ToDo&Co

Documentation technique :
Audit de qualité et performances

Auteur: Ching Yi, Pelgrims Chen
Dernière mise à jour: 14/03/2022

Interne

Catalogue

Interne

Contexte

ToDo&Co est une application de gestion des tâches quotidiennes. Cette application a dû être développée à grande vitesse afin de montrer aux investisseurs potentiels que le concept est viable (Minimum Viable Product ou MVP).

Comme **ToDo &Co** a enfin réussi à lever des fonds pour permettre le développement de l'application, l'objectif du projet est donc d'améliorer l'application, ce qui comprend :

- l'implementation de nouvelles fonctionnalités ;
- la correction de certaines anomalies ;
- et la mise en place de tests automatisés.

Introduction

Au cours du développement, il est important que les fonctions de l'application soient à la hauteur des attentes. Cependant, pour pérenniser le développement de l'application, il est également important de s'assurer que la qualité du code et la qualité des performances ont atteint un niveau élevé.

Cet audit commencera par faire le point sur la dette technique de la première version de l'application et la comparera avec la version actuelle après réalisation. Il expliquera comment le projet de la version actuelle a été amélioré ou maintenu sur les deux axes suivants : qualité du code et performance. Il suggérera également comment il peut être amélioré à l'avenir.

Codacy est utilisé comme outil d'analyse de la qualité du code, tandis que Blackfire est utilisé pour analyser la qualité des performances.

Interne

La première version

La première version fait référence à l'application qui a été construite au départ avant les actions correctives et la mise en place des nouvelles fonctionnalités.

en termes de technique :

L'application est construite avec le Framework Symfony version 3.1.6, qui est une ancienne version et n'est plus maintenue[1].

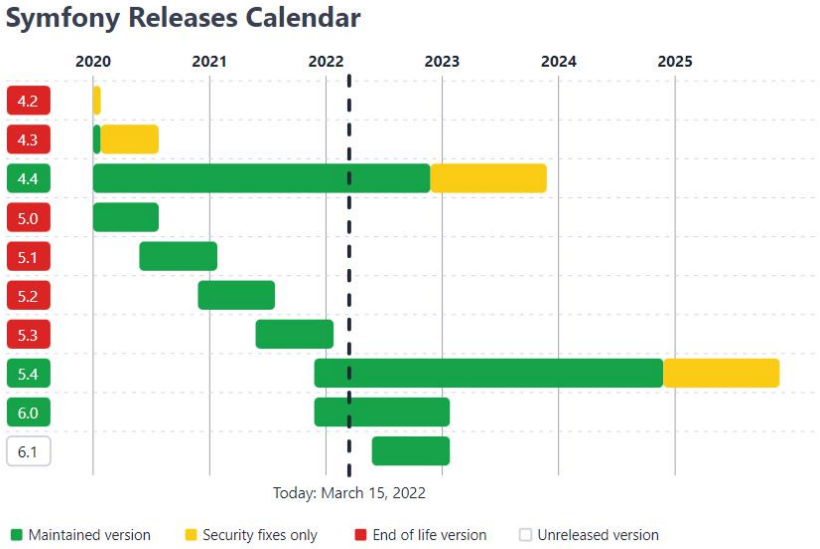
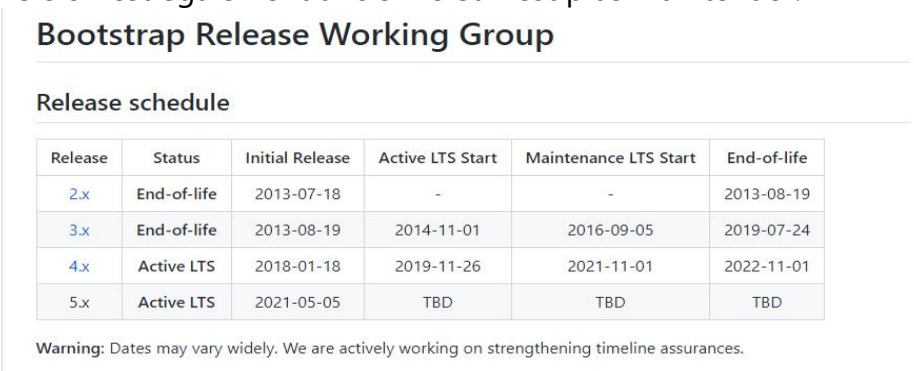


Figure 1: Calendrier de parution des dernières versions de Symfony

La librairie Bootstrap version 3.3.7 est utilisée dans le projet pour le frontend et la version est également ancienne et n'est plus maintenue[2].



Bootstrap Release Working Group

Release schedule

Release	Status	Initial Release	Active LTS Start	Maintenance LTS Start	End-of-life
2.x	End-of-life	2013-07-18	-	-	2013-08-19
3.x	End-of-life	2013-08-19	2014-11-01	2016-09-05	2019-07-24
4.x	Active LTS	2018-01-18	2019-11-26	2021-11-01	2022-11-01
5.x	Active LTS	2021-05-05	TBD	TBD	TBD

Warning: Dates may vary widely. We are actively working on strengthening timeline assurances.

Figure 2: Calendrier de parution des dernières versions de Bootstrap

[1] https://symfony.com/releases
[2] https://github.com/twbs/release

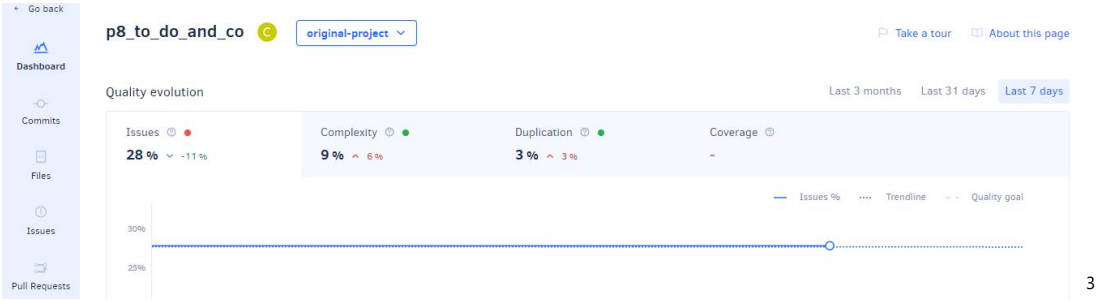
Interne

en termes de qualité du code :

La qualité du code a été évaluée par l'outil d'automatisation Codacy, qui permet d'identifier différents problèmes liés à la sécurité, aux performances ou au style du code.

L'utilisation de cet outil d'analyse a été associée au dépôt GitHub sur lequel seront stockées les modifications effectuées, afin d'établir une analyse récurrente pour chaque pull-request et de garantir le maintien d'un certain niveau de qualité au cours du développement.

L'analyse préliminaire est présentée ci-dessous :



[3]

Figure 3: Analyse Codacy initiale.

L'analyse préliminaire de Codacy montre un grade C pour l'application de sa première version. Les anomalies sont de 28%, soit 156 en nombre total. Il concernent principalement le style du code ainsi que des problèmes mettant en péril la sécurité de l'application.

[3] https://app.codacy.com/gh/JENNYPCHEN/p8_to_do_and_co/dashboard?branch=original-project

Interne

En outre, en analysant le code manuellement, il y a plusieurs aspects qui peuvent également être améliorés selon l'article 'The Symfony Framework Best Practices'[4] :

1. Utiliser l'injection de dépendances pour obtenir des services

```
$password = $this->get('security.password_encoder')->encodePassword($user, $user->getPassword());
$user->setPassword($password);
```

Figure 4: La première version n'utilise pas l'injection de dépendances lors de l'utilisation d'un service.

2. Séparer la logique métier des 'contrôleurs' par la création des services.

```
/**
 * @Route("/users/create", name="user_create")
 */
public function createAction(Request $request)
{
    $user = new User();
    $form = $this->createForm(UserType::class, $user);

    $form->handleRequest($request);



    if ($form->isValid()) {

        $em = $this->getDoctrine()->getManager();
        $password = $this->get('security.password_encoder')->encodePassword($user, $user->getPassword());
        $user->setPassword($password);

        $em->persist($user);
        $em->flush();
```

Figure 5: un service pour crypter un mot de passe peut être créé au lieu d'écrire la logique dans le contrôleur.

en termes de performance :

Les performances de l'application ont un impact direct sur l'expérience de l'utilisateur. Plus le temps de chargement d'une application est long, plus il y a de chances que cela ait un effet négatif sur l'expérience utilisateur. Bien qu'il y ait plusieurs possibilités qui peuvent affecter le temps de chargement, par exemple, le réseau internet, le trafic de l'application, etc, en tant que développeur, il est essentiel d'optimiser les performances de l'application.

l'outil d'analyse Blackfire a été donc utilisé pour analyser les performances de l'application . L'image ci-dessous montre l'analyse initiale de l'application et elle sera utilisée pour comparer les performances avec la version actuelle dans le chapitre suivant.

[4] https://symfony.com/doc/current/best_practices.html

Interne

Route:	Temps de chargement (m/s)	Pic de mémoire(mb)	Nombre de requêtes SQL
tasks/ {id}/edit	192	14.4	2
tasks/	167	11.1	2
tasks/create	182	14.4	1
users/{id}/edit	267	14.2	1
users/	292	11	2
users/create	203	10.2	0
/login	186	7.68	0
/	261	11	1

Figure 6: l'analyse initiale de l'application

Lien vers chaque analyse :

Route:	Lien:
tasks/ {id}/edit	https://blackfire.io/profiles/9b67dd4e-86cb-479c-985c-28ab30076979/graph
tasks/	https://blackfire.io/profiles/9f8ba170-bd08-4bd5-a9f9-7ed910a0e004/graph
tasks/create	https://blackfire.io/profiles/6e24b858-17d2-4a79-a0f1-e0b691888ce2/graph
users/{id}/edit	https://blackfire.io/profiles/038ec4a6-3bcc-40e5-b14a-58f636ff2084/graph
users/	https://blackfire.io/profiles/3643ce16-76d6-4533-bc8b-433021a5b65e/graph
users/create	https://blackfire.io/profiles/1170a49c-3f95-4699-9ad7-291d1dff7300/graph
/login	https://blackfire.io/profiles/2822f4e7-58d6-480a-a0ed-2581a9bf6a89/graph
/	https://blackfire.io/profiles/5361da13-54f4-44b0-baba-7ad748fa5295/graph

Interne

La version actuelle

La version actuelle fait référence à l'application après avoir corrigé les anomalies, à laquelle ont a ajouté les nouvelles fonctions demandées. Des tests sont mis en œuvre (couverture de code:90%) dans cette version pour confirmer que le fonctionnement de l'application est conforme aux attentes.

En termes de technique:

L'application a été mise à jour vers la dernière version stable de Symfony (version 6)[5]. Il y a de grandes différences entre la version 3 et la version 6. De nombreux fichiers doivent être placés différemment afin de s'adapter à la dernière version.

Bootstrap a également été remplacé par la dernière version (Bootstrap 5)[6]. Certaines syntaxes utilisées dans la version 3 de Bootstrap ne sont plus valides et ont été modifiées pour répondre à la syntaxe de Bootstrap 5.

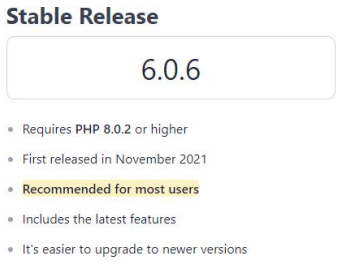


Figure 7: la dernière version stable de Symfony

En termes de qualité du code :

L'analyse préliminaire de Codacy indiquait un grade C, cependant, dans la version actuelle, la qualité du code a été améliorée suite aux suggestions de

[5] https://symfony.com/releases
[6] https://getbootstrap.com/docs/versions/

Interne

Codacy, ce qui est prouvé par le grade A indiqué dans le Codacy. Il ne reste que de 1% d'anomalies (11 en nombre total).

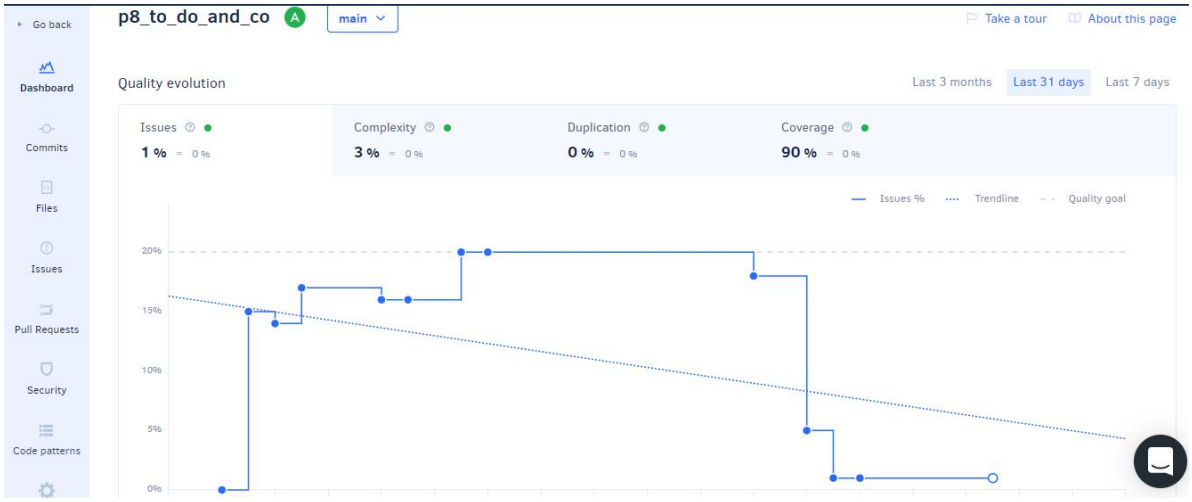


Figure 8: Analyse Codacy de la version actuelle

De nombreux efforts ont été faits dans cette partie afin de produire des codes clairs et structurés. Les anomalies reésiduelles sont tous liés aux fichiers générés automatiquement, par exemple, les fichiers quand on utilise la ligne de commande pour créer un rapport de couverture en html.

Suivi des analyses :
https://www.codacy.com/gh/JENNYPCHEN/p8_to_do_and_co/dashboard?utm_source=github.com&utm_medium=referral&utm_content=JENNYPCHEN/p8_to_do_and_co

En termes de performance :

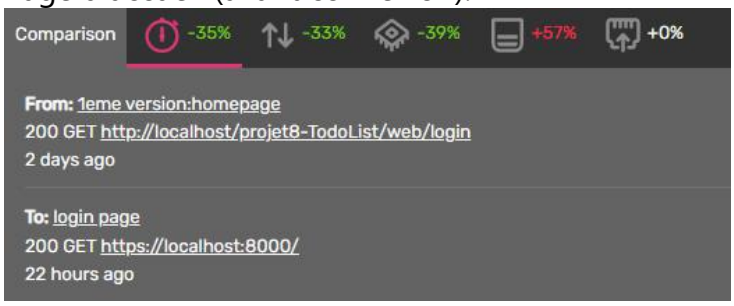
Comme mentionné, le temps de chargement d'une application peut être affecté par différents facteurs qui ne sont pas toujours contrôlables par le développeur. Cependant, un développeur peut toujours optimiser les

Interne

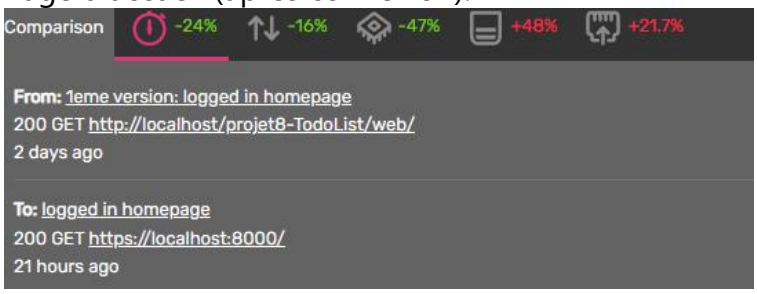
performances d'une application en fonction de ce qu'il peut contrôler lors de la création de l'application.

La figure ci-dessous illustre les comparatifs des analyses Blackfire réalisées avant et après amélioration, en termes de performance grâce à deux mesures : le temps de chargement de la page et la mémoire allouée.

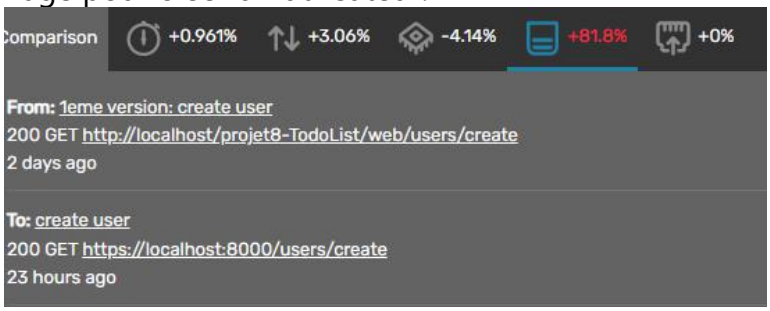
Page d'accueil (avant connexion):



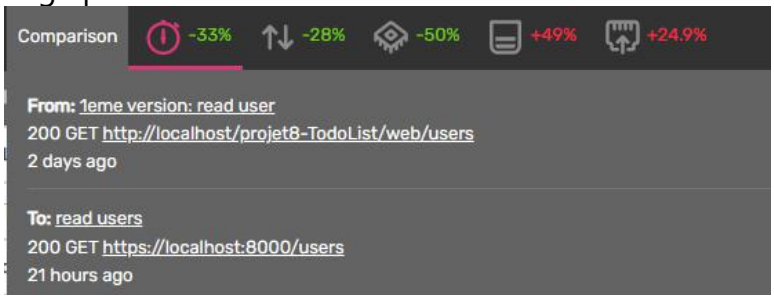
Page d'accueil (après connexion):



Page pour créer un utilisateur:

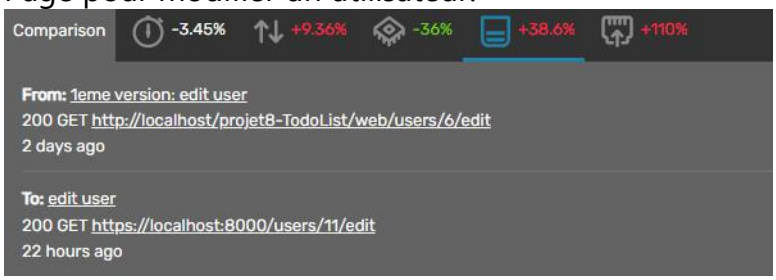


Page pour consulter la liste des utilisateurs:

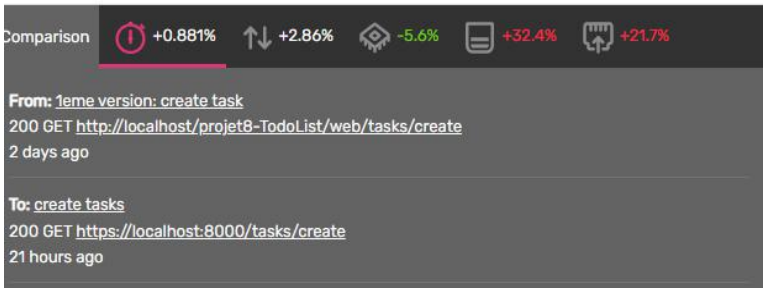


Interne

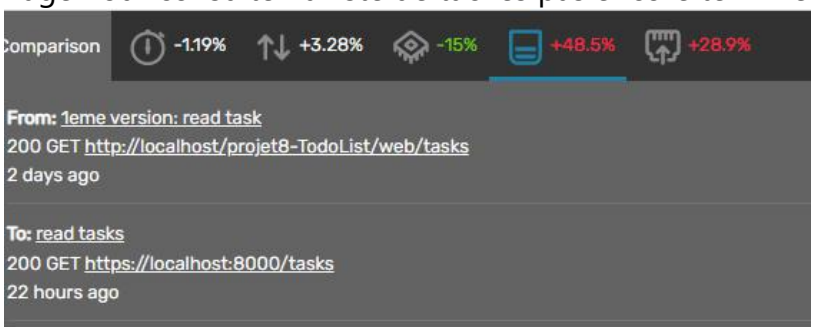
Page pour modifier un utilisateur:



Page pour créer une tâche:



Page Pour consulter la liste de tâches pas encore terminées:



Page pour modifier une tâche:

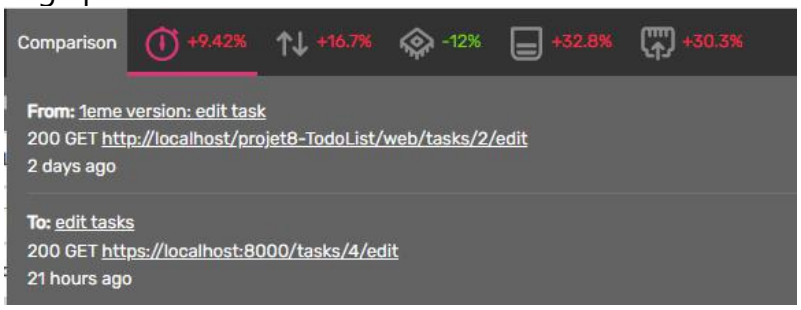


Figure 9: Comparatif de l'application avant et après améliorations en termes de performance

Comme on peut le constater, le temps de chargement des différentes pages (requêtes GET) en général a été réduit. A l'exception d'une page dont le temps de chargement a légèrement augmenté d'environ 9 %, le reste des pages a un temps de chargement inférieur à celui de la première version. 3 pages sur 8 ont même connu une grande amélioration en réduisant le temps de chargement d'environ **30,5 %** en moyenne.

Interne

D'autre part, cette analyse révèle une augmentation de la mémoire allouée. Le choix d'optimiser le temps de chargement des pages reste à privilégier car il est directement lié au client, tandis que l'augmentation de la mémoire a peu d'impact sur les performances compte tenu de la taille des serveurs actuels. La différence de mémoire pourrait s'expliquer par la version plus récente de Symfony qui est plus complexe.

Pour être plus concret, en dehors des deux pools de cache (cache.app et cache.system), qui sont toujours activés par défaut pour aider l'application à s'exécuter plus rapidement, il y a quelques méthodes que nous avons imposées pendant le développement afin d'améliorer les performances. Voici quelques exemples :

1. **Optimiser l'autoloader de Composer**

Comme dans les serveurs de production, les fichiers PHP ne devraient jamais changer, à moins qu'une nouvelle version de l'application soit déployée, c'est pourquoi nous générons la nouvelle class map pour stocker toutes les classes, de sorte qu'il n'essaie pas de trouver les classes modifiées/nouvelles à chaque fois.

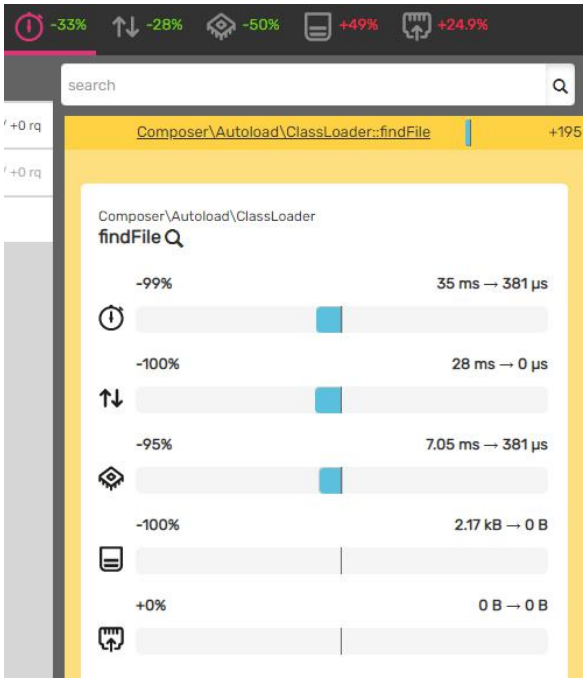


Figure 10: La comparaison de la performance sur autoload pour la page de consulter la liste d'utilisateurs.

Dans l'image ci-dessus montre que l'optimisation de l'autoloader de composer a un impact positif sur les performances.

Interne

2. Vider le conteneur de services dans un seul fichier

Symfony compile le conteneur de service dans plusieurs petits fichiers par défaut. Nous mettons ce paramètre à true pour compiler le conteneur entier dans un seul fichier afin d'améliorer les performances lors de l'utilisation du "class preloading".

Bilan et plan d'amélioration

La qualité du code et les performances ont été prises en compte et analysées lors du développement de l'application. Avec l'aide des outils, nous pouvons repérer les problèmes et les améliorer. Cela dit, il est toujours possible de les améliorer encore plus, mais il faut faire la part des choses entre l'amélioration possible et le temps à investir pour y parvenir. À l'avenir, lorsque l'application deviendra plus populaire et que de nouvelles fonctions devront être ajoutées, il sera bien sûr conseillé de surveiller et de revoir à nouveau la qualité du code et les performances.

La qualité d'une application va au-delà de la qualité du code et des performances. Les fonctions et l'esthétique de l'application elle-même font également partie de la qualité de l'application et peuvent également affecter l'expérience de l'utilisateur.

Voici quelques-unes des suggestions qui peuvent être considérées à l'avenir :

- pagination,
- fonction filtre/recherche,
- fixation d'un délai pour les tâches,
- contrôle de sécurité lors de la suppression d'une tâche

Interne