



QUALITY AND PERFORMANCE AUDIT

Amélie-Dzovinar Haladjian

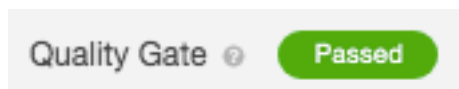
Introduction

This document compares the MVP version of Todo&Co's to-do list application before any changes were made, with the subsequent version developed to answer any overall quality and performance issues. This document includes reports regarding code quality and performance generated by SonarCloud and Blackfire but also sums up any general code improvements made throughout the code base as well as enhancements on both the user interface and user experience.

SonarCloud Quality Report

The application's code quality was analyzed using [SonarCloud](#). An analysis was performed both before and after any changes were made to the project. The following report will compare both analyses.

SonarCloud breaks down an application's code quality into 5 different measures, which are then analyzed to give an overall pass or fail status.

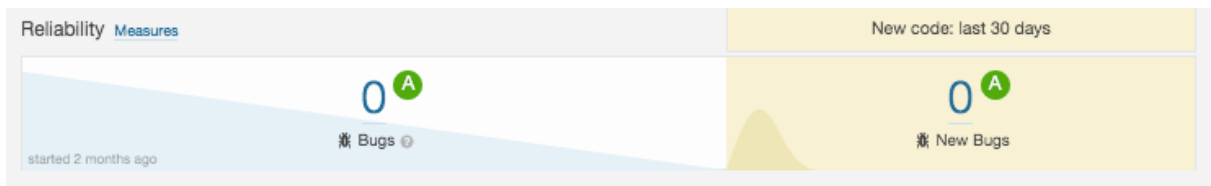


Todo&Co's to-do list application passed SonarCloud's Quality check both before and after any improvements were made. The results of the different measures are listed below. The final report is available [here](#).

1. Reliability

To assess the app's reliability, SonarCloud determines the number of bugs contained within the application. Bugs are defined as being *"A coding error that will break your code and needs to be fixed immediately"*.

The reliability metric passes the quality gate with an A rating.

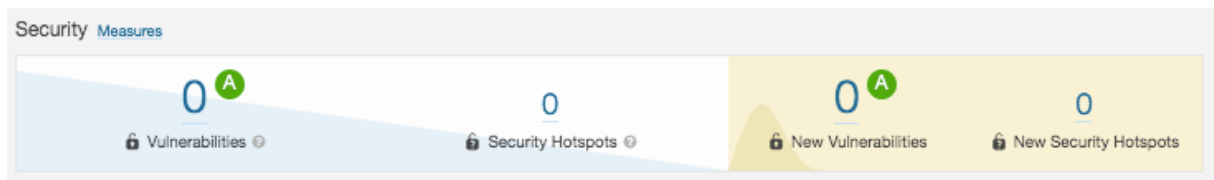


No bugs were detected during the initial analysis and no bugs were introduced after changes were made to the application.

2. Security

To determine whether the app is secure, SonarCloud checks if the code contains any vulnerabilities, which are defined as “*Code that can be exploited by hackers*”.

The security metric passes the quality gate with an A rating.

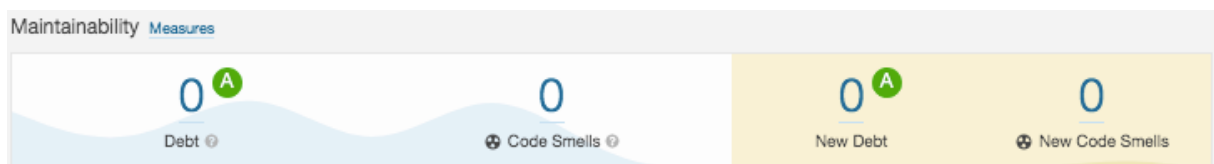


The analysis did not raise any security concerns both prior to and after any changes were made.

3. Maintainability

The app’s maintainability is assessed based on code smells and the estimated time it will take to fix them. Code smells are described as “*Code that is confusing and difficult to maintain*”.

The maintainability metric passes the quality gate with an A rating.

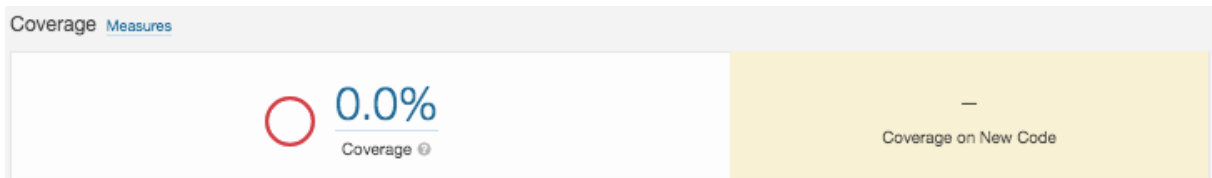


The analysis detected a few code smells in the original project, which were then fixed by using constants instead of using the same string in different places throughout the application.

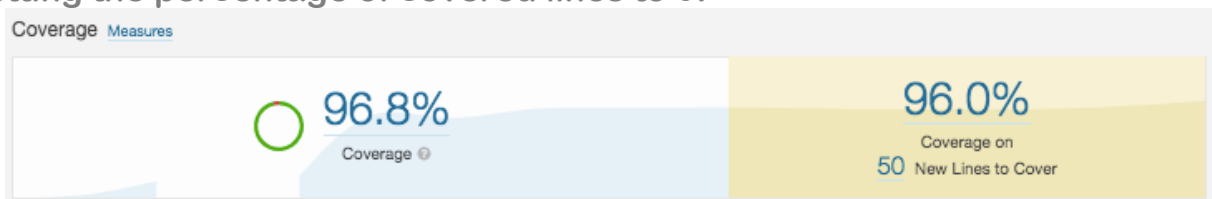
4. Coverage

The coverage indicates the percentage of lines covered by tests. The analysis is based on PHPUnit's coverage reports.

The coverage metric passes the quality gate with an 80% coverage rate.



The initial project did not contain any unit or functional tests, thus setting the percentage of covered lines to 0.

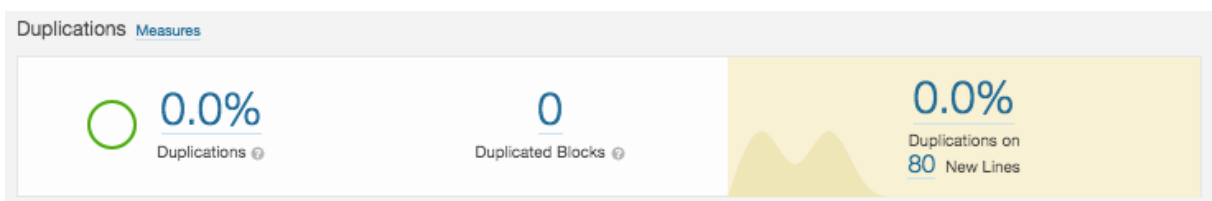


A significant improvement has been made in terms of code coverage as the source code is now at 96.8% of covered lines throughout the application.

5. Duplications

This last measure analyses the number of duplicated lines and blocks of code.

The duplications metric passes the quality gate with a duplication rate under 3%.

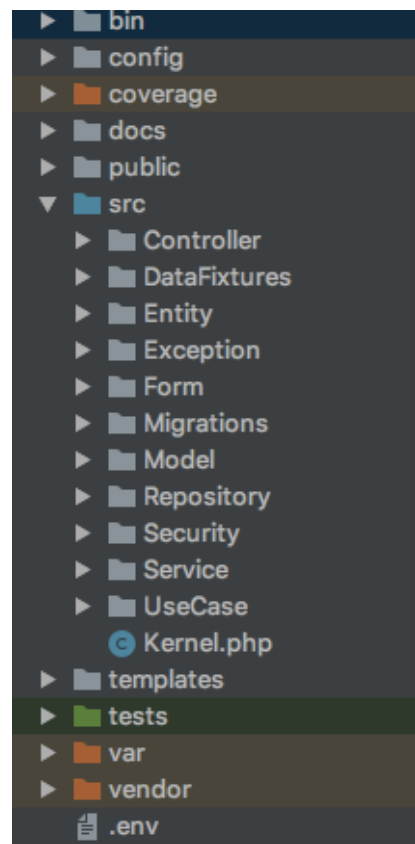
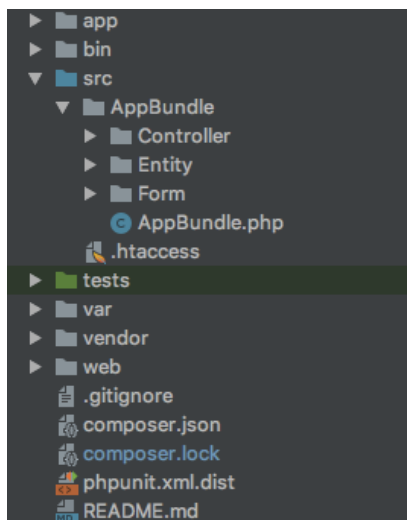


The initial analysis did not detect any duplicated code and no duplications were introduced following the refactoring and implementation of new features.

General Code Improvements

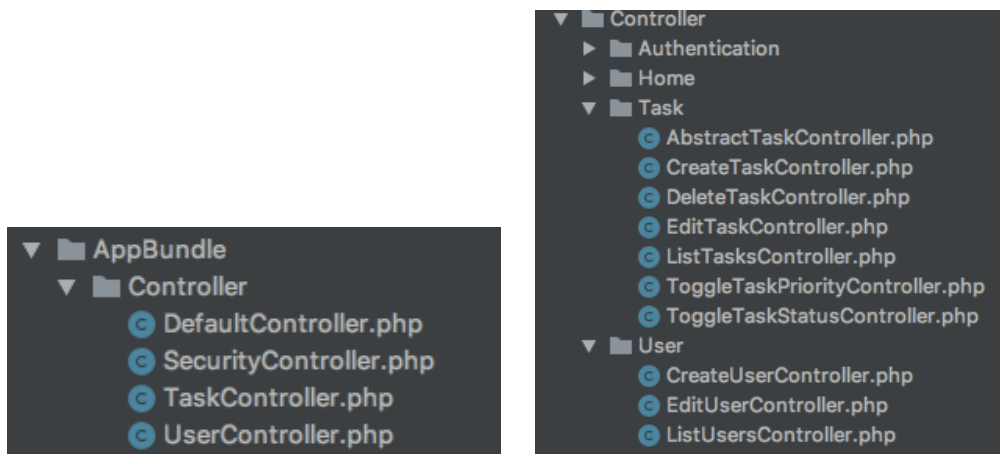
1. Symfony

In order to facilitate the code's maintainability, the application's Symfony version was updated from version 3.1, which is no longer maintained, to version 4.2. It will be necessary to update to version 4.3 but the main changes regarding Flex's new architecture were already put in place, thus facilitating any further upgrades.



2. Refactoring

Code quality cannot be fully measured by a sum of different metrics for it also refers to the developer's general experience while contributing to the project. As such, significant refactoring was performed on the MVP's code base in order to enhance the developer's experience while also contributing to improving the above-mentioned metrics.



The initial code base had all Controllers filed under the same namespace and each Controller contained multiple actions, which does not respect the Single Responsibility Principle, which dictates that a class should have only one reason to change.

```
/**
 * @Route("/tasks", name="task_list")
 */
public function listAction()
{
    return $this->render(
        view: 'task/list.html.twig',
        ['tasks' => $this->getDoctrine()->getRepository('AppBundle:Task')->findAll()]
    );
}

/**
 * @Route("/tasks/create", name="task_create")
 */
public function createAction(Request $request)
{
    $task = new Task();
    $form = $this->createForm(new TaskType(), $task);
```

These controllers were broken down into different specific classes and sorted under different namespaces according to their scope, thus resulting in smaller classes and a clearer architecture, which improves readability and navigation.

```
class ListTasksController extends AbstractTaskController
{
    /**
     * @Route("/tasks/{filter}", name="list_tasks", requirements=
     */
    public function list(GetTasks $getTasksUseCase, string $filter)
    {
        $tasks = $getTasksUseCase->execute( filters: $filter ? [$filter] : [] );
        return $this->render( view: 'task/list', ['tasks' => $tasks, ] );
    }
}
```

3. Use cases

In order to improve the platform's maintainability and reduce chances of code duplications, all the business logic contained inside the controllers was moved inside Entities or specific services called UseCases. The latter should reflect a user intention and encapsulate all the platform's application logic.

4. Security

The initial application's lacked security as it allowed visitor access to user information.

```
access_control:
  - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
  - { path: ^/users, roles: IS_AUTHENTICATED_ANONYMOUSLY }
  - { path: ^/, roles: ROLE_USER }
```

```
access_control:
  - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
  - { path: ^/register, roles: IS_AUTHENTICATED_ANONYMOUSLY }
  - { path: ^/admin, roles: ROLE_ADMIN }
  - { path: ^/, roles: ROLE_USER }
```

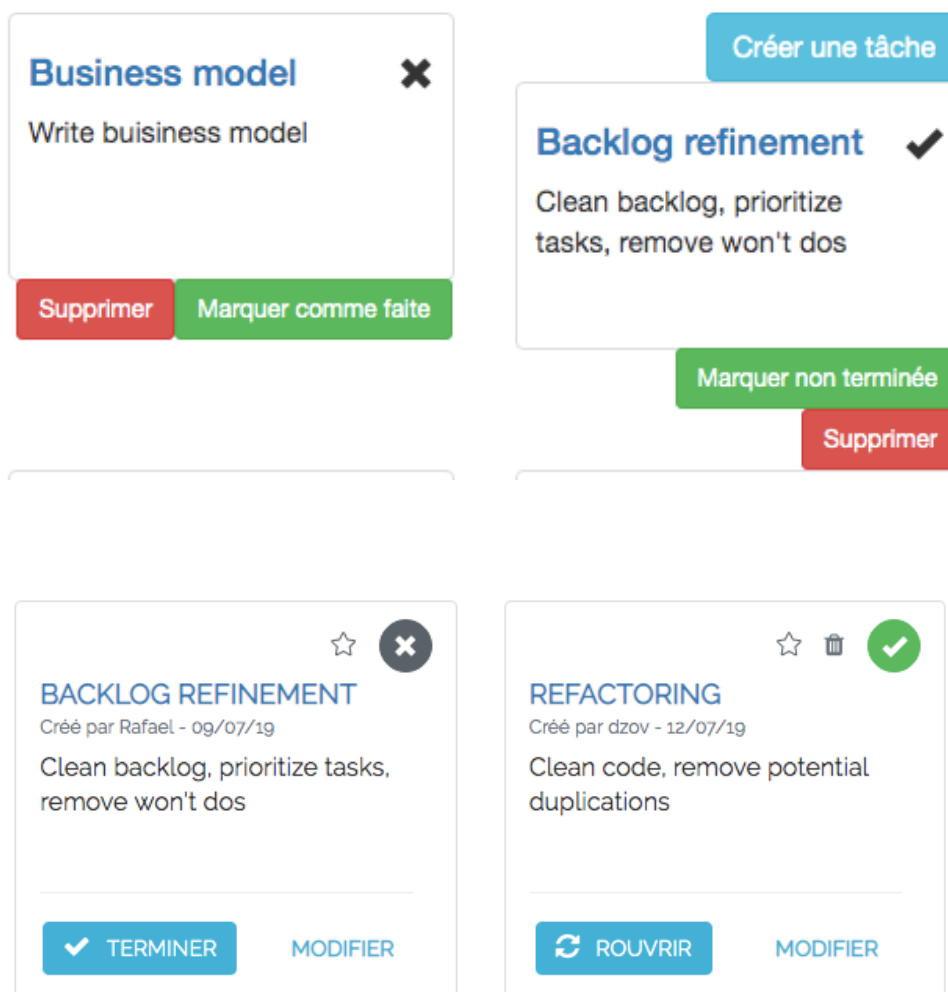
Access control was updated to properly protect the application and prevent access to users that should not be allowed access to specific parts of the application, such as the admin. Furthermore, a full authentication system using Guard was put in place. The documentation regarding this implementation can be found [here](#).

UX and UI Improvements

1. UI

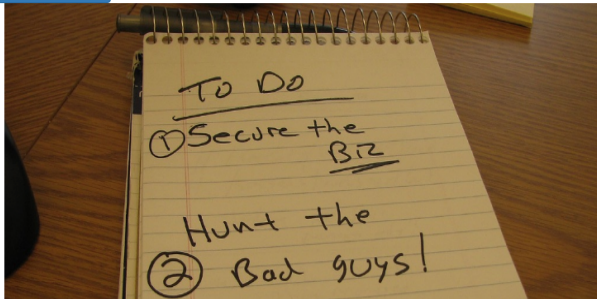
The initial state of the UI had much room for improvement. There was no consistency between pages or within the same page. The task cards were not aligned and the responsive was not properly put in place. The color scheme was not consistent and the overall aspect was not very appealing for the user.

A complete rebranding was performed and the platform's entire UI was reworked in order to improve the general aspect of the application. Bootstrap was used to facilitate the implementation of front-end enhancements.

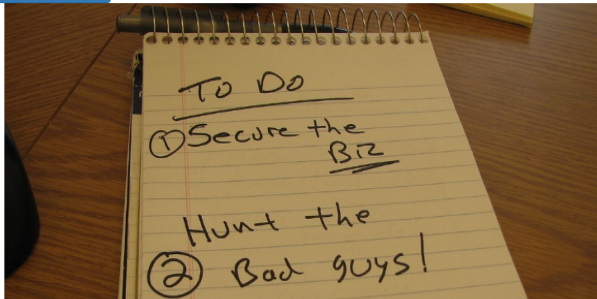


2. Templates

Templates were refactored and reorganized to ensure the interface's coherence across the whole platform.

To Do List app


Cr  er un utilisateur



Nom d'utilisateur : dzov
Mot de passe :
Se connecter

Nom d'utilisateur
dzov
Mot de passe
....

Forms used to be different throughout the platform





Nom d'utilisateur
dzov
Mot de passe :
....
SE CONNECTER

Nom d'utilisateur
Carlos
Mot de passe

A form theme was put in place to ensure that all forms look the same.

3. UX

The MVP did not leave a positive user experience. Success messages were the same regardless of the action performed by the user. The sign-up button had the same label as the button to create a new user. This can be confusing for the user who might be looking for a simple sign-up button.

As a result, significant work has been done to improve the overall application's user experience.

The success messages were adapted to the action being performed.

Créer un utilisateur

Se déconnecter

The form action labels were adapted to their context.

S'INSCRIRE

SE CONNECTER

A personalized dashboard was implemented to improve the user's overall experience with the platform.

3

TÂCHES CRÉÉES

1

TÂCHES EN COURS

2

TÂCHES TERMINÉES

+ AJOUTER UNE TÂCHE

VOIR TOUTES LES TÂCHES

Tâches prioritaires



SUPPORT REQUESTS

Créé par dzov - 12/07/19

Créer une nouvelle tâche

Consulter la liste des tâches à faire

Consulter la liste des tâches terminées

A filter system was put in place to facilitate research and navigation among all tasks, which was also destined to improve the initial user experience.

Liste des tâches

FILTRES ▼

Tâches terminées

RÉINITIALISER LES FILTRES








REFACTORING

Créé par dzov - 12/07/19

Clean code, remove potential duplications



 ROUVRIR
  MODIFIER

IMPLEMENT DASHBOARD

Créé par dzov - 07/07/19

Add user task information on dashboard. Handle responsive

 ROUVRIR
  MODIFIER

Conclusion

While the initial SonarCloud analysis did not reveal any major quality issues, improvements were made regarding the application's maintainability and a significant improvement was made regarding code coverage by raising the percentage from 0 to 96%. As such, Todo&Co's application has achieved an overall good code quality level, which is demonstrated by SonarCloud's quality gate badge. In order to maintain the current code quality, any further code addition should not decrease any of the above-mentioned metrics.

In addition, several measures were taken to improve the developer's experience while contributing to the project. A [contribution guide](#) was created in order to help future developers add new features while following the general guidelines and good practices put in place in an effort to maintain the same quality level throughout the application's evolution.

To further ensure the application's stability and quality a possible and recommended option would be to set up a continuous integration platform, thus preventing any code that does not pass the quality gate to be merged in production.

Blackfire Performance Report

The application's performance was assessed using [Blackfire](#). The existing routes in the initial project were also analyzed. Significant improvements were made in some cases as listed below.

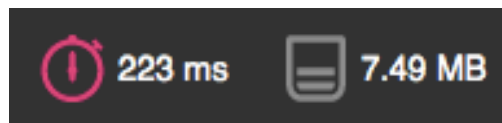
A general performance analysis was conducted for each route of the application after modifications were made to the initial code base. The results are listed below.

NB: These analyses were performed locally in production environment. The state of the machine and local server may thus impact the speed.

Login



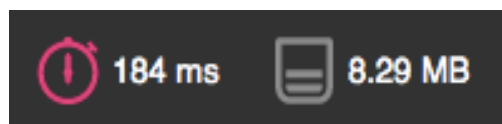
Logout



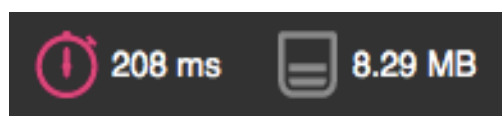
Dashboard



Create task



Delete task



Edit task

 202 ms  8.29 MB



List task

 185 ms  8.29 MB

Prioritize task

 248 ms  8.29 MB



Toggle task

 209 ms  8.29 MB

Create user

 216 ms  8.29 MB

Edit user

 198 ms  8.29 MB

List users

 422 ms  11.8 MB

 233 ms  8.29 MB

The above metrics show significant difference between the MVP's initial performance and the performance of the application after changes were made. Furthermore, the performance of the reworked application was analyzed prior to any optimization. The optimization of composer's autoloader helped to further reduce the overall response time as displayed below.

List users



Login



Conclusion

The overall changes made to Todo&Co's application helped reduce the response times of the initial application.

Despite the relatively long response time of the actual version of the application, which may be caused by the state of the machine and local server, there are no significant gaps between each route and thus no real area of concern regarding the application's performance.