



AUDIT DE QUALITE ET DE PERFORMANCE

FEVRIER 2021

Table des matières

INTRODUCTION	3
PARTIE 1: AUDIT DE QUALITE	4
ANALYSE GENERALE.....	5
1. VERSION PHP	5
2. VERSION FRAMEWORK	5
3. OUTILS D'ANALYSE	6
ANALYSE DETAILLE.....	7
1. PARTIE FRONTEND	8
2. PARTIE BACKEND.....	9
EXPERTISE.....	10
1. MISE A JOUR DE L'APPLICATION	10
2. PRINCIPES S.O.L.I.D.....	12
3. TEST AUTOMATISES	16
4. INTEGRATION CONTINUE	17
PARTIE 2 : AUDIT DE PERFORMANCE	19
MESURE DES PERFORMANCES	20
PERFORMANCE SUR LE MVP.....	20
EXPERTISE.....	22
1. OPTIMISATION DE L'AUTOLOADER DE COMPOSER	22
2. OPTIMISATION DE OPCACHE.....	23
3. CONFIGURER LE CACHE PHP REALPATH.....	23
4. PERFORMANCE ARPES OPTIMISATION.....	24
5. OPCACHE CLASS PRELOADING	26
AUTRE PISTE D'OPTIMISATION POUR L'APPLICATION	26
1. CONFIGURATION DES VARIABLES D'ENVIRONNEMENT EN PRODUCTION	26
2. PAGINATION.....	27
3. UTILISATION DU CACHE	27
BILAN.....	28

INTRODUCTION

L'objectif de ce document, est de permettre à l'entreprise d'avoir un aperçu général de la dette technique de l'application **ToDo & Co**. Ces informations vous donneront, un aperçu des priorités et de certaines réflexions concernant les futures améliorations de l'application.

Cet audit a été réalisé sur le MVP (Minimum Viable Product). Un certain nombre d'améliorations a déjà été apporté à l'application. Nous détaillerons les correctifs et améliorations qui ont été mis en place, mais aussi les différents points qui peuvent être améliorés.

Les deux principaux aspects audités concernent, **la qualité du code** ainsi que **les performances de l'application**. Les résultats présents dans ce document, se basent sur mon expérience de développeur ainsi que différents outils d'analyse.

Nous verrons dans un premier temps tout ce qui concerne la qualité du code de l'application :

- Analyse du code initial et présentation de sa dette technique
- Expertise

Nous finirons ensuite par nous intéresser aux performances de l'application :

- Analyse des performances initiales
- Comparaison des résultats
- Expertise



PARTIE 1: AUDIT DE QUALITE

ANALYSE GENERALE

1. VERSION PHP

Le code de l'application et celui du Framework ont été développé avec une version de **PHP 5**. L'application n'est pas compatible avec la version 7, un certain nombre d'erreurs apparaissent avec l'utilisation de code ou de méthodes dépréciés. Pour des questions de performance et de stabilité, il est conseillé d'utiliser les versions les plus récentes.

2. VERSION FRAMEWORK

Le projet utilise le Framework **Symfony** avec la version 3.1, cette version n'est plus maintenue à jour depuis juillet 2017, on peut voir que la dernière version à bénéficier de support à long terme est la 4.4.18.

Au moment où ce document est rédigé, la dernière version stable (**LSR**) est la 5.2.1.

Pour des questions de sécurité, de performance et de pérennité, il est important de travailler sur des versions qui sont encore maintenues. Cela permet de bénéficier des dernières fonctionnalités du Framework et aussi de le faire évoluer plus facilement vers des futures mises à jour.

Latest Stable Release	Latest Long-Term Support Release
5.2.1	4.4.18
<ul style="list-style-type: none">• First released in November 2020.• Recommended for most users.• Includes the latest features.• It's easier to upgrade to newer versions.	<ul style="list-style-type: none">• First released in November 2019.• 3 year support for bugs and security fixes.• It doesn't include the latest features.• It's harder to upgrade to newer versions.

Il arrive souvent que des applications se retrouvent « bloquées » dans leurs évolutions. Les entreprises ayant trop attendu se retrouvent face à une lourde dette technique. Il est souvent plus simple de refaire une nouvelle application que de la faire évoluer. Ceci engendre d'importants coûts qui pourraient être évité.

Au niveau du **frontend**, l'application utilise le Framework CSS **Bootstrap** 3.3.7, celui-ci ne reçoit plus de « **Long Termes Support Plan** » et ne recevra plus de mise à jour de sécurité. Dans l'application **ToDo & Co**, cela se fait ressentir. En effet certains composants qu'utilisait **Bootstrap** ont été abandonnées (**Glyphicon**), et certains noms de classes **CSS** ne sont plus utilisés sur les dernières versions.

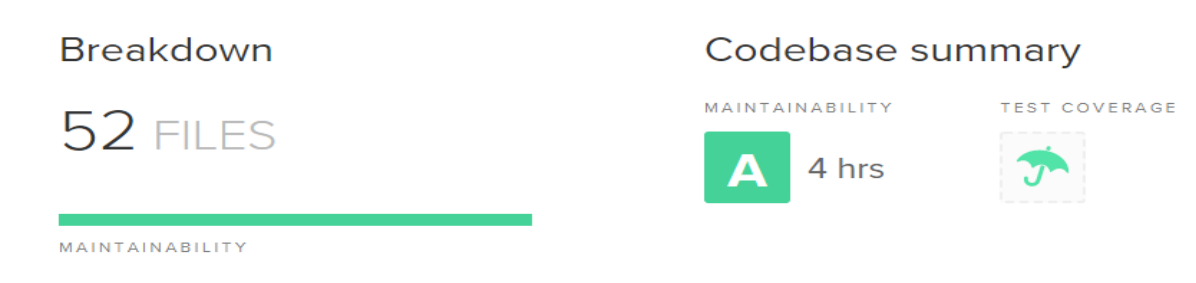
Ces changements pourraient affecter des développeurs ayant l'habitude de travailler sur des versions plus récentes. Il est donc préférable d'utiliser la version 4.6 voir la version 5.0 actuellement en Beta.

3. OUTILS D'ANALYSE

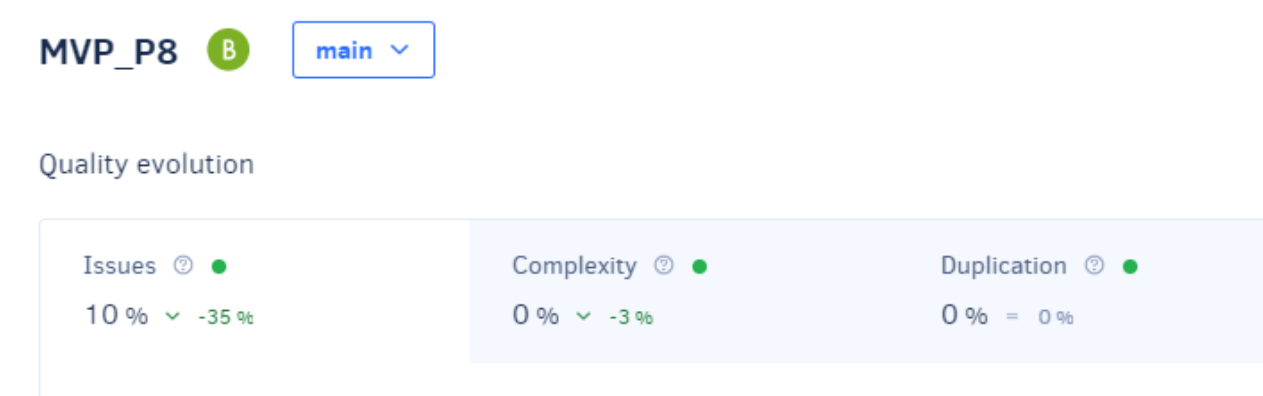
Différents outils d'analyse statique de code ont été utilisés pour avoir un aperçu de la dette technique. **Code Climat** et **Codacy** n'ont révélés aucun réel problème majeur à régler.

Code Climat :

Le niveau de maintenabilité est bon



Codacy :



Seul le paramètre « **\$request** » était inutilisé dans une méthode du **security controller**.

```

/**
 * @Route("/login", name="login")
 */
public function loginAction(Request $request)
{
    $authenticationUtils = $this->get('security.authentication_utils');

    $error = $authenticationUtils->getLastAuthenticationError();
    $lastUsername = $authenticationUtils->getLastUsername();

    return $this->render('security/login.html.twig', array(
        'last_username' => $lastUsername,
        'error'         => $error,
    ));
}

```

PHPStan et **PHP CS** ont révélés quelques problèmes notamment au niveau du coding-style (PSR-1, PSR-2 PSR-12) et d’une absence de la **PHPDoc**.

ANALYSE DETAILLE

Dans cette partie, nous allons faire un état des lieux en listant l’ensemble des problèmes majeurs trouvés dans l’application. Certains points de moindre importance ont été directement corrigés et ne seront pas présent dans liste.

Nous verrons dans un premier toute la partie **frontend** puis ensuite nous passerons à l’analyse du **backend**.

Note :

- ❌ Améliorations non effectuées.
- ✅ Corrections ou améliorations effectués sur l’application.

1. PARTIE FRONTEND

- ✖ Il n'y a pas description dans la balise méta. Les méta-descriptions peuvent être incluses dans les résultats de recherche pour résumer de manière concise le contenu de la page.
- ✖ Sur la page qui liste les tâches, le contenu de description n'a pas de limite. Il serait bien d'ajouter une fonction **Twig** pour couper le texte et ajouter « ... » afin que l'utilisateur sache que le contenu n'est pas complet.
- ✖ Problème de contraste sur différentes sections du site. Le contenu de certains boutons et headers peuvent poser des problèmes d'accessibilité aux personnes malvoyantes.
- ✖ La navigation est difficile, dans la barre de navigation les différentes sections du site ne sont pas présentes. Il est aussi difficile de revenir sur la section précédente, l'utilisateur doit cliquer sur « précédent » dans son navigateur ou alors retourner sur le menu principal.
- ✖ Le design de l'application reste très sommaire celui-ci pourrait être amélioré.
- ✖ Partie responsive de l'application à améliorer
- ✔ Un certain nombre de propriétés dans le fichier CSS ne sont pas utilisé, dans l'application.
- ✔ Le lien « To Do List » censé renvoyer vers la page principale dans la barre de navigation ne possède aucune URL.
- ✔ Il n'y a pas de lien sur le bouton « Consulter la liste des tâches terminées ».
- ✔ Le bouton « Créer un utilisateur » apparaît sur toutes les pages du site.
- ✔ Il n'y a aucun lien de navigation permettant d'accéder à la liste des utilisateurs et du coup à leurs éditions.
- ✔ Il serait judicieux d'ajouter un message flash après la connexion de l'utilisateur lui indiquant que son authentification s'est correctement déroulée.

2. PARTIE BACKEND

- ✖ Certaines fonctionnalités pourraient être implémentées pour aider les utilisateurs dans la gestion de leurs tâches. On ne peut pas savoir quand ni par qui une tâche a été créée ou modifier.
- ✖ Pas de messages de confirmation lors de la suppression d'une tâche.
- ✖ Absence de pages d'erreurs personnalisées.
- ✖ Un rôle SuperAdmin serait judicieux afin de pouvoir supprimer des comptes utilisateurs si besoin.
- ✖ Absence d'une pagination dans les pages listant les utilisateurs et les tâches.
- ✖ Il serait judicieux d'ajouter une action show afin de pouvoir juste visualiser l'intégralité de la tâche sans avoir besoin de la modifier directement.
- ✔ Pas de gestion des tâches terminées.
- ✔ Erreur dans un message flash. Lorsque l'on clique sur « marquer comme faite » ou « non terminé » on reçoit le même message flash indiquant que la tâche est marquée comme faite.
- ✔ Pas de validation concernant les données rentrer dans la création d'une tâche. Si l'on rentre un titre trop long par exemple, cela fait planter le site.
- ✔ Optimiser la sécurité et le contrôle de l'authentification avec un **Authenticator** plutôt que le système de « form_login » intégré.
- ✔ Absence de Tests unitaires et fonctionnels.
- ✔ Aucune Fixtures présentes en environnement de développement.
- ✔ Absence de **PHPDoc**
- ✔ Trop de logique fonctionnelle et de répétition dans certaines classes. Le code peut être optimisé.

EXPERTISE

1. MISE À JOUR DE L'APPLICATION

L'application étant été développée sur une version 3.1 de Symfony qui n'est plus maintenue, la dette technique était trop importante pour essayer de faire une migration vers une version 5 et plus. Il a donc été décidé de reprendre le code de l'application, et de l'intégrer sur un nouveau projet en version 5.1.9. Étant donnée la taille de l'application, ceci est la solution la plus rapide et efficace. Cela permet de repartir sur de bonne base correspondant d'avantage aux standards de Symfony. A titre d'exemple, si l'on veut profiter du formidable outil **Symfony Flex**, notre application doit respecter une certaine structure :

```
1  your-project/
2  |— assets/
3  |— bin/
4  |   └─ console
5  |— config/
6  |   |— bundles.php
7  |   |— packages/
8  |   |— routes.yaml
9  |   └─ services.yaml
10 |— public/
11 |   └─ index.php
12 |— src/
13 |   |— ...
14 |   └─ Kernel.php
15 |— templates/
16 |— tests/
17 |— translations/
18 |— var/
19 └─ vendor/
```

Utiliser **Symfony Flex** permettra de pouvoir installer rapidement à l'aide d'une simple ligne de commande les tâches les plus courantes des applications **Symfony**, telles que l'installation et la suppression de bundles et d'autres dépendances. Un autre avantage c'est qu'il permet de mettre à jour simplement les différents bundle grâce aux « recettes ». Celles-ci modifieront automatiquement certaines lignes de code ou fichiers de configurations.

Plutôt que d'utiliser une approche traditionnelle en incluant les fichiers statiques au niveau du dossier Public, il a été choisi de mettre en place **Webpack Encore**. Le **CSS**, le **Javascript** et les images seront alors situées au niveau du dossier assets (voir structure ci-dessus).

Ces fichiers seront alors compilés et leurs tailles seront réduites afin d'obtenir de meilleures performances. **Webpack Encore** permet de simplifier la configuration du javascript ou des différentes librairies utilisées par **Node**. **ToDo & Co** est une application de gestion de tâches. Lorsque l'on analyse la concurrence, on se rend compte que pour ce genre d'application le Javascript est fortement utilisé. En effet les utilisateurs veulent pouvoir effectuer rapidement et dynamiquement certaines fonctionnalités.

L'application n'utilise pas encore de **Javascript**. Ceci dit il est important d'anticiper les futurs évolutions et fonctionnalités qui pourraient être mises en place. **Webpack Encore** permettra d'avoir un environnement Javascript propre, rapide et facilement configurable dans notre application.

2. PRINCIPES S.O.L.I.D

Rappel : SOLID est un acronyme correspondant à cinq grands principes de programmations :

- Single Responsibility Principle.
- Open/Closed Principle.
- Liskov Substitution Principle.
- Interface Segregation Principle.
- Dependency Inversion Principle.

Ces principes permettent de rendre le code d'une application plus faciles à comprendre, à maintenir et à faire évoluer. Il est donc important de respecter ces principes afin de pérenniser le développement de l'application.

Dans **ToDo & Co**, un des problèmes majeurs rencontré est le non-respect du premier principe (Single Responsibility). Si l'on regarde nos controllers, on remarque que plusieurs actions sont effectuées dans nos méthodes. On s'occupe de la création, et de la soumission de nos formulaires ainsi que l'ajout des flashs messages.

```
/**
 * @Route("/tasks/create", name="task_create")
 */
public function createAction(Request $request)
{
    $task = new Task();
    $form = $this->createForm(TaskType::class, $task);

    $form->handleRequest($request);

    if ($form->isValid()) {
        $em = $this->getDoctrine()->getManager();

        $em->persist($task);
        $em->flush();

        $this->addFlash('success', 'La tâche a été bien été ajoutée.');
```

```
        return $this->redirectToRoute('task_list');
    }

    return $this->render('task/create.html.twig', ['form' => $form->createView()]);
}
```

Ceci n'est pas de la responsabilité d'un **controller**, celui-ci doit seulement traiter une requête et renvoyer une réponse. Nous avons procédé à un découplage du code. Nos classes seront plus faciles à maintenir et à réutiliser au fur et à mesure de l'évolution de l'application.

```
/**
 * @Route("/tasks/create", name="app_task_create", methods={"GET", "POST"})
 * @param Request $request
 * @return Response
 */
public function createAction(Request $request, TaskCreateHandler $handler): Response
{
    $task = new Task();

    if ($handler->handle($request, $task)) {
        return $this->redirectToRoute('app_task_list');
    }

    return $this->render('task/create.html.twig', [
        'form' => $handler->getForm()->createView(),
    ]);
}
```

Nous avons privilégié l'utilisation de services **FormHandlers** pour gérer le traitement de nos formulaires.

```
namespace App\Service\FormHandler;

use App\Form\TaskType;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Session\SessionInterface;
use Symfony\Component\Security\Core\Authentication\Token\Storage\TokenStorageInterface;

class TaskCreateHandler extends AbstractHandler
{
    private EntityManagerInterface $em;
    protected const FORMTYPE = TaskType::class;
    private TokenStorageInterface $tokenStorage;
    private SessionInterface $session;

    public function __construct(
        EntityManagerInterface $em,
        TokenStorageInterface $tokenStorage,
        SessionInterface $session
    ) {
        $this->em = $em;
        $this->tokenStorage = $tokenStorage;
        $this->session = $session;
    }

    public function process(object $data): void
    {
        $data->setAuthor($this->tokenStorage->getToken()->getUser());
        $this->em->persist($data);
        $this->em->flush();

        $this->session->getFlashBag()->add('success', 'La tâche a bien été ajoutée.');
```

Pour respecter un autre principe **DRY** (Dont Repeat Yourself), nous avons implémenté une classe abstraite pour nos différents **FormHandlers**. En effet on remarque que le traitement de nos formulaires nécessite toujours la même logique. Création du formulaire, validation, récupération de l'objet **FormInterface**. De plus ceci correspond au deuxième principe **SOLID** (Open/Closed Principle). Notre **AbstractHandler** est maintenant ouvert à l'extension, mais fermées à la modification.

Nous avons aussi typé grâce à **PHP 7.4** nos variables et nos différents paramètres.

```
<?php

namespace App\Service\FormHandler;

use Symfony\Component\Form\FormInterface;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Form\FormFactoryInterface;
use App\Service\FormHandler\AbstractHandlerInterface;

abstract class AbstractHandler implements AbstractHandlerInterface
{
    private FormFactoryInterface $formFactory;
    protected FormInterface $form;
    protected const FORMTYPE = 'abstract';

    /**
     * @required
     */
    public function setFormFactory(FormFactoryInterface $formFactory): void
    {
        $this->formFactory = $formFactory;
    }

    public function handle(Request $request, object $data, ?array $options = []): bool
    {
        $this->form = $this->formFactory->create(static::FORMTYPE, $data, $options)->handleRequest($request);

        if ($this->form->isSubmitted() && $this->form->isValid()) {
            $this->process($data);

            return true;
        }
        return false;
    }

    /**
     * @return FormInterface
     */
    public function getForm(): FormInterface
    {
        return $this->form;
    }
}
```

Enfin nous avons implémenté une **interface** à notre **AbstractHandler**. Si on a besoin d'ajouter un nouveau **FormHandler** dans l'application celui-ci devra respecter le contrat présent ci-dessous.

```
<?php

namespace App\Service\FormHandler;

use Symfony\Component\Form\FormInterface;
use Symfony\Component\HttpFoundation\Request;

interface AbstractHandlerInterface
{
    /**
     * handle
     *
     * @param Request $request
     * @param Object $data
     * @param array<null|mixed> $options
     * @return bool
     */
    public function handle(Request $request, object $data, ?array $options = []): bool;

    public function process(object $data): void;

    public function getForm(): FormInterface;
}
```

3. TEST AUTOMATISÉS

ToDo & Co ne possédait aucun test unitaire ou fonctionnel. Pour avoir un code robuste, éprouvé et maintenable la mise en place de test est primordiale. Ceux-ci garantissent la qualité de votre code et de votre application.

De plus lors de l'ajout de nouvelles fonctionnalités les développeurs pourront être plus serein lors de la mise en production et leurs changements se feront de manière plus fluide et contrôler.

Les tests unitaires et fonctionnels ont été mis en place dans l'application afin d'obtenir une couverture de code de 100%. Nous avons utilisé la librairie **PHPUnit** qui est une référence en matière de test pour le langage **PHP**. D'autre part un mini Framework a été mis en place dans **ToDo & Co** afin de faciliter le processus de Test. Les méthodes ont été documentés avec des commentaires pour aider les développeurs à son utilisation. Bien évidemment celui-ci pourra être amélioré au fur et à mesure selon les besoins de l'équipe.

Voici le rapport de couverture donné par **PHPUnit** :

```
Testing
..... 44 / 44 (100%)

Time: 01:11.628, Memory: 84.00 MB

OK (44 tests, 91 assertions)
PS C:\Users\Vincent\Desktop\Projet_8\to_do_list> php bin/phpunit
```

C:\Users\Vincent\Desktop\Projet_8\to_do_list\src / (Dashboard)

	Code Coverage			
	Lines		Functions and Methods	Classes and Traits
Total	100.00%	198 / 198	100.00%	72 / 72
■ Controller	100.00%	36 / 36	100.00%	11 / 11
■ Entity	100.00%	54 / 54	100.00%	31 / 31
■ EventListener	100.00%	10 / 10	100.00%	4 / 4
■ Form	100.00%	10 / 10	100.00%	2 / 2
■ Repository	100.00%	25 / 25	100.00%	5 / 5
■ Security	100.00%	27 / 27	100.00%	8 / 8
■ Service	100.00%	36 / 36	100.00%	11 / 11

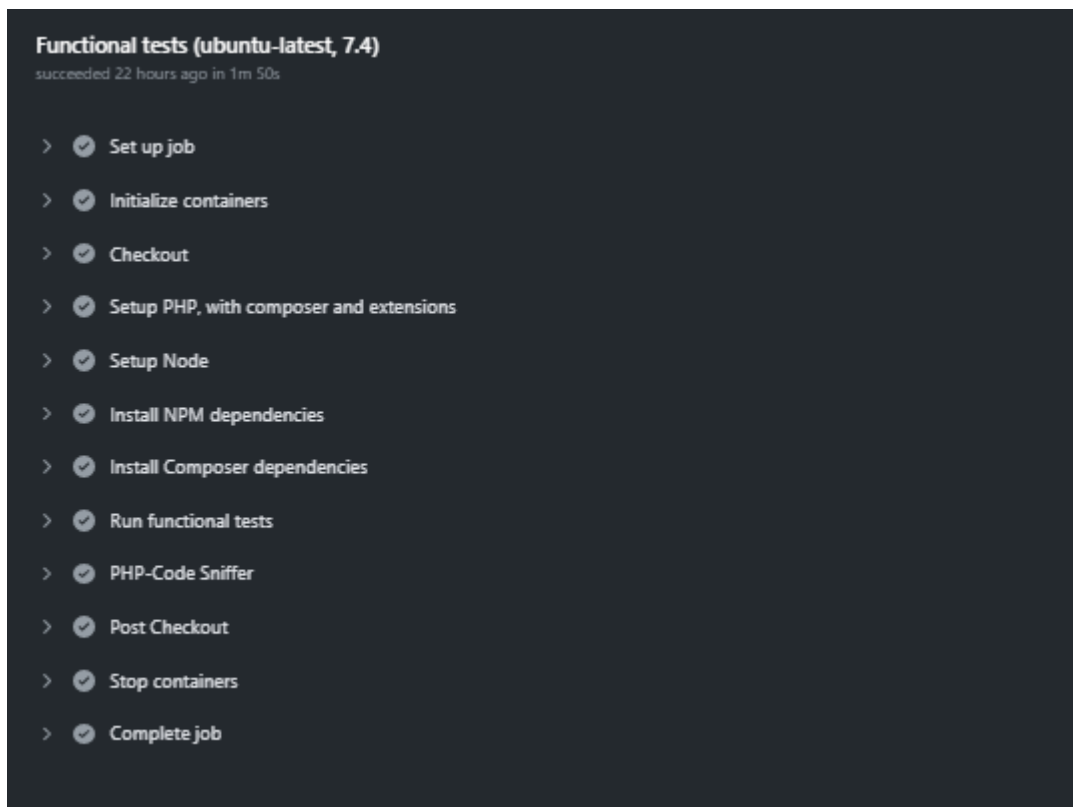
Legend

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

Generated by php-code-coverage 9.2.5 using PHP 7.4.8 with Xdebug 3.0.1 and PHPUnit 9.5.0 at Tue Jan 26 22:13:08 UTC 2021.

4. INTÉGRATION CONTINUE

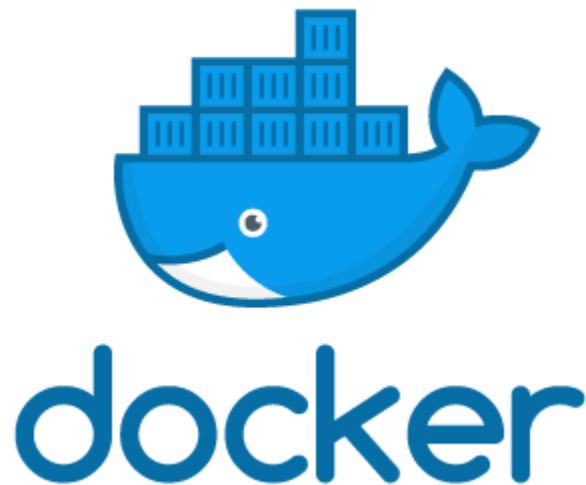
Afin de faciliter le développement de l'application et sa mise en production il a été décidé de mettre en place un workflow avec une intégration continue. Le choix à été d'utiliser **GitHub Actions**. Le projet utilisant déjà **GitHub** comme plateforme il est plus simple de centraliser le workflow et l'intégration continue sur cette même plateforme. Notre **workflow** permet notamment avant chaque **pull request** d'effectuer les tests automatisés et de vérifier le coding-style de notre application grâce à **PHP-CodeSniffer**.



5. UN MEILLEUR ENVIRONNEMENT DE DÉVELOPPEMENT

Pour éviter certains problèmes au niveau de l'équipe de développement, il est pourrait être intéressant d'utiliser un logiciel de conteneurisation comme Docker. En effet **ToDo & Co** étant une startup venant d'être créé, il est probable que l'équipe s'agrandisse.

Afin d'éviter des conflits et d'avoir un environnement de travail uniformisé et indépendant Docker s'avère être un réel atout. Cela l'est d'autant plus en cette période de COVID-19 ou le télétravail est de plus en plus rependus. Tout le monde n'a pas forcément le même environnement de travail chez soi. Lorsque l'on travaille en local sur sa machine nos version de **PHP**, de notre base de données ou même notre système d'exploitation peut être différent. Docker permettrait d'éviter certains problèmes d'incompatibilité lors de la mise en production. Cela représenterait un certain coût en termes de mise en place et de formation. Cependant, sur le long terme cet ajout réduirait les soucis auxquelles nous pourrions être confronté et améliorerait la productivité.





PARTIE 2 : AUDIT DE PERFORMANCE

MESURE DES PERFORMANCES

Dans cette partie nous verrons dans un premier temps les mesures des performances effectuées sur le **MVP**. Nous parlerons ensuite des améliorations qui ont été apporté à l'application pour optimiser ces performances. Les résultats seront présentés sous la forme d'un tableau comparatif. Enfin nous finirons par les différentes pistes possibles pour de futur optimisation.

































PERFORMANCE SUR LE MVP

Les tests de performance effectués sur l'application ont été réalisé avec **Blackfire**. Cet outil aide les développeurs à profiler, tester, déboguer et optimiser les performances de leurs applications. L'avantage est que **Blackfire** peut être utilisé à n'importe quelle étape du cycle de vie d'une application: pendant le développement, le test, la préparation et la production.

Note :

Les tests ont été effectués sur une machine utilisant comme système d'exploitation Windows en local. Les performances peuvent donc varier suivant l'environnement sur lequel ils ont été effectué. Afin de pouvoir tester. Afin d'avoir des données pertinentes des **fixtures** ont été créées sur le projet **MVP** et sur la version améliorée afin de pouvoir correctement analyser les performances notamment concernant les requêtes.

Voici les résultats des tests sur le **MVP** pour les différentes routes de l'application :

200 GET https://127.0.0.1:8001/app.php/users/create <i>[MVP] /users/create => app_user_create</i> Created less than a minute ago by vincent.signoret.mail@gmail.com	   0 rq 287 ms 224 ms 62.5 ms 12.9 MB 706 B 0 µs / 0 rq 673 µs / 1 rq  Timeline
200 GET https://127.0.0.1:8001/app.php/users/3/edit <i>[MVP] /users/{id}/edit</i> Created 1 minute ago by vincent.signoret.mail@gmail.com	   0 rq 303 ms 236 ms 66.7 ms 12.9 MB 1.2 kB 0 µs / 0 rq 1.17 ms / 2 rq  Timeline
200 GET https://127.0.0.1:8001/app.php/users <i>[MVP] /users => app_user_list</i> Created 2 minutes ago by vincent.signoret.mail@gmail.com	   0 rq 243 ms 190 ms 52.4 ms 10.6 MB 4.57 kB 0 µs / 0 rq 1.18 ms / 2 rq  Timeline
200 GET https://127.0.0.1:8001/app.php/tasks/create <i>[MVP] /tasks/create => app_task_create</i> Created 2 minutes ago by vincent.signoret.mail@gmail.com	   0 rq 307 ms 245 ms 62.8 ms 13.1 MB 706 B 0 µs / 0 rq 1.09 ms / 1 rq  Timeline
200 GET https://127.0.0.1:8001/app.php/tasks/104/edit <i>[MVP] /tasks/{id}/edit => app_task_edit</i> Created 9 minutes ago by vincent.signoret.mail@gmail.com	   0 rq 291 ms 228 ms 63.4 ms 13.1 MB 1.38 kB 0 µs / 0 rq 1.3 ms / 2 rq  Timeline
200 GET https://127.0.0.1:8001/app.php/tasks <i>[MVP] /tasks => app_task_list</i> Created 10 minutes ago by vincent.signoret.mail@gmail.com	   0 rq 270 ms 200 ms 69.6 ms 11 MB 29.5 kB 0 µs / 0 rq 2.66 ms / 2 rq  Timeline
200 GET https://127.0.0.1:8001/app.php/login <i>[MVP] /login => app_login</i> Created 11 minutes ago by vincent.signoret.mail@gmail.com	   0 rq 167 ms 130 ms 37.3 ms 6.93 MB 0 B 0 µs / 0 rq 0 µs / 0 rq  Timeline
200 GET https://127.0.0.1:8001/app.php/ <i>[MVP] / => app_homepage</i> Created 12 minutes ago by vincent.signoret.mail@gmail.com	   0 rq 240 ms 193 ms 46.8 ms 10.5 MB 706 B 0 µs / 0 rq 1.88 ms / 1 rq  Timeline

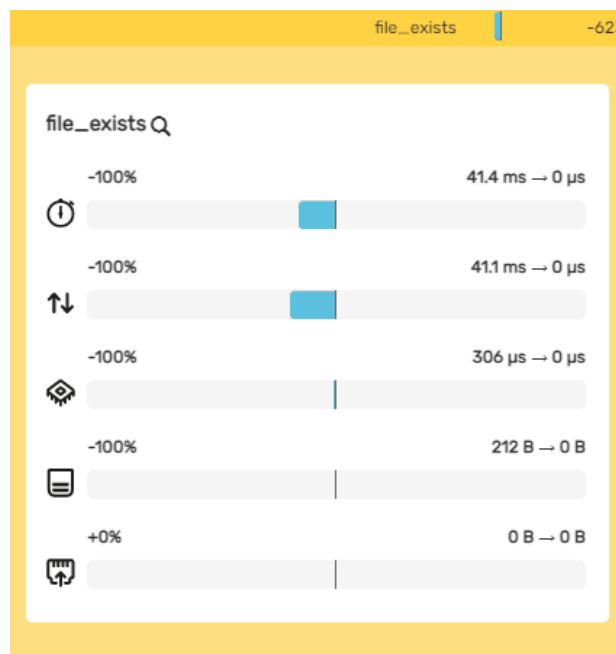
EXPERTISE

OPTIMISATION DE L'AUTOLOADER DE COMPOSER

Pendant le développement de l'application **Composer** est configuré pour optimiser la recherche de classes nouvelles et modifiées.

En production, les fichiers **PHP** ne devraient jamais changer, sauf si une nouvelle version de l'application est déployée. C'est pourquoi vous pouvez optimiser l'**autoloading** de Composer pour analyser l'ensemble de l'application une seule fois et créer un fichier optimisé comportant un tableau des emplacements de toutes les classes.

Prenons l'exemple de la page de création d'un utilisateur. Lorsque l'on regarde la version **MVP** on se rend compte que la fonction **file_exists** est appelé 623 fois. Grâce à l'optimisation de l'**autoloader** cette fonction ne sera plus appelée puisque l'ensemble des fichiers seront répertoriés dans `vendor/composer/autoload_classmap.php`.



1. OPTIMISATION DE OPCACHE

OPcache est une extension de **PHP** introduit depuis la 5.5. Lors de l'exécution d'un script PHP, celui-ci est compilé en opcode, un code compréhensible par la machine. **OPCache** stocke ce code compilé en mémoire lors de sa première exécution, afin de pouvoir le réutiliser par la suite, et donc améliorer les performances.

La configuration de **OPcache** par défaut n'est pas adaptée aux applications **Symfony**. Certains paramètres sont donc à changer pour optimiser les performances de l'application. Modifier la configuration de **PHP** :

Mémoire maximale que OPcache peut utiliser pour stocker les fichiers PHP compilés.
`opcache.memory_consumption=256`

Nombre maximum de fichiers pouvant être stockés dans le cache.
`opcache.max_accelerated_files=20000`

2. CONFIGURER LE CACHE PHP REALPATH

Lorsqu'un chemin relatif est transformé en son chemin réel et absolu, **PHP** met en cache le résultat pour améliorer les performances. Les applications qui ouvrent de nombreux fichiers PHP, tels que les projets Symfony peuvent obtenir des gains de performances. Pour cela modifier votre **php.ini**

Mémoire maximale allouée pour stocker les résultats
`realpath_cache_size=4096K`

Enregistrer les résultats pendant 10 minutes (600 secondes)
`realpath_cache_ttl=600`

3. PERFORMANCE ARPES OPTIMISATION

Voici les performances de l'application après avoir effectué les modifications ci-dessus :

200 GET https://127.0.0.1:8000/users/create <i>[ToDo_Last] /users/create => app_user_create</i> Created less than a minute ago by vincent.signoret.mail@gmail.com	Compare		
0 rq 50.7 ms 24.5 ms 26.2 ms 3.78 MB 0.978 kB 0 µs / 0 rq 430 µs / 1 rq Timeline			
200 GET https://127.0.0.1:8000/users/1/edit <i>[ToDo_Last] /users/{id}/edit => app_user_edit</i> Created 1 minute ago by vincent.signoret.mail@gmail.com	Compare		
0 rq 53.4 ms 27.1 ms 26.3 ms 3.78 MB 1.73 kB 0 µs / 0 rq 877 µs / 2 rq Timeline			
200 GET https://127.0.0.1:8000/users <i>[ToDo_Last] /users => app_user_list</i> Created 2 minutes ago by vincent.signoret.mail@gmail.com	Compare		
0 rq 45.6 ms 25 ms 20.6 ms 2.91 MB 8.97 kB 0 µs / 0 rq 847 µs / 2 rq Timeline			
200 GET https://127.0.0.1:8000/tasks/create <i>[ToDo_Last] /tasks/create => app_task_create</i> Created 3 minutes ago by vincent.signoret.mail@gmail.com	Compare		
0 rq 50.1 ms 20.6 ms 29.5 ms 3.68 MB 0.978 kB 0 µs / 0 rq 400 µs / 1 rq Timeline			
200 GET https://127.0.0.1:8000/tasks/2/edit <i>[ToDo_Last] /tasks/{id}/edit => app_task_edit</i> Created 5 minutes ago by vincent.signoret.mail@gmail.com	Compare		
0 rq 47.1 ms 21.2 ms 25.9 ms 3.71 MB 1.71 kB 0 µs / 0 rq 929 µs / 2 rq Timeline			
200 GET https://127.0.0.1:8000/tasks <i>[ToDo_Last] /tasks => app_task_list</i> Created 6 minutes ago by vincent.signoret.mail@gmail.com	Compare		
0 rq 57.2 ms 22.9 ms 34.3 ms 3.32 MB 21.6 kB 0 µs / 0 rq 1.4 ms / 2 rq Timeline			
200 GET https://127.0.0.1:8000/login <i>[ToDo_Last] /login => app_login</i> Created 9 minutes ago by vincent.signoret.mail@gmail.com	Compare		
0 rq 23.7 ms 8.66 ms 15.1 ms 2.2 MB 0 B 0 µs / 0 rq 0 µs / 0 rq Timeline			
200 GET https://127.0.0.1:8000/ <i>[ToDo_Last] / => app_homepage</i> Created 10 minutes ago by vincent.signoret.mail@gmail.com	Compare		
0 rq 24.5 ms 9.44 ms 15.1 ms 2.09 MB 0 B 0 µs / 0 rq 0 µs / 0 rq Timeline			

On remarque une nette amélioration à tous les niveaux. Voyons de plus prêt ces résultats dans un tableau comparatif.

Routes	MVP		LAST VERSION		Gain temps	Gain mémoire
	(ms)	(mb)	(ms)	(mb)	(ms)	(mb)
/users/create (app_create_user)	287	12.9	50.7	3.78	82%	71%
/users/{id}/edit (app_edit_user)	303	12.9	53.4	3.78	82%	71%
/users (app_list_user)	243	10.6	45.6	2.91	81%	72%
/tasks/create (app_create_task)	307	13.1	50.1	3.68	83%	71%
/tasks/{id}/edit (app_edit_task)	291	13.1	47.1	3.71	84%	72%
/tasks (app_list_task)	270	11	57.2	3.32	79%	70%
/login (app_login)	167	6.93	23.7	2.2	86%	68%
/ (app_homepage)	240	10.5	24.5	2.09	90%	80%

Les performances sont nettement supérieures. On a des augmentations oscillantes entre 68 et 90 pourcents.

4. OPCACHE CLASS PRELOADING

Le **preloading** avec **OPcache** est une fonctionnalité qui a été introduite dans **PHP 7.4.0** et permet de précharger des scripts dans l'**OPcache**. Ce mécanisme permet de garder des symboles spécifiques (fonctions, classes, etc.) disponibles à travers les demandes, sans avoir besoin d'être explicitement inclus.

Pour configurer le **preloading** il suffit de modifier le `php.ini` :

```
opcache.preload=/path/to/project/config/preload.php
```

Note :

Ces modifications n'ont pas été apporté à ce projet car le **preloading** ne fonctionne pas sur Windows. Comme indiqué précédemment l'ensemble des tests ont été fait en local sur un system d'exploitation Windows. Cependant il faudra prendre un compte ces modifications lors de la prochaine mise en production de l'application.

AUTRE PISTE D'OPTIMISATION POUR L'APPLICATION

1. CONFIGURATION DES VARIABLES D'ENVIRONNEMENT EN PRODUCTION

En production, les fichiers `.env` sont également analysés et chargés à chaque requête. Le moyen le plus simple de définir les variables d'environnement consiste donc à déployer un fichier `.env.local` sur vos serveurs de production avec vos valeurs de production.

Pour améliorer les performances, vous pouvez éventuellement exécuter la commande `dump-env` (disponible dans Symfony Flex 1.2 ou version ultérieure) :

```
composer dump-env prod
```

Après avoir exécuté cette commande, Symfony chargera le fichier `.env.local.php` pour obtenir les variables d'environnement et ne passera pas de temps à analyser les fichiers `.env`.

Pour faciliter ce processus, il est conseillé d'ajouter cette commande dans notre workflow (Github actions).

2. PAGINATION

Il a été annoncé dans l'audit de qualité que la liste des tâches ainsi que la liste des utilisateurs n'avaient pas de pagination. Ceci présente un problème pour l'expérience utilisateur mais peut aussi réduire les performances de l'application.

Pour le moment les performances ne sont pas vraiment impactées. Par exemple pour la liste de tâches, les seules données récupérées par la requête sont le titre et le contenu. Nous avons créé une centaine de tâches avec les fixtures. Mais si jamais nous décidons d'ajouter d'autres fonctionnalités, comme l'ajout de l'auteur, ou l'intégration d'image ou de fichier liés à la tâche. Si l'on charge directement une centaine de tâches les performances seront grandement impactées. Une pagination permettrait de contrôler le nombre d'éléments chargés et donc de soulager le nombre de requêtes et le volume des données envoyés à l'utilisateur.

3. UTILISATION DU CACHE

Pour continuer dans la même logique que la pagination, l'utilisation d'un système de cache peut permettre d'améliorer les performances de l'application. Cela permettrait de renvoyer des réponses aux utilisateurs sans même devoir effectuer des requêtes pour récupérer les données si celles-ci n'ont pas été modifiées.

Il existe plusieurs possibilités concernant la mise en place d'un système de cache.

- Utilisation du cache **Doctrine**.
- Utilisation du cache Component de **Symfony**.
- Utilisation système de gestion de base de données clé-valeur comme **Redis**.
- Utilisation du **http Cache**.

BILAN

Cet audit de qualité vous permet d'avoir une vue d'ensemble de l'état de votre application. Nous avons pu voir l'ensemble de la dette technique concernant la qualité du code ainsi que les performances. Parmi les points listés un bon nombre de correction et d'optimisation ont été apporté. Les changements qui ont été effectués ont été en priorité ciblé pour avoir un code plus robuste avec une meilleure structure. Ce choix a été fait dans le but de pouvoir faire évoluer plus facilement l'application **ToDo & Co**.