

Audit technique

ToDo & Co

Juillet 2021

Sabouret Maxime

Sommaire

1 - Contexte	3
1.1 Présentation	3
1.2 Installation du projet existant	4
1.3 Structure et architecture du projet, librairies utilisées	5
1.4 Que fait le projet, quelles sont les fonctionnalités existantes ?	8
2 - Analyse de l'existant	10
2.1 Audit de performance de l'existant	10
a - Analyse Blackfire	10
2.2 Audit de qualité de l'existant	14
a - Analyse PHPStan	14
b - Analyse PHP CS (Code Sniffer)	16
c - Analyse PHP MD (Mess Detector)	16
d - Analyse PHP CPD (Copy Paste Detector)	17
e - Analyse Code Climate	17
f - Commandes déjà incluses dans Symfony et Composer	19
g - Analyse LightHouse	19
h - Analyse du code, loi de Déméter et principes SOLID	22
i - Analyse de l'UI + versions de Jquery et Bootstrap	26
2.3 Bilan des réalisations nécessaires à l'évolution du projet	27
3 - Travail réalisé et nouvel audit	29
3.1 Organisation (issues)	29
3.2 Migration du projet	29
3.3 Correction des anomalies	30
3.4 Nouvelles fonctionnalités	34
3.5 Audit de performance	37
a - Analyse Blackfire	37
3.6 Audit de qualité	39
a - Analyse PHPStan	39
b - Analyse PHP CS (Code Sniffer)	40
c - Analyse PHP MD (Mess Detector)	40
d - Analyse PHP CPD (Copy Paste Detector)	40
e - Analyse Code Climate	40
f - Commandes déjà incluses dans Symfony et Composer	42
g - Analyse LightHouse	43
h - Analyse du code et principes SOLID	44
i - Analyse de l'UI + versions de Jquery et Bootstrap	48
3.7 Test et rapport de couverture	50
a - Tests	50
b - Rapport de couverture	53
4 - Annexes	55
4.1 Etapes de la migration	55

1 - Contexte

1.1 Présentation

ToDo & Co est une startup proposant une application permettant de gérer ses tâches quotidiennes. A la date d'écriture de ce document, l'application en est au stade de Minimum Viable Product (MVP).

Le but est d'améliorer la qualité de l'application. La qualité est un concept qui englobe bon nombre de sujets :

- qualité de code
- qualité perçue par l'utilisateur de l'application
- qualité perçue par les collaborateurs de l'entreprise
- qualité que le développeur perçoit lorsqu'il travaille sur le projet

Ici, cette amélioration de la qualité passera par :

- l'implémentation de nouvelles fonctionnalités
- la correction de quelques anomalies
- l'implémentation de tests automatisés

Nous commencerons par une analyse approfondie du projet grâce à plusieurs outils afin d'avoir une vision d'ensemble de ce qui est améliorable. Nous utiliserons notamment :

- Blackfire
- Code Climate
- PHP Stan - Niveau 6
- PHP Code Sniffer - Standard PSR-12
- PHP MD
- PHP CPD
- Des commandes déjà incluses dans Symfony et composer :
 - composer unused
 - composer valid
 - symfony twig:lint
 - doctrine:schema:valid --skip-sync
- Google LightHouse

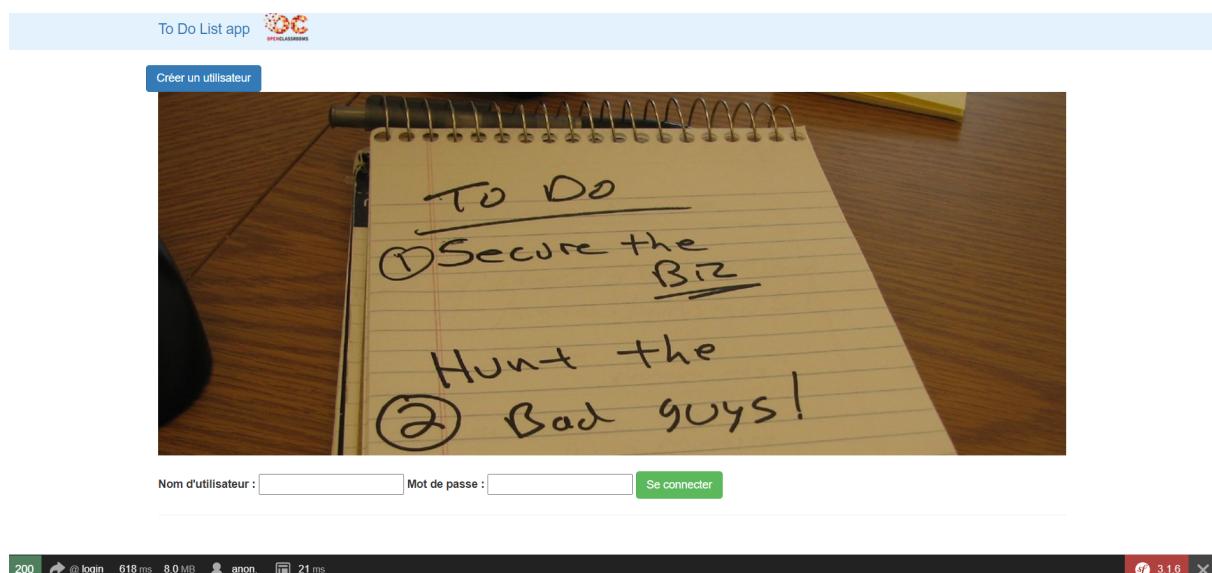
Nous effectuerons également une analyse des principes SOLID, de la version de PHP et de Symfony. Nous inspecterons l'interface du Front-office (UI, version de JQuery, de Bootstrap..).

Nous mettrons à jour le projet en réalisant une migration du vers les versions récentes stables et réutiliserons les outils d'analyse afin de connaître le gain de notre application sur chaque critère.

1.2 Installation du projet existant

Le projet de base est récupérable à l'adresse suivante :
<https://github.com/saro0h/projet8-TodoList>.

Pour l'installer, vous devez être dans une version de PHP inférieure à 7.3 mais supérieure à 5.5.9. Il suffit ensuite de fork et de cloner le projet en local. Puis vous pouvez lancer la commande "composer install" dans le terminal. Le logiciel "composer" doit être installé sur votre machine en suivant ce lien : <https://getcomposer.org/doc/00-intro.md>.



Page login

Pour que le projet fonctionne il faut également renseigner les données de la base de donnée et renseigner le schéma de celle-ci avec la commande : php bin/console doctrine:schema:create

1.3 Structure et architecture du projet, librairies utilisées

La MVP a été réalisée en Symfony 3.1. Version stable de Symfony en Mai 2016 (<https://symfony.com/releases/3.1>).

```
✓ PROJET8-TODOLIST
  ✓ app
    ✓ config
      ! config_dev.yml
      ! config_prod.yml
      ! config_test.yml
      ! config.yml
      ! parameters.yml
      ≡ parameters.yml.dist
      ! routing_dev.yml
      ! routing.yml
      ! security.yml
      ! services.yml
    > Resources\views
    ⚙ .htaccess
    📄 AppCache.php
    📄 AppKernel.php
    📄 autoload.php
  > bin
  ✓ src
    ✓ AppBundle
      > Controller
      > Entity
      > Form
      📄 AppBundle.php
      ⚙ .htaccess
    > tests\AppBundle\Con...
    > var
    > vendor
    > web
    📄 .gitignore
    { composer.json
    { composer.lock      M
    ≡ phpunit.xml.dist
    ⓘ README.md
```

Structure Symfony 3.1

Le projet en 3.1 dispose des librairies suivantes :

```
"require": {  
    "php": ">=5.5.9",  
    "symfony/symfony": "3.1.*",  
    "doctrine/orm": "^2.5",  
    "doctrine/doctrine-bundle": "^1.6",  
    "doctrine/doctrine-cache-bundle": "^1.2",  
    "symfony/swiftmailer-bundle": "^2.3",  
    "symfony/monolog-bundle": "^2.8",  
    "symfony/polyfill-apcu": "^1.0",  
    "sensio/distribution-bundle": "^5.0",  
    "sensio/framework-extra-bundle": "^3.0.2",  
    "incenteev/composer-parameter-handler": "^2.0"  
},  
"require-dev": {  
    "sensio/generator-bundle": "^3.0",  
    "symfony/phpunit-bridge": "^3.0"  
}
```

Vous trouverez dans le tableau ci-dessous le rôle de ces différentes librairies :

Noms des librairies	Rôle
php	La version de php minimale pour un projet Symfony 3.1 est 5.5.9, lorsque nous passerons à une version plus récente, la version minimale de php évoluera vers 7.2.5 (https://symfony.com/doc/current/setup.html)
symfony/symfony	Cette librairie permet d'installer symfony et ses différentes dépendances. Il ne sera cependant plus nécessaire au passage à Symfony 4 et à l'implémentation de Flex (https://symfony.com/doc/current/setup/flex.html).

doctrine/orm et doctrine/doctrine-bundle	Doctrine est l'ORM utilisé par Symfony permettant de gérer la transformation des objets (entités) de notre application PHP en requêtes SQL. La version actuelle de doctrine/orm est 2.9.3 et celle de doctrine/doctrine-bundle est 2.4.2.
doctrine/doctrine-cache-bundle	Cache de doctrine, ce bundle n'est plus maintenu (https://packagist.org/packages/doctrine/doctrine-cache-bundle). Il faut maintenant passer par symfony/cache.
symfony/swiftmailer-bundle	Swiftmailer. Ce bundle permet d'envoyer des messages à travers notre application. Il n'est plus conseillé pour les versions récentes de symfony. La documentation préconise l'utilisation du composant Mailer (https://symfony.com/doc/current/mailer.html).
symfony/monolog-bundle	Monolog permet de gérer les logs dans une application, la version récente est 3.7.0.
symfony/polyfill-apcu	Apcu est un gestionnaire de cache (https://www.php.net/manual/fr/book.apcu.php).
sensio/distribution-bundle	Flex remplace ce bundle (https://packagist.org/packages/sensio/distribution-bundle).
sensio/framework-extra-bundle	Permet de configurer les controllers avec des annotations. La version actuelle est 6.1.5.
incenteev/composer-parameter-handler	Permet, dans composer, que le script d'installation de composer copie également le contenu du fichier parameters.yml.dist dans le fichier parameters.yml. Ceci ne sera pas nécessaire avec Flex.

sensio/generator-bundle	Ce bundle permet d'automatiser certaines actions dans symfony et est maintenant remplacé par symfony/maker-bundle .
symfony/phpunit-bridge	Permet de mettre en place les tests, une version plus récente sera également mise en place étant donné qu'aucun test n'est de toute façon écrit.

1.4 Que fait le projet, quelles sont les fonctionnalités existantes ?

Le projet est organisé de la manière suivante :

Entités	
User (id, username, password, email)	Permet de représenter un utilisateur, cette entité implémente la UserInterface, elle est donc utilisée lors du processus d'authentification. Cette entité sera modifiée lors de l'évolution de l'autorisation et de l'authentification.
Task (id, createdAt, title, content, isDone)	Permet de représenter une tâche, lorsque celle-ci est réalisée, l'attribut isDone passe à true.
Controllers	
DefaultController	Affiche la page d'accueil
SecurityController	Gère l'authentification et la déconnexion

TaskController	Contient toute la logique pour lister les tâches, créer une tâche, éditer une tâche, supprimer une tâche et valider ou invalider une tâche (toggle)
UserController	Contient toute la logique pour lister les utilisateurs, créer un utilisateur, modifier un utilisateur
Formulaires	
TaskType	Permet de créer le formulaire d'ajout ou d'édition d'une tâche
UserType	Utilisé pour la création et l'édition d'un utilisateur
Vues	
Dossier app/Resources/views : Chaque vue correspond aux différents controllers explicités plus haut. Le projet utilise le moteur de template Twig avec la version 3 du framework CSS Bootstrap. Nous passerons à la version 5 de ce framework.	
Dossier Web	
L'application contient différentes images, fichiers javascripts et un fichier css principal : shop-homepage.css. Ces différentes ressources seront déplacées soit dans le dossier public soit dans les assets lors de l'implémentation de Flex et de Webpack Encore.	
Webpack Encore est une librairie conseillée depuis les nouvelles versions de Symfony pour gérer la partie frontend de nos applications (https://symfony.com/doc/current/frontend.html). Elle permet notamment de minifier les fichiers CSS et JS pour gagner en performance, mais elle permet également d'installer rapidement et simplement des outils fronts tel que Sass ou React au sein de notre application.	
Dossier config	
Le dossier app/config contient la configuration de notre application.	

Dossier test

Ce fichier contiendra l'ensemble des tests fonctionnels lorsque nous les implémenterons après la migration.

Vous retrouverez les schémas UML au lien suivant :
<https://github.com/Maxime22/projet8-TodoList/tree/master/diagrammes>.

2 - Analyse de l'existant

2.1 Audit de performance de l'existant

a - Analyse Blackfire

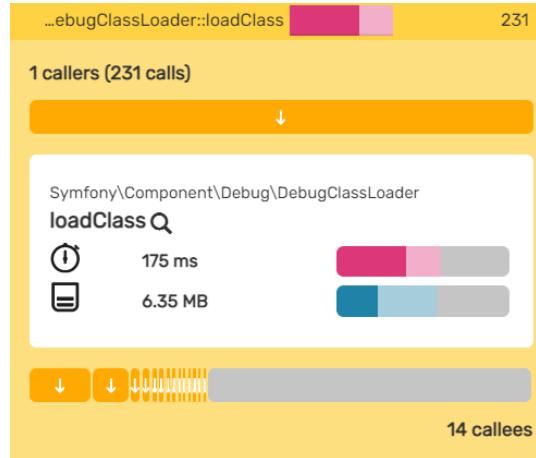
BlackFire est un logiciel développé par SensioLab permettant de tester la performance d'une application sur différents paramètres comme le temps d'exécution ou la mémoire vive consommée. Ces métriques nous sont déjà données par le profiler de Symfony mais Blackfire nous permet d'aller plus loin en analysant chaque fonction de manière individuelle (exclude) ou globale (include) et offre plus de paramètres d'analyse (temps d'accès aux fichiers, temps d'accès au processeur, quantité de données qui passe par le réseau...). De plus Blackfire est disponible en SaaS, l'outil n'a donc pas d'impact sur la performance de notre projet lorsqu'il est lancé car il ne fait pas partie intégrante de notre application comme peut l'être le profiler de Symfony.

Nous allons ainsi profiler chaque page de notre application grâce à la version gratuite de Blackfire. Nous regarderons les valeurs exclues (individuelles) pour déterminer l'impact des différentes fonctions sur l'application.

Login (/login) :

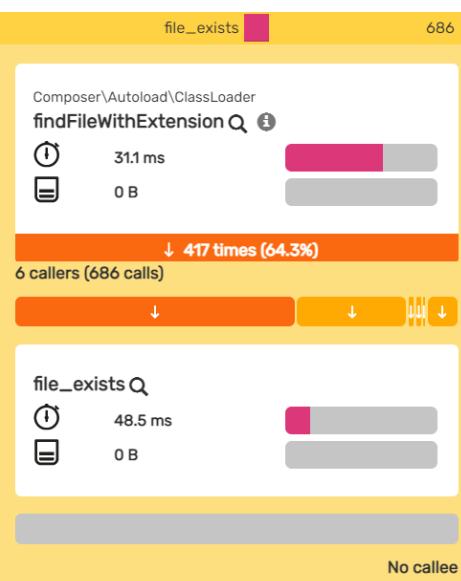


La route login met 291ms à s'afficher, la principale source de perte en performance provient de `Symfony\Component\Debug\DebugClassLoader::loadClass()` qui est une méthode appelée 231 fois et met 175ms à se charger (soit 60% du temps total de chargement de la page). Cette méthode fait partie d'un composant de Symfony qui disparaît dans les versions ultérieures. Nous ne devrions plus avoir cet énorme impact de performance suite à la migration (https://symfony.com/doc/3.3/components/class_loader.html)



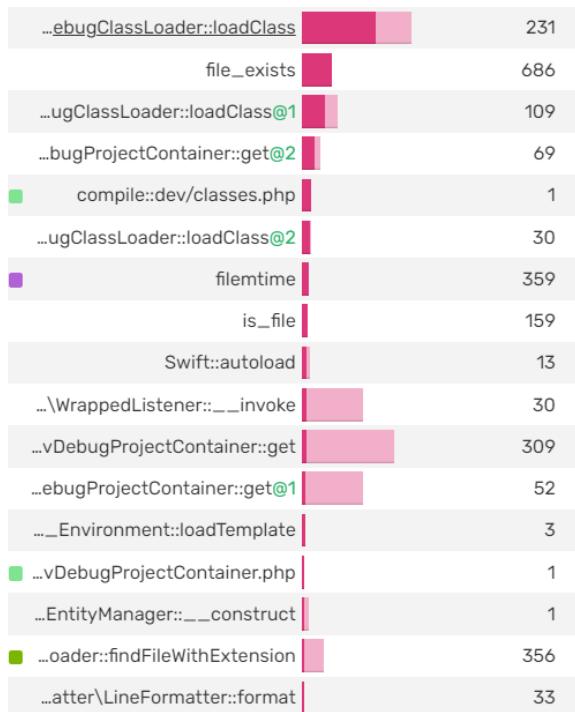
Résultats fonction `loadClass`

Une autre valeur importante est celle de la fonction `file_exists`. Celle-ci est utilisée par plusieurs fonctions dans le code mais principalement par l'autoloader de composer. Les appels de composer représentent 31.1ms soit quasiment 11% du chargement total de la page. Pour optimiser cela nous pourrions utiliser la commande `dump-autoload` de composer qui permet de mettre en cache les classes de l'application (à refaire si celle-ci sont modifiées).



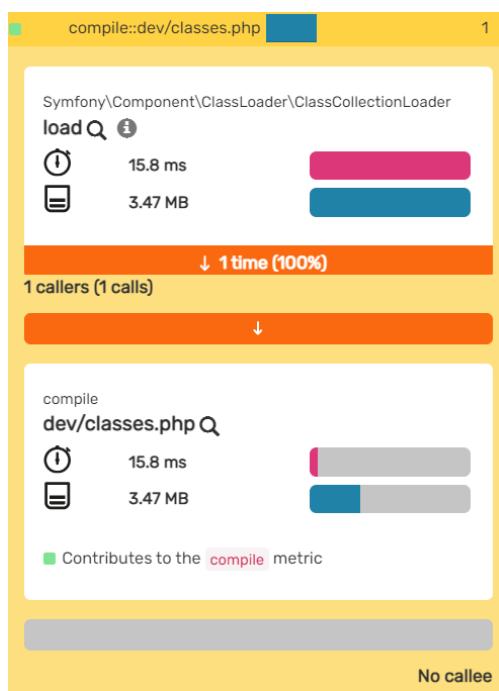
Résultats fonction `file_exists`

Autrement, il ne semble pas y avoir beaucoup de méthodes qui ont un fort impact sur notre application pour la route /login :



Résultats exclusifs des temps d'exécution

En termes de mémoire, le rapport est le même. Le ClassLoader est le plus gourmand :



Résultats mémoire la méthode load()

Accueil (/) : Le rapport est quasiment le même pour l'ensemble des routes qui suivent, en effet, comme le code de l'application ne contient pas beaucoup de logique, cette logique n'impacte pas réellement la performance de notre application. Par soucis de clarté, nous afficherons donc seulement les temps d'exécution de chaque route.



Liste des tâches (/tasks) :



Création d'une tâche (/tasks/create) :



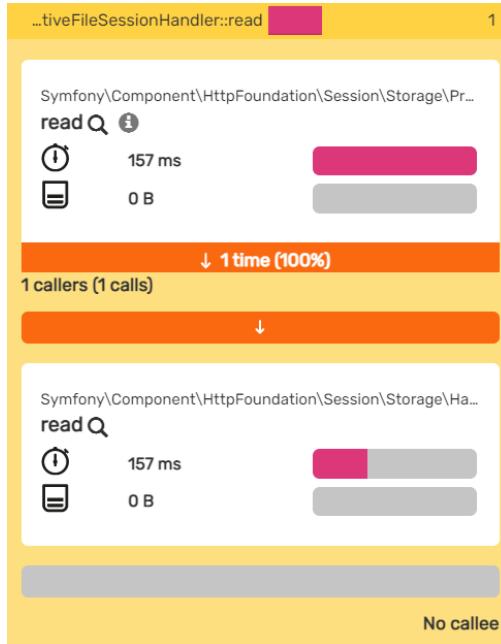
Modification d'une tâche (/tasks/{id}/edit) :



Liste des utilisateurs (/users) :

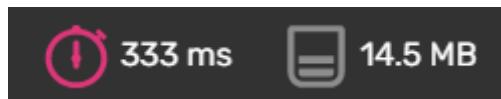


La page d'affichage des utilisateurs met un peu plus de temps à s'afficher que les autres. Ceci est dû à la fonction `read` de `Symfony\Component\HttpFoundation\Session\Storage\Proxy\SessionHandlerProxy`. C'est également un composant interne de symfony qui évoluera avec la migration.



Résultats fonction read

Création d'un utilisateur (/users/create) :



Modification d'un utilisateur (/users/{id}/edit) :



2.2 Audit de qualité de l'existant

a - Analyse PHPStan

PHPStan est un outil permettant d'analyser le code PHP (<https://phpstan.org/user-guide/getting-started>). Le but étant de connaître les erreurs que nous pourrions modifier nous même dans le code PHP, il est inutile de scanner le code Symfony car il n'est pas en notre pouvoir de le modifier ou en tout cas ce n'est pas le sujet de cet audit.

Installation de PHP Stan : composer require --dev phpstan/phpstan

Une fois installé (analyse des classes dans src), une analyse basique se fait avec la commande suivante : vendor/bin/phpstan analyse src. Cette commande fait les checks les plus basiques mais ne vérifie pas les types par exemple.

Pour une analyse plus approfondie, il faut éléver le niveau de vérification (<https://phpstan.org/user-guide/rule-levels>) : vendor/bin/phpstan analyse -l 6 src

Le résultat est le suivant :

```
Line  AppBundle\Controller\DefaultController.php
13  Method AppBundle\Controller\DefaultController::indexAction() has no return typehint specified.

Line  AppBundle\Controller\SecurityController.php
14  Method AppBundle\Controller\SecurityController::loginAction() has no return typehint specified.
30  Method AppBundle\Controller\SecurityController::loginCheck() has no return typehint specified.
38  Method AppBundle\Controller\SecurityController::logoutCheck() has no return typehint specified.

Line  AppBundle\Controller\TaskController.php
16  Method AppBundle\Controller\TaskController::listAction() has no return typehint specified.
24  Method AppBundle\Controller\TaskController::createAction() has no return typehint specified.
48  Method AppBundle\Controller\TaskController::editAction() has no return typehint specified.
71  Method AppBundle\Controller\TaskController::toggleTaskAction() has no return typehint specified.
84  Method AppBundle\Controller\TaskController::deleteTaskAction() has no return typehint specified.

Line  AppBundle\Controller\UserController.php
16  Method AppBundle\Controller\UserController::listAction() has no return typehint specified.
24  Method AppBundle\Controller\UserController::createAction() has no return typehint specified.
50  Method AppBundle\Controller\UserController::editAction() has no return typehint specified.

Line  AppBundle\Entity\Task.php
19  Property AppBundle\Entity\Task::$id has no typehint specified.
24  Property AppBundle\Entity\Task::$createdAt has no typehint specified.
30  Property AppBundle\Entity\Task::$title has no typehint specified.
36  Property AppBundle\Entity\Task::$content has no typehint specified.
41  Property AppBundle\Entity\Task::$isDone has no typehint specified.
```

Résultat analyse PHP Stan de niveau 6

Nous voyons donc qu'il est nécessaire de type hinter l'ensemble des retours dans notre application pour éviter d'éventuels mauvais retours de fonctions et d'éventuels bugs sous-jacents.

Étant sur php 7.2 au départ, la version ne supporte pas encore toutes ces façons de typer, nous réglerons certaines de ces erreurs après être passé à une version supérieure de PHP (7.4).

A ce stade, nous corrigéons 34 erreurs de typages grâce à PHPStan. Il est nécessaire de relancer ce type d'analyse constamment pour vérifier si notre application (variables, fonctions) est bien typée. Nous enlèverons les erreurs restantes après la migration.

b - Analyse PHP CS (Code Sniffer)

Code Sniffer permet de savoir si le code respecte les standards PHP (notamment le PSR 12 pour le coding style) (https://github.com/squizlabs/PHP_CodeSniffer).

Installation de PHP CS : composer require --dev squizlabs/php_codesniffer

Pour savoir si notre code respecte les standards PSR12 il suffit d'utiliser la commande :
./vendor/bin/phpcs --standard=PSR12 src

La plupart des erreurs trouvées peuvent être corrigées grâce à PHP CBF :
./vendor/bin/phpcbf --standard=PSR12 src

Les erreurs restantes concernent un trop grand nombre de caractères sur certaines lignes mais ne sont que des messages préventifs. Au cours de l'évolution du code, il sera nécessaire de répéter cette analyse régulièrement pour vérifier le coding style de l'application.

De plus, sans forcément typer, il est de bon ton d'écrire la phpdoc pour informer sur ce qui est attendu. Sans mettre d'options particulières, on peut voir la phpdoc manquante grâce à la commande suivante qui vérifie les standards PEAR (<https://pear.php.net/manual/en/standards.php>) : ./vendor/bin/phpcs src

c - Analyse PHP MD (Mess Detector)

PHPMD est un outil permettant de relever les mauvaises pratiques de conception logiciel (code smells en anglais).

Installation de PHP MD : composer require phpmd/phpmd

Il vérifie plusieurs types de problèmes variés (Available rulesets: cleancode, codesize, controversial, design, naming, unusedcode) et rend le résultat de son analyse sous différents formats : ansi, baseline, checkstyle, github, html, json, sarif, text, xml.

Pour une analyse simple, nous pouvons notamment utiliser la règle cleancode et la règle codesize afin de savoir si le code dans src présente des anomalies.

Les commandes “./vendor/bin/phpmd src text codesize” et “./vendor/bin/phpmd src text cleancode” renvoient des rapports nuls. Nous pouvons donc établir que le code de base ne présente pas de code smells importants.

d - Analyse PHP CPD (Copy Paste Detector)

Comme son nom l'indique, PHP CPD permet de savoir si nous avons du code dupliqué. Qui dit code dupliqué dit mauvaise pratique.

Installation de PHP CPD : composer require --dev sebastian/phpcpd

Après une analyse du fichier src avec “./vendor/bin/phpcpd src”, aucun clone n'est trouvé. Notre code est donc propre à ce niveau là.

e - Analyse Code Climate

Code Climate est un logiciel, comme PHP MD, qui détecte les code smells et qui, comme PHP CPD, permet d'analyser la duplication de code. C'est un outil polyvalent qui permet également de connaître le test coverage.

L'avantage majeur de Code Climate est sa simplicité d'utilisation et son interface graphique agréable. De plus, il est directement lié à une branche github, ce qui permet d'analyser rapidement l'ensemble de son projet.



Résultat de l'analyse Code Climate

Code Climate révèle qu'il existe de nombreux codes smells et de nombreuses duplications de code. Cependant ces mauvaises pratiques concernent tous des fichiers qui ne sont pas

en rapport direct avec ce que les développeurs de l'application ont écrit. En effet, toutes les duplications de code et la plupart des code smells appartiennent au fichier bootstrap.js (framework CSS externe avec sa propre dette technique). Pour les codes smells restants, ils concernent majoritairement la classe SymfonyRequirements qui ne sera plus la même lors de la migration de version.

File `bootstrap.js` has 1596 lines of code (exceeds 250 allowed).

Consider refactoring.

[OPEN](#)

```
1  /*
2   * Bootstrap v3.3.7 (http://getbootstrap.com)
3   * Copyright 2011-2016 Twitter, Inc.
4   * Licensed under the MIT license
5   */
```

•••• Found in [web/js/bootstrap.js](#) - About 4 days to fix

Method `__construct` has 316 lines of code (exceeds 25 allowed).

Consider refactoring.

[OPEN](#)

```
388     public function __construct()
389     {
390         /* mandatory requirements follow */
391
392         $installedPhpVersion = phpversion();
```

•••• Found in [var/SymfonyRequirements.php](#) - About 1 day to fix

Exemple de code smells relevés par Code Climate

Si on regarde les fichiers ajoutés par l'équipe, ils ne font pas partie de la dette technique.

[Maintainability](#) [Test Coverage](#)



[src/AppBundle/Controller/SecurityController.php](#) was added.



[src/AppBundle/Controller/UserController.php](#) was added.



[src/AppBundle/Entity/User.php](#) was added.



[src/AppBundle/Form/UserType.php](#) was added.

Résultat de l'analyse des fichiers src

f - Commandes déjà incluses dans Symfony et Composer

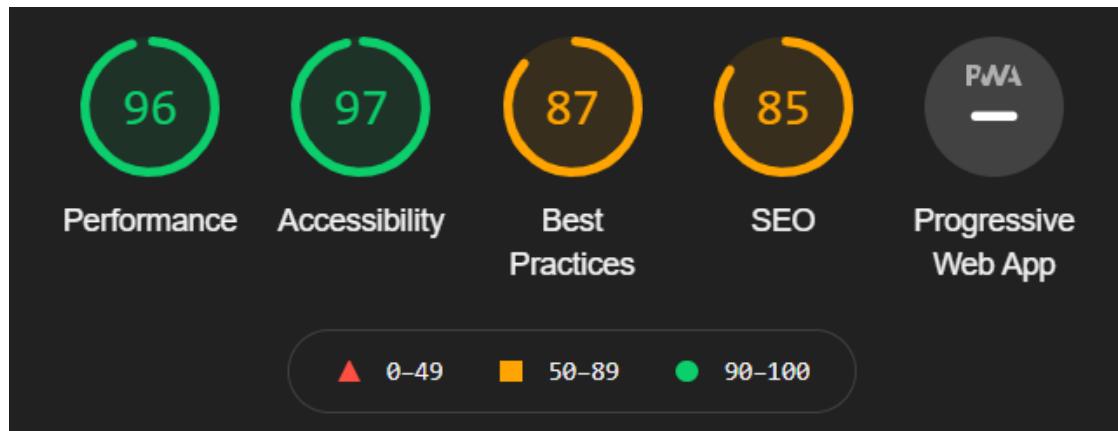
Plusieurs commandes déjà incluses dans Symfony et Composer peuvent nous permettre d'améliorer notre analyse globale :

Commande	Utilité et résultat
composer unused (https://github.com/composer-unused/composer-unused)	Permet de connaître les packages non utilisés dans composer. Nous utiliserons cet outil après la migration à PHP 7.4.
composer valid	Cette commande permet de nous indiquer si le schéma de composer est bon. Dans notre cas, le schéma du composer est bon, il manque simplement une éventuelle description et à redéfinir proprement l'autoload sans mettre "" qui peut avoir certains coûts en performance.
php bin/console lint:twig app/Resources/views (symfony twig:lint pour les versions plus récentes)	Check la syntaxe de nos fichiers twig. La syntaxe de nos 9 fichiers twig est valide
php bin/console doctrine:schema:valid --skip-sync	Permet de vérifier le schéma de notre base de données. Le schéma de notre base de données est correct.

g - Analyse LightHouse

LightHouse est un logiciel proposé par Google qui permet de calculer différentes métriques lors de l'affichage d'une page (vitesse d'affichage, SEO, etc...). La version basique s'installe en tant qu'extension sur le navigateur Chrome. LightHouse fournit une liste de recommandations sur la façon d'améliorer les performances des pages, de rendre les pages plus accessibles, de suivre les meilleures pratiques etc... Il teste également l'application sur version mobile.

L'UI étant très basique, la performance de l'ensemble des pages de l'application est de bonne qualité.



Résultat Lighthouse de la page login

Les légers problèmes de performances sont dus à une mauvaise gestion du chargement des fichiers CSS qui bloquent l'affichage de la page.

Opportunity	Estimated Savings	
▲ Eliminate render-blocking resources	0.96 s ^	
URL	Transfer Size	Potential Savings
/css/bootstrap.min.css (127.0.0.1)	118.5 KiB	1,220 ms
/css/shop-homepage.css (127.0.0.1)	1.0 KiB	620 ms

Résultat détaillé (CSS) Lighthouse de la page login

Des éléments du CSS (bootstrap) sont inutilisés :

■ Remove unused CSS	0.43 s ^	
Remove dead rules from stylesheets and defer the loading of CSS not used for above-the-fold content to reduce unnecessary bytes consumed by network activity. Learn more .		
URL	Transfer Size	Potential Savings
/css/bootstrap.min.css (127.0.0.1)	118.5 KiB	114.6 KiB

Résultat détaillé (Bootstrap) Lighthouse de la page login

Les images n'ont pas toujours une hauteur et une largeur définie :

The screenshot shows a Lighthouse audit report for an image element. At the top, a warning message reads: "Image elements do not have explicit width and height. Set an explicit width and height on image elements to reduce layout shifts and improve CLS. Learn more". Below this is a "Failing Elements" section. It lists two URLs: "/img/todolist_homepage.jpg" and "/img/Logo_OpenClassrooms.png". Each URL has a preview image and the word "img" next to it. A button labeled "Show 3rd-party resources (0)" is visible at the top right.

Résultat détaillé (images) Lighthouse de la page login

Un cache pourrait également améliorer les performances :

The screenshot shows a Lighthouse audit report for static assets. A warning message says: "Serve static assets with an efficient cache policy — 5 resources found. A long cache lifetime can speed up repeat visits to your page. Learn more". Below this is a "Cache TTL" table. It lists five resources with their current cache settings and sizes:

URL	Cache TTL	Transfer Size
/css/bootstrap.min.css (127.0.0.1)	None	119 KiB
/img/todolist_homepage.jpg (127.0.0.1)	None	112 KiB
/js/bootstrap.min.js (127.0.0.1)	None	36 KiB
/img/Logo_OpenClassrooms.png (127.0.0.1)	None	7 KiB
/css/shop-homepage.css (127.0.0.1)	None	1 KiB

A button labeled "Show 3rd-party resources (0)" is located at the top right.

Résultat détaillé (cache) Lighthouse de la page login

En ce qui concerne l'accessibilité, Google nous informe que le contraste de couleur peut être gênant pour l'utilisateur.

Pour les best practices, LightHouse nous rappelle qu'il faudrait corriger les erreurs javascripts et mettre une image de logo plus adaptée.

Enfin, la partie SEO n'est pas réellement ce qui nous importe dans ce type d'application même si les recommandations sont toujours intéressantes.

Liste des tâches (/tasks) : Pour cette page en particulier, LightHouse prévient qu'il serait judicieux d'utiliser <link rel=preload> pour demander à charger en priorité une "glyphicon".

Autrement, les autres pages retournent toutes le même type de rapport.

h - Analyse du code, loi de Déméter et principes SOLID

La loi de Déméter (Principe de connaissance minimale) est une règle de conception pour développer un logiciel qui peut être résumée en « Ne parlez qu'à vos amis immédiats ». Un objet A peut requérir un service (appeler une méthode) d'un objet B, mais A ne peut pas utiliser B pour accéder à un troisième objet (C) et requérir ses services.

La loi de Déméter dans notre code :

Aucune de nos classes n'utilise des classes par l'intermédiaire d'autres classes. Lorsqu'un controller instancie une tâche par exemple, il n'utilise pas cette tâche pour instancier via son intermédiaire d'autres instances qu'il utiliserait. Les méthodes d'une Task n'instanciant de toute manière aucune classe avec des méthodes particulières.

En programmation orientée objet, **SOLID** est un acronyme mnémonique qui regroupe cinq principes de conception destinés à produire des architectures logicielles plus compréhensibles, flexibles et maintenables. Rappelons rapidement les différents acronymes :

- S (Responsabilité Unique) : une classe ne devrait avoir qu'une et une seule raison de changer
- O (Principe Ouvert/Fermé) : les classes d'un projet devraient être ouvertes à l'extension, mais fermées à la modification.
- L (Principe de Liskov) : il doit être possible de substituer une classe "parente" par l'une de ses classes enfants
- I (Ségrégation d'interface) : une interface = un besoin
- D (Inversion des dépendances) : les classes de haut niveau ne devraient pas dépendre directement des classes de bas niveau, mais d'abstractions

Les principes SOLID dans notre code :

De manière globale, assez peu de code est écrit ce qui laisse peu de place à d'éventuelles erreurs. De plus, la structure même du framework nous invite souvent à respecter ces principes pour des actions simples. Dans le code, chaque fonction correspond à une route et donc à une action particulière (S de SOLID).

Pour ce qui est du principe de Liskov, il n'existe pas de classes enfants qui réutilisent des méthodes des classes parents pour le moment. Les classes écrites n'ont pas pour but d'être étendues. Cependant, il faut avoir en tête la mise en place d'interfaces pour respecter ce principe si le code se complexifie.

De même pour la ségrégation d'interface, pour avoir de la séparation d'interfaces, il faudrait déjà avoir des interfaces et que celles-ci soient utiles à notre code.

Pour ce qui est du problème d'inversion de dépendance, aucune de nos classes ne dépend d'autres classes dans son constructeur. **Nous n'utilisons pas l'injection de dépendance ce qui est contraire au DIP.**

Autrement, pour rentrer plus dans le détails, quelques remarques sont à prendre en compte concernant les controllers, les entités et les formulaires. :

Controller	Détail
DefaultController	La classe ne comporte qu'une fonction qui n'a qu'une seule responsabilité, celle d'afficher la vue, l'ensemble des principes SOLID est respecté pour le moment.
SecurityController	La classe comporte 3 fonctions liées à la sécurité dont 2 sans code car juste utilisées lors de la connexion et de la déconnexion. La seule fonction ayant du code s'occupe uniquement de la connexion d'un utilisateur et délègue l'authentification à un service authenticationUtils. Elle respecte l'ensemble des principes SOLID.

TaskController	<ul style="list-style-type: none"> - listAction : cette méthode ne s'occupe que de rendre la liste des tâches en utilisant un repository - createAction et editAction : ces méthodes ont deux conditions semblables qui vérifient la validité du formulaire (if (\$form->isValid())). Nous pourrions mettre cette condition dans une autre fonction afin de limiter le code dupliqué. De plus ces routes du controller ne devraient pas implémenter la logique de traitement des formulaires. Ceci est contraire au SRP. - toggleTaskAction et deleteTaskAction : rien à signaler de particulier
UserController	<p>Mêmes remarques que pour TaskController, les fonctions sont les mêmes hormis toggle et delete qui n'existent pas.</p>

Entité	Détail
Task	<p>La classe ne fait que déclarer des attributs, créer des getters et des setters de ces attributs. Créer des classes TaskGetters et TaskSetters feraient partie d'une suroptimisation non nécessaire. Autrement, les éléments de cette classe sont très simples et respectent donc les principes SOLID.</p>
User	<p>La seule différence avec Task dans sa structure est le fait qu'elle implémente l'interface UserInterface. Les remarques de Task sont applicables à User.</p>

Formulaire	Détail
TaskType et UserType	<p>Les formulaires n'ont qu'une méthode build qui est une méthode étendue de la classe AbstractType. Elle respecte donc son implementation. Pour TaskType, les options de title pourraient être plus précises.</p>

Tout comme les principes SOLID sont un ensemble de bonnes pratiques, les pratiques **STUPID** représentent des indices de code dont la qualité est insuffisante :

- S (Singleton) : On évite ce “design pattern” qui consiste à empêcher de pouvoir créer plusieurs instances d’une même classe.
- T (Couplage fort) : Les classes ne doivent pas dépendre d’autres classes. On privilégie le polymorphisme avec les interfaces et les classes abstraites.
- U (Non testable) : On fait des méthodes simples, qui ne font qu’une chose et qui sont testables facilement.
- P (Optimisation prématuée) : On évite de vouloir optimiser trop tôt sous peine de complexifier sans utilité.
- I (Nommage non descriptif) : On met des noms qui définissent ce que la classe, la fonction ou la variable fait.
- D (duplication) : On évite la duplication de code.

Les principes STUPID dans notre code :

S : Il n’existe aucun Singleton pour le moment dans l’ensemble du code. Toutes les classes peuvent être appelées plusieurs fois.

T : Aucun couplage fort n’est visible. Les classes ne dépendent pas d’autres classes dans leurs constructeurs.

U : Les fonctions sont toutes facilement testables pour le moment.

P : Il ne semble pas y avoir d’optimisation prématuée.

I : DefaultController est un nom qui ne veut pas dire grand chose, On peut le remplacer par HomepageController.

Les noms comme listAction, editAction ne seront plus nécessaire après la migration du framework. Le “Action” ne voulant pas spécialement dire quelque chose, on peut remplacer ce nommage par listTasks par exemple.

D : Suite à l’analyse de l’existant, on a vu qu’aucun code n’était dupliqué hormis dans des fichiers sources de Bootstrap notamment.

i - Analyse de l'UI + versions de Jquery et Bootstrap

Le but de cette mise à jour n'est pas de refaire totalement l'UI du projet. Cependant nous pouvons relever quelques points intéressants qu'il serait utile de changer pour améliorer l'expérience utilisateur.

Page d'accueil (/) :



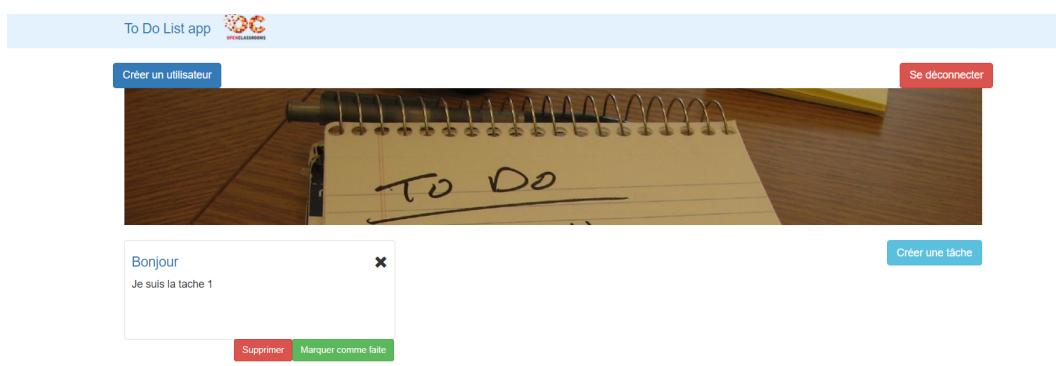
The screenshot shows the homepage of the "Todo List app". At the top, there is a blue header bar with the text "To Do List app" and the "OC OPENCLASSROOMS" logo. Below the header, there are two buttons: "Créer un utilisateur" (Create user) on the left and "Se déconnecter" (Logout) on the right. The main content area features a photograph of an open spiral-bound notebook. The left page has the handwritten text "TO DO" at the top, followed by a circled heart symbol and the text "Secure the Biz". The right page has the handwritten text "Hunt the" and a circled number "2" followed by "Bad guys!". Below the photograph, there are three buttons: "Créer une nouvelle tâche" (Create a new task) in green, "Consulter la liste des tâches à faire" (Check the list of tasks to do) in blue, and "Consulter la liste des tâches terminées" (Check the list of completed tasks) in grey.

Page d'accueil

Les marges sont respectées, cependant, il serait plus agréable pour l'utilisateur d'avoir la gestion d'utilisateur dans la navbar et la création des tâches en dessous de l'image. Nous pouvons également mettre les boutons de connexion et déconnexion dans la navbar.

Le bouton en haut à gauche "TodoList App" ne renvoie pas, pour le moment, vers la page d'accueil. Ce qui serait profitable.

Liste des tâches (/tasks) :



The screenshot shows the "Tasks" page of the "Todo List app". At the top, it features the same blue header bar with the "To Do List app" and "OC OPENCLASSROOMS" logo. Below the header, there are two buttons: "Créer un utilisateur" (Create user) on the left and "Se déconnecter" (Logout) on the right. The main content area shows a photograph of the same spiral-bound notebook with the "TO DO" list. In the foreground, a modal window is open. The modal has a white background and contains the text "Bonjour" and "Je suis la tâche 1". At the bottom of the modal, there are two buttons: "Supprimer" (Delete) in red and "Marquer comme faite" (Mark as done) in green. To the right of the modal, there is a blue button labeled "Créer une tâche" (Create a task).

Page liste des tâches

Il manque des marge sur la plupart des boutons. Nous voudrions quelque chose de plus aéré et ergonomique. Nous décollerons donc l'ensemble de ces boutons.

Création d'une tâche (/tasks/create) : Mêmes remarques que pour /tasks

Modification d'une tâche (/tasks/{id}/edit) : Mêmes remarques que pour /tasks

Liste des utilisateurs (/users) : L'UI est correcte mais aucun bouton ne permet d'accéder à cette page. Comme dit plus haut, un bouton dans la navbar serait judicieux.

Création d'un utilisateur (/users/create) : L'UI est correcte

Modification d'un utilisateur (/users/{id}/edit) : L'UI est correcte

Version de Bootstrap et Jquery : Le projet utilise la version 3 de Bootstrap. La version actuelle est la version 5, nous mettrons le projet à jour à ce niveau là. En ce qui concerne Jquery, lorsqu'on lance le navigateur, on trouve l'erreur GET <http://127.0.0.1:8000/js/jquery.js> net::ERR_ABORTED 404 (Not Found). Le jquery de bootstrap n'est donc pas implémenté car pas utilisé dans le projet. Nous mettrons à jour cette dépendance.

2.3 Bilan des réalisations nécessaires à l'évolution du projet

Migration vers 5.3 :

Comme nous l'avons spécifié en amont, le projet ne peut rester en Symfony 3.1. Nous allons donc migrer l'ensemble de l'application vers la version 5.3 en enlevant toutes les dépréciations et en passant à une structure Flex.

Régler les anomalies :

Une tâche doit être attachée à un utilisateur

- Actuellement, lorsqu'une tâche est créée, elle n'est pas rattachée à un utilisateur. Il faut qu'automatiquement, à la sauvegarde de la tâche, l'utilisateur authentifié soit rattaché à la tâche nouvellement créée. Il manque pour cela un attribut "author" dans l'entité Task que l'on liera à notre entité User.
- Lors de la modification de la tâche, l'auteur ne peut pas être modifié.
- Pour les tâches déjà créées, il faut qu'elles soient rattachées à un utilisateur anonyme.

Choisir un rôle pour un utilisateur

- Lors de la création d'un utilisateur, il doit être possible de choisir un rôle pour celui-ci. Pour cela, il va falloir créer un attribut "roles" dans l'entité User qui n'existe pas pour le moment. Les rôles listés sont les suivants :
 - rôle utilisateur (ROLE_USER) ;
 - rôle administrateur (ROLE_ADMIN).
- Lors de la modification d'un utilisateur, il est également possible de changer le rôle d'un utilisateur.

Régler les autres problèmes relevés dans l'analyse de l'existant :

Outre la migration qui résoudra nombre de problèmes, il conviendra de changer la version de Jquery, la version de Bootstrap et de mettre à jour les différentes pages pour que le site soit toujours fidèle à son UI d'origine. A la suite de ces modifications, une analyse complète sera refaite afin de pouvoir comparer les deux versions de l'application.

Implémentation de nouvelles fonctionnalités, lesquelles ?

Autorisation

- Seuls les utilisateurs ayant le rôle administrateur (ROLE_ADMIN) doivent pouvoir accéder aux pages de gestion des utilisateurs.
- Les tâches ne peuvent être supprimées que par les utilisateurs ayant créé les tâches en question.
- Les tâches rattachées à l'utilisateur "anonyme" peuvent être supprimées uniquement par les utilisateurs ayant le rôle administrateur (ROLE_ADMIN).

Implémentation de tests automatisés

- Tests unitaires et fonctionnels) nécessaires pour assurer que le fonctionnement de l'application est bien en adéquation avec les demandes => Ces tests doivent être implémentés avec PHPUnit
- Rapport de couverture de code au terme du projet supérieur à 70 %.

3 - Travail réalisé et nouvel audit

3.1 Organisation (issues)

L'ensemble des tâches suivantes ont été réalisées dans l'ordre, elles sont détaillées dans la suite de cette section :

- Migration
- UML (les diagrammes ne sont pas dans ce document mais dans la section /diagrammes du code sur le repository Github)
- Anomalies et nouvelles fonctionnalités :
 - Fixtures
 - Attacher une tâche à un utilisateur lors de la création de la tâche et faire en sorte qu'à la modification de la tâche, l'utilisateur ne change pas
 - Tâches déjà présentes liées à un auteur anonyme (fixtures)
 - Lors de la création et la modification d'un utilisateur, possibilité de choisir un rôle
 - Gestion des utilisateurs seulement pour les admins
 - Suppression des tâches seulement par les créateurs de leurs propre tâche (voters)
 - Suppression des tâches anonymes seulement par les admin (voters)
- Relancer et corriger les analyses (qualité et performance) :
 - Améliorer la performance (<https://symfony.com/doc/current/performance.html>)
 - Corriger l'UI et Bootstrap 5
 - Ajouter de la PhpDoc
 - Faire un code plus SOLID
- Tests, rapport de couverture > 70%
- Readme et contributing.md

3.2 Migration du projet

Nous sommes aujourd'hui en Juillet 2021. La version 3.1 de Symfony est donc obsolète au regard des évolutions que nous envisageons. Nous devons migrer le projet vers la version actuellement stable de Symfony : 5.3 (<https://symfony.com/releases/5.3>).

Cette migration se déroule en plusieurs étapes :

- Symfony 3.1 => Symfony 3.4
- Symfony 3.4 => Symfony 4.4 (changement de la structure pour Flex)
- Symfony 4.4 => Symfony 5.3

Flex (<https://symfony.com/doc/current/setup/flex.html>) est le nouveau moyen de gérer ses packages dans un projet. Il automatise l'installation des dépendances et leur configuration. Plus besoin d'aller manuellement rechercher la configuration par défaut et d'enregistrer soi-même le bundle. Pour le mettre en place, il faut cependant changer la structure d'une application si celle-ci a été créée avant la version 4 de Symfony. Pour plus d'informations sur Flex :

<https://afsy.fr/avent/2017/08-symfony-flex-la-nouvelle-facon-de-developper-avec-symfony>

La version 5.3 n'est cependant pas une TLS. Il faudra donc migrer vers la version stable 5.4 lorsque celle-ci sortira.

Réalisation et étapes de la migration : [voir annexe](#).

Comme nous sommes sur un petit projet, il aurait été également possible de tout supprimer et de créer un symfony 5.3 tout neuf et de copier nos fichiers au fur et à mesure. Je rappelle qu'il faudra passer à la version 5.4 dès que celle-ci sera disponible car c'est la prochaine version stable.

3.3 Correction des anomalies

Fixtures (src/DataFixtures) :

Avant même de commencer à corriger les anomalies, nous devons mettre en place des données pour l'installation et l'utilisation du projet. Nous allons donc modifier ce dont nous avons besoin dans la base de donnée et installer des fixtures (<https://symfony.com/doc/current/bundles/DoctrineFixturesBundle/index.html>) : composer require --dev orm-fixtures

Nous créons ainsi des données pour les utilisateurs et pour les tâches afin d'avoir de la matière sur laquelle travailler.

Nous avons vu précédemment qu'il manque l'attribut "roles" dans la classe User. Nous devons l'ajouter. Pour nous simplifier la tâche, nous installons le maker de Symfony (<https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html>).

```

PS C:\wamp64\www\projet8-TodoList> php bin/console make:entity User

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> roles

Field type (enter ? to see all types) [string]:
> json

Can this field be null in the database (nullable) (yes/no) [no]:
>

Not generating User::getRoles(): method already exists
Not generating User::setRoles(): method already exists
updated: src/Entity/User.php

```

Ajout de l'attribut roles

Ce maker nous permet de créer l'attribut simplement grâce au wizard de Symfony. Utiliser le type JSON permet que lorsque nous enregistrons notre rôle en base de données, Doctrine encode automatiquement notre tableau en json_encode et le stockera dans un seul champ. De même lors de la récupération.

Dans la classe Task, il manque l'attribut "author" lié à la classe User qui permettrait de faire le lien entre la tâche et l'utilisateur créateur. Nous utilisons le maker pour créer cet attribut.

```

PS C:\wamp64\www\projet8-TodoList> php bin/console make:entity Task

Your entity already exists! So let's add some new fields!

New property name (press <return> to stop adding fields):
> author

Field type (enter ? to see all types) [string]:
> relation

What class should this entity be related to?:
> User

What type of relationship is this?
-----
Type      Description
-----
ManyToOne  Each Task relates to (has) one User.
           Each User can relate to (can have) many Task objects

ManyToOne  Each Task can relate to (can have) many User objects.
           Each User relates to (has) one Task

ManyToMany  Each Task can relate to (can have) many User objects.
           Each User can also relate to (can also have) many Task objects

OneToOne   Each Task relates to (has) exactly one User.
           Each User also relates to (has) exactly one Task.

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToOne

Is the Task.author property allowed to be null (nullable)? (yes/no) [yes]:
>

Do you want to add a new property to User so that you can access/update Task objects from it - e.g. $user->getTasks()? (yes/no) [yes]:
>

A new property will also be added to the User class so that you can access the related Task objects from it.

New field name inside User [tasks]:
>

updated: src/Entity/Task.php
updated: src/Entity/User.php

```

Ajout de l'attribut author (relation Task-User)

Nous changeons également le setCreatedAt() afin de faciliter son utilisation. Pour que celui-ci enregistre directement un new \Datetime lors de son appel :

```
public function setCreatedAt(): self  
  
{  
  
    $this->createdAt = new DateTime();  
  
    return $this;  
  
}
```

Une fois l'ensemble des attributs ajoutés, nous pouvons effectuer une migration (“composer require migrations” puis “php bin/console make:migration”) de notre base de données afin de faire évoluer son schéma. Au terme de “php bin/console doctrine:migrations:migrate”, nous obtenons :

```
[notice] Migrating up to DoctrineMigrations\Version20210712130907  
[notice] finished in 542.6ms, used 18M memory, 1 migrations executed, 4 sql queries
```

Migration de la base de données effectuée

Il est alors temps de créer les données en utilisant les fixtures (php bin/console doctrine:fixtures:load) :

```
> purging database  
> loading App\DataFixtures\TaskFixtures  
> loading App\DataFixtures\UserFixtures
```

Fixtures chargées

Ce qui a pour résultat dans la base de donnée :

	id	created_at	title	content	is_done	author_id
<input type="checkbox"/>	2	2021-07-12 14:17:03	Tâche 1	Faire du sport	1	NULL
<input type="checkbox"/>	3	2021-07-12 14:17:03	Tâche 2	Ranger la maison	0	NULL
<input type="checkbox"/>	4	2021-07-12 14:17:03	Tâche 3	Faire la vaisselle	1	NULL
<input type="checkbox"/>	5	2021-07-12 14:17:03	Tâche 4	Aller courir	0	NULL
<input type="checkbox"/>	6	2021-07-12 14:17:03	Tâche 5	Faire une session CodinGame	1	NULL

Table tâches

	<input type="button" value="← T →"/>		<input type="button" value="id"/>	<input type="button" value="username"/>	<input type="button" value="password"/>	<input type="button" value="email"/>	<input type="button" value="roles"/>
<input type="checkbox"/>	<input type="button" value="Éditer"/>	<input type="button" value="Copier"/>	<input type="button" value="Supprimer"/>	2	admin	\$2y\$12\$ymoHO.wxqOTFRVVoCaskf.wbQW2gKUE160Us9caqLhq...	admin@hotmail.fr ["ROLE_ADMIN"]
<input type="checkbox"/>	<input type="button" value="Éditer"/>	<input type="button" value="Copier"/>	<input type="button" value="Supprimer"/>	3	demo	\$2y\$12\$F8ozh/UMsyJDLN1Ck9UUGu1uUZCtuerQQiyBeaRKmbF...	demo@hotmail.fr ["ROLE_USER"]

Table utilisateurs

Nous avons attaché les tâches créées à un author_id null car il est demandé que les tâches déjà créées soient rattachées à un utilisateur “anonyme”.

Une tâche doit être attachée à un utilisateur :

Pour qu'une tâche soit rattachée à un utilisateur, il suffit de remplir l'attribut author de la tâche. Pour cela, nous allons simplement ajouter l'utilisateur courant dans un \$task->setAuthor(\$security->getUser()) lors de la création d'une tâche (le composant security de Symfony nous permettant d'avoir l'utilisateur connecté) :

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

    $form->handleRequest($request);

    if ($form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $task->setAuthor($security->getUser());
        $em->persist($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()]);
}
```

Route task_create

De cette façon, l'auteur est lié à la tâche lors de la création et il n'y a aucun moyen que, lors de la modification de la tâche, l'auteur puisse être modifié.

Choisir un rôle pour un utilisateur :

Afin de pouvoir choisir le rôle d'un utilisateur lors de sa création ou de sa modification, il est préférable de modifier le FormType associé. Effectivement, la création et la modification d'un utilisateur passent tous deux par le UserType.

Il suffit d'ajouter un ChoiceType qui permettra de choisir une ou plusieurs options d'un tableau :

```
->add('roles', ChoiceType::class, [
    'choices' => User::ROLES,
    'multiple' => true,
    'required' => false
])
```

ChoiceType roles

User::ROLES est une constante définie dans l'entité et donc facilement réutilisable ::

```
const ROLES = [
    'ROLE_ADMIN' => 'ROLE_ADMIN',
    'ROLE_USER' => 'ROLE_USER'
];
```

Constante User::ROLES

3.4 Nouvelles fonctionnalités

Seul les admins peuvent gérer les utilisateurs :

Afin que seuls les utilisateurs ayant le rôle administrateur (ROLE_ADMIN) soient en pouvoir d'accéder aux pages de gestion des utilisateurs, il suffit de bloquer l'accès aux routes commençant par /users dans la configuration.

```
access_control:
    - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/users, roles: ROLE_ADMIN }
    - { path: ^/, roles: ROLE_USER }
```

Access control

Nous créons également une hiérarchie des rôles afin que les admins puissent avoir au moins les mêmes droits que les utilisateurs basiques :

```
role_hierarchy:  
|   ROLE_ADMIN:      ROLE_USER
```

Role hierarchy

Seul l'auteur d'une tâche peut supprimer sa tâche et les tâches “anonymes” peuvent être seulement supprimées par les admins :

Ces deux conditions concernent la suppression d'une tâche et la permission d'accéder à cette suppression. Nous allons donc créer un voter (<https://symfony.com/doc/current/security/voters.html>) qui aura toute la logique et qui établira si oui ou non nous pouvons supprimer la tâche.

Un voter est notamment appelé lorsque nous implémentons la méthode `denyAccessUnlessGranted` :

```
/**  
 * @Route("/tasks/{id}/delete", name="task_delete")  
 */  
public function deleteTaskAction(Task $task): Response  
{  
    $this->denyAccessUnlessGranted('task_delete', $task);  
  
    $em = $this->getDoctrine()->getManager();  
    $em->remove($task);  
    $em->flush();  
  
    $this->addFlash('success', 'La tâche a bien été supprimée.');//  
    return $this->redirectToRoute('task_list');
```

Route task_delete avec appel du voter

Nous pouvons alors coder un voter qui teste si l'auteur est anonyme ou correspond à l'utilisateur connecté et qui vérifie si l'utilisateur est admin dans le cas où l'auteur est à null :

```

class TaskVoter extends Voter
{
    const TASK_DELETE = "task_delete";

    protected function supports(string $attribute, $task): bool
    {
        return in_array($attribute, [self::TASK_DELETE])
            && $task instanceof Task;
    }

    protected function voteOnAttribute(string $attribute, $task, TokenInterface $token): bool
    {
        $user = $token->getUser();

        // If the user is anonymous, do not grant access
        if (!$user instanceof UserInterface) {
            return false;
        }

        // No author => permission if admin
        if ($task->getAuthor() === null) {
            if (in_array("ROLE_ADMIN", $user->getRoles())) {
                return true;
            } else {
                return false;
            }
        }

        // ... (check conditions and return true to grant permission) ...
        switch ($attribute) {
            case self::TASK_DELETE:
                return $this->canDelete($task, $user);
                break;
        }

        return false;
    }

    private function canDelete(Task $task, User $user)
    {
        return $user === $task->getAuthor();
    }
}

```

Voter

Ainsi, toutes les nouvelles fonctionnalités sont mises en place.

3.5 Audit de performance

a - Analyse Blackfire

Nous avons déjà présenté Blackfire dans l'analyse de l'existant. Nous allons cette fois nous pencher sur l'analyse après migration et suivre les instructions de performance de Symfony (<https://symfony.com/doc/current/performance.html>) pour voir l'influence des différentes actions à effectuer. Il est important de désactiver la configuration xdebug de php.ini utilisée lors de nos tests avec phpunit car cela rentre en conflit avec l'outil Blackfire.

Nous allons ainsi comparer les tests avant et après optimisation pour chaque page, afin d'estimer le gain.

Nos manipulations d'optimisation :

- Dans l'application Symfony :
 - `container.dumper.inline_factories` : true dans le fichier `service.yaml` : Depuis PHP 7.4, cela permet que le container ne soit plus chargé en plusieurs fichiers mais en un seul, ce qui améliore les performances
- Dans le `php.ini` de notre serveur :
 - `opcache.preload=C:/wamp64/www/projet8-TodoList/config/preload.php` et `opcache.preload_user=www-data` : à partir de PHP 7.4, OPcache peut compiler et charger des classes au démarrage et les rendre disponibles pour toutes les requêtes jusqu'au redémarrage du serveur, améliorant considérablement les performances
 - `opcache.memory_consumption=256` et `opcache.max_accelerated_files=20000` : la configuration basique d'OPCache n'étant pas optimale pour Symfony, ces valeurs sont conseillées par le framework
 - `opcache.validate_timestamps=0` : dans les serveurs de production, les fichiers PHP ne doivent jamais changer, à moins qu'une nouvelle version de l'application ne soit déployée. Cependant, par défaut, OPcache vérifie si les fichiers mis en cache ont changé leur contenu depuis leur mise en cache. Cette vérification introduit une surcharge qui peut être évitée en mettant le timestamp à 0.
 - `realpath_cache_size=4096K` et `realpath_cache_ttl=600` : 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, comme les projets Symfony, doivent utiliser au moins ces valeurs d'après le framework

- Dans composer :
 - `composer dump-autoload --no-dev --classmap-authoritative` : dans les serveurs de production, les fichiers PHP ne doivent jamais changer, à moins qu'une nouvelle version de l'application ne soit déployée. C'est pourquoi on peut optimiser le chargeur automatique de Composer pour analyser l'ensemble de l'application une fois et créer une "carte de classe" optimisée. L'option `--no-dev` exclut les classes qui ne sont nécessaires que dans l'environnement de développement (c'est-à-dire les dépendances `require-dev` et les règles `autoload-dev`). L'option `--classmap-authoritative` crée une mappe de classe pour les classes compatibles PSR-0 et PSR-4 et empêche Composer d'analyser le système de fichiers pour les classes qui ne sont pas trouvées dans la carte de classe. Cette commande doit être utilisée à chaque fois que des classes sont ajoutées ou que le `composer.json` est modifié. La carte de classe est enregistrée dans le fichier `vendor/composer/autoload_classmap.php`.

Après l'ensemble de ces manipulations, nous pouvons retester l'ensemble des routes de notre application :

Route	Avant optimisation		Après optimisation		Gain	
	Temps d'exécution (ms)	Mémoire utilisée (MB)	Temps d'exécution (ms)	Mémoire utilisée (MB)	Temps d'exécution (ms)	Mémoire utilisée (MB)
Login (/login)	291	11	210	11,5	27,84%	-4,55%
Accueil (/)	335	13,5	185	13,7	44,78%	-1,48%
Liste des tâches (/tasks)	350	13,6	186	15,2	46,86%	-11,76%
Création d'une tâche (/tasks/create)	427	16,3	193	17,8	54,80%	-9,20%

Modification d'une tâche (/tasks/{id}/edit)	419	16,4	200	17,9	52,27%	-9,15%
Liste des utilisateurs (/users)	474	13,5	176	15	62,87%	-11,11%
Création d'un utilisateur (/users/create)	333	14,5	209	18,1	37,24%	-24,83%
Modification d'un utilisateur (/users/{id}/edit)	409	16,3	215	18,1	47,43%	-11,04%
Moyenne					46,76%	-10,39%

Ainsi, nous pouvons noter un gros gain de performance concernant le temps d'exécution. Celui-ci est en effet en moyenne 2 fois plus rapide qu'avant l'optimisation. En revanche, ceci est lié à une légère augmentation de la mémoire utilisée (10%). La fonction la plus gourmande est la fonction includeFile de composer qui charge un grand nombre de fichiers, cependant le gain en temps d'exécution en vaut la peine.

3.6 Audit de qualité

a - Analyse PHPStan

Nous refaisons une analyse PHPStan avec le niveau 6 d'analyse afin de ne pas laisser d'éventuelles erreurs dans notre code (<https://phpstan.org/user-guide/rule-levels>) : vendor/bin/phpstan analyse -l 6 src.

La plupart des erreurs concernent la documentation de nos fonctions et de nos variables. Nous devons alors ajouter de la PHPDoc sur les composants qui le demandent. Suite à cela, le code respecte le niveau 6 de l'analyse :

```
PS C:\wamp64\www\projet8-TodoList> vendor/bin/phpstan analyse -l 6 src
12/12 [=====] 100%

[OK] No errors
```

Résultat analyse PHP Stan

Il ne faut pas oublier de relancer les tests pour voir si rien n'a été bouleversé.

b - Analyse PHP CS (Code Sniffer)

Après avoir lancé les commandes PHP CS suivantes :

- ./vendor/bin/phpcs --standard=PSR12 src
- ./vendor/bin/phpcs src

Et les avoir corrigées avec les commandes :

- ./vendor/bin/phpcbf --standard=PSR12 src
- ./vendor/bin/phpcbf src

Nous obtenons un code qui respecte les standards que nous nous sommes fixés dans l'analyse de l'existant.

c - Analyse PHP MD (Mess Detector)

Les commandes “./vendor/bin/phpmd src text codesize” et “./vendor/bin/phpmd src text cleancode” retournent des rapports nuls. Nous pouvons donc établir que le code de base ne présente pas de code smells importants.

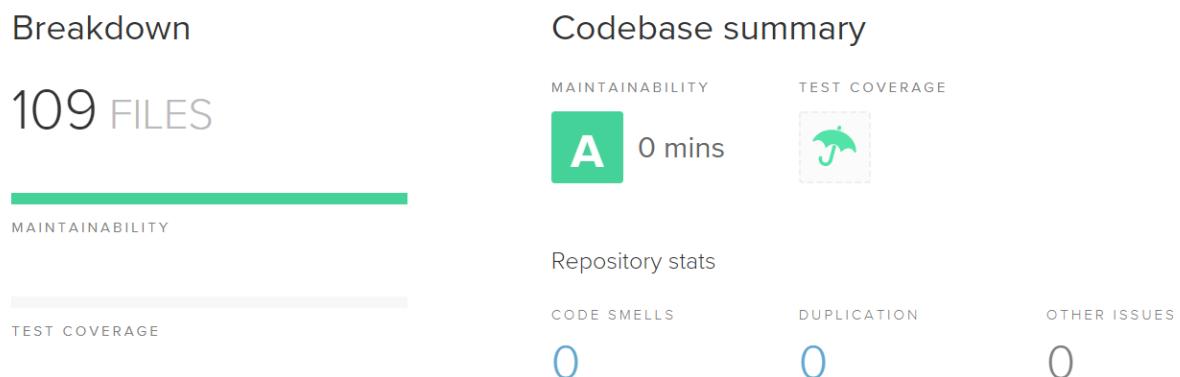
d - Analyse PHP CPD (Copy Paste Detector)

Après une analyse du fichier src avec la commande “./vendor/bin/phpcpd src”, aucun clone n'est trouvé. Notre code est donc propre à ce niveau là.

e - Analyse Code Climate

Nous avons vu dans l'analyse de l'existant que les fichiers posant problème (code smell et duplication) étaient des fichiers qui allaient disparaître lors de la migration (bootstrap.js et SymfonyRequirements). L'équipe n'avait, jusque-là, ajouté aucun code problématique. Nous devons maintenant vérifier que la migration a bien eu pour effet de minimiser la dette technique et que nous n'avons pas rajouter nous même, avec nos modifications de problèmes éventuels.

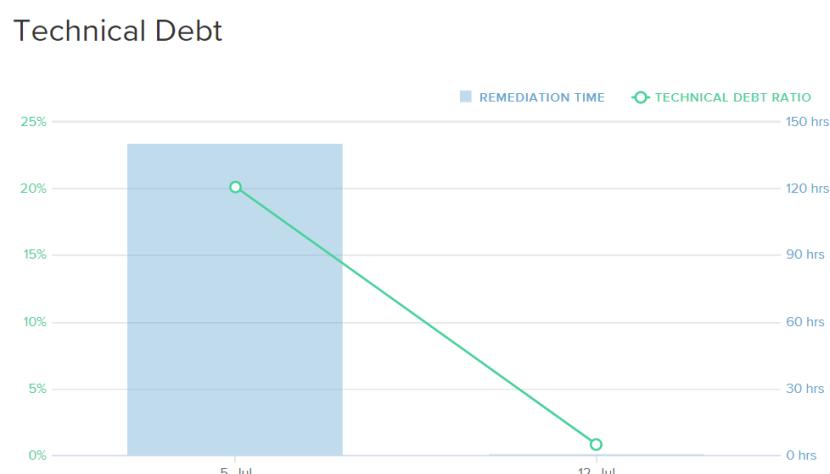
Analysons rapidement le rapport après migration et optimisation :



Rapport Code Climate après optimisation

Ainsi, nous observons qu'il n'y a plus de problèmes de code smell ou de duplication. Le code contient beaucoup plus de fichiers mais ces derniers ne remontent aucune anomalie. Le code est donc propre à ce niveau là.

Code Climate nous permet également de vérifier l'évolution de la dette technique de notre projet :



Evolution de la dette technique

Nous pouvons voir que nous avons drastiquement réduit la dette technique du projet. Il faut toutefois faire attention à refaire ce genre de vérification continuellement lors des évolutions futures. Garder un niveau d'exigence élevé et de manière régulière permet de garder un code maintenable et évolutif sur le long terme.

f - Commandes déjà incluses dans Symfony et Composer

Nous allons ici utiliser les commandes présentées dans la même partie de l'analyse de l'existant afin de vérifier que rien n'a été oublié et que tout reste fonctionnel. Ces commandes sont des outils intégrés ou dérivés de Symfony et Composer qui permettent de vérifier les points suivants :

- packages inutilisés
- schéma de composer valide
- syntaxe des fichiers twig
- schéma de la base de donnée cohérent

Nous avons déjà présenté **composer unused** dans l'analyse de l'existant (<https://github.com/composer-unused/composer-unused>). L'outil permet de connaître les packages non utilisés dans composer :

```
! [NOTE] Found 17 package(s) to be checked.
Scanning files from basedir C:\wamp64\www\projet8-TodoList
-----
1545/1545 [=====] 100%
Results
-----
Found 16 used, 1 unused and 1 ignored packages

Used packages
✓ php
✓ doctrine/orm
✓ doctrine/doctrine-bundle
✓ symfony/dotenv
✓ symfony/webpack-encore-bundle
✓ symfony/security-bundle
✓ symfony/twig-bundle
✓ twig/extras-bundle
✓ twig/twig
✓ symfony/monolog-bundle
✓ symfony/yaml (suggested by: doctrine/orm)
✓ symfony/validator
✓ doctrine/annotations
✓ doctrine/doctrine-migrations-bundle
✓ symfony/form
✓ sensio/framework-extra-bundle

Unused packages
✗ symfony/polyfill-apcu

Ignored packages
○ symfony/flex (Invalid package type)
```

Analyse composer unused

L'ensemble des paquets sont utilisés sauf polyfill-apcu car nous utilisons OPCache pour améliorer les performances.

composer valid : Dans notre cas, le schéma du composer est bon, on pourrait éventuellement ajouter une description au projet.

php bin/console lint:twig templates (symfony twig:lint pour les versions plus récentes) : La syntaxe de nos 10 fichiers twig est valide.

php bin/console doctrine:schema:valid --skip-sync : Le schéma de notre base de donnée est toujours correct :

g - Analyse LightHouse

Précédemment, nous avons vu que la performance de l'ensemble des pages de l'application était de bonne qualité. Nous avons cependant légèrement changé l'UI (voir partie analyse de l'UI ci-dessous) et devons vérifier que le rapport de Lighthouse est toujours convenable.

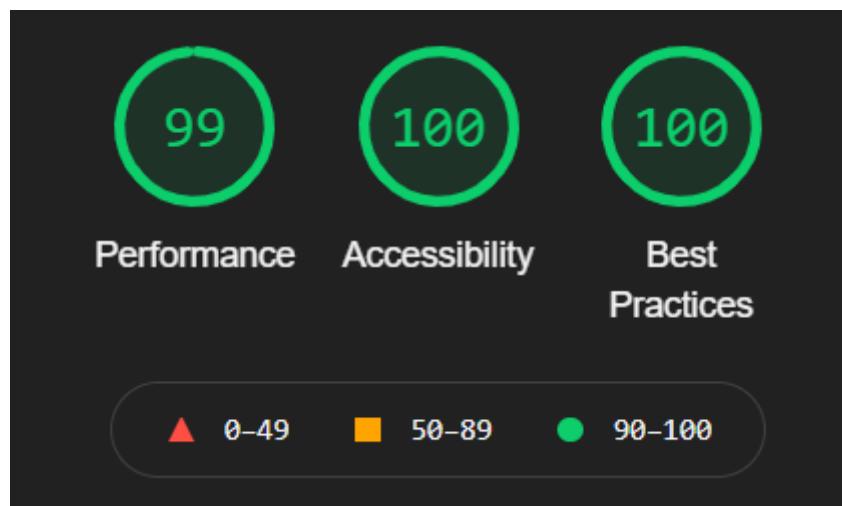
Les légers problèmes relevé à l'analyse de l'existant étaient dus à :

- Une mauvaise gestion du chargement des fichiers CSS qui bloquent l'affichage de la page : nous avons réglé ce problème grâce à l'utilisation de Webpack Encore et d'un build en production (yarn build)
- Des images qui n'ont pas toujours une hauteur et une largeur définie : ce critère n'influe pas directement sur la performance
- L'image OpenClassRooms est trop petite
- Un cache : configuration d'OPCache
- Quelques erreurs JavaScript
- Un problème de chargement des glyphicon pour la page "liste des tâches" : nous avons remplacé cela par un balise link FontAwesome

L'ensemble des pages relevaient les mêmes types de rapport. Avec les paramètres suivants :

Categories	Device
<input checked="" type="checkbox"/> Performance	<input type="radio"/> Mobile
<input type="checkbox"/> Progressive Web App	<input checked="" type="radio"/> Desktop
<input checked="" type="checkbox"/> Best practices	
<input checked="" type="checkbox"/> Accessibility	
<input type="checkbox"/> SEO	

Paramètres LightHouse



Résultats de l'analyse après migration et optimisation

Comme pour les autres analyses, il est important de relancer LightHouse quand des modifications de l'UI ont été apportées.

h - Analyse du code et principes SOLID

Nous avons vu dans l'analyse de l'existant que le code écrit est minime et qu'il ne détient pas beaucoup d'erreurs. Nous avons notamment vu que les principaux axes d'amélioration concernaient le S et le D de SOLID.

Concernant le S :

L'ensemble de la logique des formulaires était contenu dans les contrôleurs. Or, le but du premier principe est de simplifier au maximum les fonctions et les classes pour que ces dernières n'aient qu'un seul but et qu'une seule raison de changer.

Nous avons donc créé des FormHandler afin de contenir notre logique. La première classe est la classe abstraite FormHandler mère qui définit les méthodes communes et les méthodes à implémenter pour les classes filles.

Afin de respecter le principe S, nous créons une classe pour chaque action. Nous avons donc 4 classes filles de FormHandler : TaskCreateFormHandler, TaskUpdateFormHandler, UserCreateFormHandler, UserUpdateFormHandler.

Nous ne créons pas un constructeur pour le FormHandler mais nous utilisons les méthodes calls pour avoir plus de flexibilité sur leur instanciation (https://symfony.com/doc/current/service_container/calls.html). Ceci permet de récupérer, quand on le souhaite, une instance de formFactory (pour créer le formulaire) et une instance requestStack (pour récupérer la requête courante).

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

/**
 * @return void
 */
public function setRequestStack(RequestStack $requestStack){
    $this->requestStack = $requestStack;
}
```

Setters de FormHandler

Suite à cela, nous devons créer le formulaire et l'affecter à notre classe. Pour cela on crée une fonction createForm qui récupère le type et l'entité correspondante à partir du contrôleur :

```
/**
 * @param object $entity
 * @return FormInterface<FormInterface>
 */
public function createForm(object $entity)
{
    $this->entity = $entity;
    return $this->form = $this->formFactory->create($this->getFormType(), $entity);
}
```

Méthode createForm de FormHandler

Vient ensuite une fonction générale (`handle`) qui permet de capter la soumission du formulaire. Cette fonction se charge de valider le formulaire et de lancer ensuite la méthode `process` qui détient la logique de MAJ de la base de données. Que le formulaire soit valide ou non, `handle` indique au contrôleur l'état de validation par un booléen :

```
/**  
 * @return boolean  
 */  
public function handle()  
{  
    $this->form->handleRequest($this->requestStack->getCurrentRequest());  
    if ($this->form->isSubmitted() && $this->form->isValid()) {  
        $this->process();  
        return true;  
    }  
    return false;  
}
```

Méthode handle du FormHandler

Les fonctions `process` et `getFormType` dépendent de chaque formulaire, nous implémentons donc des classes abstraites obligatoires :

```
/**  
 * @return void  
 */  
abstract protected function process();  
  
/**  
 * @return string  
 */  
abstract protected function getFormType();
```

Classes abstraites process et getFormType de FormHandler

Les classes filles doivent donc implémenter ces méthodes au minimum. Nous ajoutons également des setters spécifiques à l'utilisation de chaque classe fille . Pour exemple, la classe `TaskCreateFormHandler` a besoin du composant `security` pour ajouter un auteur, de l'`entity manager` et des setters du parent pour fonctionner :

```

    /**
     * @return void
     */
    public function setEntityManager(EntityManagerInterface $entityManager){
        $this->em = $entityManager;
    }

    /**
     * @return void
     */
    public function setSecurity(Security $security){
        $this->security = $security;
    }

    /**
     * @return void
     */
    public function process()
    {
        $this->entity->setAuthor($this->security->getUser());
        $this->entity->setCreatedAt();
        $this->em->persist($this->entity);
        $this->em->flush();
    }

    /**
     * @return string
     */
    public function getFormType()
    {
        return TaskType::class;
    }

```

Classe TaskCreateFormHandler

```

App\FormHandler\TaskCreateFormHandler:
    calls:
        - setEntityManager: ['@doctrine.orm.default_entity_manager']
        - setSecurity: ['@security.helper']
        - setFormFactory: ['@form.factory']
        - setRequestStack: ['@request_stack']

```

Configuration calls de TaskCreateFormHandler pour initialisation dans services.yaml

Les autres classes filles créées suivent le même schémas avoir leurs propres besoins et donc leurs propres setters. Ainsi, toute la logique des formulaires se retrouve dans des classes spécifiques et les contrôleurs s'en retrouve grandement allégés :

```

/**
 * @Route("/tasks/create", name="task_create")
 */
public function createAction(TaskCreateFormHandler $taskCreateFormHandler): Response
{
    $form = $taskCreateFormHandler->createForm(new Task());
    if ($taskCreateFormHandler->handle()) {
        $this->addFlash('success', 'La tâche a été bien ajoutée.');
        return $this->redirectToRoute('task_list');
    }
    return $this->render('task/create.html.twig', ['form' => $form->createView()]);
}

```

Route task_create du TaskController

Nous respectons ainsi au mieux le SRP lors de l'externalisation de la logique du FormHandler,

Concernant le D :

Avant nous ne respections pas le D car nous faisions directement appel au container avec la méthode get, nous n'utilisions pas l'injection de dépendance et donc par extension l'inversion de dépendance. Avec Symfony 5.3 et l'injection de l'interface UserPasswordHasherInterface nous avons réglé ce problème de DIP.

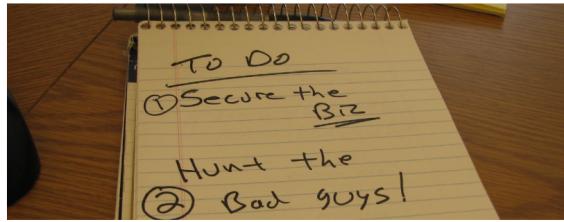
Tout cela nous permet d'avoir une application respectant ces principes. Il faut toujours avoir en tête ces derniers afin de garder un facile à comprendre et à faire évoluer.

i - Analyse de l'UI + versions de Jquery et Bootstrap

Page d'accueil (/) :

Nous avons mis la gestion d'utilisateur dans la navbar et la création des tâches en dessous de l'image. Les boutons de connexion et déconnexion se retrouvent maintenant dans la navbar. Le bouton en haut à gauche et l'image "TodoList App" renvoient vers la page d'accueil.

Bienvenue sur Todo List, l'application vous permettant de gérer l'ensemble de vos tâches sans effort !



Tâches

[Créer une nouvelle tâche](#)

[Consulter la liste des tâches à faire](#)

[Consulter la liste des tâches terminées](#)

Copyright © OpenClassrooms

Page d'accueil

Liste des tâches (/tasks et /tasks/ending) : Les boutons ont été espacés et un page "tâches terminées" a été ajoutée et reliée à la route /tasks/ending. Cette dernière reprend le template de /tasks ci dessous :

Liste des tâches

Tâches à faire

[Créer une tâche](#)

x

Tâche 2

Ranger la maison

[Marquer comme faite](#)

[Supprimer](#)

x

Tâche 3

Faire la vaisselle

[Marquer comme faite](#)

[Supprimer](#)

x

Tâche 4

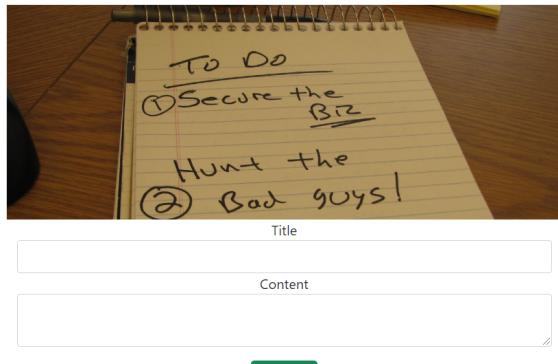
Aller courir

[Marquer comme faite](#)

[Supprimer](#)

Page liste des tâches

Création d'une tâche (/tasks/create) et modification d'une tâche (/tasks/{id}/edit) : L'UI est maintenant centrée et plus aérée :



Copyright © OpenClassrooms

Page d'édition d'une tâche

Gestion des utilisateurs : Les UI étaient déjà bonnes. Le problème était surtout qu'aucune page ne permettait d'y accéder. Ce problème est maintenant résolu grâce au lien dans la navbar qui ne s'affiche que si l'utilisateur est admin :

```
{% if is_granted('ROLE_ADMIN') %}  
<li class="nav-item me-3">  
|   <a class="nav-link" aria-current="page" href="{{ path('user_list') }}>Gestion utilisateur</a>  
</li>  
{% endif %}
```

Affichage d'un lien de gestion utilisateur pour les admins

Version de Bootstrap et Jquery : Les versions de Bootstrap et de Jquery ont été mises à jour à l'installation de Webpack Encore (<https://symfony.com/doc/current/frontend/encore/installation.html>). Nous sommes actuellement en version 5 de Bootstrap (<https://getbootstrap.com/docs/5.0/getting-started/introduction/>) qui utilise la version 3.6.0 de Jquery.

3.7 Test et rapport de couverture

a - Tests

Nous mettons en place un maximum de tests afin de pouvoir rendre notre application la plus fiable possible. Nous devons tester toutes les fonctionnalités de l'application, à savoir :

- Le CRUD des tâches

- Le CRUD des utilisateurs
- Les validateurs des entités
- Les autorisations

Pour cela nous installons la dernière version de PHPUnit : composer require --dev phpunit/phpunit

Nous avons également besoin d'une base de données de test afin de ne pas impacter notre base de données principale. Nous créons alors un fichier .env.test.local avec les paramètres de cette base de données de test.

Pour ne pas que les modifications faites dans les tests soient prises en compte dans la base de donnée, nous installons le doctrine-test-bundle (<https://packagist.org/packages/dama/doctrine-test-bundle>) qui permet de faire des rollback de la base de donnée à chaque action.

Nous créons ensuite la base de donnée dans MySql : “php bin/console doctrine:database:create --env=test” + “php bin/console doctrine:schema:update --env=test --force” et nous chargeons les fixtures de tests équivalentes à nos fixtures de base : php bin/console doctrine:fixtures:load --env=test. Mais nous ne voulons pas que le dossier des DataFixtures se retrouve dans la couverture de test. Nous modifions le fichier xml de phpunit en ce sens.

```
<coverage processUncoveredFiles="true">
    <include>
        <directory suffix=".php">src</directory>
    </include>
    <exclude>
        <directory suffix=".php">src/DataFixtures</directory>
    </exclude>
</coverage>
```

Modification phpunit.xml

A partir de là, nous pouvons créer l'ensemble de nos tests et les lancer avec la commande : ./vendor/bin/phpunit

Une des petites subtilités que nous allons mettre en place concerne la connexion. En effet, afin qu'il soit simple de simuler une connexion dans l'ensemble de nos tests, nous allons créer un trait réutilisable par tous nos fichiers.

```

namespace App\Tests;

use App\Entity\User;
use Symfony\Component\BrowserKit\Cookie;
use Symfony\Bundle\FrameworkBundle\KernelBrowser;
use Symfony\Component\Security\Core\Authentication\Token\UsernamePasswordToken;

trait NeedLoginTrait
{
    public function login(KernelBrowser $client, User $user)
    {
        $session = $client->getContainer()->get('session');
        $token = new UsernamePasswordToken($user, null, 'main', $user->getRoles());
        $session->set('_security_main', serialize($token));
        $session->save();

        $cookie = new Cookie($session->getName(), $session->getId());
        $client->getCookieJar()->set($cookie);
    }
}

```

NeedLoginTrait

De cette façon, tous nos fichiers pourront connecter facilement un utilisateur en utilisant le trait :

```

class HomepageControllerTest extends WebTestCase
{
    use NeedLoginTrait;
}

```

Utilisation de NeedLoginTrait

Puis :

```

$user = $this->em->getRepository(User::class)->findOneBy(["username" => "admin"]);
$this->login($this->client, $user);

```

Utilisation de la méthode login de NeedLoginTrait

Autrement, les tests fonctionnels étendent le WebTestCase afin d'avoir accès facilement au client et au crawler. Pour les validateurs nous utilisons uniquement le KernelTestCase et son service validator.

Pour écrire l'ensemble de nos tests, nous nous basons sur ce qui est déjà écrit dans les entités et dans les controllers. Pour plus de sécurité, nous rajoutons une regex sur le password ainsi qu'un test pour savoir si celui-ci n'est pas null.

L'écriture des tests révèle quelques anomalies :

- tester si le formulaire est soumis avant de tester si il est valide
- le password hasher doit être implémenté dans les routes qui utilisent le mot de passe

b - Rapport de couverture

Pour lancer un rapport de couverture, xdebug (<https://xdebug.org/>) doit être installé et activé.

Dans le php.ini, nous ajoutons les lignes suivante :

```
[xdebug]  
  
xdebug.mode=coverage
```

Ensuite, il suffit de lancer la commande suivante afin de générer une page HTML correspondant à notre rapport : [./vendor/bin/phpunit --coverage-html public/test-coverage](#)

Avec cette commande, le rapport est dans le dossier public/test-coverage (le dossier test-coverage n'a pas besoin d'être créé, PHPUnit s'en charge pour nous).

Le rapport de couverture est le suivant :

	Lines	Code Coverage			Classes and Traits
		Functions and Methods		Classes and Traits	
Total	98.73%	155 / 157	98.33%	59 / 60	92.86%
Controller	100.00%	51 / 51	100.00%	11 / 11	100.00%
Entity	100.00%	36 / 36	100.00%	25 / 25	100.00%
FormHandler	100.00%	45 / 45	100.00%	19 / 19	100.00%
Form	100.00%	12 / 12	100.00%	2 / 2	100.00%
Security	84.62%	11 / 13	66.67%	2 / 3	0.00%

Nous avons 98.73% de couverture globale. La seule fonction qui n'est pas testée totalement concerne les return false dans le voter que nous avons créé :

```

protected function voteOnAttribute(string $attribute, $task, TokenInterface $token): bool
{
    $user = $token->getUser();

    // If the user is anonymous, do not grant access
    if (!$user instanceof UserInterface) {
        return false;
    }

    // No author => permission if admin
    if ($task->getAuthor() === null) {
        if (in_array("ROLE_ADMIN", $user->getRoles())) {
            return true;
        } else {
            return false;
        }
    }

    // ... (check conditions and return true to grant permission) ...
    switch ($attribute) {
        case self::TASK_DELETE:
            return $this->canDelete($task, $user);
            break;
    }

    return false;
}

```

Rapport de couverture de la fonction voteOnAttribute

Etant donné que la route Delete user n'est accessible que par les administrateurs, il est pour le moment impossible de rentrer dans le premier false de manière normale (l'utilisateur ne pouvant être anonyme). Cependant, il est préférable de laisser ce type de condition dans le cas où le code évoluerait. De même pour le return false à la fin, il n'est jamais appelé car la fonction supports vérifie déjà que l'attribut sera l'un des attributs du switch. Cependant, il est préférable de laisser ce return false toujours pour des évolutions supposées sur le voter.

Ce rapport est disponible dans le dossier public/test-coverage/index.html. Attention toutefois, un rapport de couverture de 100% ne signifie pas que tout est bien testé. Des erreurs peuvent subsister. C'est une très bonne chose d'avoir un code qui passe par toutes les fonctions mais ça ne veut pas dire que tous les cas sont gérés. Il faut donc rester vigilant.

4 - Annexes

4.1 Etapes de la migration

Les étapes ci-dessous sont les étapes suivies, dans l'ordre, pour réaliser la migration :

Symfony 3.1 => 3.4

- "symfony/symfony": "3.4.*", dans composer puis composer update

Symfony 3.4 => 4 (https://symfony.com/doc/4.0/setup/upgrade_major.html)

Avant de passer à la version 4, nous devons enlever les différentes dépréciations (au nombre de 11) :

Info. & Errors	2	Deprecations	11	Debug	28	PHP Notices	0	Container	582
Log messages generated by using features marked as deprecated.									
Time Channel Message									
08:30:56 php User Deprecated: Symfony\Component\HttpKernel\Kernel::loadClassCache() is deprecated since Symfony 3.3, to be removed in 4.0. Show context Show trace									
08:30:56 php User Deprecated: Symfony\Component\HttpKernel\Kernel::doLoadClassCache() is deprecated since Symfony 3.3, to be removed in 4.0. Show context Show trace									
08:30:56 php User Deprecated: Creating Doctrine\ORM\Mapping\UnderscoreNamingStrategy without making it number aware is deprecated and will be removed in Doctrine ORM 3.0. Show context Show trace									
08:29:57 - Using the unquoted scalar value " !event " is deprecated since Symfony 3.3 and will be considered as a tagged value in 4.0. You must quote it in " C:\wamp64\www\projet8-TodoList\app\config\config_dev.yml " on line 20. Show context Show trace									
08:29:57 - Using the unquoted scalar value " !doctrine " is deprecated since Symfony 3.3 and will be considered as a tagged value in 4.0. You must quote it in " C:\wamp64\www\projet8-TodoList\app\config\config_dev.yml " on line 23. Show context Show trace									
08:29:57 - Using the unquoted scalar value " !doctrine " is deprecated since Symfony 3.3 and will be considered as a tagged value in 4.0. You must quote it in " C:\wamp64\www\projet8-TodoList\app\config\config_dev.yml " on line 23. Show context Show trace									
08:29:57 - The " framework.trusted_proxies " configuration key has been deprecated in Symfony 3.3. Use the Request::setTrustedProxies() method in									

Exemple de dépréciations

- User Deprecated: Symfony\Component\HttpKernel\Kernel::loadClassCache() is deprecated since Symfony 3.3, to be removed in 4.0. => on enlève \$kernel->loadClassCache(); dans app.php et app_dev.php (<https://symfony.com/blog/new-in-symfony-3-3-deprecated-the-classloader-component>)

- User Deprecated: Creating Doctrine\ORM\Mapping\UnderscoreNamingStrategy without making it number aware is deprecated and will be removed in Doctrine ORM 3.0. => naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware à la place de doctrine.orm.naming_strategy.underscore dans config.yml
- !event et !doctrine => ["!event", "!doctrine"] in config_dev
- The "framework.trusted_proxies" configuration key has been deprecated in Symfony 3.3. Use the Request::setTrustedProxies() method in your front controller instead. => <https://symfony.com/blog/fixing-the-trusted-proxies-configuration-for-symfony-3-3>
- Not setting "logout_on_user_change" to true on firewall "main" is deprecated as of 3.4, it will always be true in 4.0. => logout_on_user_change: true dans firewalls main (<https://codereviewvideos.com/course/upgrade-symfony-3-to-symfony-4/video/fixing-generic-symfony-deprecations>)
- Autowiring-types are deprecated since Symfony 3.3 and will be removed in 4.0. Use aliases instead for "Psr\Log\LoggerInterface". et Symfony\Component\HttpKernel\DependencyInjection\Extension::addClassesToCompile() is deprecated since version 3.3, to be removed in 4.0. => changer la version de monolog-bundle dans composer => de 2.8 à 3.1

A ce stade, le code en 3.4 n'a plus de dépréciations, nous pouvons passer à la version 4 de Symfony:

- "symfony/symfony": "^4.0", dans composer puis composer update "symfony/symfony" => A ce moment symfony/symfony rentre en conflit avec "sensio/distribution-bundle", "sensio/framework-extra-bundle" et "sensio/generator-bundle", nous supprimons alors ces bundles qui ne sont plus maintenus
- Nous retirons également swiftmailer qui n'est pas utilisé dans notre projet, si jamais nous avons besoin d'un mailer nous prendrons le composant Mailer de Symfony
- composer update "symfony/symfony"
- Les use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route; sont alors remplacés par use Symfony\Component\Routing\Annotation\Route;
- Symfony\Bundle\FrameworkBundle\Controller\Controller; est remplacé par Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

A ce moment là, le projet est fonctionnel avec la version 4.4.26 de symfony (qui requiert une version de PHP >=7.1.3), cependant, 36 dépréciations apparaissent. de plus nous ne sommes pas encore passé à l'architecture Flex

- Après un composer update, nous réalisons que les versions de doctrine ne sont pas à jour, on passe alors de "doctrine/orm": "^2.5" et "doctrine/doctrine-bundle": "^1.6" à "doctrine/orm": "^2.9.3" et "doctrine/doctrine-bundle": "^2.4" => nous n'avons plus que 31 dépréciations

Passage à l'architecture flex :

- composer require symfony/flex (<https://symfony.com/doc/4.0/setup/flex.html#upgrade-to-flex>)
- composer remove symfony/symfony
- Ajout de "conflict": {"symfony/symfony": "*"} dans composer
- On change ensuite toute la structure comme indiqué dans le lien suivant entre les étapes 4 et 12 : <https://symfony.com/doc/4.0/setup/flex.html#upgrade-to-flex>
 - Tous les fichiers du type app/config/config_dev.yml vont dans les équivalents dans config/packages/dev/*.yaml
 - Ce qui est dans config.yml se retrouve dans package/*.yml
 - Les fichiers de routing_dev.yml se retrouvent dans config/routes/dev/web_profiler.yaml mais config/routes/dev/framework.yaml doit être supprimé
 - Aucun service n'est configuré dans service.yaml, ce qui simplifie la tâche
 - app/Resources/views/ -> templates/
 - Les fichiers dans le dossier src/AppBundle/ vont dans le dossier src/ sauf AppBundle.php => ceci entraîne également la modification des namespaces où AppBundle est remplacé par App, nous remplaçons dans le même temps tous les AppBundle du code par App
 - La partie autoload de composer se simplifie (<https://github.com/symfony/skeleton/blob/8e33fe617629f283a12bbe0a6578bd6e6af417af/composer.json#L24-L33>)
 - web/img => public/img
 - web/fonts => public/fonts
 - Les .htaccess sont supprimés
 - Les .js et .css se retrouvent dans la configuration Webpack encore suite à son installation : composer require symfony/webpack-encore-bundle puis yarn install
 - Tout ce qui a rapport avec les assets dans les .twig doivent être remplacés par des entrées webpack exemple : <link href="{{ asset('css/bootstrap.min.css') }}" rel="stylesheet"> devient {% block stylesheets %} {{ encore_entry_link_tags('app') }}{% endblock %} =>idem pour les scripts

- composer require symfony/dotenv, on met notamment ce qui était avant dans le parameters.yml et dans le config.yml dans le .env et le .env.local
- Nous retirons la partie "scripts" et "extra" de composer qui correspondent à l'ancienne façon de fonctionner, cependant, nous devons ajouter des indications pour flex dans le extra(<https://symfony.com/doc/current/setup/flex.html>) :

```

"extra": {
    "symfony": {
        "allow-contrib": false,
        "require": "4.4.*"
    }
}

```
- On met à jour le bin/console avec une version plus récente et on supprime le bin/symfony_requirements
- Certains composants ne sont plus installés à ce stade, nous devons donc faire les composer require nécessaires, dans le même temps, nous supprimons le dossier vendor qui contient encore des fichiers de l'ancienne configuration et nous faisons un composer update pour tout mettre à jour :
 - composer require security
 - composer require twig
 - composer require --dev profiler
 - composer require monolog
 - composer require symfony/yaml
 - composer require symfony/validator
 - composer require symfony/form
 - composer require sensio/framework-extra-bundle (permettra d'utiliser le paramConverter pour la transformation en objet de ce qu'on met dans les paramètres d'une route)
- On enlève aussi "doctrine/doctrine-cache-bundle": "^1.2" qui est abandonné
- Pour que la route login fonctionne à nouveau il faut lui envoyer dans ses paramètre Symfony\Component\Security\Http\Authentication\AuthenticationUtils, il n'est plus possible et il est devenu déconseillé de faire \$this->get('security.authentication_utils')
- Nous retirons également "incenteev/composer-parameter-handler": "~2.0" et toute sa configuration qui ne sera plus utile à l'installation de Flex
- Nous déplaçons shop-homepage.css dans app.css

- Nous permettons l'implémentation de Scss à travers Webpack (<https://symfony.com/doc/current/frontend/encore/simple-example.html#using-sass-less-stylus>) en installant : yarn add sass-loader@^12.0.0 sass --dev et en le configurant
- Nous ajoutons le framework Bootstrap à Webpack : yarn add bootstrap --dev et nous l'importons dans app.css : @import "~bootstrap/scss/bootstrap";

A cet instant il reste uniquement 6 dépréciations pour symfony 4.4 :

- The "security.firewalls.main.logout_on_user_change" configuration key has been deprecated in Symfony 4.1 => logout_on_user_change: true doit être supprimé dans security.yml
- The "twig.exception_controller" configuration key has been deprecated in Symfony 4.4, set it to "null" and use "framework.error_controller" configuration key instead. => Ajouter exception_controller: null dans les différents twig.yaml
- Relying on the default value ("false") of the "twig.strict_variables" configuration option is deprecated since Symfony 4.1. You should use "%kernel.debug%" explicitly instead, which will be the new default in 5.0. => Ajouter strict_variables: "%kernel.debug%" dans les différents twig.yaml

Symfony 4 => 5.3

A ce stade, le code en 4.4 n'a plus de dépréciations, nous pouvons passer à la version 5 de Symfony. La différence de version est moins grande entre 4 et 5 qu'entre 3 et 4. Il reste donc peu de choses à migrer. A cet instant nous changeons de version de PHP et passons en 7.4.9 car Symfony 5.3 recommande minimum PHP 7.2.5.

- composer update avec require 5.3.* dans l'option extra symfony de composer, on met également tous les composants commençant par symfony/ à la version 5.3.*
- On passe phpcpd à la version 6.0.3
- composer update

A ce stade l'application 5.3 est fonctionnelle, il reste quelques dépréciations :

- Dans security.yml l'option encoders est remplacée par password_hashers (<https://symfony.com/blog/new-in-symfony-5-3-passwordhasher-component>)
- Dans security.yml, on rajoute enable_authenticator_manager: true et on supprime l'option anonymous
- Using type "Symfony\Component\Routing\RouteCollectionBuilder" for argument 1 of method "App\Kernel:configureRoutes()" is deprecated, use

"Symfony\Component\Routing\Loader\Configurator\RoutingConfigurator" instead => on utilise un fichier Kernel.php plus récent pour enlever ces dépréciations

- Class "App\Entity\User" should implement method "Symfony\Component\Security\Core\User\UserInterface::getUserIdentifier()": returns the identifier for this user (e.g. its username or e-mailaddress) => rajouter cette méthode dans notre classe User
- Since symfony/security-bundle 5.3: Configuring an encoder for a user class that does not implement "Symfony\Component\Security\Core\User>PasswordAuthenticatedUserInterface" is deprecated, class "App\Entity\User" should implement it. => on implémente l'interface PasswordAuthenticatedUserInterface dans la classe User

Lors de l'ensemble de ces modifications il faut régulièrement faire un php bin/console clear:cache et parfois supprimer le cache du dossier var pour ne pas avoir des erreurs non significatives.