

ToDo & Co .

Audit technique de l'application.



## Table des matières

<b>ToDo &amp; Co .....</b>	<b>1</b>
<b>Audit technique de l'application. ....</b>	<b>1</b>
<b>I- Introduction. ....</b>	<b>3</b>
<b>II- Audit de la qualité du code. ....</b>	<b>3</b>
1- Introduction à Codacy. ....	3
2- Analyse du projet par Codacy.....	3
<b>III- Audit des performances de l'application. ....</b>	<b>4</b>
1- Travaux d'optimisation réalisés.....	4
2- Analyse de l'application avant et après optimisation. ....	5

## I- Introduction.

Cet audit comprend l'analyse du code et celle des performances du projet. Ci-dessous, le processus d'analyse, ainsi que les outils utilisés seront détaillés.

## II- Audit de la qualité du code.

### 1- Introduction à Codacy.

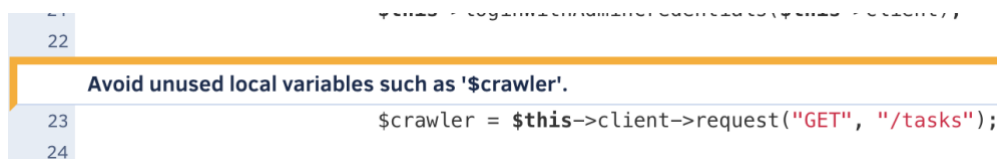
Afin de pérenniser le projet et de vérifier la qualité de celui-ci en minimisant la dette technique de l'application, il convient d'analyser la qualité du code produit.

Pour réaliser cet audit, nous allons nous appuyer sur une solution tierce : **Codacy**.

**Codacy** est un portail qui permet d'analyser le code produit automatiquement et génère une revue sur ce qu'il doit être apporté comme correctif afin d'avoir le projet le plus pérenne possible.

Le rapport liste les problèmes selon leurs types : sécurité, erreurs, duplication, code inutilisé, performance et selon différents niveaux Infos, Issues (problèmes de convention, code inutilisée et Errors (lignes susceptibles de poser des soucis dans le futur).

### Exemple de revue par Codacy :





L'analyse est faite automatiquement à chaque push de notre dépôt vers Github. Chaque commit est analysé, revu puis noté.

**Note :** La configuration de l'analyse exclu donc tous les fichiers de Symfony. Seul notre code produit est analysé (ici : src/ et tests/). Attention à bien exclure tout autre fichier généré par Symfony.

### 2- Analyse du projet par Codacy.

Après analyse du repository Github, notre projet a obtenu le **grade A**, soit le plus haut grade possible, ce qui démontre que notre projet ne possède pas d'erreurs et suit les bonnes pratiques de développement.

Repository name	Grade
 <b>ToDoAndCo</b>	

### III- Audit des performances de l'application.

#### 1- Préface.

L'application ToDo lors du début du projet était en Symfony 3.1.6. Cette version n'étant plus maintenue, elle a été portée vers une version LTS (Long Term Support) en Symfony 4.4.21.

Pourquoi choisir une version LTS ?

- Support allongée concernant les mises à jour de bugs pendant 3 ans.
- Support allongée concernant les mises à jour de sécurité pendant 4 ans.

#### 2- Travaux d'optimisation réalisés.

Pour analyser les performances de notre application un service tiers a été utilisé :

**Blackfire.io.**

C'est un profiler plus performant que celui disponible par défaut dans le Web Profiler d'une application en Symfony et permet :

- De trouver les causes exactes des soucis de performances.
- De stocker plusieurs analyses de profils selon différentes situations et de les comparer (comme ce qui a été fait dans la section juste après).
- D'afficher le temps de génération de la page en PHP et la mémoire utilisée.

Suite à l'analyse de la version de production sans optimisations, deux améliorations ont été apportées :

==> **Optimisation de l'autoloader de Composer pour notre projet :**

À la racine du projet, ouvrez le terminal et entrez la commande `composer dump-autoload --no-dev --classmap-authoritative`

→ Par défaut, notre gestionnaire de dépendance **Composer** va analyser une liste de répertoires inscrits dans le `composer.json`, et pour chaque fichier des répertoires inscrits, il va lister toutes les classes dans un conteneur, et va les charger automatiquement, dès qu'une classe est appelée.

→ **En production**, notre application doit être épurée de tous fichiers et donc classes non nécessaires au fonctionnement en production. Pour cela, entrer ci-dessus dans notre terminal avec deux options :

- **--no-dev** : Exclu tous les fichiers qui sont utilisés pour le développement.
- **--classmap-authoritative** : créer le conteneur (classmap) qui liste les classes qui aide Composer à ne pas analyser les fichiers pour les classes qui ne sont pas dans le mappage des classes.

## ==> Activation de l'extension de PHP, Zend OPcache :

Dans un environnement de production, il convient d'activer **OPCache** par défaut, explication :








→ En temps normal, sans l'extension **OPCache**, le script PHP est analysé et transformé en langage machine (opcode) puis détruit à chaque sollicitation.

→ Avec l'extension, vérifie si le binaire correspondant existe et le chargera s'il l'est. Sinon, il le mettra dans une mémoire partagée jusqu'à ce qu'il soit re-sollicité. Ce processus exclut donc la partie analyse du script et allègera le chargement global de notre application en production.










### 3- Analyse de l'application avant et après optimisation.

Ci-dessous, les rapports d'analyses par le client Blackfire avant et après optimisation. Le rapport détaillé est disponible en cliquant sur chacune des analyses.








Globalement, après optimisation, on distingue -22 à -78% sur le temps de chargement de chacune des pages, et un gain de -71 à -76% de mémoire partagée allouée en moins, ce qui est non négligeable.












Page de login (path : login)			
  login (before)  106 ms  12.7 MB	  login (after)  25.3 ms  3.06 MB		
Comparaison :	 Comparaison  -76%  -76%		












  












Page d'accueil (path : home)			
  home (before)  130 ms  14.7 MB	  home (after)  30.3 ms  3.9 MB		
Comparaison :	 Comparaison  -77%  -73%		












  












Liste des tâches (path : task_list)			
  task_list (before)  145 ms  14.9 MB	  task_list (after)  47 ms  4.03 MB		
Comparaison :	 Comparaison  -68%  -73%		

Ajout d'une tâche (path: task_create)	
  task_create (before)  174 ms  16.6 MB	  task_create (after)  43 ms  4.71 MB
Comparaison :	 Comparison  -75%  -72%

Suppression d'une tâche (path : task_delete)	
  task_delete (before)  161 ms  17.3 MB	  task_delete (after)  35.8 ms  4.25 MB
Comparaison :	 Comparison  -78%  -75%

Tâche terminée (path : task_toggle)	
  task_toggle (before)  123 ms  14.1 MB	  task_toggle (after)  32.8 ms  3.81 MB
Comparaison :	 Comparison  -73%  -73%

Liste des utilisateurs (path : user_list)	
  user_list (before)  129 ms  14.7 MB	  user_list (after)  32.3 ms  3.89 MB
Comparaison :	 Comparison  -75%  -73%

Modification d'un utilisateur (path : user_edit)	
  user_edit (before)  627 ms  17 MB	  user_edit (after)  488 ms  4.88 MB
Comparaison :	 Comparison  -22%  -71%