

Audit de qualité de code

Application To Do List

Décembre 2020

Caroline Dirat



Formation OpenClassrooms

« Développeur d'application PHP/Symfony »

Projet 8 - **Améliorez une application existante**

TABLE DES MATIERES

Mesures de qualité de code	2
Sécurité	3
Dette technique	3

MESURES DE QUALITE DE CODE


CODECLIMATE

 maintainability **A**

Code Climate est une application qui évalue la maintenabilité du code.

Le code de **To Do List** obtient la note de maintenabilité **A** sur Code Climate.

CODACY

 code quality **A**

Codacy est une application qui analyse la qualité de code, tout en étant paramétrable. Ici, Codacy analyse le style de code de trois langages :

- **PHP** (avec PHP Code Sniffer et PHP Mess Detector)
- **CSS** (avec CSSLint)
- **Javascript** (avec JSLint)

Codacy évalue aussi la **complexité** du code et les **duplications** de code, qui ont un impact sur la **maintenabilité** du code.

Le code de l'application **To Do List** obtient la note d'évaluation **A**, avec **0% de Complexité** de code et **0% de Duplication** de Code.

À l'issue de notre travail d'amélioration de l'application To Do List, nous obtenons donc une **bonne note de qualité de code**.

TEST UNITAIRES ET FONCTIONNELS

 coverage **100.00%**

Le **taux de couverture de code** par des **tests unitaires** et **fonctionnels** via **PHPUnit** est de **100%**. Cela ne signifie pas que le code est testé à 100%, tandis que l'on ne teste pas le code propre au framework Symfony et celui des dépendances dans le dossier vendor/. Mais les tests sont présents sur les **besoins métiers critiques** (comme par exemple, l'accès à la gestion des utilisateurs uniquement par les utilisateurs avec le rôle ROLE_ADMIN).

L'application présente également des **tests fonctionnels** via **Behat**, afin de tester les **scénarios utilisateurs** que l'on peut lire (en anglais) dans les fichiers .feature du dossier features/.

Les tests PHPUnit garantissent la qualité de code de l'application, et en particulier sa **maintenabilité** puisqu'ils permettent d'apprécier l'impact d'une modification de code sur les fonctionnalités existantes.

En outre, intéressons-nous à la gestion de la sécurité dans l'application.

SECURITE

PROTECTION DES FORMULAIRES

- Tous les formulaires sont protégés par un jeton (token) contre les attaques de type « Cross-Site Request Forgery », y compris le formulaire de connexion.
- Les données entrantes sont filtrées par des Contraintes de validation.

PROTECTION CONTRE LA FAILLE XSS - CROSS SITE SCRIPTING

- ▶ Filtrage des entrées (grâce contraintes de validation de **Symfony**)
- ▶ Echappement des données en sorties (par **Twig**)

VULNERABILITE DES DEPENDANCES

La version du **framework Symfony** utilisée pour construire l'application était initialement sur une version qui n'est plus maintenue (la version 3.1). On utilise désormais la **version lts** (4.4.17), qui offre 3 ans de support pour les bugs et correctifs de sécurité.

Aussi, le binaire de Symfony offre un outil de vérification des vulnérabilités de sécurité des dépendances de l'application :

```
symfony check:security
Symfony Security Check Report
=====
No packages have known vulnerabilities.
```

Donc, les **dépendances** du projet ne présentent **pas de vulnérabilité de sécurité**.

DETTE TECHNIQUE

BONNES PRATIQUES DE SYMFONY

Deux points sont à améliorer pour être en accord avec les [bonnes pratiques de Symfony](#), afin d'optimiser la sécurité de connexion pour l'un et la maintenabilité du code pour l'autre.

➤ Le fournisseur d'authentification du formulaire de connexion

Comme il est expliqué dans la [documentation technique sur l'implémentation de l'authentification](#), la méthode d'authentification à l'application To Do List utilise le [fournisseur d'authentification form_login](#).

Symfony recommande de créer une authentification de connexion par formulaire avec **Guard**. En effet, l'**authentificateur Guard** permet un **contrôle total** sur le **processus d'authentification** (ce qui n'est pas le cas avec form_login) et permet donc d'**optimiser la sécurité** de la connexion à l'application.

Pour implémenter un authentificateur Guard, le procédé est expliqué dans la [documentation officielle](#) de Symfony.

➤ Utiliser **Webpack Encore** pour traiter les ressources Web

Le développement reste très sommaire côté frontend. Lorsque l'application va s'étoffer de ce côté-là, il sera intéressant de **faciliter** la **gestion** des fichiers **CSS** et **Javascript** en installant **Webpack Encore**.

Webpack Encore permettra aussi d'intégrer **Bootstrap** à l'application, au lieu de charger ses fichiers via les CDN pointés depuis le template de base (templates/base.html.twig). Voir la [documentation officielle](#) de Symfony.

En facilitant la gestion des fichiers CSS et JavaScript, on améliore la **maintenabilité** de l'application.

SEO (SEARCH ENGINE OPTIMISATION)

L'application aura aussi besoin d'améliorations du point de vue du SEO, c'est-à-dire sa capacité à être référencée dans les moteurs de recherche. Le site [d'opquast](#) liste 80 bonnes pratiques à suivre.

Un bon nombre de ces bonnes pratiques sont déjà en vigueur dans l'application To Do List :

- ✓ Chaque image est dotée d'une alternative textuelle.
- ✓ La longueur des alternatives textuelles est inférieure ou égale à 80 caractères.
- ✓ Chaque page du site contient un élément de titre de section H1.
- ✓ Le site comporte autant de titres de section H1 différents que de pages.
- ✓ La page d'accueil expose la nature des contenus et services proposés.
- ✓ Les images sont cohérentes avec les contenus de la page.
- ✓ Le libellé de chaque hyperlien décrit sa fonction ou la nature du contenu vers lequel il pointe.
- ✓ Les adresses URL ne contiennent pas d'indication concernant les paramètres de session.
- ✓ Les URL des liens internes contiennent exclusivement des caractères alphanumériques ou considérés comme sûrs.
- ✓ Tous les hyperliens du site sont valides. (et vérifié dans les tests fonctionnels Behat)
- ✓ Les liens internes utilisent une URL unique pour chaque page.
- ✓ Le titre de chaque page (élément TITLE) permet d'identifier son contenu.
- ✓ Le code source de chaque page contient une métadonnée qui définit le jeu de caractères.
- ✓ Le contenu de l'élément TITLE de chaque page ne commence pas par le nom du site.
- ✓ La navigation est possible via des liens HTML.
- ✓ Les contenus HTML sont mis en forme à l'aide de styles CSS externalisés.
- ✓ Le serveur envoie un code HTTP 404 pour les ressources non trouvées.

Mais on peut encore améliorer le code HTML (des fichiers du dossier templates/).

Par exemple :

- ▶ L'alternative textuelle des images devrait être plus explicite que le simple « todo list »
- ▶ Les termes présents dans l'alternative textuelle des images doivent être également présents dans le contenu de la page.
- ▶ Les mises en majuscules à des fins décoratives doivent être effectuées à l'aide des styles CSS.

- ▶ Le contenu visé pour le référencement doit être mis en exergue (avec les balises **strong** et **em**).
- ▶ De nombreuses balises **div** pourraient être remplacées par des balises HTML5 plus spécifiques (**section, article, main...**)
- ▶ Dans la balise `<meta name="description">`, il faudrait renseigner son attribut content avec la description du contenu propre de la page (une phrase significative, et non une liste de mots).
- ▶ Dans la balise `<meta name="author">`, il faudrait renseigner son attribut content.
- ▶ Les titres de section HTML doivent comporter des mots clefs contenus dans la balise **meta keywords**.