps d'exécution total : 603 ms, ce qui est relativement rapide.

Initialisation de Symfony : 224 ms, ce qui représente environ 37% du temps d'exécution total.

Utilisation de mémoire : 6,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- controller.get_arguments : 158,1 ms / 2 MiB
- EntityValueResolver::resolve : 144,7 ms / 2 MiB
- controller : 149,9 ms / 6 MiB
- Ce qui représente environ 26%, 24%, et 25% du temps d'exécution total.



Page Delete Task :

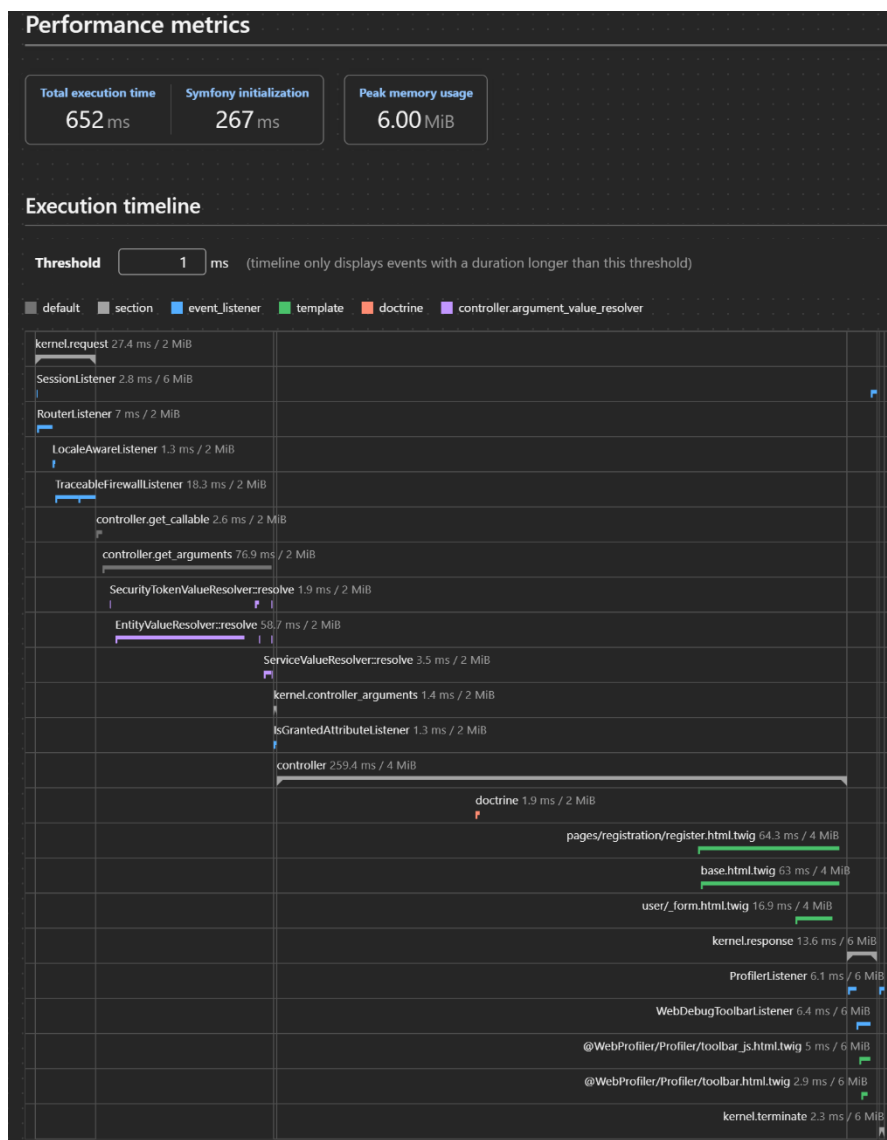
Temps d'exécution total : 450 ms, ce qui est relativement rapide.

Initialisation de Symfony : 218 ms, ce qui représente environ 48% du temps d'exécution total.

Utilisation de mémoire : 8,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- kernel.request : 181,8 ms / 6 MiB
- TraceableFirewallListener : 173,9 ms / 6
- Ce qui représente environ 40%, et 39% du temps d'exécution total.



Page Création d'un User :

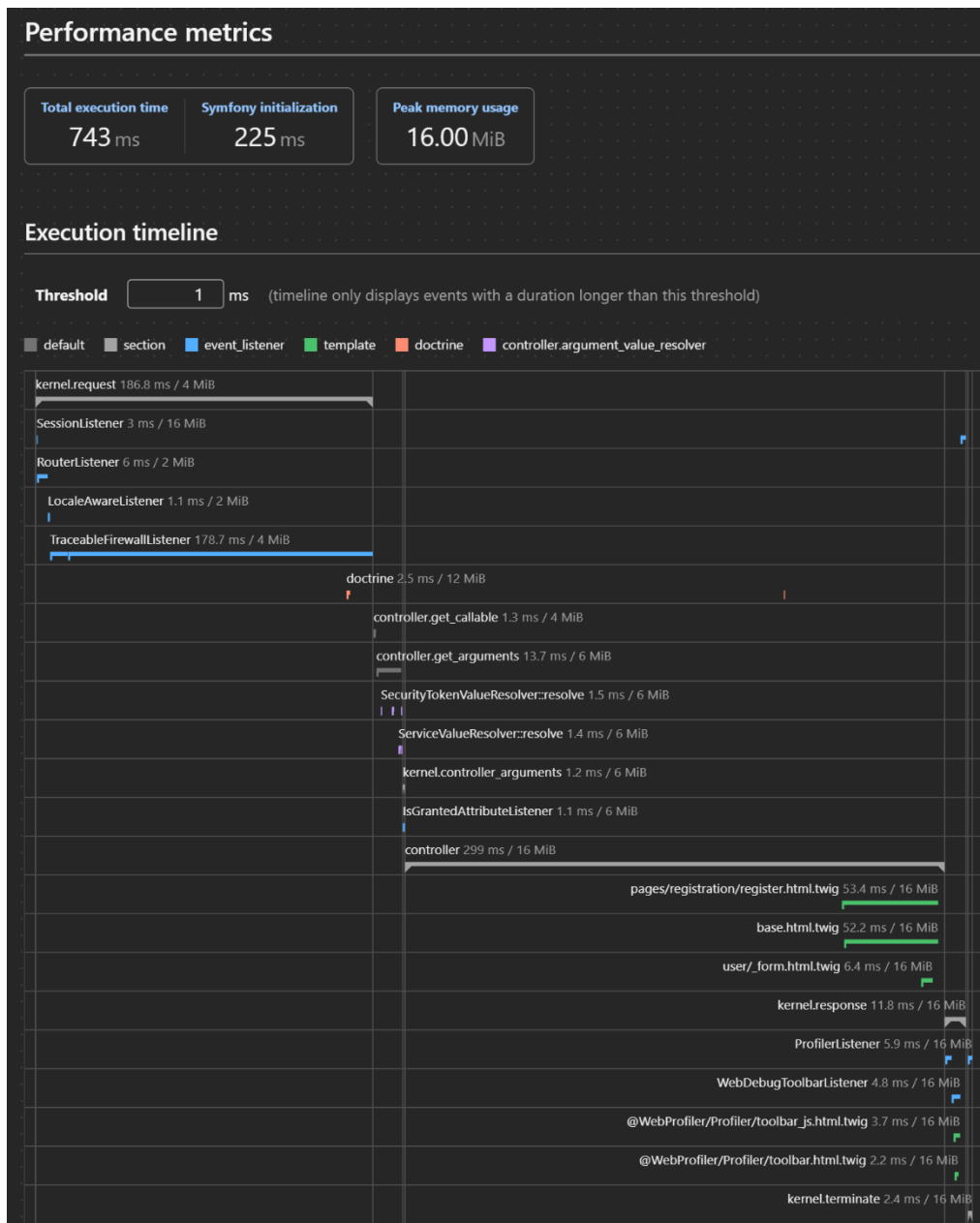
Temps d'exécution total : 652 ms, ce qui est relativement rapide.

Initialisation de Symfony : 267 ms, ce qui représente environ 41% du temps d'exécution total.

Utilisation de mémoire : 6,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- controller : 259,4 ms / 4 MiB,
- controller.get_arguments : 76,9 ms / 2 MiB
- EntityValueResolver::resolve : 58,7 ms / 2 MiB
- Ce qui représente environ 40%, 12% et 9% du temps d'exécution total.



Création d'un User Non Valid :

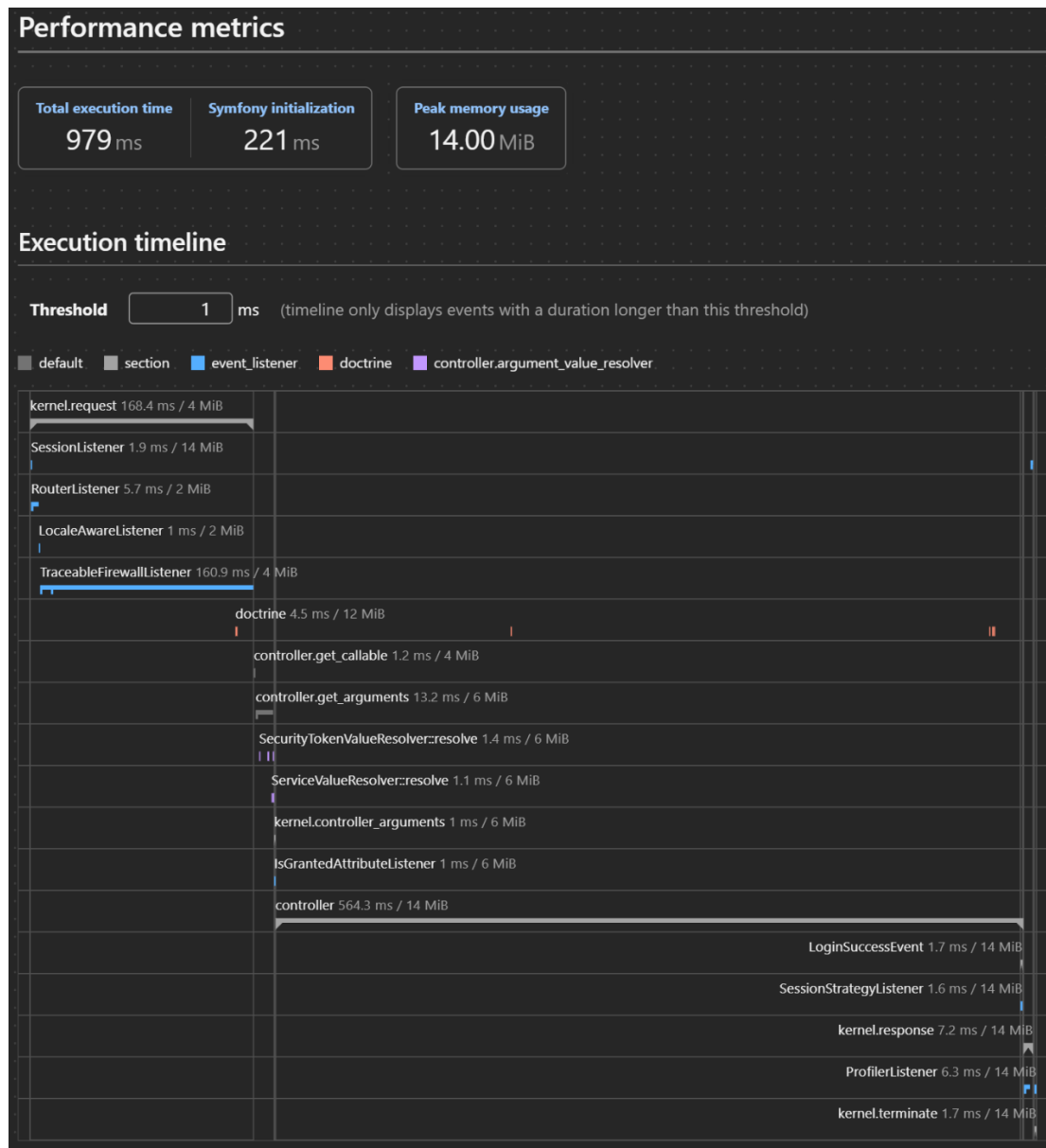
Temps d'exécution total : 743 ms, ce qui est relativement rapide.

Initialisation de Symfony : 225 ms, ce qui représente environ 30% du temps d'exécution total.

Utilisation de mémoire : 16,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- controller : 299 ms / 16 MiB,
- kernel.request : 186,8 ms / 4 MiB,
- TraceableFirewallListener : 178,7 ms / 4 MiB,
- Ce qui représente environ 40%, 25% et 24% du temps d'exécution total.



Création d'un User Valid :

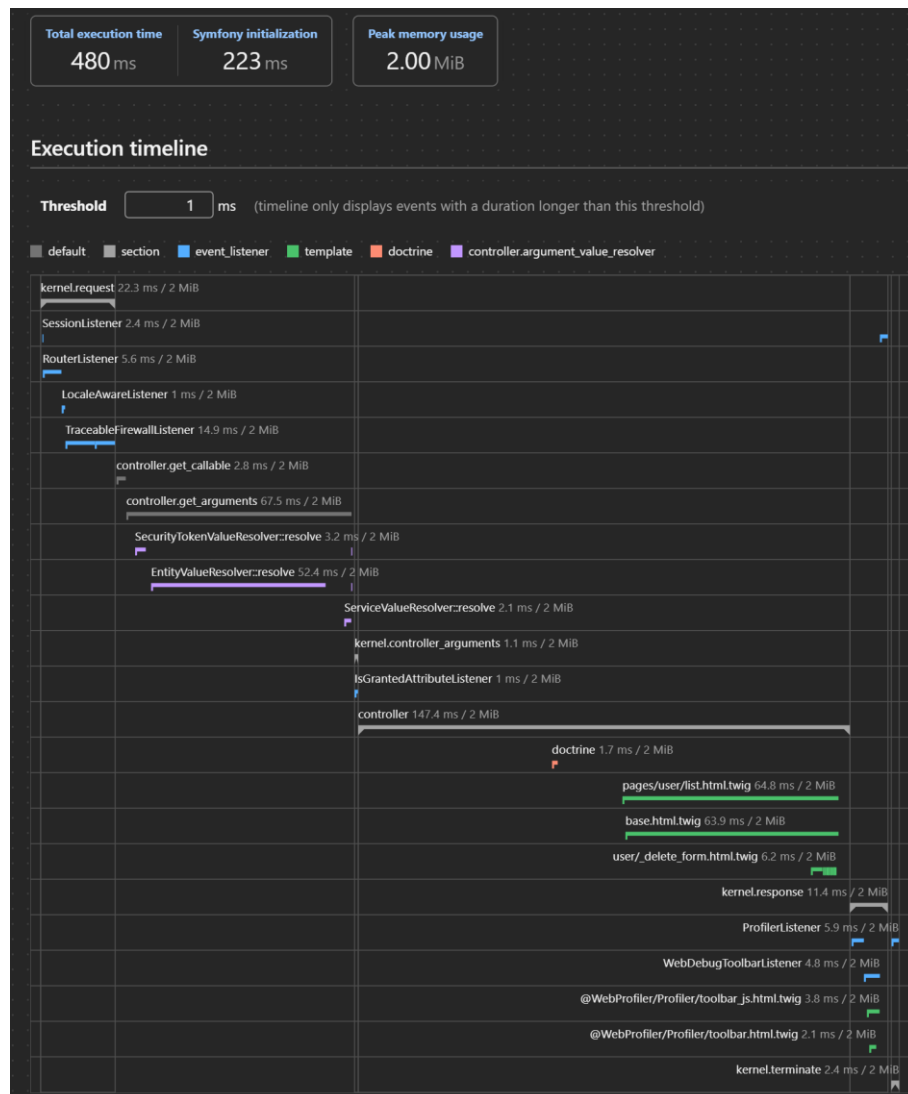
Temps d'exécution total : 979 ms, ce qui est relativement rapide.

Initialisation de Symfony : 221 ms, ce qui représente environ 23% du temps d'exécution total.

Utilisation de mémoire : 14,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- controller : 564,3 ms / 14 MiB,
- kernel.request : 168,4 ms / 4 MiB,
- TraceableFirewallListener : 160,9 ms / 4 MiB,
- Ce qui représente environ 58%, 17% et 16% du temps d'exécution total.



Page List des User :

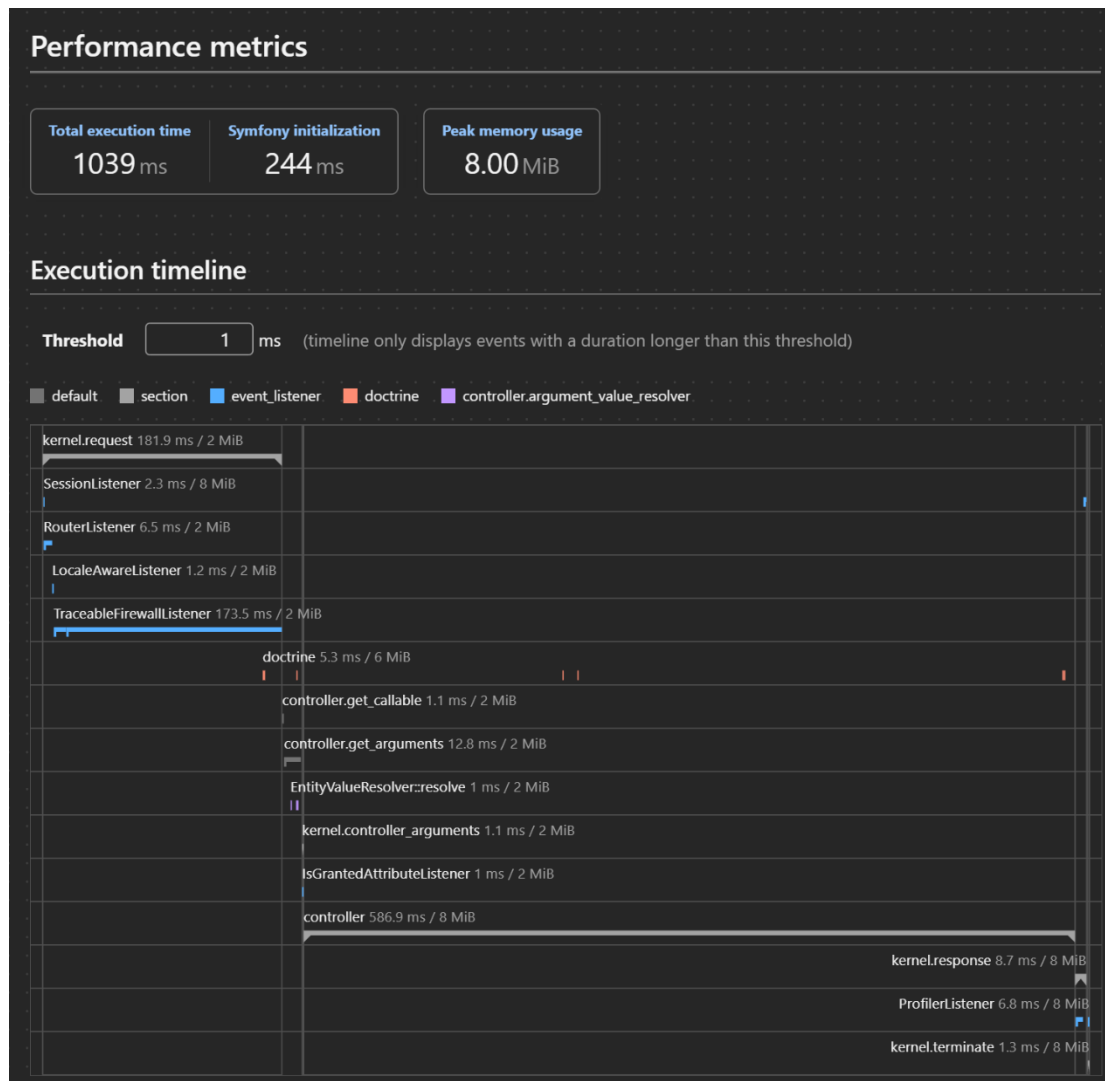
Temps d'exécution total : 480 ms, ce qui est relativement rapide.

Initialisation de Symfony : 223 ms, ce qui représente environ 46% du temps d'exécution total.

Utilisation de mémoire : 2,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- controller.get_arguments : 67,5 ms / 2 MiB,
- EntityValueResolver::resolve : 52,4 ms / 2 MiB,
- controller : 147,4 ms / 2 MiB,
- Ce qui représente environ 14%, 11% et 31% du temps d'exécution total.



Page Edit User :

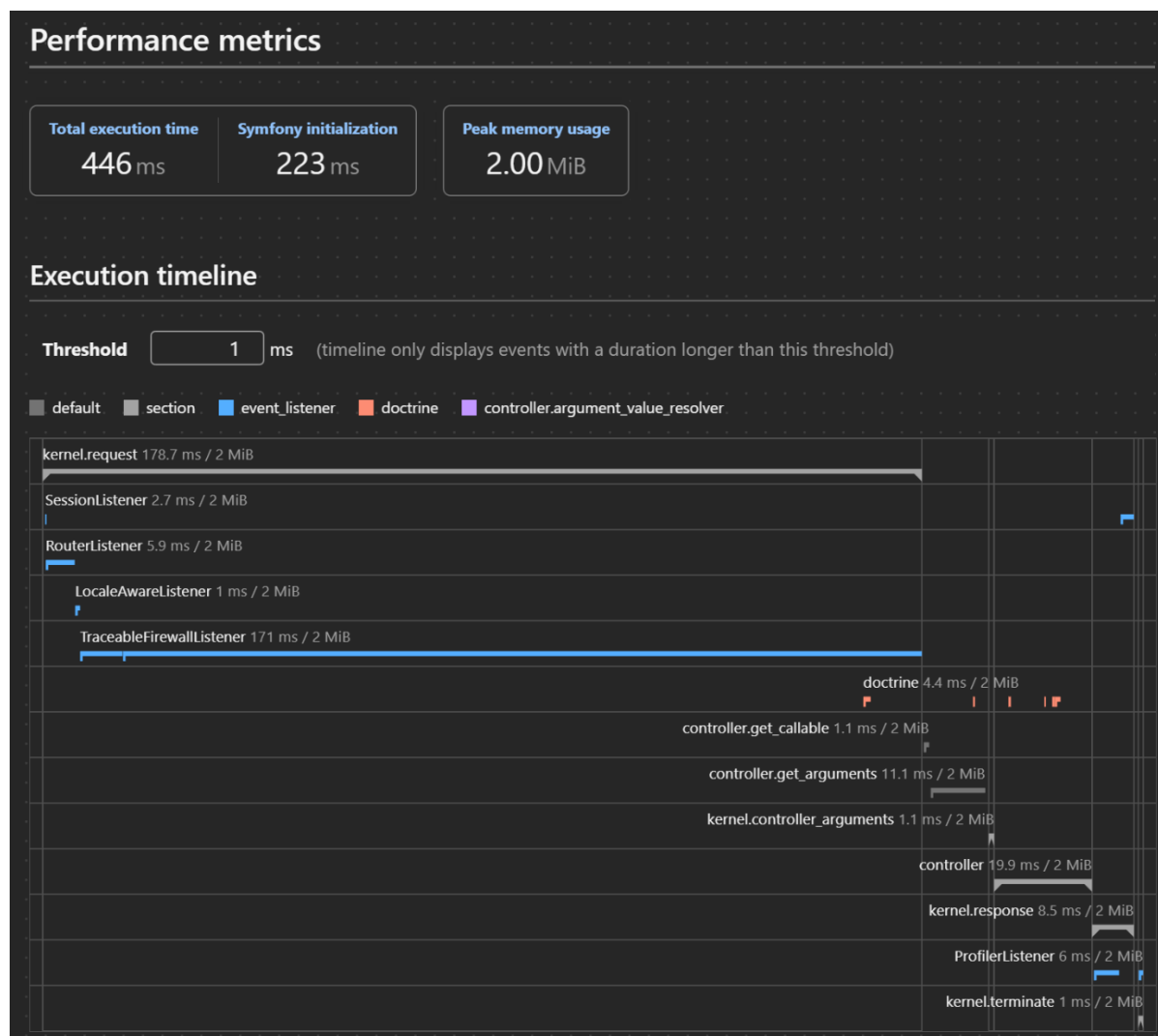
Temps d'exécution total : 1039 ms, ce qui reste relativement rapide mais à contrôler lors de nouvelle mise à jour.

Initialisation de Symfony : 244 ms, ce qui représente environ 24% du temps d'exécution total.

Utilisation de mémoire : 8,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- controller : 586,9 ms / 8 MiB,
- kernel.request : 181,9 ms / 2 MiB,
- TraceableFirewallListener : 173,5 ms / 2 MiB,
- Ce qui représente environ 57%, 18% et 17% du temps d'exécution total.



Page Deleted User :

Temps d'exécution total : 446 ms, ce qui reste relativement rapide mais à contrôler lors de nouvelle mise à jour.

Initialisation de Symfony : 223 ms, ce qui représente environ 50% du temps d'exécution total.

Utilisation de mémoire : 2,00 MiB, ce qui est une quantité raisonnable pour une page web.

Événements les plus coûteux :

- kernel.request : 178,7 ms / 2 MiB,
- TraceableFirewallListener : 171 ms / 2 MiB,
- Ce qui représente environ 40% et 38% du temps d'exécution total.

Conseils d'optimisation :

- Vérifiez la résolution des arguments du contrôleur et la résolution de la valeur de l'utilisateur pour voir si elles peuvent être optimisées.
- Vérifiez les opérations de doctrine pour voir si elles peuvent être optimisées
- Utilisez des techniques de mise en cache pour réduire le nombre d'appels à la base de données.
- Utilisez des techniques de réduction du nombre d'appels à la base de données pour réduire la charge sur la base de données.