Symfony

Audit de Qualité

Vincent Lescot, décembre 2018

Openclassrooms projet n°8 - Améliorer une application existante

Sommaire

Introduction

ToDo & Co est une startup dont le cœur de métier est une application permettant de gérer ses tâches quotidiennes. L'entreprise vient tout juste d'être montée, et l'application a dû être développée à toute vitesse pour permettre de montrer à de potentiels investisseurs que le concept est viable (on parle de Minimum Viable Product ou MVP) avec le Framework Symfony dans sa version 3.1.

J'ai été chargé d'apporté de nouvelles fonctionnalités à l'application, et d'analyser la qualité et les performances.

Ce document a pour but d'exposer cette analyse dans l'état actuel de l'application et des pistes pour sa bonne évolution.

I. Versions utilisées

1) PHP

Version utilisée : >= 5.6

Un gain significatif peut être apporté en migrant le projet sur PHP 7.2 (la version 7.3 sortie récemment peut aussi ajouter de légers gains).

D'après le site *www.phpbenchmarks.com*, le temps de réponse d'une requête entre les deux versions est largement divisé.

	Hello World Temps de réponse	API REST Temps de réponse
PHP 5.6	4.6 ms	13.7 ms
PHP 7.2	1.8 ms	3.0 ms

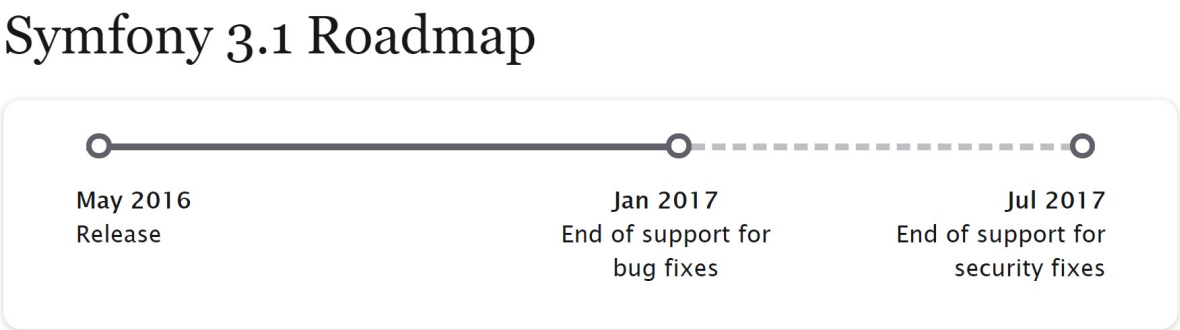
La mémoire utilisé est aussi très grandement réduite jusqu'à 50%.

De plus, l'évolution du langage apporte sont lot d'améliorations tel que :

- le typage des paramètres des fonctions,
- les classes anonymes et,
- l'opérateur «vaisseau».

2) Symfony

*<u>Version utilisée</u> = 3.1.**



La version 3.1 de Symfony n'est plus maintenue pour les bugs et pour la sécurité depuis juillet 2017. **La migration du projet vers une version plus récent est une priorité**.

Je propose deux recommandations alternatives :

- Une migration vers la version 3.4 LTS, maintenue en sécurité jusqu'en novembre 2021.
    - Avantages :
        - Support long terme,
        - Requis PHP 7.1,
        - Autowiring.
- Une migration vers la version 4.2, maintenue en sécurité jusqu'en janvier 2020.
    - Avantages :
        - Requis PHP 7.1,
        - Autowiring,
        - Flex.
    - Inconvénient :
        - Pas de version LTS avant la version 4.4 (novembre 2019 ➔ novembre 2023).

Qualité du code

II. Qualité du code

1) Outils de qualité du code

Des outils d'analyse de la qualité du code peuvent aussi être utilisés afin de révéler des problèmes de complexités, de styles ou de duplicate dans le code. Pour ce faire, nous utilisons Codacy et CodeClimate.

Puis nous feront des tests de performances de l'application avec l'outil BlackFire.

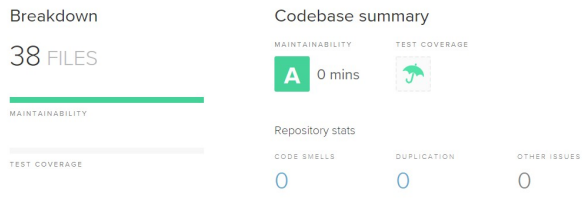
Comme nous allons le voir, les analyses des fichiers PHP du projet initial sont bonnes. Notre but sera donc de maintenir le niveau de qualité.

CodeClimate

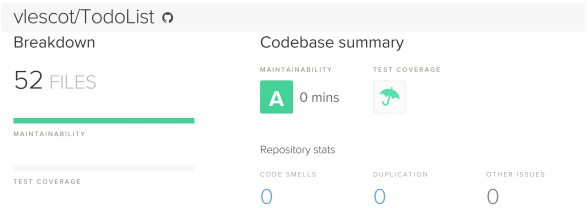
L'analyse de CodeClimate sur le projet initial ne révèle aucun problème.

Avec huit fichiers supplémentaires avec l'ajout de fonctionnalités et une révision de la cohérence du code, CodeClimate ne montre toujours aucun problème dans le code.

Avant



Après

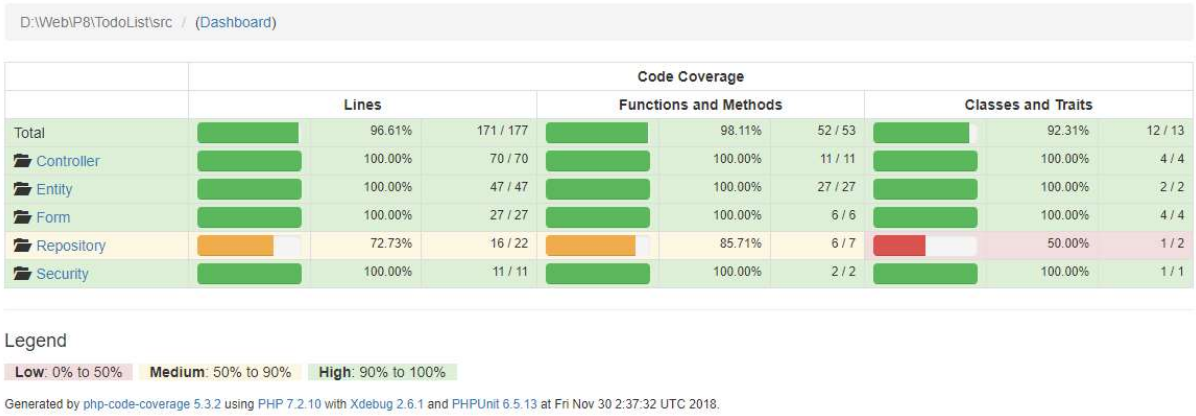


2) Tests et couverture

Les tests permettent de garantir le bon fonctionnement de l'application et font partie intégrante de la qualité d'une livraison.

La couverture de code testé et un indicateur pour vérifier la proportion de code exécuté lors des tests et d'assurer une meilleure maintenabilité du code, car **ils permettent d'éviter toute régression**.

La couverture de code l'application est supérieure à 95%.



D:\Web\P8\TodoList\src / (Dashboard)

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		96.61%	171 / 177		98.11%	52 / 53		92.31%	12 / 13
Controller		100.00%	70 / 70		100.00%	11 / 11		100.00%	4 / 4
Entity		100.00%	47 / 47		100.00%	27 / 27		100.00%	2 / 2
Form		100.00%	27 / 27		100.00%	6 / 6		100.00%	4 / 4
Repository		72.73%	16 / 22		85.71%	6 / 7		50.00%	1 / 2
Security		100.00%	11 / 11		100.00%	2 / 2		100.00%	1 / 1

Legend

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

Generated by php-code-coverage 5.3.2 using PHP 7.2.10 with Xdebug 2.6.1 and PHPUnit 6.5.13 at Fri Nov 30 2:37:32 UTC 2018.

Le code non couverture est appelé par le Framework lui-même lors du login d'un utilisateur.

```
35    public function loadUserByUsername($username): ? UserInterface
36    {
37        return $this->createQueryBuilder('u')
38            ->where('u.username = :username OR u.email = :email')
39            ->setParameter('username', $username)
40            ->setParameter('email', $username)
41            ->getQuery()
42            ->getOneOrNullResult();
43    }
```

Une forte couverture est indispensable pour déclenché les éventuelles bogues. Cependant, l'indicateur ne prend pas en considération les paramètres injectés dans les tests ce qui le rend facilement faillible et peut remplacer des tests pertinents.

Les tests réalisés :

- Sur le maintien de la sécurité, avec des tests sur l'accès des utilisateurs pour chaque rôle sur chacune des routes.
- Sur le maintien du fonctionnement de l'application, avec des tests sur les statuts code et le contenu complet des réponses de chacune des requêtes présentent ainsi que la modification des données sur l'administration du site.
- Des tests unitaires sur les entités et les formulaires.

```
Testing TodoList
.............................................   48 / 48 (100%)

Time: 30.84 seconds, Memory: 110.13MB

OK (48 tests, 113 assertions)
```

III. Analyse de Performance

Selon plusieurs études, la fidélisation des utilisateurs d'une application web dépend en grande partie de la qualité et de la performance de l'application. Sur ce point, l'expérience utilisateur est très importante et on peut noter trois axes à surveiller particulièrement :

- La rapidité d'affichage des pages d'un site web (40% des visiteurs abandonnent la navigation si le temps de réponse dépasse 3 secondes)
- Le temps nécessaire au téléchargement des éléments d'une page
- La fluidité du navigateur dans la manipulation des éléments d'une page

Il est également bon de rappeler qu'il ne s'agit pas là des seuls éléments de performances sur lesquels il faut se pencher. La performance d'une application web dépend effectivement du temps de réponse, mais également de la disponibilité du système, de la robustesse de celle-ci, et de sa capacité à monter en charge.

Certaines de ces données ne sont pas testables compte tenu de l'environnement local sur lequel les tests ont été effectués.

L'analyse BlackFire permet de mesurer le temps d'exécution du code entre le moment où la requête est envoyé par le client jusqu'au moment où le serveur renvoi la réponse.

Nous comparons le projet dans son état initial et à la livraison.

A noter, les analyses de performances suivants sont réalisées après :

- Optimisation du serveur par l'activation de l'OPcache et du realpath cache.
- Optimisation de l'autoloader de Composer. Les appels de la fonction file_exists sont supprimés de l'autoloader par la création du fichier « autoload_classmap.php ».

Route	**INITIAL**		**PROJECT**		**Gain/Perte**	
Method GET*	**Time (ms)**	**Memory (MB)**	**Time (ms)**	**Memory (MB)**	**Time (%)**	**Memory (%)**
homepage	32,6	2,66	20,3	2,45	37,73%	7,89%
login	17,8	1,76	12,7	2,65	28,65%	-50,57%
task_list	33,9	2,74	18,4	2,7	45,72%	1,46%
task_create	40,1	3,4	20,9	3,22	47,88%	5,29%
task_edit	41	3,43	19,8	3,25	51,71%	5,25%
user_list	39,7	2,67	13,7	2,59	65,49%	3,00%
user_create	28,6	2,53	20,6	3,4	27,97%	-34,39%
user_edit	33,3	2,67	18,8	3,4	43,54%	-27,34%

** Notre compte BlackFire permet seulement d'analyser via la méthode GET*

Les performances de l'application sont globalement améliorées. La mise à jour de la version de Symfony accélère les processus notamment son initialisation. Le fichier « /public/index.php » apporte un gain de vitesse par rapport au fichier « /web/app.php ». L'accélération est parfois compensée par les fonctionnalités apportées.

L'augmentation de la mémoire utilisée est produite par la fonction handle du Kernel de Symfony.

IV. Dette Technique

1. Back-End

1) Bonnes Pratiques

Validation sur le mot de passe

Aucune validation n'était effectuée sur le mot de passe saisi par l'utilisateur. Symfony fournit une contrainte de validation sur les mots de passe qui a été implémentée.

Controller extends AbsctractController

Plutôt qu'hériter de la classe Controller, l'AbstractController est considéré comme une bonne pratique car il ne fournit pas d'accès aux services public et oblige d'explicité la déclaration dans les dépendances.

Supprimer le suffixe « Action »

Dans les premières versions, le suffixe était requis. Le suffixe est à présent devenu optionnel et non recommandé.

Utilisation du @ParamConverter

Dans les cas simple, le ParamConverter est une bonne pratique car il allège les Controllers en interrogeant automatiquement la base de données pour une entité et en le passant en argument de l'action.

```php
use App\Entity\Post;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route("/{id}", name="admin_post_show")
 */
public function show(Post $post)
{
    $deleteForm = $this->createDeleteForm($post);

    return $this->render('admin/post/show.html.twig', [
        'post' => $post,
        'delete_form' => $deleteForm->createView(),
    ]);
}
```

Accèder aux Repositories via l'injection de dépendances dans les Actions

Avec Symfony version 4.2, l'accès aux Repositories via le RegistryManager ne sera plus une bonne pratique.

```php
$taskRepository = $this->getDoctrine()->getRepository(Task');
```

Pour cela, il est préférable d'utiliser l'injection de dépendance, comme suivant :

```php
public function list(TaskRepository $repository): Response
```

Respect des PSR-1 et PSR-2

Bien que le projet initial ne manque pas de respecter la plupart des PSR-1 et PSR-2, les commentaires manquaient intégralement.

Type-Hinting

Dans le projet initial, la version 5.6 de PHP était utilisée et ne permettait pas de type-hinter les paramètres ou de déclaré des type de retour de méthodes. En travaillant avec une version >= 7.0, ces déclarations sont possibles et recommandés.

Form : Séparer le Front du Back

Le bouton submit des formulaires doivent être définit dans le template du front-end et non du back-end.

Inversement, l'attribut « action » des balises form doit être définit dans les options du FormFactory.

Définir les routes « login_check » ou « logout » dans routes.yaml

Pour fonctionner, Symfony a besoin que ces routes soient définies. Les définir via annotation dans des actions vides alourdies le code inutilement et demande parfois de créer une ou plusieurs classes dans ce but.

2) Incohérences

Connecter l'utilisateur après l'enregistrement

Dans le projet initial, suite à l'enregistrement d'un membre, ce dernier avait besoin de s'authentifier.

Validation des données

Aucune restriction via validation sur le mot de passe n'était présente.

Protection contre la faille CSRF

Le formulaire de connexion n'était pas protégé contre cette faille. Symfony créer et gère la protection par défaut à condition de l'intégrer dans le formulaire, par exemple via {{ form_rest(form) }}.

Méthode incohérente dans l'entité Task

```
$task->toggle(!$task->isDone());
```

Est remplacé par :

```
$task->toggle();
```

No comment…

2. Front-End

Une barre de navigation

Un manque de cohésion dans les liens internes du site rendait la navigation pénible et créait des incohérences, comme la présence de deux boutons pour une créer une tâche dans la page de liste des pages ou l'absence de lien pour mettre à jour un utilisateur.

Une barre de navigation a permis de rentre le site plus ergonomique.

Les formulaires avec Twig

Couplé à Symfony, Twig permet d'afficher les formulaires avec facilités et de géré automatiquement le token CSRF.

Nous avons donc implémenté les formulaires avec Twig plutôt qu'en HTML.

Tri dans les assets

Le fichier shop-homepage.css comportant de nombreuses classes inutilisées, un tri à été effectué pour les supprimer.

Nous avons aussi privilégié le chargement des libraisons via CDN pour accèlérer leur chargement grâce au cache des navigateurs clients.

V. Recommandations

Pour la suite du projets, nos recommandations sont les suivantes :

- Utiliser des CDN pour les librairies fronts tières,
- Créer des pages d'erreurs personnalisés (401, 403, 404, 500),
- Optimiser l'autoload de Composer après chaque modification de dépendances,
- Implémenter un cache http,
- Activer le cache Doctrine,
- Implémenter une pagination pour les listes des tâches et des utilisateurs.