Audit de qualité et performances

TodoList



[**1 - Audit de qualité :**](#_ewm6rasmqnf8) **2**

[1 - 1 Préambule :](#_fwqpfwn4sj0) 2

[1 - 2 Structure du code :](#_gnkc29yi75oz) 2

[1 - 3 Le rapport de SensiolabInsight :](#_vm6tmuwuvawf) 4

[1 - 3 - 1 : Généralités :](#_mk2qywhmge0n) 5

[1 - 3 - 2 : Les critiques :](#_f78sai6ierbn) 5

[The Symfony version should be maintained :](#_gjtxbksnzc6t) 5

[Symfony secret should be changed :](#_y52s6gwgpekq) 5

[1 - 3 - 3 : Les majeures :](#_s4hkbu9kie5n) 6

[Public methods in controller classes should only be actions :](#_ufarxfj43660) 6

[The composer.json file should be valid :](#_p2pyuboph5o7) 6

[Symfony applications should not contain a config.php file :](#_skt3r0yf1rnm) 6

[1 - 3 - 4 : Les mineures :](#_irw3u9p1urrn) 6

[Source code should not contain TODO comments :](#_rqeakf121rvr) 6

[Form types should be in Form/Type folders :](#_qr1ydsizyjh7) 7

[Commented code should not be committed :](#_ghzv1vip4uoy) 7

[Default session cookie's name should be changed :](#_sl5037ja3yvk) 7

[1 - 3 - 5 : Les infos :](#_h2s2zx4sykbv) 7

[Deprecated class found in service definition :](#_7gfo74ox8vv1) 7

[The composer.json file should not raise warnings :](#_abwmjo9bidrw) 8

[**2 - Audit de performance :**](#_bw1pu7b0dzq0) **8**

[2 - 1 : Prérequis :](#_kyaydo1760fe) 8

[2 - 2 : Backend :](#_qv8fvthttnrx) 9

[2 - 3 : Frontend :](#_d8f647rqhm4t) 10

[**3 -Conclusions :**](#_tfb9c1ey4he4) **10**

# 1 - Audit de qualité :

## 1 - 1 Préambule :

Cet audit de qualité s’appuie sur l’outil Sensiolab Insight, édité par la société Sensiolab, créatrice du framework Symfony, et spécialisé dans l’analyse de qualité de code PHP, et tient compte des spécificités du framework. A cela s’ajouterons des commentaires non relevés par l’outil, mais tirés mon expérience personnelle. L’application étant une MVP, je ne parlerais pas de la qualité visuelle, qui n’a pas encore été abordée et reste encore à travailler.

## 1 - 2 Structure du code :structure.png

Comme le montre la figure ci-contre, la structure du projet est une structure “standard” pour un projet symfony.

**A savoir:**

* un dossier **app** contenant les configurations et ressources applicables à l’ensemble de l’application.
* Un dossier **bin** contenant les ressources nécessaires à l’exécution de la ligne de commande.
* Un dossier **src** contenant les différents bundles, c’est le coeur de l’application.
* Un dossier **tests** contenant l’ensemble des tests de l’application.
* Un dossier **var** contenant les fichiers de cache, les logs et les fichiers de session.
* Un dossier **vendor** contenant les librairies externes utilisées par l’application.
* Un dossier **web** contenant les fichiers à afficher à l’utilisateur de l’application (fichiers de bootstrap, javascript, css, images etc…).

**A cela s’ajoute trois dossiers ajoutés par mes soins :**

* Un dossier **coverage** contenant le rapport de code coverage.
* Un dossier **diagramms** contenant les diagrammes de classe de l’application.
* Un dossier **documents** contenant le présent audit, ainsi que la documentation technique de l’application.

Il est cependant bon de relever un point sur cette structure. En regardant d’un peu plus près les contenu des dossiers et particulièrement le dossier **src** contenant les bundles de l’application, il est à noter qu’elle n’est composée que d’un seul bundle. Hors, il aurait été mieux venu de créer un SecurityBundle. Donc, un bundle séparé pour la sécurité.

Ainsi, le SecurityController et l’entité User auraient été séparés dans un bundle à part, ce qui constitue une bonne pratique.

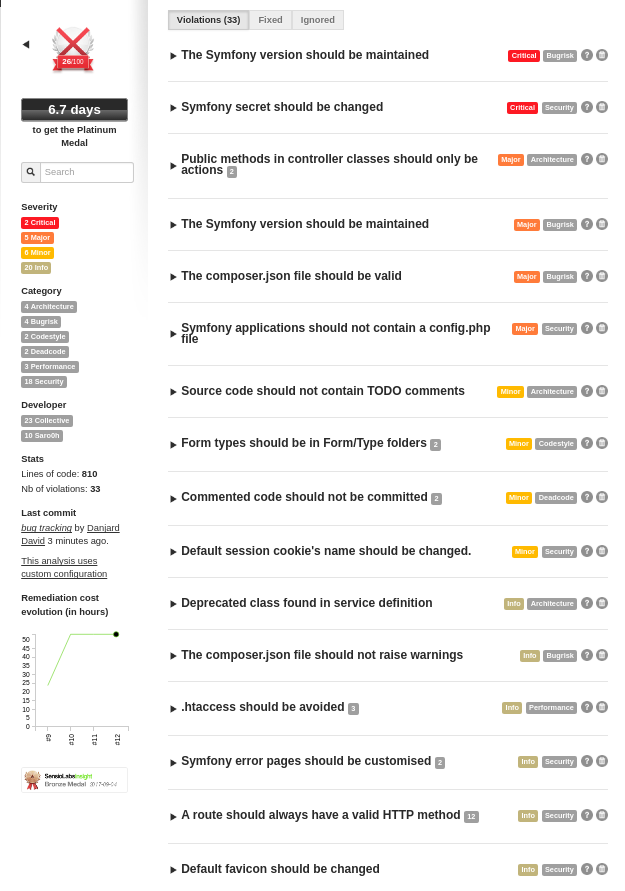
Il est toujours conseillé de garder une certaine logique dans la structure de son application afin de faciliter la maintenabilité, et la scalabilité future de l’application.

De plus, définir à “fr” la variable “locale” du fichier “App/Config/config.yml”, permettrait de faciliter d’éventuels futurs besoins en traduction.

Ensuite, les formulaires sont uniquement validés par la méthode *$form->isValid().* Cette méthode de validation est maintenant déconseillée. Il vaut mieux vérifier si le formulaire à été soumis avant de vérifier s’il est valide. Pour cela, on utilise la méthode *$form->isSubmitted()* avant. Pour l’instant cette façon de faire est juste depricated, mais pour la version 4 de Symfony, elle lèvera une alerte de type Warning.

Pour finir, un README.md qui explique réellement le projet, celui en place n’est pas très explicite et ne permet donc pas au développeur, de situer le projet.

## 1 - 3 Le rapport de SensiolabInsight :



### **1 - 3 - 1 : Généralités :**

Comme le montre la capture d’écran ci-dessus, le rapport de l’outil sensiolabsInsight remonte un certain nombre de points.

Ils sont au nombre de 16, classés en 4 catégories:

* **Critique :** indique un point risquant de mettre en péril de bon fonctionnement de l’application.
* **Major :** indique un point qui risque de créer des bugs, ou des points de sécurité moins graves que ceux marqués en critique.
* **Minor :** Essentiellement des points moins importants tels que problèmes de style ou de lisibilité.
* **Info :** Sont des points d’informations remontés, risquant de poser problème un jour ou l’autre. C’est le dernier niveau d’information de l’outil.

Cet outil d’analyse attribue aussi un score sur 100, afin de mieux voir d’un coup d’oeil, l’état de qualité du code. Le score actuel est de 22/100. Ceci car de nombreux points sont relevés. Nous allons voir lesquels en détail.

### 1 - 3 - 2 : Les critiques :

## The Symfony version should be maintained :

Cette erreur est due au fait que la version de Symfony choisie pour le développement de l’application n’est plus maintenue. En effet, la version 3.1.10 n’est plus maintenue depuis août 2017. Donc, aucun correctif ne sera apporté, ce qui risque, à terme de poser des soucis de compatibilité, voire de sécurité. Pour y remédier, il est conseillé de passer à une version ultérieure de Symfony (actuellement la 3.3).

## Symfony secret should be changed :

Ce paramètre de sécurité à été laissé à sa valeur par défaut. Il est conseillé de choisir une clef unique autour de 32 caractères. Ce token est utilisé à des fins de sécurité pour la génération des tokens présents dans la protection des failles CSRF par exemple. Ce point est donc important, et classé critique.

[ Note ] : La protection contre les failles CSRF est par défaut activée : <https://symfony.com/doc/3.1/form/csrf_protection.html>

### **1 - 3 -** 3 **: Les majeures :**

## Public methods in controller classes should only be actions :

Ce point d’architecture est levé à cause des méthodes **loginCheck()** et **logoutCheck()**, qui ne devraient pas se trouver dans le contrôleur. Les contrôleurs ne devraient contenir que des Actions.

Pour ne plus lever ce point, il vaudrait mieux configurer le routage via un fichier YML (par exemple), afin de pouvoir créer la route “/login” et “/logout”, sans avoir à créer les méthodes loginCheck() et logoutCheck() puisque, comme expliqué dans la documentation technique, ces routes sont interceptées par Symfony. C’est notamment sur ce point que nous rendons compte de l’utilité du SecurityBundle qui aurait permis de gérer les routes liées à la sécurité via un fichier YML, tout en laissant les routes de AppBundle configurées via les annotations.

[ Plus d’informations ] : <https://symfony.com/doc/3.1/security.html>

## The composer.json file should be valid :

Dans ce cas, il s’agit d’une petite erreur, le fichier composer.json, devrait contenir un champ description.

[ Plus d’informations ] : <https://getcomposer.org/doc/04-schema.md#description>

## Symfony applications should not contain a config.php file :

Ce point est important, le fichier **config.php** situé à la racine du dossier /web sert à initialiser le projet. Il devrait être supprimé une fois ceci fait, car il crée une faille de sécurité en donnant des informations qui devraient rester confidentielles à d’éventuels visiteurs malveillants.

Par défaut ce fichier n’est accessible que par le localhost, mais il est tout de même fortement conseillé de le supprimer une fois en production.

### **1 - 3 -** 4 **: Les mineures :**

## Source code should not contain TODO comments :

Certains fichiers contiennent des commentaires de TODO (à faire). Ce qui traduit un code incomplet ou en cours de développement. Ce qui n’est pas une chose souhaitable pour une application en production. Il est donc déconseillé de laisser ainsi le code.

Par contre, il s’agit dans ce cas particulier d’une librairie externe (bootstrap) hébergée au sein de notre application. Dans ce cas, il est difficile d’apporter des modifications. Il serait cependant judicieux d’essayer de trouver une version complète. Il serait aussi encore plus judicieux d’inclure bootstrap depuis un CDN et non de l’héberger sur notre serveur, pour des raisons de performances évoquées plus loin.

## Form types should be in Form/Type folders :

Ce point d’architecture est levé car les formulaires sont situés directement dans le dossier /Form. Il aurait fallu créer un dossier /Type dans ce dossier /Form, et créer les formulaires à l’intérieur de ce dossier.

## Commented code should not be committed :

Ce point est levé, car par défaut, le fichier /web/app.php contient deux lignes commentées destinées (en les décommantant) à activer le cache de symfony.

Il est donc conseillé pour des questions de propreté de code, de les retirer si la gestion du cache n’est pas retenue, ou de les commenter si nous souhaitons activer le cache (en général, un système de cache n’est pas une mauvaise idée en termes de performances).

[ Plus d’informations ] : <https://symfony.com/doc/3.1/http_cache.html>

## Default session cookie's name should be changed :

Ce point de configuration crée un risque en termes de sécurité, il est toujours conseillé de personnaliser au maximum les points sensibles de l’application. Dans notre cas, les noms des cookies de session sont laissés par défaut. Il est conseillé de les personnaliser afin de ne pas faciliter l’action de visiteurs malveillants qui souhaiteraient y accéder.

[ Plus d’informations ] : <https://symfony.com/doc/current/reference/configuration/framework.html#name>

### **1 - 3 -** 5 **: Les** infos **:**

Les infos sont des points moins importants, donnés à titre indicatifs. Il est cependant conseillé de régler aussi ces points. En général les noms des points levés sont assez explicatifs en eux même, sans avoir besoin d’explications supplémentaires. Cependant, je m’attarderai sur deux points particuliers dont le nom est peu descriptif :

## Deprecated class found in service definition :

Il s’agit d’un petit bug de sensiolabinsight qui dit que depuis la version 3.3 de Symfony, le service “event\_dispatcher” utilise Symfony\Component\EventDispatcher\EventDispatcher au lieu de Symfony\Component\EventDispatcher\ContainerAwareEventDispatcher auparavant.

Ce service est un service Symfony, ce bug sera fixé avec les prochaines mise à jour de Symfony.

## The composer.json file should not raise warnings :

Ce warning en question, est le fait que l’autoload à été défini avec un namespace vide. Ce qui veut dire que, lors d’un appel à une classe, l’autoload vas rechercher dans l’ensemble des fichiers en partant de la racine. Pour éviter cela et donc gagner en performances, il est conseillé de définir un namespace dans lequel l’autoload ira chercher les classes appelées.

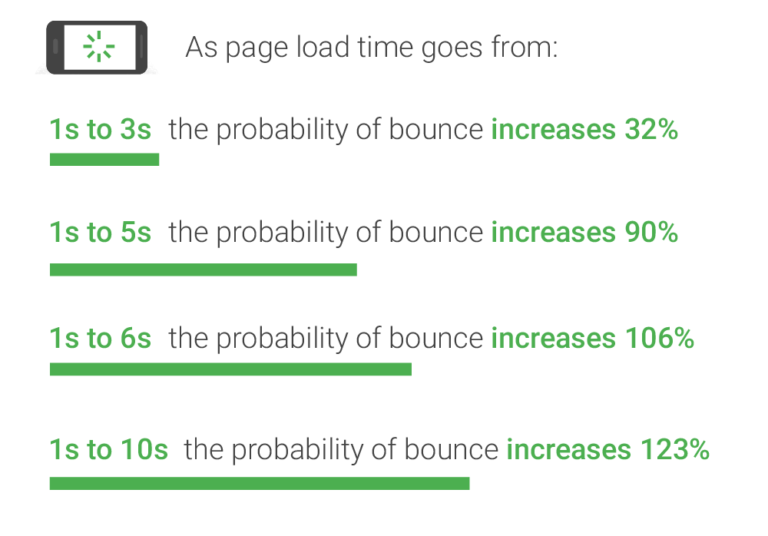
[ Plus d’informations ] : <https://stackoverflow.com/questions/36158929/why-specify-the-namespace-when-using-psr-4-autoloading-with-composer>

# 2 - Audit de performance :

## 2 - 1 : Prérequis :

**Qu’est ce que la performance ?**

Quand on parle de performance, on parle de la rapidité de rendu d’une application. Il est important de préciser ce point, car aujourd’hui l’utilisateur est de moins en moins patient en ce qui concerne le temps d’attente pour accéder à une ressource sur internet.



Cette infographie montre, d’après la dernière [étude google](https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/), l’impact du temps de chargement d’une page sur le taux de rebond. Une seconde étant la limite à partir de laquelle l’utilisateur n’a plus l’impression de fluidité.

Nous voyons donc que la performance est un point important, qui ne doit surtout pas être négligé.

Cependant, l’amélioration des performances d’une application finit par avoir un coût, il est donc important de savoir pourquoi chercher à améliorer les performances d’une application, afin de mettre en face le coût de l’amélioration, des retombées économiques potentielles.

Tout est donc une question de juste équilibre.

## 2 - 2 : Backend :

Les métriques côté backend ont été relevés grâce à BlackFire, l’outil d’audit réalisé par sensioLab, société créatrice du framework Symfony.

Récapitulatif des métriques obtenues via l’outil BlackFire :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **URI** | **Temps execution Total** | **I/O** | **CPU** | **Mémoire** | **Réseau** |
| **/users/create** | 247 ms | 187 ms | 60.2 ms | 17.8 MB | 719 B |
| **/tasks/create** | 250 ms | 191 ms | 59 ms | 17.7 MB | 1.28 KB |
| **/tasks** | 351 ms | 292 ms | 58.9 ms | 16.3 MB | 1.39 KB |
| **/login** | 218 ms | 173 ms | 44.6 ms | 8.96 MB | 0 B |
| **/** | 244 ms | 193 ms | 50.2 | 14.1 MB | 719 B |

Comme le montre le tableau récapitulatif ci-dessus, les temps d’exécutions côté serveur sont très raisonnables.

Ces résultats restent à pondérer tout de même, car testés en local. Ils ne laissent cependant pas voir de problème particulier côté serveur.

Cependant, il serait sûrement bienvenue d’utiliser les capacités de [cache de symfony](https://symfony.com/doc/3.1/http_cache.html) voire [varnish](http://symfony.com/doc/current/http_cache/varnish.html), afin de prévenir à l’avance d’éventuels problèmes de surcharge sur le serveur lors de la mise en production. D’autant plus que le site n’en est qu’au début de son développement, et risque bien de s’enrichir en terme de fonctionnalités, donc de devenir plus lourd à traiter côté serveur.

L’utilisation du [cache doctrine](http://blog.kaliop.com/blog/2014/10/06/doctrine-symfony2-2/) peut aussi être une bonne pratique. Elle permet de diminuer le nombre de requêtes à la base de données, et donc d'accélérer grandement le traitement côté serveur.

Pour finir, il serait une très bonne pratique, que d’intégrer les tests de performances aux tests unitaires, [comme décrit dans la documentation de blackfire](https://blackfire.io/docs/integrations/phpunit). Ceci permettant de diminuer les risques qu’une future release ne cause des soucis de latence sur le serveur.

## 2 - 3 : Frontend :

Chercher à obtenir des métriques sur une application hébergées en local, ne serait pas représentatif. Ce qui fait le temps d'exécution côté frontend, c’est le temps d’exécution du navigateur, mais aussi et surtout les temps de transfert sur le réseau, que nous ne pouvons pas mesurer en local.

Cela n’empêche cependant pas de relever quelques points.

Tout d’abord, les fichiers jquery et bootstrap sont hébergés en local. Il est mieux conseillé de travailler avec des fichiers appelés depuis un CDN. Ceci évite de surcharger son propre serveur, permet de télécharger ces fichiers depuis des serveurs puissants, disposant d’une très bonne connection, et d’un très bon temps de réponse, et pour finir, si le navigateur est déjà passé sur un site faisant appel à ce CDN, il disposera des fichiers en cache et n’ira donc pas du tout les chercher sur le réseau.

Ensuite, les fichiers CSS et JS ne sont ni minifiés, ni regroupés. Du coup, l’appel à chaque fichier se fera séparément, ce qui augmente significativement le temps de transfert, et ont une taille individuelle plus importante que s’ils avaient été minifiés, ce qui tend encore à augmenter les temps de transfert.

Afin de répondre à ces problématiques, il peut, par exemple être utilisé l’outil [GULP](https://gulpjs.com/) afin d’automatiser ces tâches. Un très bon article est disponible sur cette page : <https://blog.pabuisson.com/2014/05/automatisez-workflow-avec-gulpjs/>

# 3 -Conclusions :

Pour conclure, les principaux besoins en termes de refactoring, sont d’ordre qualitatif. J’irai un peu plus loin en disant qu’il serait risqué de mettre en production l’application dans l’état actuel des choses.

La performance n’est pas à laisser en reste, mais aucun problème n’a été relevé pour l’instant. Cependant certaines améliorations pourraient être menées de manière préventive. Ceci afin d’apporter une meilleure expérience utilisateur, et éviter les éventuels risques de surcharge sur le serveur en cas de fort trafic.

**Annexe :**

Après avoir mis en place quelques recommandations, de ce document, une nouvelle analyse de performance à été réalisée.

Il a été implémenté :

* L’appel à un CDN pour les librairies Bootstrap et JQuery.
* Mise en place du cache doctrine.
* Mise en place du cache Symfony.
* Améliorations de qualité de code ([sensiolabInsight](https://insight.sensiolabs.com/projects/058ea154-4ff1-4fd4-bb74-564fc451a215/analyses/20)).

Ceci nous permet de relever de nouvelles métriques, et de les comparer aux anciennes :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **URI** | **Temps execution (Ancien)** | **Temps execution (Nouveau)** | **Evolution**  **(ms)** | **Evolution**  **(baisse en %)** |
| **/users/create** | 247 ms | 19 ms | -228 ms | 1300 % |
| **/tasks/create** | 250 ms | 20.4 ms | -229.6 ms | 1225 % |
| **/tasks** | 351 ms | 20 ms | -331 ms | 1755 % |
| **/login** | 218 ms | 19.7 ms | -198.3 ms | 1106 % |
| **/** | 244 ms | 24.7 ms | -219.3 ms | 988 % |

Comme nous pouvons le voir dans les relevés de métriques ci-dessus, la baisse est considérable. En moyenne, une requête s’exécute 12.7 fois plus rapidement qu’avant, côté serveur.

Tout simplement, car, grâce au cache, la page appelée n’est pas régénérée à chaque requête, mais issu du cache directement en mémoire. Elle ne sera régénérée qu’en cas d’invalidation (délai expiré ou modification de son contenu).

Au final, le nouveau code-coverage est de 96.3 %.