



DOCUMENTATION TECHNIQUE
AUDIT DE QUALITE ET DE PERFORMANCE

Auteur : Bourguine Bérenger FAGADE

Date de la dernière mise à jour : 15 Juin 2020

Satut : confidentiel
V1

Sommaire

Table des matières

I. METHODOLOGIE.....	3
II. LE PROJET	3
A. CONTEXTE	3
B. OBJECTIFS	3
III. AUDIT DE QUALITE	4
A. ANALYSE DE CODE INITIAL	4
1. <i>Environnement technique.....</i>	<i>4</i>
2. <i>Déploiement de l'application.....</i>	<i>4</i>
3. <i>Analyse manuelle de l'application (revue de code)</i>	<i>5</i>
4. <i>Analyse automatisée de l'application</i>	<i>5</i>
B. POINTS DES AMELIORATIONS (APPORTEE)	7
1. <i>Migration vers sf5.....</i>	<i>7</i>
2. <i>Mise en place d'outils de suivi de qualité de code.....</i>	<i>10</i>
3. <i>Mise en place des tests unitaires et fonctionnels.....</i>	<i>10</i>
IV. AUDIT DE PERFORMANCE.....	11
A. AMELIORATIONS APPORTEES.....	11
B. ANALYSES ET COMPARATIFS	11

I. METHODOLOGIE

Pour réussir un audit de qualité de site web, il est indispensable de définir le périmètre de l'audit en fonction des enjeux de l'exercice. Dans le cas d'espèce, nous allons évaluer la qualité et les performances techniques de notre application web.

Pour une évaluation plus objective, nous ferons non seulement une analyse manuelle de l'application (revue de code) mais nous utiliserons également les outils d'automatisation de l'analyse de code et de performance.

Nous ferons dans un premier temps un audit de qualité de code et dans un second temps un audit de performance.

L'approche de ces audits se déroulera en 3 phases :

- L'analyse de l'existant ;
- Les améliorations apportées ;
- Puis une analyse après les améliorations.

II. LE PROJET

A. Contexte

Vous venez d'intégrer 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).

Le choix du développeur précédent a été d'utiliser le framework PHP Symfony, un framework que vous commencez à bien connaître !

Bonne nouvelle ! **ToDo & Co** a enfin réussi à lever des fonds pour permettre le développement de l'entreprise et surtout de l'application.

Votre rôle ici est donc d'améliorer la qualité de l'application. La qualité est un concept qui englobe bon nombre de sujets : on parle souvent de qualité de code, mais il y a également la qualité perçue par l'utilisateur de l'application ou encore la qualité perçue par les collaborateurs de l'entreprise, et enfin la qualité que vous percevez lorsqu'il vous faut travailler sur le projet.

B. Objectifs

Nous avons comme mission :

- L'amélioration de la qualité du code

- L'amélioration des performances de l'application
- L'implémentation de nouvelles fonctionnalités
- La mise en place de tests automatisés

III. AUDIT DE QUALITE

A. Analyse de code initial

1. Environnement technique

Symfony	3.1
Doctrine	2.5
Swiftmailer	2.3
Bootstrap	3.3.7
PHP	7.2.4

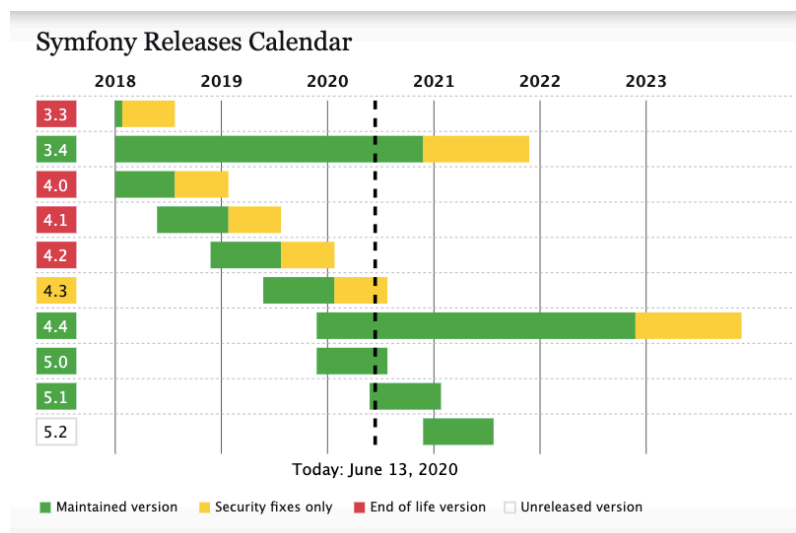
2. Déploiement de l'application

Une fois l'application récupérée depuis github, nous avons constaté entre autre :

- des erreurs fonctionnelles au lancement de l'application ;
- la dépréciation de plusieurs composants du framework (voir annexe) ;
- la dépréciation de la version du framework utilisé ;
- des problèmes de sécurité de l'application.

Dépréciation de la version de Symfony et PHP utilisée

D'après la roadmap de symfony (ci-dessous), la version 3.1 utilisé dans le cadre de notre projet n'est plus maintenue depuis 2018.



Source : <https://symfony.com/releases>

3. Analyse manuelle de l'application (revue de code)

En plus des anomalies remontées dans l'énoncé du projet, nous avons remarqué d'autres anomalies.

Au niveau fonctionnel :

- Absence d'un bouton « Accueil ».
- Le bouton « consulter la liste des tâches terminées » n'affiche rien même lorsqu'il existe de 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

Au niveau du code :

- L'utilisation de AppBundle dans les namespaces. On utilisera plutôt 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.

Au niveau sécurité:

- 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.
- Cliquer sur « afficher la liste des tâches à faire » affiche toutes les tâches de tous les utilisateurs.

4. Analyse automatisée de l'application

a. Github

B. Points des améliorations (apportées)

1. Migration vers sf5

L'application est sous une version de Symfony 3.1 (plus supportée). L'application ne profite pas des améliorations du framework. Pour résoudre les problèmes d'obsolescence de certains composants et méthodes utilisées, il est indispensable de passer à une version plus récente du framework. Lorsque nous avons initialisé ce projet, la version la plus récente était la version 5.08. Nous avons donc migré vers cette dernière version.

Cependant, de façon à mitiger la nécessité de changements fréquents de version du framework et donc de ne pas mettre l'application en péril, un upgrade vers une version LTS est à prévoir à long terme. La version LTS de la série 5 est la 5.4. La roadmap de symfony ne mentionne pas encore le date de lancement de cette release.

Après une migration, nous avons apporté des modifications indispensables au niveau de plusieurs composants (controllers, entités, sécurité, authentification, ...)

a. Controllers

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

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class DefaultController extends AbstractController
```

Nous avons également mise en place le Type Hinting dans les contrôleurs :

```
/**
 * @Route("/tasks", name="task_list")
 */
public function listAction(TaskRepository $taskRepository): Response
{
    return $this->render('task/list.html.twig', [
        'tasks' => $taskRepository->findAll()
    ]);
}
```

d. Entity

A ce niveau, nous avons :

- ajouté un champ author à l'entité Task. Ce champ est de type relation avec l'entité User.
- Ainsi lors de la création d'un task l'utilisateur connecté est assigné au Task. Pour les

anciens Tasks n'ayant pas d'utilisateur, la fonction retourne « anonyme ».

```
public function getAuthor()
{
    return $this->author !== NULL ? $this->author: "anonyme";
}
```

- mise en place l'unicité sur les champs email et username de l'entité User.

```
* @UniqueEntity(fields={"email", "username"}, message="Ce compte existe déjà.")
```

e. Securité

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

Un système d'autorisation a été mis en place avec un ROLE_USER et un ROLE_ADMIN que l'on peut choisir à la création du compte.

L'administration des comptes est restreinte aux utilisateurs possédant le rôle ROLE_ADMIN.

```
# Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
access_control:
- { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
- { path: ^/users, roles: ROLE_ADMIN }
- { path: ^/, roles: ROLE_USER }
```

Nous avons également mise en place un système de voter limitant la suppression de Task uniquement par l'auteur du Task et la suppression d'un Task anonyme uniquement par les utilisateurs ayant le ROLE_ADMIN.

// Dans la Classe TaskVoter

```
// ... (check conditions and return true to grant permission) ...
switch ($attribute) {
    case Self::DELETE:
        return $user === $subject->getAuthor();
        break;
    case Self::DELETE_ANONYME:
        //l'auteur est vide et le user est un admin
        if ($subject->getAuthor() === 'anonyme' && in_array("ROLE_ADMIN",
$user->getRoles())) {

return true;
```



```

        } else { return false; }
        break;
    }
}

```

// Dans le controller

```

/**
 * @Route("/tasks/{id}/delete", name="task_delete")
 */
public function deleteTaskAction(Task $task, EntityManagerInterface $em)
{
    if ($task->getAuthor() === 'anonyme')
    {
        $this->denyAccessUnlessGranted('TASK_DELETE_ANONYME', $task,
$message='Vous devez être admin.');
```

```

    } else {
        $this->denyAccessUnlessGranted('TASK_DELETE', $task, $message='Vous
ne disposez pas des droits de suppression.');
```

```

    }
}

```

f. Authentication

Un authenticator personnalisé a été mis en place pour le formulaire de login :

```

guard:
    authenticators:
        - App\Security\AppAuthenticator

```

Cela permet d'avoir des messages d'erreurs plus précis en cas de problème d'authentification :

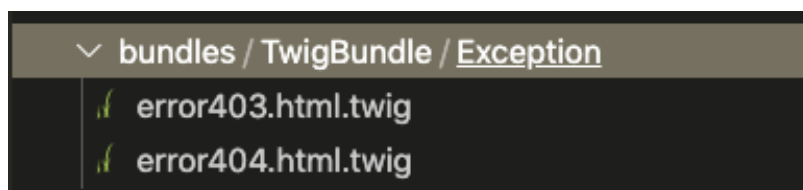
```

public function getUser($credentials, UserProviderInterface $userProvider)
{
    $token = new CsrfToken('authenticate', $credentials['csrf_token']);
    if (!$this->csrfTokenManager->isTokenValid($token)) {
        throw new InvalidCsrfTokenException();
    }
    $user = $this->entityManager->getRepository(User::class)-
>findOneBy(['username' => $credentials['username']]);

    if (!$user) {
        // fail authentication with a custom error
        throw new CustomUserMessageAuthenticationException('Identifiants
incorrects');
    }
    return $user;
}

```

g. Mise en place de Page d'erreur personnalisé dans le dossier template



2. Mise en place d'outils de suivi de qualité de code
















Une série de contrôles est mises en place de façon à réduire la dette technique de l'application.

En résumé pour assurer le suivi qualité du code, nous avons mise en place une série d'actions:

- Implémentation de nouveaux tests unitaires et fonctionnels
- L'utilisation constante d'un outil de revue du code avant déploiement (codacy et codeclimate, Travis)
- Ecriture d'un contribute.md qui établis une culture code de qualité par l'utilisation de best practice.
- Implémentation d'outils à savoir: PHP csfixer.

3. Mise en place des tests unitaires et fonctionnels.

Nous avons réalisé des tests unitaires et fonctionnels avec PHPUNIT. Nous avons un taux de couverture de 98,22% pour les lignes, 93,88% pour les méthodes et 80% pour les classes.

/Users/berengerbourgine/Todolist/src / (Dashboard)									
	Code Coverage								
		Lines		Functions and Methods			Classes and Traits		
Total		98.22%	166 / 169		93.88%	46 / 49		80.00%	8 / 10
■ Controller		100.00%	66 / 66		100.00%	10 / 10		100.00%	4 / 4
■ Entity		100.00%	48 / 48		100.00%	27 / 27		100.00%	2 / 2
■ Form		100.00%	17 / 17		100.00%	3 / 3		100.00%	2 / 2
■ Security		92.11%	35 / 38		66.67%	6 / 9		0.00%	0 / 2

Legend

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

IV. AUDIT DE PERFORMANCE

Il est important de préciser que cette analyse doit être réalisée en environnement de production car le profil de l'environnement de dev influe beaucoup sur les performances de l'application. Dans les grandes lignes, voici la liste des mesures rapportées par Blackfire, ainsi que les fonctionnalités que nous utiliserons:

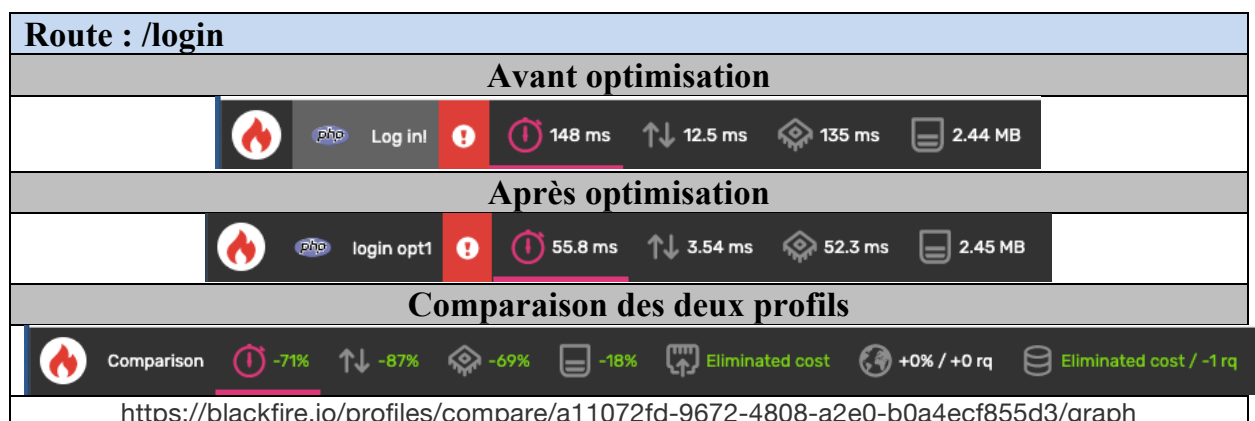
Metrics:

- Wall Time => le temps global que PHP prend pour exécuter le code (les instructions exécutées, le processus par le CPU, le volume de données en mémoire, le temps de réponse des services appelés sur le réseau etc...)
- CPU time => Le temps de réponse du processeur (temps que va mettre le processus I/O et le réseau)
- I/O time => Ce sera le temps de réponse du réseau (base de données, requête HTTP pour web service etc...) et/ou la lecture sur le disque
- MB => la mémoire utilisée par PHP pour générer la page
- Network => temps de transfert de données sur le réseau

A. Améliorations apportées

- Optimisation de l'autoloader composer du projet
- OPcache activé dans PHP

B. Analyses et comparatifs



Route : /homepage
Avant optimisation

	php	Homepage	!	123 ms	↑↓ 13.9 ms	109 ms	3.02 MB	0.803 kB
Après optimisation								
	php	Homepage Opt	!	66.3 ms	↑↓ 4.57 ms	61.7 ms	3.02 MB	0.803 kB
Comparaison des deux profils								
	Comparison		!	-46%	↑↓ -67%	-43%	+0.001%	+0%
https://blackfire.io/profiles/compare/4cce7536-22a6-4dfc-a2ed-43cf09a2ae84/graph								

Route : /tasks (La liste des tâches)								
Avant optimisation								
	php	Task_liste	!	136 ms	↑↓ 16.7 ms	119 ms	3.24 MB	2.17 kB
Après optimisation								
	php	tasks_liste_opt	!	88.4 ms	↑↓ 10.3 ms	78.1 ms	3.24 MB	2.17 kB
Comparaison des deux profils								
	Comparison		!	-35%	↑↓ -38%	-34%	+0.001%	+0%
https://blackfire.io/profiles/compare/71f0a0a6-c8be-4caa-ac53-940706421885/graph								

Route : /users (La liste des utilisateurs)								
Avant optimisation								
	php	user_list	!	176 ms	↑↓ 20.7 ms	156 ms	3.16 MB	1.79 kB
Après optimisation								
	php	Users_liste_opt	!	75.7 ms	↑↓ 9.97 ms	65.7 ms	3.16 MB	1.79 kB
Comparaison des deux profils								
	Comparison		!	-57%	↑↓ -52%	-58%	-0.01%	+0%
https://blackfire.io/profiles/compare/8442a154-217a-4878-a12b-4460a4b336a2/graph								

Commentaires

La page d'accueil est générée en 148 ms et consomme 2,44Mb de mémoire.

En analysant le call graph, (<https://blackfire.io/profiles/5a733b9b-c3f6-40a9-b161-0a3f3bc0a082/graph>) on s'aperçoit que la classe Composer\Autoload\ClassLoader

Prend le plus de temps de réponse et qu'elle est appelée 213 fois.

On pourrait considérablement réduire ce coût en dumpant l'autoloader de composer grâce à la commande « `composer dump-autoload --optimize` ».

La page des tâches est générée en 55,8 ms et consomme 2,45 Mb de mémoire.

Comme on peut le constater, le temps de génération de la page s'est bien réduit mais au détriment de la mémoire qui a légèrement augmentée. Cependant il est préférable de privilégier la vitesse à la mémoire car les serveurs en possèdent généralement une grande quantité. L'optimisation de l'autoloader explique dans une large mesure ce gain de rapidité.