Rapport de Conception Logicielle

The Cookie Factory

Soulaiman Zabourdine
Valentin Campello
Titouan Le Mao
Julien N'Diaye
Leo Marache

Année 2020-2021

Sommaire

Introduction

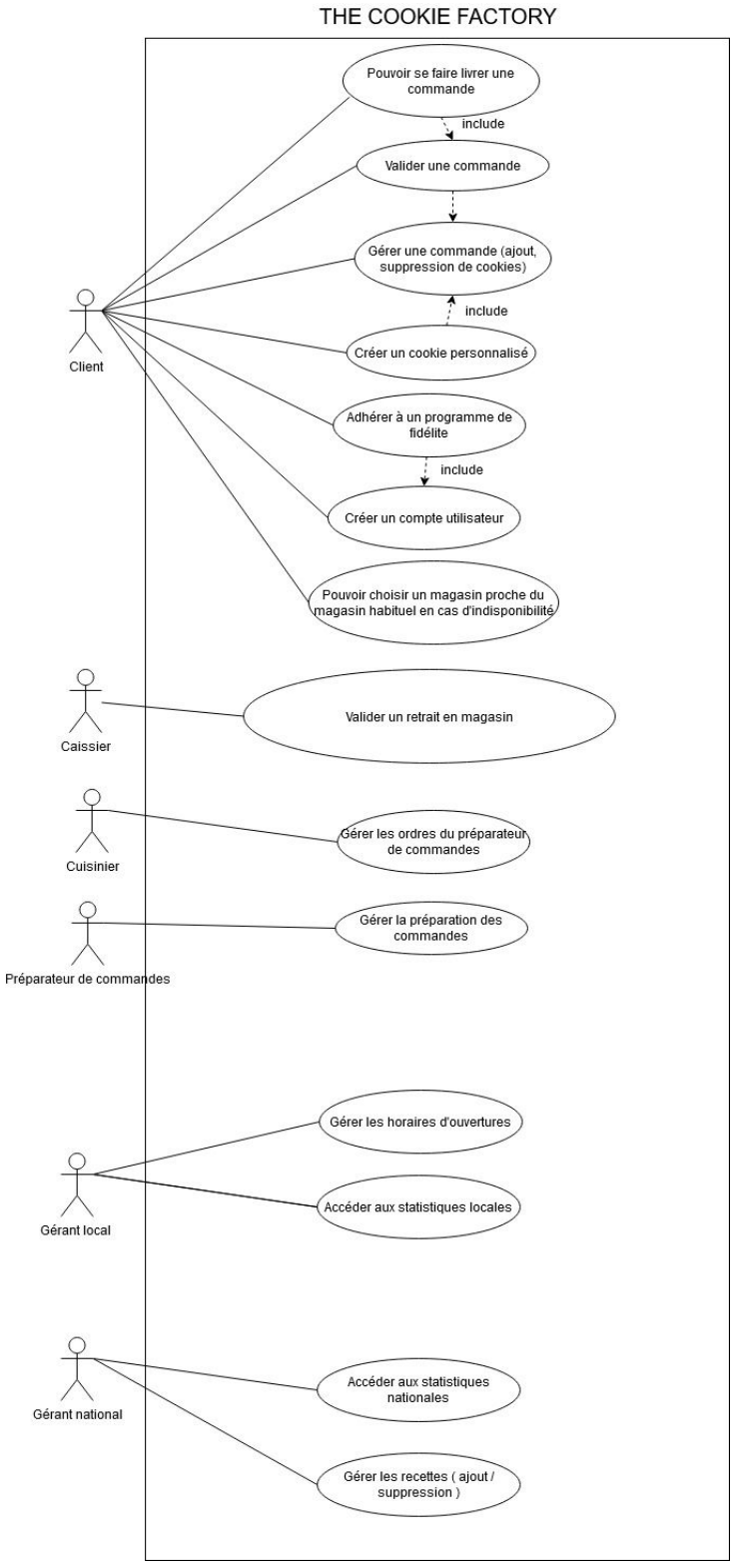
The Cookie Factory™ est une entreprise de fabrication de cookie leader aux USA depuis 1953. Devant faire face à de nouveaux défis, nous avons décidé de revoir intégralement son système de fonctionnement, de la chaîne de production à la livraison.

C'est dans ce contexte que nous vous révélons son nouveau système de fonctionnement, entièrement développé en Java.

La solution générique d'implémentation que nous avons créé réinvente complètement le mode de fonctionnement de *The Cookie Factory™*.
Cette solution permet de redéfinir une importante chaîne de production de cookies en un service innovant à la croisée du *e-Shop* et du *drive-in*.

À travers ce rapport, nous verrons le langage de modélisation unifié et les différents patrons de conceptions sur lesquels *The Cookie Factory™* repose. Nous verrons également comment nous avons fait évoluer la conception de notre application et comment se sont déroulées les différentes étapes de développement du projet.

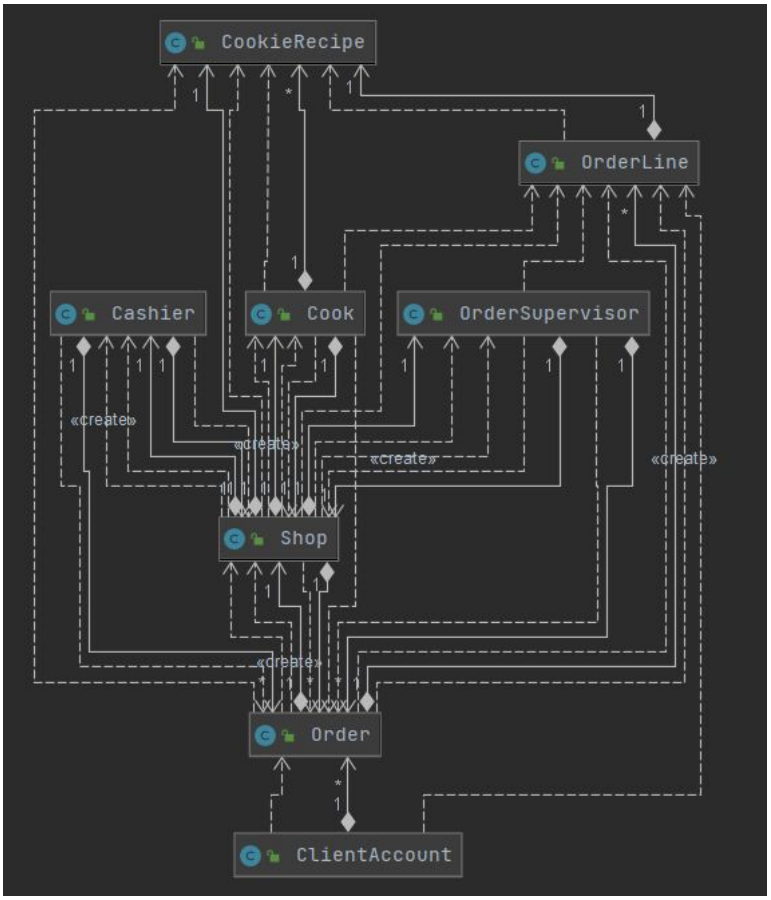
II - Diagramme de cas d’utilisation



III - Diagramme de classe

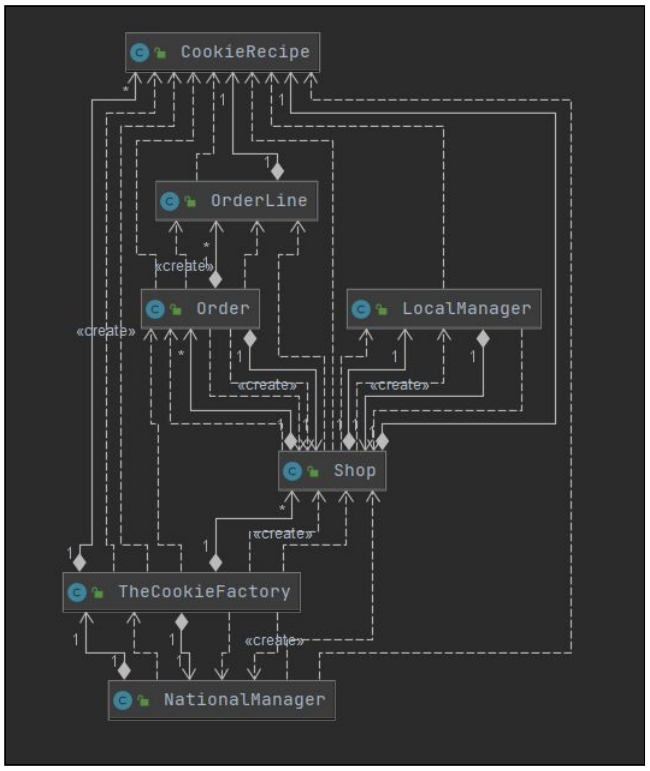
Le diagramme de classe que nous avons fait pour représenter la structure de *The Cookie Factory™* comporte 16 classes. À travers ces 16 classes, nous avons trouvé une façon juste d'augmenter la cohésion du code en créant des classes qui contiennent chacune des fonctionnalités bien définies.
Cela nous a permis de minimiser le couplage du code tout en gardant une structure efficace permettant une extensibilité, flexibilité, maintenabilité et réutilisabilité du code.

Pour mieux comprendre notre diagramme de classe, nous l'avons découpé en plusieurs parties. Chacune de ces parties permet d'avoir une représentation assez claire des classes qui entrent en jeu pour un scénario.
Le but, ici, n'est pas de réduire la taille du diagramme de classe mais de l'expliquer à travers le prisme de certains exemples.



Ici, nous voyons une sous-partie du diagramme de classe qui représente les principales classes interagissant directement entre elles lors d'une commande.

On peut noter que la classe shop est celle qui contient le plus de dépendances envers les autres classes.
Cela s'explique en partie par le fait que pour une commande, quasiment toutes les classes sont en relation avec un magasin. La classe Shop est donc le principal point principal d'accès aux données.



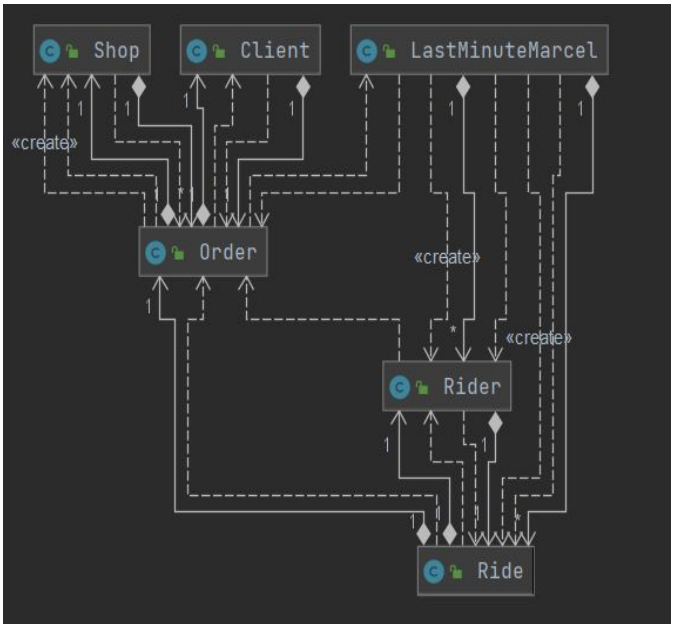
Dans cette sous-partie, nous avons un aperçu qui nous permet de voir les classes qui entrent en jeu lors de la création et l'utilisation des statistiques au sein de *The Cookie Factory™*.

Les 3 classes ayant le plus de dépendances sont les classes CookieRecipe, TheCookieFactory et Shop. C'est dans ces classes que la majorité des données sont récupérées pour faire les statistiques.

Il est à noter qu'une interface est utilisée pour la transformation de données en statistiques mais n'est pas représentée ici.

Cette dernière sous-partie nous permet d'avoir un aperçu des principales classes qui permettent de lier les deux services suivants :

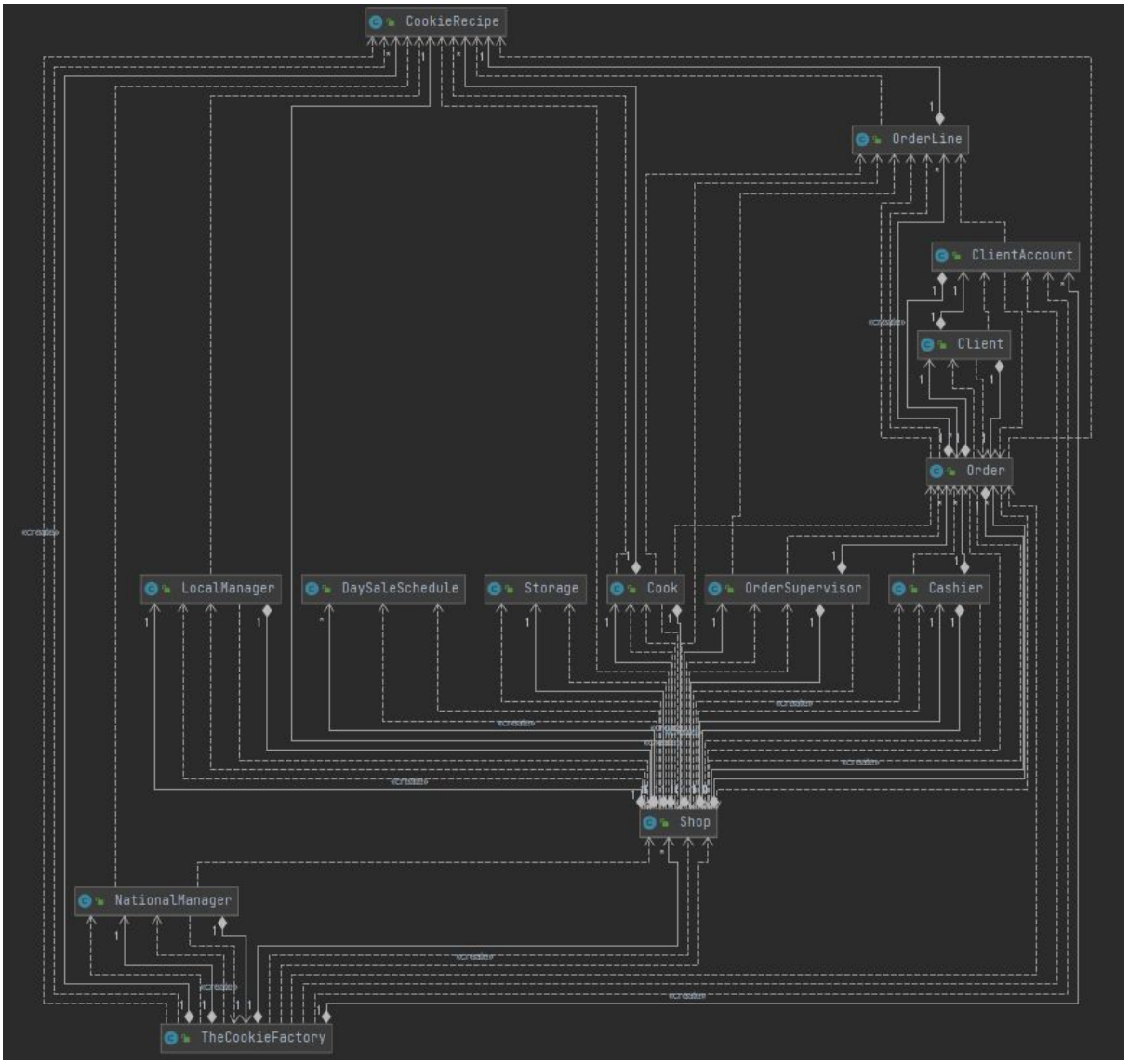
- *Cookies on Demand*
- *Marcel Eat*



Le seul et unique chemin de données qui met en relation ces deux services se fait via la classe Order.

Nous avons décidé de ne pas pouvoir relier le service Marcel Eat au service Cookies on Demand autrement que par l'intermédiaire de la classe Order. Marcel *Eat* étant un service externe à *Cookies on Demand* , il n'était pas juste de le mettre en relation avec par l'intermédiaire de plusieurs classes sachant que ce sont deux services qui sont construits avec deux diagrammes de classe différents. Cela a été un choix de conception qui nous a limités dans nos patrons de conception mais qui fut nécessaire pour que *The Cookie Factory™* garde une excellente extensibilité.

Voici le diagramme de classe complet que nous avons obtenu par rétro-ingénierie de notre projet grâce à l'environnement de développement Intellij.



Pendant la conception du système entier de *The Cookie Factory™*, nous avons créé des classes et nous les avons articulées entre elles de façon à diminuer le couplage du code tout en ayant une grande cohésion du code.
De plus, notre diagramme de classe a permis tout au long de notre projet de respecter les points suivants : la réutilisabilité, la maintenabilité, l'extensibilité et la flexibilité.
Cela nous a permis d'ajouter de plus en plus de nouvelles fonctionnalités sans changement au niveau du diagramme de classe.

IV - Patrons de conception

Nous avons dû faire beaucoup de choix quant aux patrons de conception que nous avons utilisés dans le but d'obtenir un système répondant le mieux aux approches que nous nous sommes faites d'un service à la croisée du *e-Shop* et du *drive-in*.

Les patrons de conception de conception nous ont aidés tout au long de notre projet.
De ceux que nous avons déterminés au début pour avoir un guide lors de l'écriture de notre code source à ceux utilisé après la programmation d'une partie de notre code pour relier les différents modules de code source déjà écrit, ils nous ont permis d'avoir un service qui répond au mieux possible au problème du *e-Shop* et du *drive-in*.

Chaque patron de conception que nous avons mis en place correspond à une approche différente qui ne répète pas les stratégies présentes dans d'autres patrons de conception. Cela nous a permis de résoudre chaque aspect des problèmes que nous avons rencontré d'une façon organisée. Ce n'est que par la combinaison de tous nos patrons de conception que nous sommes arrivés à notre solution finale qu'est *The Cookie Factory™*.

Un des patrons de conception que nous avons utilisé se nomme chain of responsibility.
Nous voulions que *The Cookie Factory™* soit capable d'assurer dans la mesure du possible la fabrication de cookies et la délivrance des commandes de cookies. En effet, certaines recettes peuvent être indisponibles dans certains magasins par faute d'ingrédients manquants. Ou bien, elles peuvent être stockées mais les horaires d'ouvertures du magasin ne permettent pas au client de pouvoir commander et récupérer ses cookies.

La solution à ce problème était de permettre à d'autres magasins de prendre le relai pour pouvoir assurer le bon fonctionnement de *The Cookie Factory™*. Ainsi, nous avons créé des méthodes qui permettent de savoir si une commande est réalisable par un magasin ou non. Si la commande n'est pas réalisable par un magasin, elle est transmise aux magasins les plus proches pour permettre la préparation et la délivrance de la commande. Nous avons donc créé dans chaque magasin une liste de magasins à proximité pour permettre la mise en place de cette solution.

Une conséquence s'est tout de suite dégagée, la meilleure gestion de nos données. En effet, il est désormais possible d'optimiser nos stocks d'ingrédients et de pouvoir

légèrement redéfinir les horaires d'ouvertures et de fermeture de nos magasins sans pénaliser le service Cookies on Demand.
Ensuite, nous nous sommes basés sur le patron de conception Flyweight.
Ici, le but était de minimiser la création excessive de données lorsque les commandes ne pouvaient pas être passées dans un magasin par manque d'ingrédients ou par faute d'erreurs intrinsèques à la commande (récupérer sa commande avant l'ouverture du magasin par exemple).

C'est pour cela que nous avons défini des objets (commandes) qui sont réutilisables par le même ou par d'autres magasins. Cela permet d'éviter la suppression puis la recréation d'une commande si cette dernière n'est pas conforme ou d'éviter la duplication de commandes entre les magasins. Il est d'un ordre logique que si une commande n'est pas réalisable par un magasin, elle doit pouvoir être transférée dans un autre magasin sans pour autant se dupliquer. La commande reste donc la même et peut être traitée par un autre magasin.

L'application de ce patron de conception nous a permis de réduire la masse de données circulant dans l'enseigne *The Cookie Factory™*. De plus, elle a permis une grande réutilisabilité et flexibilité des commandes.

Enfin, le dernier patron de conception que nous avons utilisé se nomme Mediator.
Avec l'utilisation des précédents patrons de conception, nous avons implémenté le service Cookies on Demand mais il fallait le coupler au service Marcel Eat. Étant deux services différents, il ne fallait pas que les classes de ses deux services puissent communiquer librement entre elles mais utiliser un médiateur dans une classe.

Ainsi est donc le médiateur object Order qui permet de contrôler la façon dont Marcel Eat interagit avec Cookies on Demand. Cette solution nous a permis de garder une bonne extensibilité de notre service Cookies on Demand qui est le service principal de l'enseigne *The Cookie Factory™*.

Une conséquence à tout cela fut la simplicité de couplage du service Cookies on Demand au service Marcel Eat tout en respectant le diagramme de classe fait au début. Comme certains patrons de conception peuvent être déterminés après la programmation d'une partie de notre code, une des seules façons de respecter l'architecture que nous avions initialement donnée à *The Cookie Factory™* au vue de la façon dont nous avons implémenté nos concepts était ce patron de conception.

Les bénéfices globaux des patrons de conception ont été la standardisation de certains concepts en modélisation, la réutilisation de solutions efficaces vis-à-vis

des problèmes qu'a rencontré *The Cookie Factory™* et la facilité de maintenance que nous aborderons dans notre point suivant.

V - Rétrospection de notre application

Avant de nous lancer dans le développement de notre application, nous avons créé un diagramme de classe, un diagramme de séquence et un diagramme de cas d'utilisation. Puis lorsque nous nous sommes lancés dans le développement de notre application, nous nous sommes appuyés sur les patrons de conceptions que nous avions définis à l'avance.

Tout au long du développement de notre application, nous avons testé chaque méthode pour s'assurer du bon fonctionnement de celle-ci.
Lorsqu'une user story était totalement implémentée, nous procédions systématiquement au test de cette dernière via les cucumbers tests.

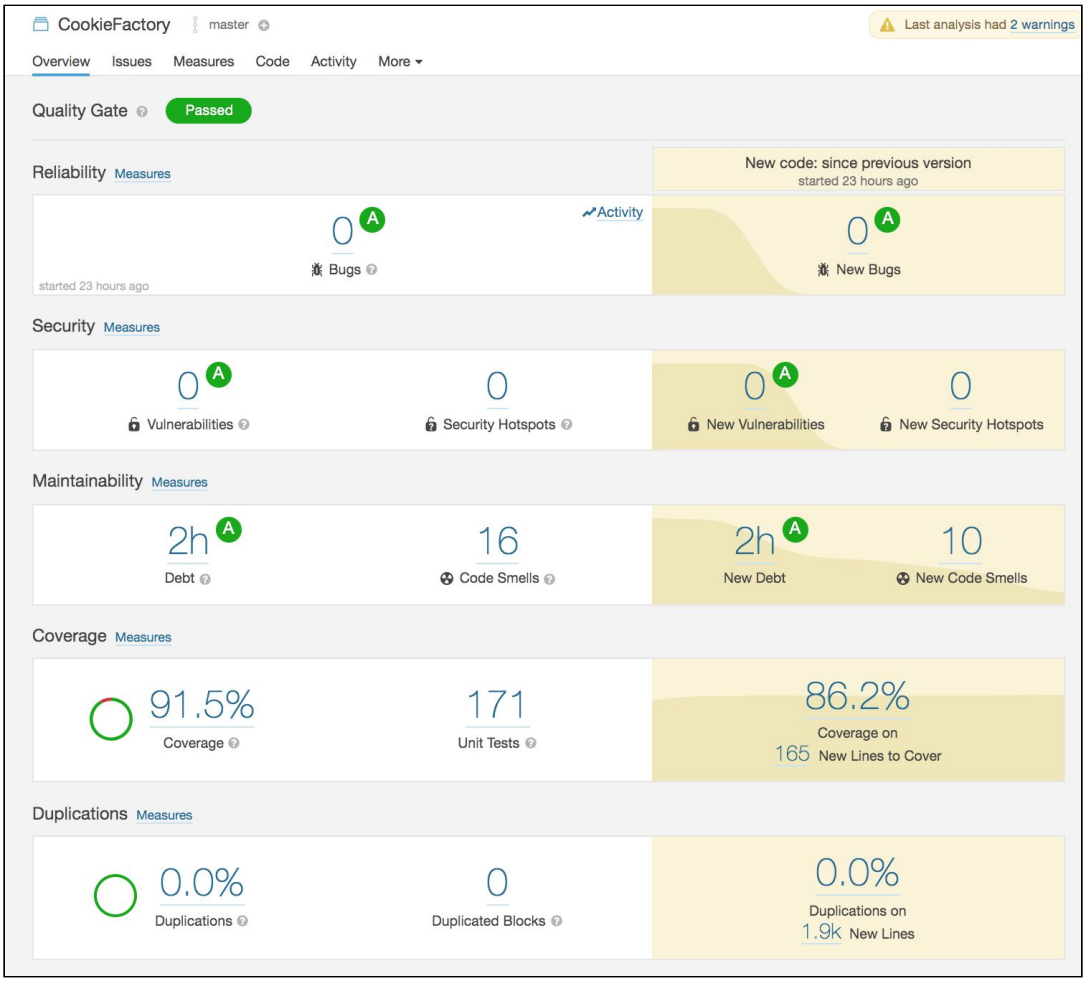
Même après avoir marqué comme close une user story, nous continuâmes de renforcer le test cucumber associé pour être sûr du bon fonctionnement de nos méthodes.

En codant de cette façon, les concepts que nous implémentions furent bien testés mais parfois ne respectaient pas spécialement le diagramme de classe que nous avions défini au début du projet. Nous avons donc dû faire du refactoring pour remettre notre application dans le droit chemin pour ne pas qu'elle perde son extensibilité au fur et à mesure de l'implémentation des fonctionnalités.

Les découpages que nous avons fait nous ont donc globalement permis de développer notre application tout en gardant son extensibilité, sa flexibilité, sa maintenabilité et sa réutilisabilité. Par contre, beaucoup de refactoring se sont imposés lors du développement de *The Cookie Factory™*, surement dû à nos découpages imparfaits qui n'était donc pas assez concis et précis.

À la fin du développement de notre application, nous avons utilisé la plate-forme open source SonarQube pour déterminer la qualité du code de notre projet.

La qualité de notre application était bonne même si elle comportait certains bugs, certaines vulnérabilités et quelques *"codes smell"*. Nous avons donc corrigé les bugs et nettoyé le code afin d'avoir un projet de meilleure qualité. En ce qui concerne le coverage de notre projet, il est de 91.5% et est au-dessus des 90% que nous nous étions fixés au début du projet.



Rapport Sonar de notre projet

Enfin, pour améliorer notre conception lors de nos prochains projets, il faudrait que l'on commit de moins gros bouts de code sur github, que l'on consulte Sonar tout au long de notre projet et non qu'à la fin. Il en va de même pour les users story, elles

devront être plus précis pour éviter de faire beaucoup de refactoring conséquents sur notre projet.

VI - Auto-évaluation

Pour ce travail, nous nous sommes tous investis selon nos moyens. C'est pourquoi nous donnons la répartition des points ci-dessous :

Valentin CAMPELLO: 100
Titouan LE MAO: 100
Léo MARACHE: 100
Julien N'DIAYE: 100
Soulaiman ZABOURDINE: 100

Nous avons travaillé tous ensemble pour établir les diagrammes de cas d'utilisation et de classe. En ce qui concerne le développement de notre application, nous avons tous travaillé dessus régulièrement et nous nous sommes entraidés pour surmonter les difficultés que nous avons rencontré. Enfin, pour le rapport nous avons intégralement travaillé à distance par visioconférence.