Rapport Final SAE S4

Equipes:

Armand Simon

Brando Titouan

Gikapa Christian

Gribanova Sofia

Pelletier Alex

Sommaire

Introduction

Ce rapport a pour but de faire part du travail que nous avons fait durant tout le S4 pour optimiser notre application développée au S3.  
 Dans un premier temps, nous avons analysé l'application d'un autre groupe, ce qui nous a appris à utiliser Runner SonarQube pour analyser le code et déterminer quelles parties devaient être modifiées ou quels aspects de notre code devaient être changés.

Ensuite, nous avons lu le rapport d’audit qu’une autre équipe avait fait sur notre projet pour savoir quels étaient les points les plus importants, et nous avons commencé à optimiser, refactorer, recoder notre application. Tout ce travail a été fait en utilisant GitLab pour organiser nos tâches à effectuer.

Ce rapport est donc un bref compte-rendu de toutes les améliorations apportées à l’application que nous avons rendue en fin de S3, en montrant un avant/après, ainsi que les raisons de chaque modification.

I. Synthèse globale de l’optimisation produite

Pour améliorer notre application, nous avons décidé de nous concentrer sur 4 points qui nous paraissaient plus importants :

* Finir le code de la page ‘‘historique des points’’ pour que l’ensemble des fonctionnalités de l’application soient opérationnelles.
* Faire un refactoring de l’ensemble des Dao et du fichier EsporterManagerInitBDD.java
* Mettre en place un système de log pour toutes les erreurs et exceptions pour éviter de les afficher directement dans la console
* Limiter au maximum la présence de code smell pour avoir un code plus lisible

Nous allons dans ce rapport détaillé ces différents points petit à petit, en mettant en lumière l’intérêt du changement que nous avons apporté.

Dans l’application que nous avons rendue au S3, le système de classement des équipes n’étaient pas tout à fait complet, ce qui créait un manque dans les fonctionnalités demandées. **Titouan Brando** a donc repris ce que nous n’avions pas eu le temps de finir pour compléter cette page. Cela englobe la compréhension d’un code partiellement fait, avec l’aide de **Simon Armand** qui l’avait commencé, ainsi que d’en tirer ce qui marche pour finir la fonctionnalité. Il a donc refait les fichiers ModeleSaison.java et ModeleSaison.java pour que le code corresponde à la demande. Cela lui aura pris au total environ 7h de travail, et l’User Story n°8 Consulter l’historique de points est maintenant complétée. Une image contenant texte, Police, capture d’écran, blanc

Description générée automatiquement

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement Le second point est le refactoring des Dao, qui servait en particulier à éviter les répétitions pour rendre le code plus lisible, et plus facilement modifiable pour le futur. Pour cela, **Alex Pelletier** a d’abord commencé par supprimer tous les duplicatas d’appel des méthodes, avant de créer une classe permettant de définir des constants utiles dans toutes les Dao (le nom des colonnes des tables de la base de données pour faire les requêtes, évitant les valeurs en brut dans le code), et de finalement créer une classe SuperDao.java dont tous les fichiers Dao héritent, ce qui permet de faire une seule fois la connexion avec la BD, et de lier la classe Constants.java en évitant les répétitions. Cela lui aura pris environ 20h de travail.

Une image contenant texte, capture d’écran, menu, Police

Description générée automatiquement Ce point comprend aussi le refactoring du fichier EsporterManagerInitBDD.java. En effet, ce fichier était jusque-là une ‘‘Monster class’’, un fichier avec environ 400 lignes de code très répétitives, comme le montre l’image ci-contre :

Cela est effectivement normal pour un fichier initBDD, mais nous nous sommes dit qu’il valait mieux diviser chaque parties de ce fichier dans des sous-fichiers init pour permettre une meilleure lecture du code. **Sofia Gribanova** a donc décomposé cette ‘‘Monster class’’ en plusieurs petites classes faisant chacun l’initialisation d’une partie de la base de données. Par exemple pour l’image ci-contre, elle a créé un fichier nommé initEquipe.java, et fait juste appel à ce fichier dans EsporterManagerInitBDD.java. Cela lui aura pris environ 5h.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement Pendant tout notre développement du S3, nous avions utilisé des System.out.println() pour vérifier notre code, ce qui n’est pas très recommandé, les bonnes pratiques favorisant plutôt la mise en place d’un système de log. C’est **Sofia Gribanova** et **Simon Armand** qui se sont occupé de créer ce système et de remplacer toutes les occurrences de System.out.println() par un appel à java.util.logging.Logger, permettant de gérer des logs au lieu d’afficher nos tests directement dans la console. Cela leur aura pris environ 2h chacun.

La dernière grande partie de notre optimisation avait pour but de limiter les codes smell, pour rendre l’application plus lisible et limiter l’apparition de bugs. Comme nous pouvons le voir sur le graphique ci-dessous, nous avions au début environ 800 codes smell, nombre que nous avons réussi à diminuer petit à petit.Une image contenant texte, ligne, capture d’écran, diagramme

Description générée automatiquement

Pour améliorer cette partie, toute l’équipe a participé, en particulier **Titouan Brando** pour les codes smell de la vue, et **Christian Gikapa** pour le reste. Cela nous aura pris environ 20h au total.

II. Qualité de l’optimisation

Nous allons maintenant vous présenter quelques métriques permettant d’affirmer que notre code est maintenant plus propre.

Tout d’abord, pour ce qui est de la fiabilité de l’application, nous étions à la fin du S3 avec 98 problèmes détectés par sonarQube, dont 3 problèmes de haute importance, contre 16 problèmes détectés donc 0 de haute importance en cette fin de S4 :

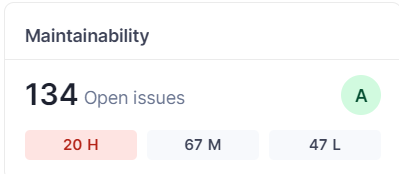
Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

*Avant*

*Après*

Pour ce qui est de la maintenabilité, nous sommes passé de 795 problèmes à 134 problèmes, et même s’il reste 20 problèmes de haute importance, il y en avait 151 au début, ce qui représente une énorme amélioration :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

*Avant*

*Après*

Une image contenant texte, capture d’écran, Police

Description générée automatiquement En outre, les risques de sécurité sont passés de 42 à 6, et ce ne sont que des éléments qu’il est conseillé de supprimer avant le rendu du code, mais dont nous avons encore besoin pour nos tests :

Une image contenant texte, Police, blanc, capture d’écran

Description générée automatiquement

*Avant*

*Après*

Malheureusement, nous n’avons pas enregistré les tests comme tel sur sonarQube, et il nous comptabilise donc une couverture de test de 0%. Nous n’avons donc pas de avant/après pour cette métrique, mais en regardant sur , notre couverture de test est aujourd’hui de …%.

III. Organisation interne dans le groupe

Nous avons mis en gras dans la première partie le nom des personnes de l’équipe qui ont participé à chaque partie des modifications présentées dans ce rapport. Pour rajouter les informations manquantes, nous nous devions de préciser :

* Christian Gikapa s’est occupé de déployer, configurer et tester sonarQube pour que nous puissions nous en servir de base des modifications à apporter
* Sofia Gribanova a également fait le refactoring de tous les tests de la Dao pour les passer sous JUnit
* Simon Armand a rédigé le rapport petit à petit en prenant compte du travail de chacun et en essayant d’être le plus clair possible

Conclusion