Audits

Audit de qualité du code & performance de l'application

Dans cette partie nous allons comparer l'application initiale (au début de reprise du projet) et l'application améliorée

Audit de qualité du code

Plusieurs outils ont été utilisés pour tester la qualité du code. Nous allons présenter des analyses de code réalisées sur Codacy et CodeClimate. Nous avons aussi mis en place des tests unitaires et fonctionnels pour vérifier la qualité du code mais aussi détecter les bugs et dépréciations.

Codacy & CodeClimate

Les analyses Codacy et CodeClimate n'ont pas révélés de problèmes majeurs sur le code de l'application. Néanmoins, des erreurs de sécurités ont été relevés sur la version initiale de l'application. Parmis les plus problématiques, nous retrouvons l'utilisation de variable superglobale sans filtre php particulier ($_SERVER) ainsi que l'affichage de code HTML via la méthode php `echo`, ce qui est déconseillé.

L'analyse des deux projets (initial et amélioré) sont disponibles en suivant les liens suivants : projet initial / projet amélioré

PHPUnit

Concernant PHPUnit, 61 tests ont été réalisés pour un rapport de couverture de l'application supérieur à 85%. Le rapport de tests est disponible ici.

Conclusion

L'audit de qualité de code des deux projets (initial et amélioré) nous as permis d'observer que la montée de version a eu des bénéfices en terme de qualité de code mais aussi de sécurité. Ce n'est pas pour autant que le projet initial n'était pas sécurisé, mais la version de Symfony 3.1 n'était plus supportée et des failles de sécurité commencaient à apparaitre, comme nous l'a rappelé GitHub lors de l'installation du projet :

Dans le même temps, qui dit montée de version Symfony, dit aussi montée de version PHP. Nous sommes passé de la version 5.4 à la version 7.2, ce qui nous as permis d'éliminer bon nombre de méthodes PHP dépréciées.

Audit de performance de l'application

Nous allons présenter dans cette partie un comparatif de performance entre l'application initiale et l'application améliorée.

Pour les deux outils utilisés (BlackFire, Chrome Dev Tools), nous faisons 1 test de performance par application sur les pages centrales de l'application (login / accueil / liste utilisateurs / liste de tâches).

Après chaque test de performance, chaque application est rechargée entièrement, y compris le cache de l'application (via un `php bin/console cache:clear`).

devtool chrome

Blackfire

Dans cet audit de performance, nous allons utilisé BlackFire pour relever le temps de chargement et le pic d'utilisation de la mémoire vive nécessaire pour charger la page demandée.

Malheureusement, nous ne pouvons pas avoir accès à d'autres métriques avec un plan gratuit.

before

L'analyse de Blackfire donne : le temps de chargement du site, la mémoire utiliser pour son chargement, les fonctions utilisées et son nombre d'appels. Chaque fonction est détaillée avec son temps d'exécution et la mémoire utilisée. Un schéma est aussi consultable pour suivre le plan d'exécution du site, avec une couleur spécifique pour les fonctions les plus gourmande. Les tests ici présents, sont faits avec la licence gratuite de blackfire, ce qui limite l'analyse du site. À savoir: avec la version payante des recommandations d'optimisation sont proposés.

Optimisation

Pour optimiser cette partie de l'application il faut exécuter une simple commande dans le terminal à la racine du projet :

```
composer dump-autoload --no-dev –classmap-authoritative
```

Cette commande sert à mettre en cache les classes utiles à l'application, toutefois, ci de nouvelles classes sont ajoutés il faudra absolument relancer cette même commande ! • `--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 autoloaddev) • `--classmap-authoritative` crée un mappage de classe pour les classes compatibles PSR-0 et PSR-4 utilisées dans votre application, et empêche Composer d'analyser les classes qui ne se trouvent pas dans le mappage de classe

L'autoloader charge automatiquement les classes utiles à l'application. Dans le cas de composer, l'autoloader est dans un dossier « vendor » à la racine du projet. Avec la commande écrite un peu plus haut, des fichiers sont créés pour faire une carte des classes à charger pour l'application, ce qui évite de charger tous les chemins à chaque chargement de pages. Documentation

Bilan

before

Voici les résultats de l'optimisation de l'autoloader, le temps a eu une baisse de -35 % et la consommation de données -15 %. Beaucoup de fonctions ont eu une réduction significative d'appel, voir même n'est plus du tout appeler.