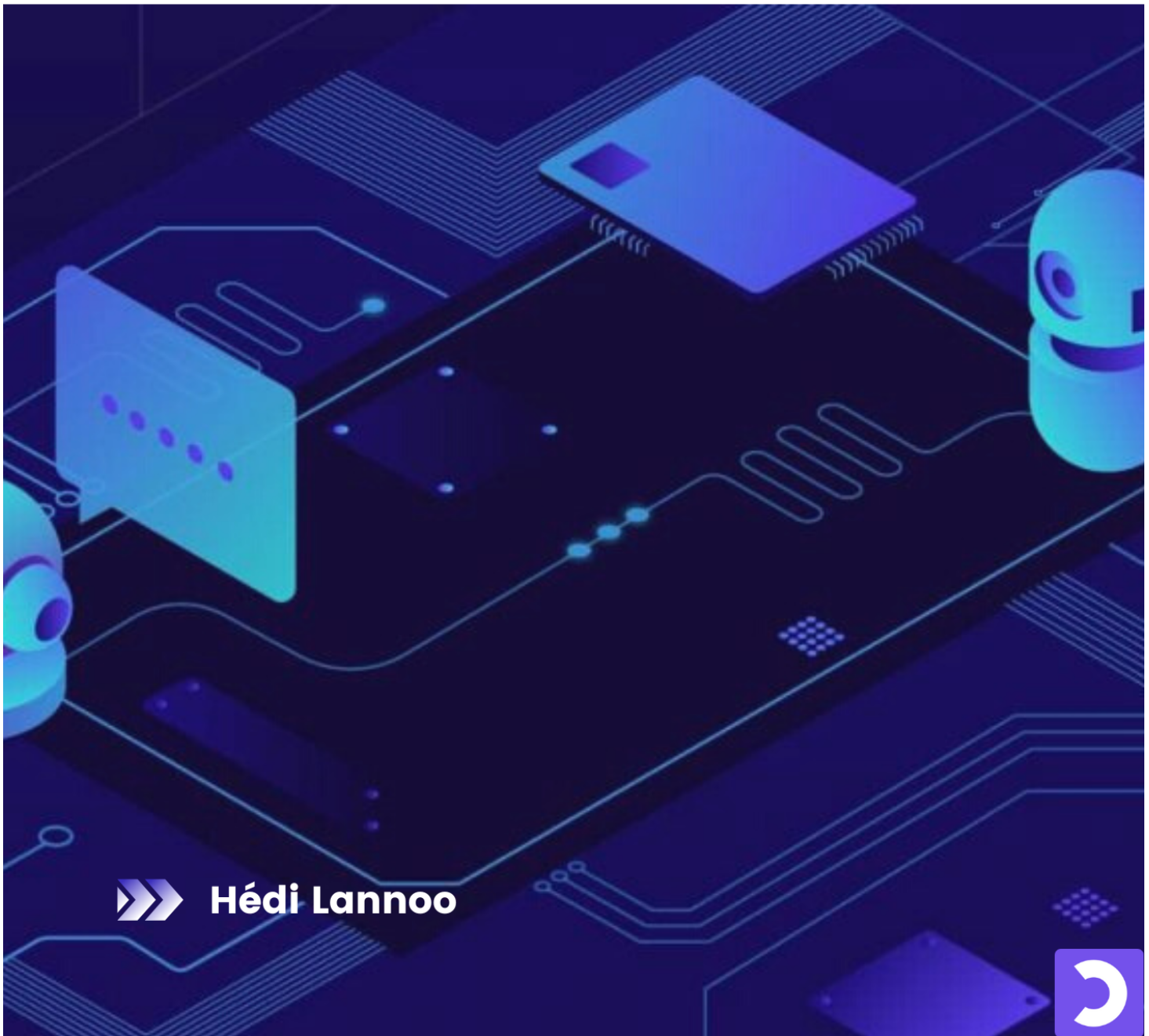
2022

AUDIT : TODO & CO



Hédi Lannoo

AVANT-PROPOS

ToDo & Co est une toute nouvelle startup dont le cœur de métier est une application permettant de gérer ses tâches quotidiennes.

Cette dernière ayant été réalisée à toute vitesse afin d'élaborer un MVP (Minimum Viable Product) pour de potentiels investisseurs, elle concède certaines lacunes.

Ainsi, après une levée de fonds, l'entreprise a décidé d'améliorer la qualité et la performance de son application.

C'est pourquoi l''audit réalisé ici pour la société ToDo & Co à pour but de faire un état des lieux de l'application réalisée sous PHP avec le Framework Symfony, avant et après optimisation de cette dernière.

Ainsi, cet audit comparera bien entendu de manière qualitative le code, la performance et prendra également en compte l'aspect expérience utilisateur.

SOMMAIRE

I - ETAT DES LIEUX

1.1 - Environnement

Les spécificités initiales du MVP sont disponibles sur Github à l'adresse suivante : Lien

Version de PHP	5.5.9
Version de Symfony	3.1

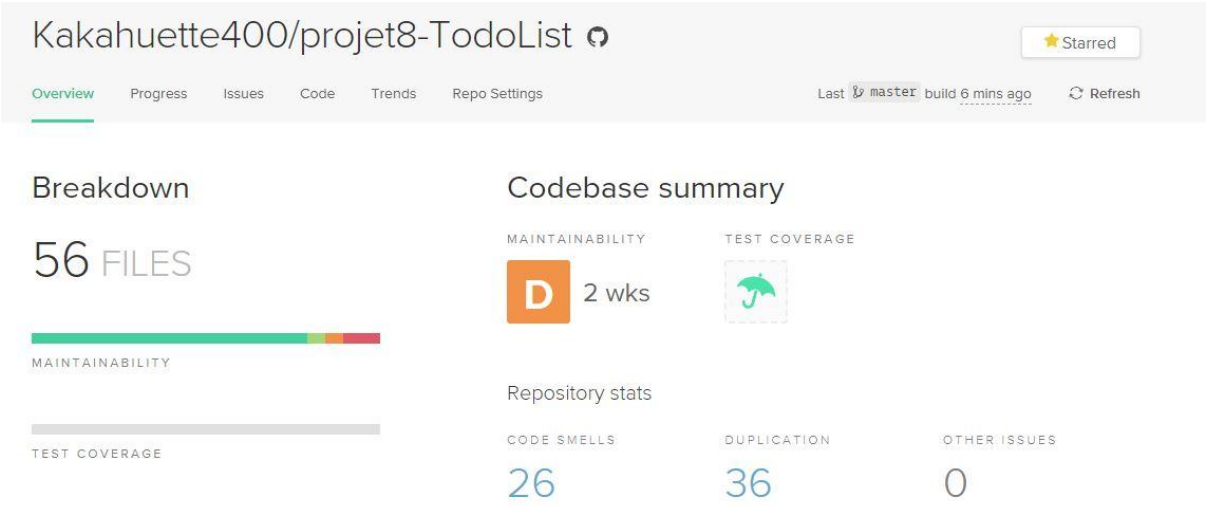
Après installation du projet, celui-ci soulève de nombreuses exceptions au lancement de ce dernier, la version de Symfony et de PHP n'étant plus maintenue :

- Last Supported releases Symfony : Lien
- Last Supported versions PHP : Lien

Cependant ce projet semble fonctionner sous PHP 7.2.34 et Symfony 3.4, ainsi le projet a été update via composer sur cette version.

1.2 - Analyse de la qualité du code

Pour analyser la qualité du code initial, CodeClimate a été utilisé, ainsi le site produit ce rapport :



Globalement le projet obtient le grade D, ce qui en soit n'est pas une très bonne évaluation, cependant à la vue du peu de code enregistré sur le projet, ce grade peut très aisément être amélioré.

Sans surprise, les principaux problèmes sont relatifs à la maintenance globale du projet, certaines utilisations étant totalement dépréciées aujourd'hui avec la version de PHP et de son framework Symfony.

On retrouve ainsi principalement des entraves au principe SOLIDE par le biais d'utilisation de code dupliqué ici et là.
De la même manière, une grande partie du code est à refactorer car beaucoup trop long.

Enfin on retrouve également quelques problèmes d'indentation du code relativement mineurs ainsi que que du code inutilisé (ex : swiftmailer).

1.3 - Analyse de la performance

Pour analyser la performance du code nous allons cette fois-ci utiliser le profiler intégré à Symfony. Pour faire cela nous allons tester et lister la vitesse moyenne de chargement de différentes pages ainsi que la mémoire utilisée que nous comparerons plus tard avec la version du projet optimisée.

Analyse 1 : Mise en cache inactive (first landing)

URL	Vitesse d'exécution (/ms)	Initialisation Symfony (/ms)	Mémoire (MiB)	Render Time (/ms)	cache calls	Requêtes
/login	173	38	2.0	5	5	0
/users	211	39	2.0	5	19	1
/users (Data)	292	56	4.0	7	33	1
/users/create	254	50	2.0	18	4	0
/users/{id}/edit	260	38	4.0	17	18	1
/	217	37	2.0	6	18	1
/tasks	312	53	2.0	7	66	2
/tasks(data)	530	72	4.0	10	66	2
/tasks/create	519	55	2.0	21	116	1
/tasks/{id}/edit	386	46	2.0	19	116	2

Analyse 2 : Mise en cache active (second landing)

URL	Vitesse d'exécution (/ms)	Initialisation Symfony (/ms)	Mémoire (MiB)	Render Time (/ms)	cache calls	Requêtes
/login	169	38	2.0	5	4	0
/users	194	37	4.0	5	18	1
/users (data)	215	38	4.0	5	18	1
/users/create	197	37	4.0	15	4	0
/users/{id}/edit	235	40	4.0	15	18	1
/	197	38	2.0	5	18	1
/tasks	214	37	4.0	5	33	2
/tasks(data)	220	37	4.0	5	33	2
/tasks/create	267	37	2.0	15	53	1
/tasks/{id}/edit	245	39	2.0	15	54	2

Après analyse des différents pages de l'application, ce qui en ressort globalement, c'est une vitesse d'exécution moyenne plutôt bonne (> 300 ms), celle-ci est néanmoins à nuancer sur certaines pages avant la mise en cache de certains composant de celle-ci, notamment sur les urls liées aux "tasks".

On remarquera ainsi l'absence de mise en cache sur les pages effectuant des requêtes, ce qui alourdit considérablement le temps de chargement.

1.4 - Sécurité

Sur le projet initial il existe un certains nombre de problèmes liés à la sécurité de l'application et des données qui y sont liées :

- Aucun test n'a été effectué sur l'application.
- Un utilisateur non authentifié peut modifier/créer un autre utilisateur.
- Le formulaire d'édition/création d'un utilisateur ne possède pas de contraintes particulières sur les champs "nom d'utilisateur" et "mot de passe".
- Il n'existe aucune redirection lorsque l'url est erronée.
- Il n'existe aucune vérification au niveau de l'email entré lors de "l'inscription".
- Un utilisateur authentifié peut modifier ou supprimer n'importe quelle tâche. (Aucun utilisateur attaché à la tâche)

1.5 - Expérience Utilisateur

Au-delà des différents problèmes de qualité, de sécurité, de performances il y a également un problème qui est central dans cette application : “l’expérience utilisateur”.

En effet, bien que cette application soit destinée à l’utilisation de professionnels, celle-ci se doit d’être un minimum intuitive et cohérente, cependant il existe de nombreuses améliorations à réaliser :

Problèmes UX	**Réponse à apporter**
Site peu responsive	Faire une application responsive friendly
Pas de réel page/processus d’inscription	Créer une page/processus dédiée uniquement à cela
Peu de dissociation tâche faites/ non faites	Créer un code couleur et un tri automatique
Design peu professionnel / amateur	Utiliser un template professionnel
Liens dysfonctionnels (ex: liste tâches terminées)	Supprimer les liens défectueux ou ne renvoyant vers rien.
Pas de CRUD complets pour les users/tasks	Ajouter l’option delete sous conditions
Pas de notion de hiérarchie d’entreprise (admin/users) dans les actions possibles	Créer différents rôles avec des permissions et droits différents
Informations manquantes à l’affichage	Ajouter la date et le nom de l’utilisateur sur une tâche créer.
Édition du mot de passe et des autres champs dans le même temps.	Créer des formulaires séparés pour l’édition de ces champs
Pas de lien de navigation	Ajouter une barre de menue adaptative
Page d’erreur inexistante	Créer des pages d’erreurs personnalisées

1.6 - Synthèse

L’analyse du MVP ToDo & Co montre principalement un problème lié à l’obsolescence des versions de PHP/SYMFONY et des différents bundles associés et utilisés.
On notera également qu’une refactorisation est nécessaire afin d’optimiser la longueur du code pour ainsi améliorer la vitesse de chargement des pages et dans le même temps de respecter les principes SOLID.

Enfin on retravaillera globalement l’UX afin d’avoir un produit plus simple d’utilisation et plus intuitif pour le consommateur.

II - RÉDUIRE LA DETTE TECHNIQUE

2.1 - Nouveau projet et nouvelles fonctions

Afin d'éviter tout conflit pour arriver à une version "stable" de Symfony qui serait la 6.1.3, j'ai décidé de partir sur un tout nouveau projet en reprenant simplement l'idée de chaque fonction et en les adaptant à la version en cours.

Comme explicité dans le chapitre précédent, de nombreuses améliorations sont à mettre à place, certaines se traduiront par des optimisations visuelles, d'autres seront concrètement implémentées à l'aide de nouvelles fonctions.

Ainsi à la liste de l'état des lieux, vient s'ajouter des fonctionnalitées concrètes sur les autorisations :

- Mettre en place un système d'autorisation/restriction selon le rôle de l'utilisateur :
  Ex : Seul l'administrateur peut accéder à la page gestion utilisateur.

- Seul l'utilisateur ayant créé la tâche peut supprimer cette dernière (hors administrateur).

- Les tâches reliées à l'utilisateur "anonyme" ne peuvent être supprimées uniquement que par un administrateur

2.2 - Corrections d'anomalies

De la même manière que sur le précédent point, afin d'améliorer la cohérence du produit et créer une notion de hiérarchie avec autorisation et restriction associé à un rôle, il sera ajouté* :

- L'ajout d'un utilisateur associé à chaque tâche créé, celui-ci ne pouvant être changé ultérieurement, si ce n'est par la suppression de l'utilisateur lui-même.

- Les tâches déjà créées doivent être rattachés à un utilisateur "anonyme".
  (**Pour aller plus loin :** Si un utilisateur est supprimé, les tâches seront attribuées à l'utilisateur "anonyme").

- Lors de l'ajout/modification d'un utilisateur il doit être possible de modifier son rôle : ROLE_USER ou ROLE_ADMIN.
-

*Nous ajouterons également quelques éléments de sécurité supplémentaires.

2.3 - Tests automatisés

Afin de s'assurer du bon fonctionnement de l'application, des tests automatisés seront rédigés à l'aide PHPUnit de manière unitaire mais surtout fonctionnelle (majorité de l'application), cela afin d'assurer la pérennité et la fiabilité de l'application.
Ainsi, il sera également demandé aux prochains contributeurs de rajouter des tests à toutes nouvelles méthodes ajoutées.

III - ACTIVATION DU PLAN ET COMPARATIF

3.1 - Environnement

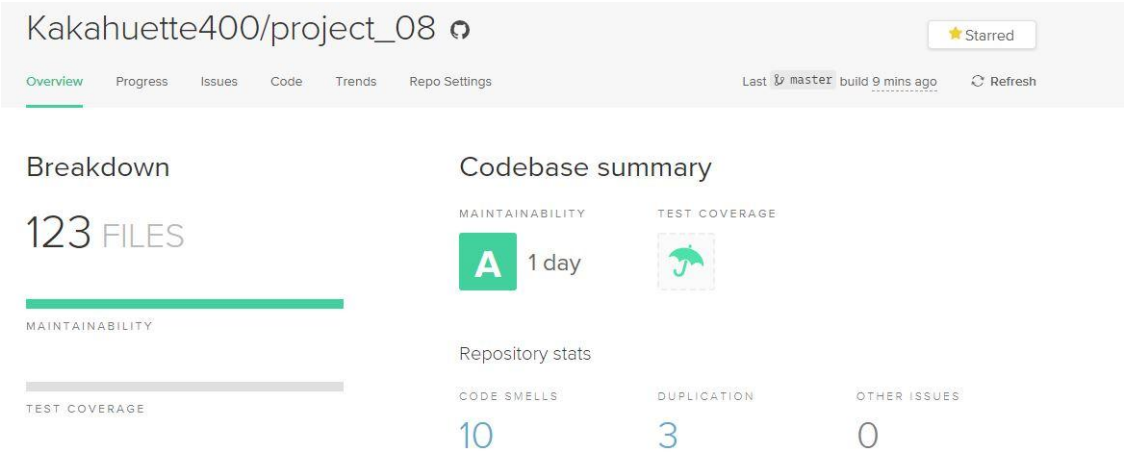
Pour ce nouveau projet voici les différentes versions utilisées :

Version de PHP	8.0.13
Version de Symfony	6

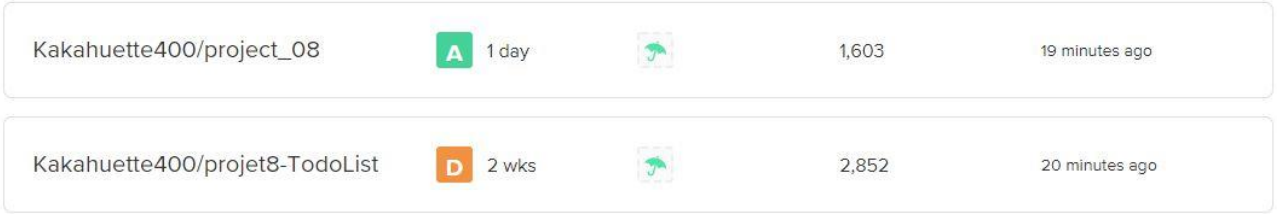
Il s'agit des versions les plus stables respectivement pour PHP et Symfony, cela permet d'avoir un produit en capacité d'accueillir les toutes dernières fonctionnalités et bundles du framework.

3.2 - Analyse de la qualité du code

Pour analyser la qualité du code de la version optimisée, CodeClimate a été également utilisé, ainsi le site produit ce rapport :



Ainsi, nous obtenons la note de A sur le projet optimisés, ce qui nous permet d'avoir une très bonne maintenabilité sur celui-ci.
La mise en perspective des deux projets nous donnes ainsi, ce résultat :



On obtient ainsi un code mieux maintenu, avec plus de fonctionnalités et avec plus de 1200 lignes de code en moins.

3.3 - Analyse de la performance

Analyse 1 : Mise en cache inactive (first landing)

URL	Vitesse d'exécution (/ms)	Initialisation Symfony (/ms)	Mémoire (MiB)	Render Time (/ms)	cache calls	Requêtes
/login	155	36	4	32	0	0
/users	85	25	4	7	1	2
/users (Data)	118	38	4	14	1	2
/users/create	217	35	6	39	24	1
/users/{id}/ edit	256	47	6	39	0	2
/tasks	120	37	2	16	1	2
/tasks(data)	141	35	4	22	1	3
/tasks/create	167	25	6	28	0	1
/tasks/{id}/ edit	201	42	6	36	0	3

Analyse 2 : Mise en cache active (second landing)

URL	Vitesse d'exécution (/ms)	Initialisation Symfony (/ms)	Mémoire (MiB)	Render Time (/ms)	cache calls	Requêtes
/login	79	23	4	9	0	0
/users	76	23	4	7	1	1
/users (data)	78	23	4	8	1	1
/users/create	92	11	4	12	12	1
/users/{id}/ edit	144	33	2	14	12	2
/tasks	73	23	4	6	1	1
/tasks(data)	79	25	4	12	1	1
/tasks/create	87	23	4	10	0	1
/tasks/{id}/ edit	71	11	4	12	0	3

Différentes optimisations ont été mises en place pour accélérer le temps de chargement, notamment lors du rechargement d'une page déjà visitée.

1) Refactoring :
   - Le refactoring du code à permis de gagner quelques milliseconde sur le temps de chargement lors d'une première arrivée.

2) Activation OPCACHE
   - Permet d'alléger la lecture du script en mettant en cache ce qui permet de transformer le script PHP en langage machine, c'est effectif dès son activation.

3) Mise en cache des requêtes (CacheInterface)
   - Certaines requêtes sont parfois assez gourmandes, c'est pourquoi il est préférable de les mettre en cache, c'est le cas pour le listing des users et des différentes tâches par exemple.

Ainsi, si l'on compare les différents URLs avant et après optimisations, sur certains critères nous obtenons ce tableau :

Analyse 1 : Mise en cache inactive (landing)

URL	Vitesse d'exécution (/ms) V1	Vitesse d'exécution (/ms) V2	Mémoire (MiB) V1	Mémoire (MiB) V2	Requêtes V1	Requêtes V2
/login	173	155	2	4	0	0
/users	211	85	2	4	1	2
/users (data)	292	118	4	4	1	2
/users/create	254	217	2	6	1	2
/users/{id}/edit	260	256	4	6	1	2
/tasks	312	120	2	2	2	2
/tasks(data)	530	141	4	4	2	3
/tasks/create	519	167	2	6	1	1
/tasks/{id}/edit	386	201	2	6	2	3
MOY/ TOTAL	326	162	2,66	4,66	1	2

Pour résumer, l'analyse avant la mise en cache, on voit que globalement le nombre de requête ainsi que la mémoire requise pour le chargement est supérieur sur la version améliorée du projet, cependant la vitesse d'exécution moyenne des pages à été réduite quasiment de moitié (50,31 %), passant de 326 ms à 162 ms.

Analyse 2 : Mise en cache active (second landing)

URL	Vitesse d'exécution (/ms) V1	Vitesse d'exécution (/ms) V2	Mémoire (MiB) V1	Mémoire (MiB) V2	Requêtes V1	Requêtes V2
/login	169	79	2	4	0	0
/users	194	76	4	4	1	1
/users (data)	215	78	4	4	1	1
/users/create	197	92	4	4	0	1
/users/{id}/ edit	235	144	4	2	1	2
/tasks	214	73	4	4	2	1
/tasks(data)	220	79	4	4	2	1
/tasks/create	267	87	2	4	1	1
/tasks/{id}/ edit	245	71	2	4	2	3
MOY/ TOTAL	217	87	3,33	3,7	1	1

Sur cette seconde analyse, après mise en cache on se retrouve avec un nombre de requêtes moyenne similaire et un écart de mémoire réduit.
Concernant la vitesse d'exécution, après optimisation on constate une amélioration de 59,90 % passant de 217 ms à 87 ms.
Pour résumer, bien que notre application soit plus fournie d'un point de vue visuel et en données, ce qui explique la mémoire supplémentaire allouée et le nombre de requêtes de départ, notre optimisation a permis d'arriver à des performances similaires à ce niveau.

Le point le plus marquant étant la vitesse d'exécution puisque celle-ci connaît une amélioration de la performance de 50,31% et de 59,90% respectivement avant et après mise en cache.
Ainsi, c'est près de 10 points d'amélioration supplémentaire constatés dues notamment à la mise en place de certains caches sur les requêtes et d'Opcache.

Pour aller plus loin :

- La mise en cache peut s'effectuer à différents niveaux afin d'améliorer la performance, nous aurions également pu utiliser APCu afin de compiler du code intermédiaire.

3.4 - Test du code

Afin d'améliorer la qualité finale de l'application et d'éviter tout problème lors de la mise en production, des tests unitaires et fonctionnels ont été exécutés.

Voici le coverage de l'application terminée (81,44%):

	Lines	
Total	84.43%	244 / 289
Controller	87.69%	114 / 130
Entity	92.16%	47 / 51
Form	87.88%	29 / 33
Security	70.42%	50 / 71
Services	100.00%	4 / 4

	Functions and Methods	
Total	78.48%	62 / 79
Controller	75.00%	12 / 16
Entity	90.62%	29 / 32
Form	83.33%	10 / 12
Security	52.94%	9 / 17
Services	100.00%	2 / 2

	Classes and Traits	
Total	50.00%	10 / 20
Controller	25.00%	1 / 4
Entity	66.67%	2 / 3
Form	83.33%	5 / 6
Security	16.67%	1 / 6
Services	100.00%	1 / 1

Les tests unitaires ont été principalement réalisés sur les entités pour le reste il s'agit principalement de tests fonctionnels.

Pour les tests unitaires, un setUp est réalisé afin de faire appel aux différentes fonctions une à une et les tester par la suite.
Pour les tests fonctionnels, des fixtures sont générées et supprimées de manière automatique après chaque test afin de garantir l'indépendance et la fiabilité de chacun d'entre eux.

```
Testing
.....                                               52 / 52 (100%)

Time: 00:17.183, Memory: 86.00 MB

OK (52 tests, 148 assertions)
```

Ainsi c'est un total de 52 tests qui sont effectués pour un total de 148 assertions, soit un temps total de 17,183 secondes et 86 MB de mémoire utilisés.

3.5 - Sécurité

A) Autorisations et restrictions

Les différentes Urls sont désormais restreintes aux utilisateurs authentifiés, seule la page de login reste accessible aux personnes non authentifiées.
Ainsi, pour toutes les Urii commençant par /user seul un administrateur peut y avoir accès.

```
access_control:
    - { path: ^/task, roles: [ROLE_USER, ROLE_ADMIN]  }
    - { path: ^/user, roles: ROLE_ADMIN  }
```

De la même manière seul un administrateur à le pouvoir d'effectuer un CRUD sur l'ensemble des utilisateurs.
Cette limitation est effectués par le biais de voters :

1) Attribut ajouté au-dessus de la fonction concernée.

```
#[Security("is_granted('CAN_VIEW', user)")]
```

2) conditions du voter pour réaliser l'action.

```
switch ($attribute) {
    case 'CAN_EDIT':
        return $subject->getRoles()[0] === "ROLE_ADMIN";
        break;
    case 'CAN_VIEW':
        return $subject->getRoles()[0] === "ROLE_ADMIN";
        break;
    case 'CAN_CREATE':
        return $subject->getRoles()[0] === "ROLE_ADMIN";
        break;
    case 'CAN_DELETE':
        return $subject->getRoles()[0] === "ROLE_ADMIN";
        break;
}
```

Sur la partie tâches, des modifications ont également été effectués :

- Un administrateur peut réaliser un CRUD sur l'ensemble des tâches. (tâche appartenant à l'utilisateur anonyme aussi)
- Un utilisateur ne peut modifier que ses propres tâches.

1) Attribut ajouté au-dessus de la fonction concernée.

```php
#[Security("is_granted('ROLE_ADMIN') or is_granted('TASK_EDIT', task)")]
```

2) conditions du voter pour réaliser l'action.

```php
switch ($attribute) {
    case 'TASK_EDIT':
        return $user === $subject->getCurrentUser() && $user->getRoles()[0] === "ROLE_USER";
        break;
    case 'TASK_DELETE':
            return $user === $subject->getCurrentUser() && $user->getRoles()[0] === "ROLE_USER";
        break;
    case 'TASK_TOGGLE':
            return $user === $subject->getCurrentUser() && $user->getRoles()[0] === "ROLE_USER";
        break;
}
```

- Un utilisateur supprimé par un administrateur voit ses tâches attribuées à l'utilisateur anonyme.

```php
public function deleteUser($id, TaskRepository $taskRepository): response
{
    $users = $this->userRepository->findOneBy(['id' => $id]);
    $tasks = $taskRepository->findBy(array('currentUser' => $users));
    $anonymousUser = $this->userRepository->findOneBy(['username' => "Anonyme"]);

    foreach ($tasks as $task){
        $task->setCurrentUser($anonymousUser);
        $this->em->persist($task);
    }
    $this->em->remove($users);
    $this->em->flush();
```

B) Vérification Email

Lors de l'inscription un mail de validation est envoyé (envoi d'un token) et est comparé à celui enregistré en base afin de n'autoriser l'accès de l'application qu'à un utilisateur bien authentifié.

```php
public function sendEmailConfirmation(string $verifyEmailRouteName, UserInterface $user, TemplatedEmail $email): void
{
    $signatureComponents = $this->verifyEmailHelper->generateSignature(
        $verifyEmailRouteName,
        $user->getId(),
        $user->getEmail()
    );

    $context = $email->getContext();
    $context['signedUrl'] = $signatureComponents->getSignedUrl();
    $context['expiresAtMessageKey'] = $signatureComponents->getExpirationMessageKey();
    $context['expiresAtMessageData' ] = $signatureComponents->getExpirationMessageData();
    $url = $context['signedUrl'] = $signatureComponents->getSignedUrl();
    $components = parse_url($url);
    parse_str($components['query'], &result: $results);
    $this->updateToken($results['token'], $user);

    $email->context($context);

    $this->mailer->send($email);
}
```

C) Modification du Mot de passe

La modification du mot de passe ne se fait désormais que par le biais d'une confirmation email, Elle peut être initiée par un administrateur depuis la page affichant la liste des utilisateurs ou par l'utilisateur lui-même depuis le bouton "mot de passe oublié" sur la page de connexion.

Celui-ci renvoie vers une page qui envoie un e-mail de réinitialisation à l'utilisateur associé à l'email.

3.5 - Synthèse

Le MVP avait de nombreuses lacunes, le plan d'optimisation à permis de combler celles-ci par la mise en place de nouvelles fonctionnalités et la réparation de certains problèmes.

Ainsi, tant sur le plan technique que fonctionnel et de l'UX l'application est désormais plus performante et de meilleure qualité comme le prouvent nos différents tests et analyses misent en place.

Cependant de nombreux éléments sont encore perfectibles, voici quelques exemples de ce qui améliorerait la performance, la qualité ou l'UX du projet :

- Mettre en place un filtre des tâches
- Mettre en place une pagination
- Mise en place d'un bundle gérant davantage de caches
- Jalonnement des Todos
- Refactorer davantage le projet