**Audit de qualité**

**et de performance**

**du code de l'application**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Amélioration de la qualité de l'application** | | | |
| **Projet** | Todolist | | |
| **Pris en charge le** | 20/10/17 | **MAJ** | V1.0 |
| **Par** | Aurélien Thériot | **Version** | MVP |
| **Destinataires** | **ToDo & Co** | | |
| **Fichier** | Audit-de-code-qualite-performances.docx | | |

[**Aller à l'index**](#index)

INDEX

*\*Maintenir Ctrl et cliquez sur le lien voulu pour accéder aux pages correspondantes.*

[I / Qu'est-ce que Todolist ?](#todolist)

[1. Contexte](#contexte)

[2. Plateforme technique](#technique)

[II / Audit de qualité](#qualite)

[1. Préambule](#preambule1)

[2. Architecture du projet](#architecture)

[2.1 Fichiers et apports au dossier app](#app)

[2.2 Fichiers et apports aux dossiers src et tests](#src)

[3. Première analyse Sensiolab Insight et légende des erreurs.](#analyse1)

[3.1 Première violation majeure](#majeur1)

[3.2 Deuxième violation majeure](#majeur2)

[3.3 Troisième violation majeure](#majeur3)

[3.4 Quatrième violation majeure](#majeur4)

[4. Seconde Analyse après correction des erreurs majeures](#analyse2)

[4.1 Première nouvelle violation Critique avec une violation majeure associée](#critique1)

[4.2 Deuxième nouvelle violation Critique](#critique2)

[5. Troisième Analyse après correction des erreurs critiques](#analyse3)

5.1 [Erreurs mineures](#mineur1)

[5.2 Infos](#info1)

[5.3 Infos ignorées](#infoignore)

[6.Quatrième et Analyse finale après correction de quelques erreurs mineures](#analyse4)

[III / Audit de performance du code](#Auditperformance)

[1. Préambule](#preambule2)

[2. Performances](#performances)

[3. Profilage](#profilage)

[3.1 Interpréter les métriques de Blackfire](#profilageinterprete)

[3.2 Analyse des relevés de Profilage](#analyseprofilage)

[3.3 Recommandations](#recommandations)

[4 Bilan](#conclusion)

**I/** **Qu'est-ce que Todolist ?**

**1.Contexte**

Todolistest une application permettant de gérer ses tâches quotidiennes.

L'application était à un stade de prototype (MVP) conçu sous le framework Symfony.

Le but de la manœuvre à présent est d'améliorer la qualité de l'application.

**2.Plateforme technique**

**Framework Symfony :**

**Version du projet MVP sous version 3.1.\***

**Version actuelle du projet mise à jour sous Symfony 3.3.\***

**Version de développement en locale:**

Version Wamp 3.0.6 64bit

Version Apache 2.4.23

Version de PHP 7.0.10

Version de MySQL : 5.7.14

Ajout de l'utilisation des bundles suivants :

"knplabs/knp-paginator-bundle":"2.6.\*",

"phpunit/phpunit": "^6.4.3"

Knp-paginator-bundle servira à créer une pagination pour certaines pages.

PhpUnit servira à la mise en place de test unitaires et fonctionnels.

[**Retour à l'index**](#index)

**II/** **Audit de Qualité**

**1. Préambule**

L' audit de qualité fournit tout au long de ce document, s’appuie sur l’outil [Sensiolab Insight](https://insight.sensiolabs.com/?utm_source=openclassroom&utm_medium=parteners&utm_campaign=PARTENARIATS).

Outil édité par la société Sensiolab, également créatrice du framework Symfony, 4

d'ou la raison du choix de cet outil pour son apport supplémentaire concernant la qualité du code PHP au sein de l'environnement Symfony lui même.

Des screens tiré des rapports d'analyse figureront tout le long de ce document, s'y ajouteront en plus des commentaires relevés par l'outil, mes explications et solutions pour ces differentes relevés.

**2.Architecture du projet sous Symfony**

|  |  |
| --- | --- |
|  | **Voici la structure générale du code du projet Todolist**  [**app**](#app) : Contient la configuration et les templates du site.  **bin** : Permet l'execution des lignes de commandes pour la création de l'application  **Diagrams** : Contient les schemas UML de l'application  **Documentation** : Contient la couverture de code des tests PhpUnit,  le resultat des tests fonctionnels,  un document sur le système d'authentication de l'application,  un autre sur la procèdure de contribution au projet  ainsi que ce document même.  [**Src**](#src): Contient le code source de l'application  [**tests**](#src): Contient les tests PhpUnit  **var** : Contient les fichiers de cache, logs, et de session  **web** : Contient les fichiers nécessaires à l'affichage de l'application pour les internautes. |
|  |  |

[**Retour à l'index**](#Cadre1|frame)

|  |  |
| --- | --- |
| **2.1** **Les fichiers du dossier app et apports** | |
|  | **Legende :**  **La couleur bleu du texte indique une information concernant le projet sous sa forme MVP.**  La couleur orange elle, indique ce qui a été implémenté par la suite, pour améliorer la qualité du code.  **Security.yml :**  **Le fichier sécurité contenait un sytème sécurisé**  **sans distinction dans ses autorisations.**  Pour l'amélioration de qualité, il a été ajouté un système d'authentification avec les rôles USER et ADMIN.  **Dossier Resources :**  **L'application ne contenait pas de pages d'erreurs personnalisées.**  Il a été ajouté un dossier TwigBundle dans lequel il a été créé des pages d'erreurs personnalisées.  Ainsi qu'un layout genéral pour ces pages d'erreur à la racine du dossier Resources/view .  Certains fichiers twig ont également subit des transformation afin d'améliorer le menu de navigation ou de permettre ou restraindre l'acces aux utilisateurs spécifiques à du contenu (en plus de la gestion d'acces côté controller).  Certains templates ont été également été adapté pour recevoir un système de pagination.    [**Retour à l'index**](#index) |

|  |  |
| --- | --- |
| **2.2 Les fichiers du dossier src et tests et apports** | |
| < Suite sur la page suivante > | **Legende :**  **La couleur bleu du texte indique une information concernant le projet sous sa forme MVP.**  La couleur orange elle indique ce qui a été implémenté par la suite, pour améliorer la qualité du code.  **Controller:**  **L'application ne contenait pas de système d'autentification pour différencier les utilisateurs membres.**  Il a été ajouté divers modifications afin de permettre ou dispenser l'accès à certaines pages selon le rôle de l'utilisateur. (System de rôle qui a été mis en place récemment.)  **TaskController:**  **La méthode listAction permettait seulement l'affichage de la liste entière des tâches.**  Il a été ajouté un système de tag afin de permettre l'affichage de la liste entière ou seulement les la liste des tâches terminée ou encore en cours.  **La méthode CreateAction enregistrait les tâches sans les assimiler à son auteur.**  Il a été ajouté la récupération de l'utilisateur de la session et son assimilation directe à l'enregistrement de la tâche créé.  (ceci n'a pas été répété dans la méthode EditAction afin de ne pas modifier l'auteur de la tâche).  **UserController:**  **L'application ne contenait pas de système de pagination pour la liste des utilisateurs.**  Il a été ajouté l'implémentation du bundle knp-paginator afin de lister les utilisateurs (ainsi que plus tard les taches).  La méthode listAction désormais, renvoit des informations supplémentaires liées à la pagination  ainsi que le template visé.  **Entity:**  **La méthode CreateAction enregistrait les tâches sans les assimiler à son auteur.**  Une variable author en relation ManyToOne à été ajouté à l'entité Task vers User afin de ratacher les tâches à leur créateur.  **L'application ne contenait pas de système d'autentification pour différencier les utilisateurs membres.**  Une variable rolename à été ajoutée à User afin qu'un admin du site puisse changer le rôle des utilisateurs  **Form:**  **La méthode CreateAction enregistrait les tâches sans les assimiler à son auteur.**  Il a été ajouté au sein de UserType un champ rolename caché permettant d'enregistrer automatiquement tout nouvel utiliateur en tant que ROLE\_USER.  Il a été ajouté un UserEditType qui permet d'ajouter le champ rolename sous forme de liste d'option permettant à un administrateur du site de changer le role des utilisateurs.  [**Retour à l'index**](#index) |

**3)** **Analyse du code par Sensiolabs Insight**

**après implémentation des nouvelles features et amélioration du projet.**

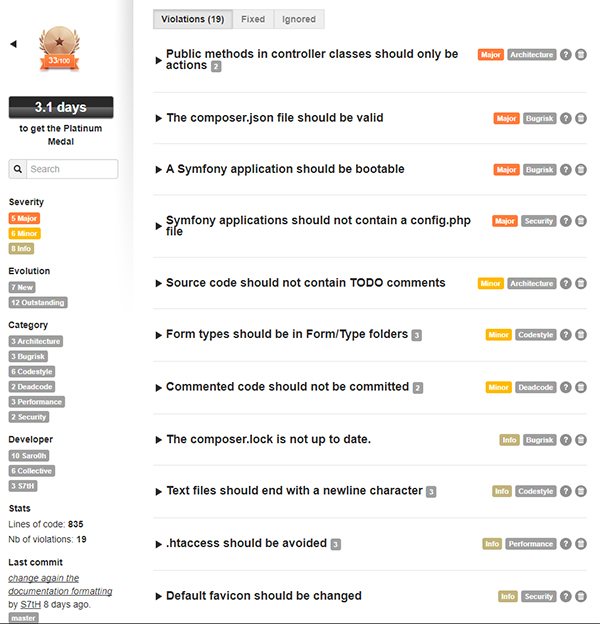
**Legende des types d'erreurs pouvant être relevées :**

Critique : risque de mettre en péril le bon fonctionnement de l’application.

Major : risque de créer des bugs, ou des points de sécurité, cependant moins graves que ceux marqués en critique.

Minor : détails moins importants tels que des problèmes de style ou de lisibilité.

Info : Informations sur les bonnes procédures en vigueur ou conseils pouvant améliorer le contenu.



**3.1** **Première violation majeur**



Les contrôleurs ne doivent contenir que des méthodes nommées avec le suffixe Action.

Les méthodes loginCheck() et logoutCheck() ne respecte pas la convention de nommination des méthodes appartenant au Controllers.

Cette convention permet de distinguer les méthodes qui vont être appelées par le noyau des autres méthodes qui pourrait être crée au sein d'un contrôleur.

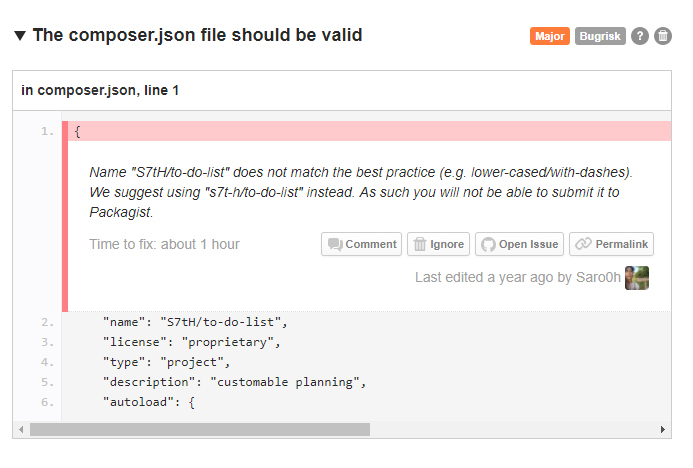
Il suffit donc, soit de rajouter le suffixe Action aux deux méthodes ciblées,

ou encore de créer ces routes dans un fichier yml prévu pour le routing afin de fixer le problème.

Ces routes étant interceptées par Symfony, il serait judicieux de choisir la seconde solution, mais dans un soucis d'homogénéité, toutes les routes de l'application étant appelées par notations au sein des controllers, je vais opter pour la première solution.

[**Retour à l'index**](#index)

**3.2** **Seconde violation majeur**

****

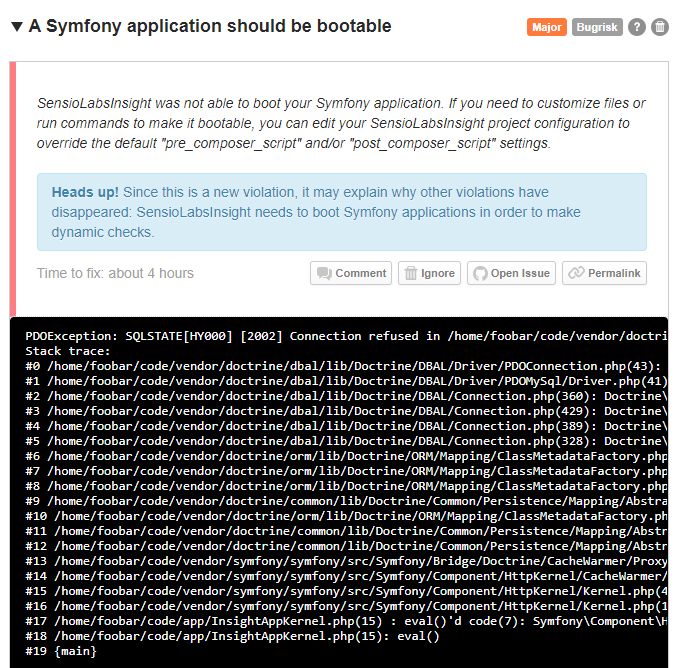
Le problème vient du format de la casse qui se trouve ici avec deux capitales dans le nom « S7tH » qui devrait figurer en minuscule.

Le problème vient du format de la casse qui se trouve ici avec deux capitales dans le nom « S7tH » qui devrait figurer en minuscule.

Pour le fixer il suffit donc de remettre le nom en cause, entièrement en minuscule.

[**Retour à l'index**](#index)

**3.3** **Troisième violation majeur**

******

SensioLabsInsight utilise un moteur d'analyse dynamique pour démarrer des applications afin de trouver des bogues et des erreurs impossibles à trouver par une simple analyse statique. C'est pourquoi, lorsque votre application n'est pas amorçable, SensioLabsInsight déclenche une violation de la qualité du code et désactive certaines règles d'analyse.

Dans des circonstances normales, cette erreur est rarement visible dans les analyses effectuées par nos utilisateurs, mais un changement récent introduit par la version 2.5 du projet DBAL est de casser beaucoup de projets sur SensioLabsInsight.

Fondamentalement, lors de l'appel de la méthode getDatabasePlatform() dans DBAL 2.5, cet appel doit établir une connexion afin d'évaluer la classe de plate-forme de base de données appropriée Doctrine\DBAL\Connection si elle  n'est pas déjà connectée.

**< suite sur la page suivante >**

**< suite Troisième violation majeur >**

La solution consiste à configurer explicitement la version du serveur de base de données utilisée par Doctrine. Dans les applications Symfony, c'est aussi simple que de déclarer le server\_version paramètre de Doctrine DBAL dans le app/config/config.ymlfichier. La valeur de ce paramètre doit être le numéro de version de votre serveur de base de données. Par exemple, si vous utilisez MySQL, exécutez la commande mysql -V pour obtenir la version et configurez-la comme suit:

# app/config.yml

doctrine:

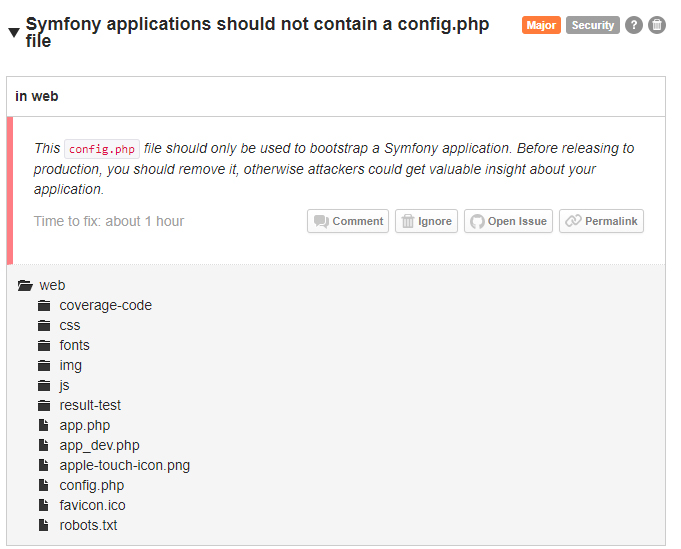
dbal:

server\_version: 5.6

Il suffit donc de déclarer la version serveur de mysql utilisée, la 5.7.14 pour ce projet, dans le fichier config.yml de symfony comme décrit ci-dessus.

[**Retour à l'index**](#index)

**3.4** **Quatrième violation majeure**



Le fichier config.php situé dans le dossier /web sert à initialiser le projet.

Une fois le projet initialisé, il est préférable de le supprimmer.

Sa présence crée une faille de sécurité pouvant donner des informations confidentielles à une personne malveillante.

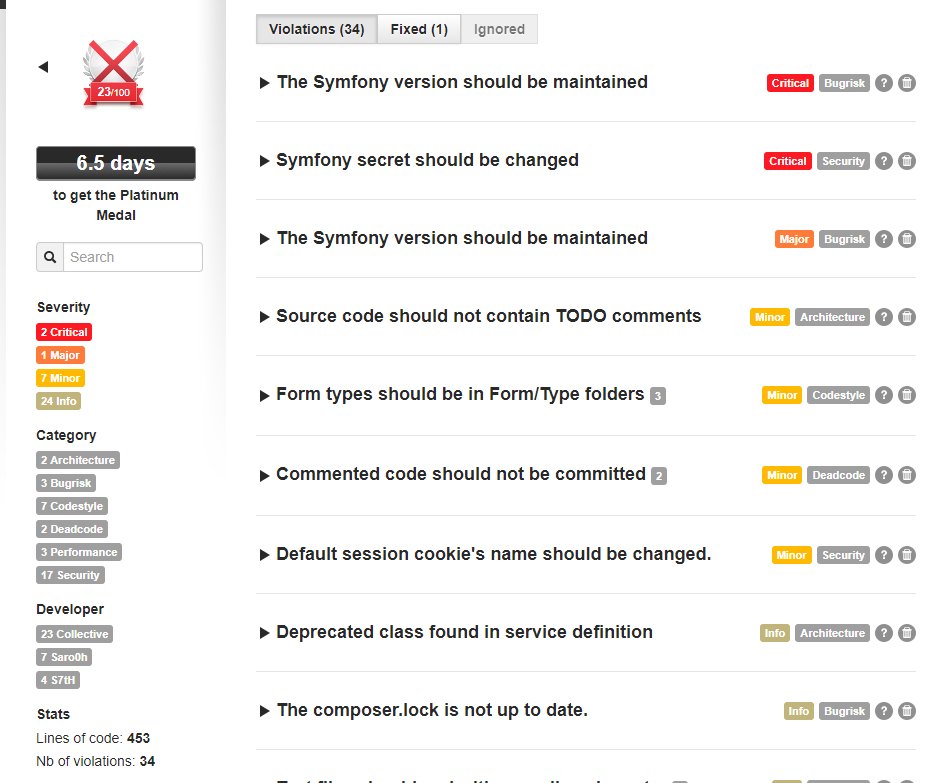
Par défaut ce fichier n’est accessible qu'en localhost, mais il est fortement conseillé de le supprimer une fois en mode production.

Il suffit donc d'ajouter le fichier web/config.php dans le fichier gitignore pour résoudre le problème ou tout simplement de le supprimer.

[**Retour à l'index**](#index)

**4)** **Résultat de la nouvelle analyse**

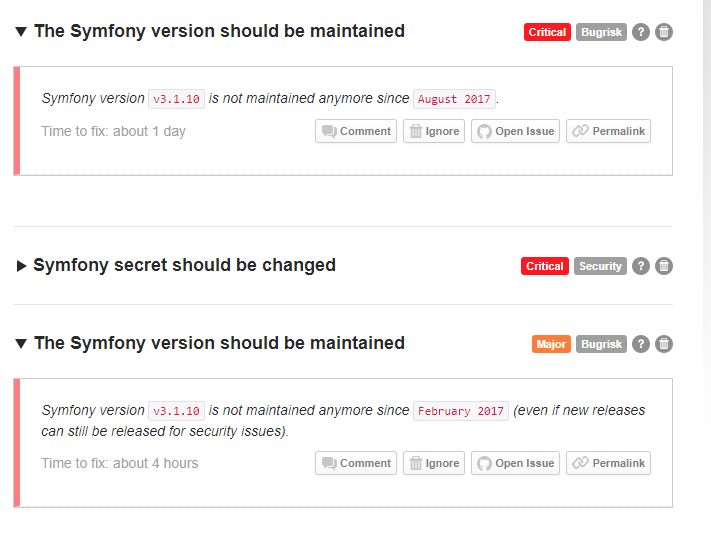
**après avoir corrigé les erreurs majeures.**

****

**Apparition de deux erreurs critiques, mais les erreurs majeures précédentes on bien disparuent et sont donc par conséquent bien réglées.**

[**Retour à l'index**](#index)

**4.1** **Première nouvelle violation Critique avec une violation majeure associée**



Ces deux erreurs son liées et indiquent que la version de symfony actuelle n'est plus à jour.

La version 3.1.10 n’est plus maintenue depuis août 2017.

Aucun correctif ne sera apporté.

A terme, cela risque de poser des soucis de compatibilité.

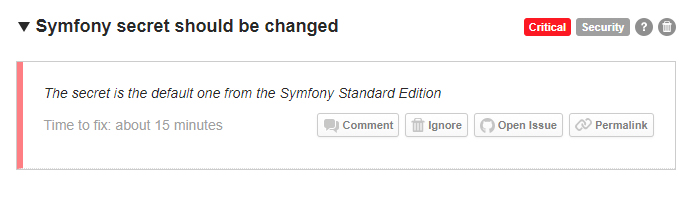
Dans le fichier composer.json, il suffit d'adapter la version demandée de symfony qui était jusque là vérouillée au maximum aux version comprenant 3.1\* en version trouvée sur packagist 3.3.\*

Pour finir il faut saisir dans la console :

php composer.phar update et le problème est résolu.

[**Retour à l'index**](#index)

**4.2** **Deuxième nouvelle violation Critique**



La norme Symfony Edition est livré avec un secret par défaut: ThisTokenIsNotSoSecretChangeIt.

Ce secret est utilisé pour renforcer les fonctions de sécurité dans Symfony, il est donc important que la valeur par défaut soit modifiée.

C'est l'une des premières choses à faire lors de la configuration d'un nouveau projet Symfony.

Il est préférable de le faire avant la mise en ligne d'un projet, mais il peut également être modifié après la mise en ligne d'un projet, bien que certains utilisateurs puissent subir une perturbation mineure.

Je pourrais ignorer tout simplement cette erreur qui ne gêne pas en l'occurence tant que le futur utilisateur de l'application pense a mettre une clé dans son fichier paramètre.

Cependant je vais ajouter une clef autour de 32 caractères differentes de la mienne dans le fichier paramètre.

Afin de customiser cette règle ont peut ajouter ces lignes au fichier de configuration :

rules:

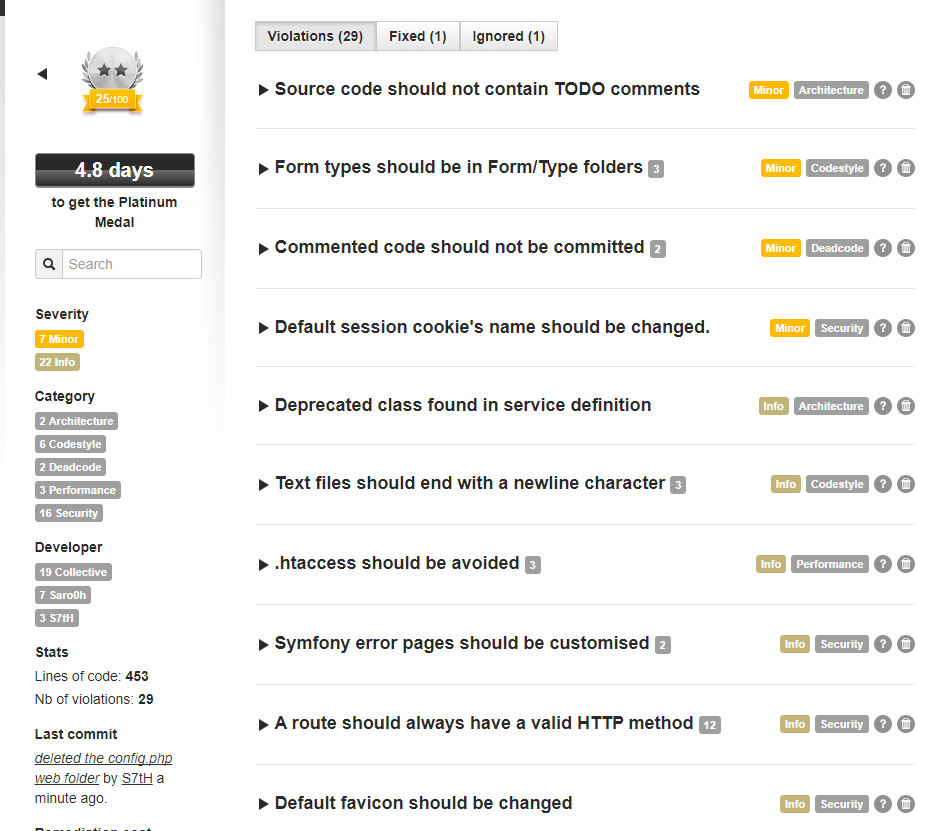
# ...

symfony.obvious\_csrf\_key:

enabled: true

[**Retour à l'index**](#index)

**5)** **Nouvelle Analyse après correction des erreurs critiques**

****

Il ne reste a présent plus que des erreurs mineures et info, la qualité du code est à présent correcte.

[**Retour à l'index**](#index)

**5.1 Erreurs mineures**

-Source code should not contain TODO comments :

Certains fichiers contiennent des commentaires de TODO pouvant signifier la présence d'un code incomplet.

Pour une application en production, cela n'est pas souhaitable.

Il est donc déconseillé de laisser le code ainsi.

Dans le cas présent, il s’agit de la librairie externe Bootstrap.

N'étant pas propriétaire de cette librairie, il est difficile d’apporter des modifications durables.

Il serait judicieux judicieux d’inclure Bootstrap depuis un CDN et non de l’héberger sur notre serveur.

Ainsi ce problème ne nous sera plus relevé et de plus cela peut améliorer les performances.

Dans notre cas, il suffit de retirer le commentaire TODO dans [web / js / bootstrap.js](https://github.com/S7tH/OPC_Projet8/blob/110086d7d1652b6bb3ee04ac8c902c20bd941124/web/js/bootstrap.js#_blank) , ligne 1576, ou de retirer Bootstrap de notre librairie et de l'inclure depuis un CDN.

-Form types should be in Form/Type folders 3

Lors du premier développement d'un projet, il est tentant de sauvegarder les types de formulaires directement dans le Form/répertoire du bundle, mais vous ne devriez pas le faire.

Cela semble agréable et simple, mais au fur et à mesure que le développement progresse, vous devrez probablement étendre le composant de forme d'autres manières. Outre les types de formulaire, vous pouvez étendre le composant de formulaire avec:

Ajout de transformateurs de données personnalisés

Ajout de listes de choix personnalisées

Ajout d' écouteurs d'événement personnalisés

Toutes ces personnalisations, ainsi que les types de formulaires personnalisés, doivent être bien organisées dans la structure de répertoires de votre bundle. La meilleure pratique recommandée consiste à organiser les extensions de formulaire dans les sous-répertoires à l'intérieur du Form/répertoire, en fonction de leur point d'extension.

Il faut de créer un document Type dans le document src / AppBundle / Form et de déplacer les fichier Type.php à l'intérieur.

Il faut également penser à adapter les namespaces de ces fichiers ainsi que l'appel use

de ceux-ci dans les controllers correspondant.

**< suite sur la page suivante >**

**< suite des erreurs mineures >**

-Default session cookie's name should be changed.

Le nom par défaut utilisé pour un cookie de session en PHP est "PHPSESSID".

Il est important que ce nom soit changé en quelque chose de spécifique à chaque application pour éviter que les sessions ne saignent d'une application à l'autre.

Il est conseillé donc de les personnaliser afin d'éviter qu'une personne malveillante puisse

y accéder.

Il suffit de modifier le nom de la session dans app/config/config.yml:

framework:

session:

name: acme\_session

Pour personnaliser cette règle, vous pouvez ajouter la section suivante à la configuration de votre projet:

rules:

# ...

symfony.request.session\_cookie\_default\_name:

enabled: true

-Commented code should not be committed 2

Le code commenté réduit la lisibilité et réduit la confiance du code pour les autres développeurs.

S'il s'agit d'un usage courant pour le débogage, il ne doit pas être validé.

En utilisant un système de contrôle de version, un tel code peut être retiré en toute sécurité.

Dans le cas présent, par défaut, le fichier /web/app.php contient deux lignes destinées

à offrir le choix d'activer ou non le cache de Symfony.

Il est conseillé de les retirer, ou dans le cas contraire de les commenter si activons le cache.

En mettant en place le cache Symfony les deux lignes visées doivent être dé-commentée pour l’activer, ce qui a été fait, problème résolu.

**[Retour à l'index](#index)**

5.2 Infos

-Text files should end with a newline character 3

La façon dont Unix se comporte en général à la fin des fichiers est la suivante:

\ n les caractères ne démarrent pas les lignes; Au lieu de cela, ils les finissent.

Donc, \ est un terminateur de ligne et non un séparateur de ligne.

La première ligne (comme toutes les lignes) n'a pas besoin de \ n pour la démarrer.

La dernière ligne (comme toutes les lignes) a besoin d'un \ n pour y mettre fin.

Un \ n à la fin du fichier ne crée pas de ligne supplémentaire.

Cependant, parfois les éditeurs de texte ajouteront une ligne vide visible à la fin du document.

Il suffit de créer un saut de ligne à la fin des dossiers suivant :

[app/config/config\_test.yml](https://github.com/S7tH/OPC_Projet8/blob/110086d7d1652b6bb3ee04ac8c902c20bd941124/app/config/config_test.yml#_blank), line 21

[app/Resources/views/task/listcontent.html.twig](https://github.com/S7tH/OPC_Projet8/blob/110086d7d1652b6bb3ee04ac8c902c20bd941124/app/Resources/views/task/listcontent.html.twig#_blank), line 30

[Documentation/Authentication.md](https://github.com/S7tH/OPC_Projet8/blob/110086d7d1652b6bb3ee04ac8c902c20bd941124/Documentation/Authentication.md#_blank), line 113

-Symfony error pages should be customised 2

Les pages d'erreur de symfony par défaut ne sont pas détaillées et n'ont aucune mise en forme particulière.

Il suffit de customiser les pages d'erreurs du site pour y remédier en créant un dossier TwigBundle/view/Exceptionet d'y créer des fichiers de type error.html.twig

-Default favicon should be changed

Le favicon dans le fichier /web est par défaut le sigle de Symfony.

Il serait judicieux de créer le sigle propre au site.

Il suffit d'ajouterun favicon customisé à l'application en écrasant le favicon par défaut.

-A route should always have a valid HTTP method 12

Le routeur Symfony prend en charge la requête RESTful en acheminant les demandes de la même ressource vers différentes actions, en fonction de la méthode de requête HTTP.

Par exemple, une requête GET /widgetspeut rendre une liste de widgets, tandis qu'une requête POST /widgets peut insérer un nouveau widget dans la base de données.

Il est important que chacune de vos routes soit spécifique quant aux méthodes autorisées.

Il suffirait d'ajouter les méthodes http GET PUT POST correspondantes aux routes avec :

use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;

[**Retour à l'index**](#index)

**5.3 Infos ignorées**

-Deprecated class found in service definition

Sensiolabinsight dit que depuis la version 3.3 de Symfony, le service “event\_dispatcher” utilise à présent Symfony\Component\EventDispatcher\EventDispatcher au lieu de Symfony\Component\EventDispatcher\ContainerAwareEventDispatcher.

Il suffirait d'utiliser la class Symfony\Component\EventDispatcher\EventDispatcher

au lieu de Symfony\Component\EventDispatcher\ContainerAwareEventDispatcher

utilisée actuellement mais qui n'est plus appréciée depuis la version 3,3 de symfony.

Cependant ce service Symfony fixera certainement ce bug dans les prochaines mise à jour de ce dernier et ne gêne en rien l'application.

Je vais donc l'ignorer.

-.htaccess should be avoided 3

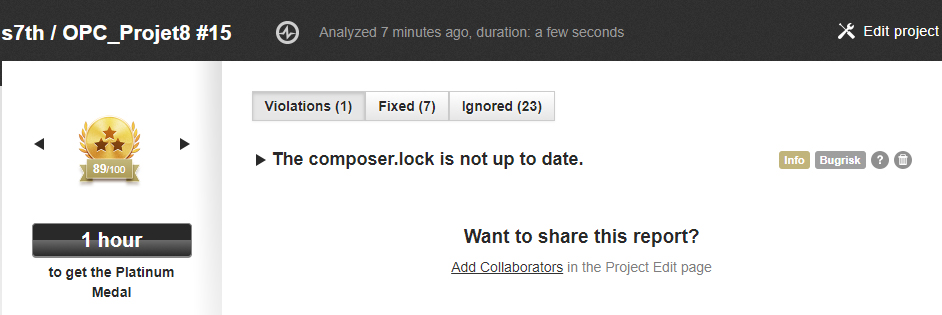
Pour des questions de performances globales, il vaut mieux déplacer les fichiers .htaccess au sein de la configuration du serveur.

Il suffirait de mettre les .htacces dans la configuration serveur

Pour le moment, j'ai décidé d'ignorer ce cas et de le remettre à plus tard.

**[Retour à l'index](#index)**

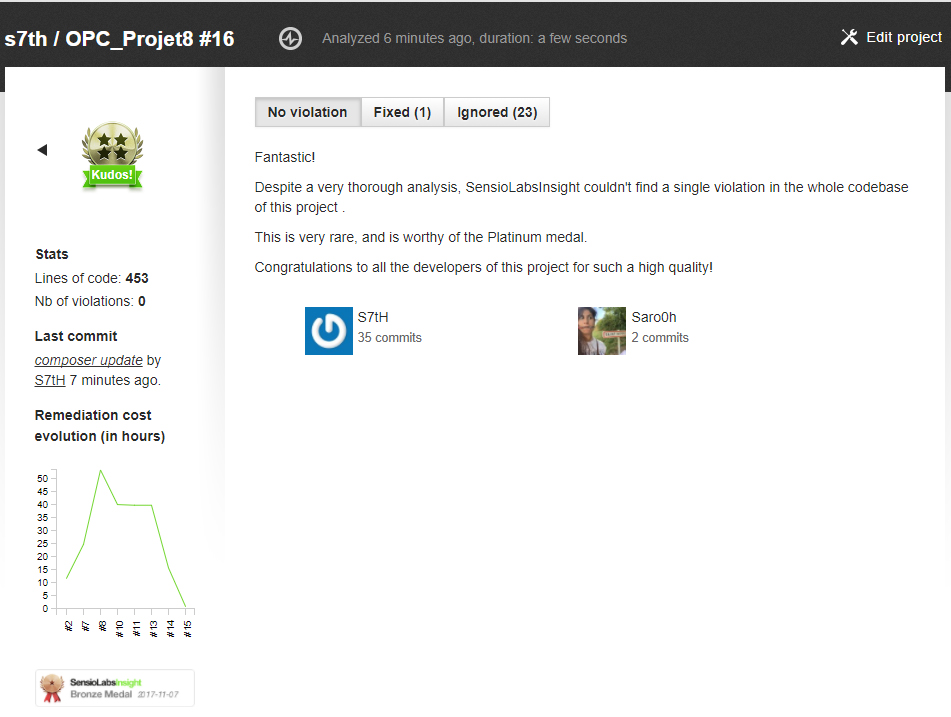
**6) Nouvelle Analyse après correction des erreurs mineures essentielles et application de certaines infos**



SensioLabInsight m'indique que le fichier composer.lock n'est pas à jour.

Dans le cas présent il suffit de mettre composer à jour avec la commance php composer.phar update et le reste se fera automatiquement .

Toutes les erreurs ciblées importantes sont corrigées

****

**III/ Audit de performance du code**

**1.Preambule**

L' audit de performance fournit tout au long de ce document, s’appuie sur l’outil [Blackfire](https://blackfire.io/)

Outil édité par la société Sensiolab, également créatrice du framework Symfony, choisit pour les même raisons que l'outil d'audit de qualité concernant ce projet.

Des screens tiré du profilage des différentes pages du site figureront tout le long de ce chapitre,

s'y ajouteront en plus mon analyse pour améliorer les performances.

**2.Performances**

A partir d'une seconde de temps de chargement d'une page, la sentation de fluidité auprès de l'utilisateur commence à s'estomper.

C'est pour cela, qu'il est important de réduire la charge au mieux pour éviter cette désagréable sensation procurée envers l'utilisateur.

La performance visée est donc la rapidité d'execution de application à transmettre ses informations à l'utilisateur.

Cependant, il faut garder en tête que l'amélioration des performances ont un coût.

C'est du temps en plus passé sur l'application à remodeler cette dernière afin de gagner de la performance. Tâche qui peut vite s'averer coûteuse et très délicate dans bien des situations.

C'est pourquoi il est important de bien analyser ce qui est crutial de ce qui ne ferait gagner que quelques micro secondes et ne s'avèrerait pas rentable sur le plan du coût de l'amélioration.

[**Retour à l'index**](#index)

**3.Profilage**

3.1 Interpréter les métriques Blackfire

**Données de temps :**



**Blackfire nous donne deux sortes de mesures de temps:**

**1)** Le temps exclusif (Colonne %Excl. Dans le métrique) est le temps passé dans la fonction elle-même, à l'exclusion du temps passé chez les enfants.

Il nous indique quels sont les nœuds les plus consommés par eux-mêmes.

Ce sont probablement ceux que l'on pourrait vouloir optimiser en premier.

On parlera dans nos relevés, de coût exclusif qui est la quantité de ressources consommées par le nœud lui-même, sans les coûts de ses enfants.

**2)** Le temps inclusif, (Colonne %Incl. Dans le métrique) quand le temps inclut l'heure de l'appel de la fonction, y compris le temps inclus pour tous ses enfants.

Il nous permet de déterminer le chemin critique de l'application.

On parlera dans nos relevés, de coût global ou inclusif qui est la somme des ressources consommées par le nœud lui-même plus les coûts de tous ses enfants (y compris les enfants des enfants et ainsi de suite).

**En plus des mesures temporel nous avons les calls:**

Les données de la liste Fonction / Métrique incluent également une colonne appelée Calls.

Cette colonne correspond au nombre de fois qu'une fonction/méthode particulière a été appelée.

Cela peut s'averer utile pour vérifier qu'il n'y ait pas de ligne avec une valeur disparate, ce qui pourrait être le signe de la présence d'un code défectueux.

**< suite à la page suivante >**

**< suite de l'interprétation des métriques Blackfire >**

**Le temps est en fait composé de deux parties:**



**1)** Le temps d'I/O est l'heure à laquelle le CPU a attendu les opérations d'entrée / sortie (Input / Output) (réseau, disque, …)

C'est un excellent moyen de repérer les nœuds où se déroulait des activités d' I/O intensives du code qui attendait le réseau (appels de base de données, appels Redis, appels HTTP, ...) ou des disques (inclusions de fichiers, …)

Gardons à l'esprit que l'I/O n'est presque jamais à 0 car elle inclut également des activités non significatives comme l'accès à la mémoire.



**2)** Le temps CPU est la durée pendant laquelle le CPU a été utilisée pour le traitement des instructions.

[**Retour à l'index**](#index)

**3.2** **Analyse des relevés de Profilage**

**Les différents éléments qui nous interessent sur les métriques relevés :**

|  |  |
| --- | --- |
|  | **Temps d'execution** |
|  | **Le temps d'I/O** |
|  | **Le temps CPU** |
|  | **La mémoire utilisée** |
|  | **Le réseau utilisé** |

**Les différents métriques des pages de l'application**

**Page de connexion**

**/login**

****

**Homepage**

**/**

****

**Créer un utilsateur**

**/users/create**

****

**Liste utilisateurs**

**/users**



**< suite à la page suivante >**

**< suite des métriques >**

**Edition d'utilisateur**

**/users/{id}/edit/**

**Liste des tâches**

**/tasks**

****

**Création d'une tâche**

**/tasks/create**

****

**Edition d'une tâche**

**/tasks/{id}/edit**

****

Comme on peut le constater les temps d'executions permettent au site de garder sa fluidité auprès de l'utilisateur tout en gardant à l'esprit que ces données pourront varier une fois en ligne selon la performance du serveur.

aucun problème n’a été relevé en l'état actuel des choses et l'on pourrait presque aussi bien s'en contenter pour une application restreinte à une petite communauté d'utilisateurs.

Cependant certaines améliorations pourraient être menées de manière préventive afin d'augmenter la qualité de l'application en terme de gestion de ressources côté serveur, évitant ainsi au mieux des surcharges de ce dernier pendant une utilisation massive.

Dans la suite de ce document, je vais vous présenter quelques recommandations relevées par l'outil Blackfire permettant donc d'améliorer cette qualité de performance.

[**Retour à l'index**](#index)

**3.3** **Recommandations suggérées par BlackFire**

**a) La classmap autoloader de Composer devrait être déchargée en production**



Composer est l'outil de gestion de dépendance pour PHP.

Il permet aux développeurs de déclarer les bibliothèques dont dépendent les projets et les gère automatiquement (installation / mise à jour).

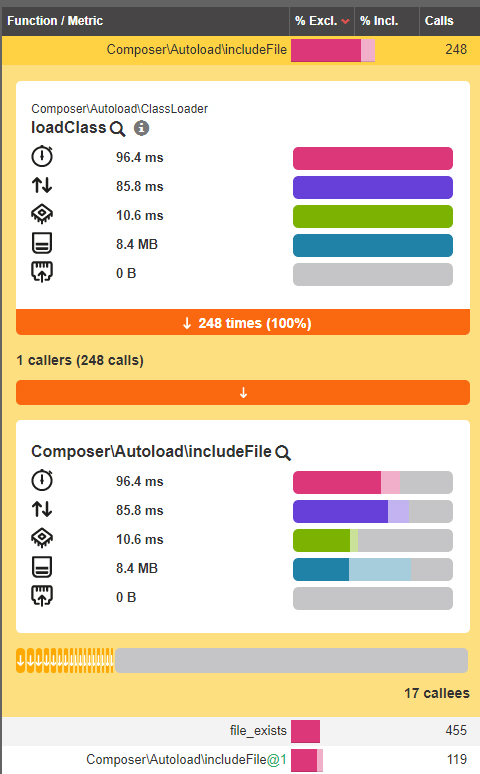
Il gère également la classe [autoloading](https://getcomposer.org/doc/01-basic-usage.md#_blank) en générant une classe autoload dans le dossier vendor/autoload.php quand composer install, composer update ou composer dump-autoload est exécuté.

Chaque fois que PHP charge une classe, Composer doit rechercher le fichier correspondant sur le système de fichiers, ce qui est un processus lent.

Mais ceci peut être optimisé pour la production car la structure du répertoire ne devrait jamais changer.

L'utilisation de composer dump-autoload --optimize décharge une une version optimisée de l'autoloader par génération d'une classmap.

Cependant, la classmap peut être énorme, il est fortement recommandé d'avoir un cache d'opcode PHP installé (comme Zend OPcache).



Comme on peut le constater sur le graph ci-contre, la classe autoload est la plus lourde a charger en temps exclusif et cela sur toutes pages confonduent de l'application.

Dans l'état actuel des choses, l'application restant raisonnable en termes de ressources, cela n'est pas un problème directe.

Cependant, cette dernière étant encore en cours de développement se verra certainement enrichie de nouveau contenu par la suite.

Il est donc indéniable que l'autoloader soit optimisé en production afin de prévenir des coûts ultérieurs.

**< suite à la page suivante >**

**< suite des recommandations >**

**b) Les annotations Doctrine devraient être mis en cache en production**



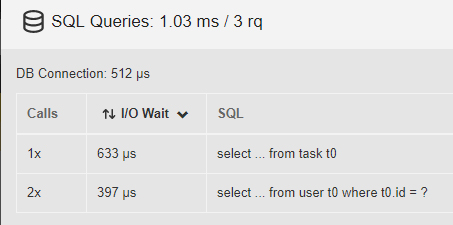
Doctrine ORM est la bibliothèque ORM la plus utilisée dans les applications PHP.

Grâce à cette bibliothèque, vos applications peuvent manipuler les informations stockées dans la base de données en utilisant des objets PHP au lieu des instructions SQL.

Avant d'utiliser l'ORM, vous devez créer ces objets PHP et définir une configuration pour aider Doctrine à transformer les objets en enregistrements de base de données. Cette configuration est appelée "mapping d'entité" et peut être définie dans différents formats (XML, YAML, PHP).

Beaucoup d'applications définissent le mappage des entités à l'aide d'annotations PHP, car elles sont très pratiques.

Cependant, l'utilisation d'annotations pour le mappage a un coût: Doctrine doit transformer cette configuration en code PHP normal exécuté par l'application. Dans les applications réelles avec beaucoup d'entités complexes, ce processus de conversion a un impact sévère sur les performances. C'est pourquoi vous devez mettre en cache l'analyse des annotations Doctrine en production.



Toujours en prévision de l'agrandissement de l'application, l'utilisation du cache Doctrine ferait économiser d'avantage de temps aux traitements des données par le serveur en diminuant le nombre de requête envoyées à la base de données.

Ce qui en soit améliorerait les performances de l'application en augmentant la rapidité d'execution du serveur en allégeant ses tâches.

**< suite à la page suivante >**

**< suite des recommandations >**

**c) Vous devriez executer moins de requêtes SQL**

Les bases de données relationnelles sont l'un des moyens les plus populaires de stocker des informations pour les applications Web.

Par conséquent, il est courant d'effectuer des requêtes SQL pour récupérer les informations nécessaires pour rendre le contenu affiché aux utilisateurs finaux.

La grande majorité des requêtes SQL sont très rapides lors de leur exécution dans le serveur de base de données. Cependant, chaque requête SQL introduit une surcharge dans votre application:

Si la base de données est située sur un serveur distinct, vous devez prendre en compte la communication aller-retour nécessaire pour envoyer la requête et obtenir les résultats.

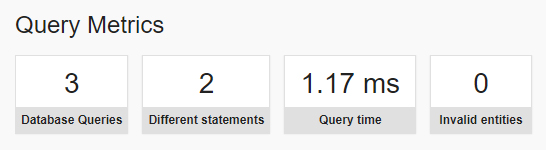
Si le serveur de base de données est sous forte charge, l'exécution de la requête doit attendre que le serveur libère certaines de ses ressources.

Si votre application utilise une bibliothèque d'abstraction de base de données, telle qu'un ORM, la requête d'origine doit être analysée pour que la requête SQL réelle soit exécutée et les résultats doivent être traités pour les transformer en la structure requise par l'application.

Ce sont quelques-unes des raisons pour lesquelles vous ne devriez pas exécuter trop de requêtes SQL pour générer votre contenu.



Dans le cas actuel de l'application, le nombre de requêtes executées reste raisonnable, mais pourrait certainement être amélioré si l'application n'avait pas besoin de récupérer l'utilisateur à chaque exécution.

Cela peut déjà se régler en partie en mettant en place le cache Doctrine, ce qui évitera au serveur de redemander à chaque rafraichissement des informations que l’utilisateur possède déjà.

*L’on remarque ci-dessus que deux requêtes sont faites pour l’appel de l’utilisateur*

[**Retour à l'index**](#index)

**4 Bilan**

**1-** Utiliser la commande composer dump-autoload --optimize pour la mise en production afin d'optimiser l'autoloader afin de réduire au mieux sa charge côté serveur.

(Tâche n°1 non exécutée pour le moment, l’application étant encore en phase de développement)

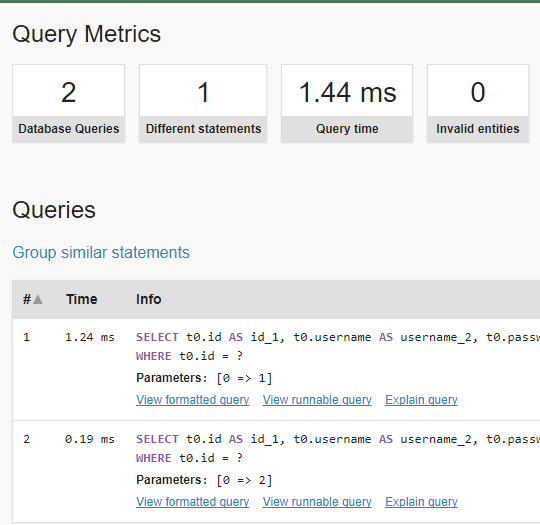
**2-** La mise en place du cache Doctrine s'avèrerait bénéfique à la mise en production de l'application pour alleger le serveur au niveau de ses tâches concernant les requêtes vers une base de données.

La mise en place du cache Symfony pourrait également s’avérer utile,

ainsi les pages ne seront pas regénérées à chaques requêtes étant mis en mémoire dans le cache tant que son contenu reste le même.

Ce qui gagnerait dont en rapidité d'execution ainsi qu'en ressources déployées.

(Tâche n°2 exécutée le 22/11/2017) Cache Doctrine et Symfony mis en place. Installation de l’extension APCu sous Wamp, création du dossier Repository Task afin de configurer l’enregistrement temporaire des requêtes sql dans le cache Doctrine. Activé une ligne de code dans le fichier web/app.php pour le cache Symfony et rendu les réponses http des contrôleurs cacheables.

Notre requête concernant l’utilisateur sur la page de la liste des tâches, n’est appelé qu’une fois désormais au lieu de deux comme c’était le cas avant la mise en place du cache Doctrine.

Comme l’attestent les deux graphs ci-dessous, la mise en cache Doctrine augmente la rapidité d’exécution en mod prod dans le cas ou un utilisateur s’amuserait à faire plusieurs rafraichissements de la page en simultanés.

Voici le test en mod prod de la page d’affichage de la liste des tâches avant la mise en cache Doctrine

A présent voici celui après avoir mis en place le cache Doctrine.

talk_list_cache.jpg

L’on constate que le temps d’exécution est diminué sur plusieurs tableaux (le dernier icône représente le temps d’exécution des requêtes à une base de données) au détriment d’une légère hausse en ressource mémoire.

**< Suite à la page suivante >**

**< Suite du bilan >**

Sur le projet à une plus grande ampleur, il faudra considérer ce coût en mémoire qui pourrait peut être, demander à terme un serveur plus puissant.

A présent voyons cette même page rafraichie avec le cache Symfony mis en place

tasl_list_cachesymf.jpg

L’on constate que le serveur est grandement soulagé de ses tâches à effectuer, ce qui nous fait gagner en performance.

**3-** Il pourrait être interressant sans être primordial de minifier le fichier CSS ainsi que d'appeler Jquery et Bootsrap depuis leur adresse CDN afin de gagner en temps de chargement des pages côté utilisateur. Déchargeant ainsi quelque peu le serveur.

(Tâche n°3 exécutée le 21/11/2017) Fichier shop-homepage.min.css créé, adresses CDN mis en place dans le fichier base.html.twig et suppression des fichiers physique en rapport avec Bootstrap et JQuery.

[**Retour à l'index**](#index)