RAPPORT FINAL DU PROJET FIL ROUGE : EQUIPE F

1- Nom des étudiants et rôles (Equipe F) : 2
2- Périmètre fonctionnel : 3
2.1- Hypothèses de travail: 3
2.2- Extensions choisies et éléments spécifiques: 3
2.3- Points non implémentés relativement la spécification et des extensions choisies: 4
3- Conception : 5
3.1- Le glossaire: 5
3.2- Le diagramme de cas d'utilisation et User Stories: 5
3.3- Le diagramme de classes: 7
3.4- Un diagramme de séquence: 8
4- Design Patterns appliqués ou pas 9
4.1- Design Patterns en détail: 9
4.2- Autres Design Patterns: 10
5- Qualité des codes et gestion de projets: 12
6- Rétrospective et auto-évaluation: 14
6.1-Travail effectué selon le rôle: 14
6.2- Bilan du fonctionnement de l'équipe: 15
6.3- Auto-évaluation: 15

1- Nom des étudiants et rôles (Equipe F) :

Pour réaliser le projet de conception logiciel dans le cadre de notre cours de quatrième année à l'école Polytech Nice Sophia, nous avons créé notre équipe UberTech qui est constituée de 4 membres qui sont :

- **BEUREL Simon** : Responsable de la partie OPS du projet.
- **MAUROIS Quentin** : Product Owner
- **AZIKI Tarik** : Software Architect
- **FROMENT Lorenzo** : Quality Assurance Engineer

2- Périmètre fonctionnel :

2.1- Hypothèses de travail:

- **Toutes les livraisons se passent correctement:**
    - Nous sommes partis du principe que la livraison s'effectue sans problèmes. Le livreur livre la commande au bon endroit, dans un délai raisonnable, en étant courtois et sans avoir endommagé la commande. Grâce à cette hypothèse, nous enlevons la possibilité au client de signaler un livreur.
- **Avis sur le restaurant/livreur:**
    - Nous avons choisi de ne pas traiter le moment où l'utilisateur donne son avis et/ou note un restaurant ainsi qu'un livreur. Nous partons du principe que l'action est effectuée en « arrière-plan » ce qui permet donc aux administrateurs du campus de voir des statistiques et prendre des décisions.
- **Pas de notion de temps de préparation/livraison de la commande:**
    - Dans notre projet il n'y a pas de notion de temps de préparation d'une commande, les employés du restaurant confirment la préparation lorsqu'ils ont terminé. Cela se passe de la même manière pour la livraison, le client n'a pas de temps d'estimation de livraison.
- **Le paiement s'effectue par carte et avec succès:**
    - Nous avons fait l'hypothèse que les clients ne pouvaient payer que par carte et qu'il n'y avait jamais de problème de paiement, donc la fonction appelée pour payer ne fait rien à part dire que le paiement s'est bien effectué.
- **Le client choisit son livreur :**
    - Contrairement à ce que peuvent proposer les différentes applications de livraison de nourriture actuelles, notre application impose au client de choisir le livreur qui lui amènera sa commande. Pour cela, après avoir payé sa commande, on lui propose une liste de livreurs disponibles.

2.2- Extensions choisies et éléments spécifiques:

Les extensions qui ont été choisies pour le projet sont:

- La possibilité pour un livreur de signaler un usager. Effectivement, cela peut être agaçant pour un livreur de devoir attendre un client alors qu'il pourrait être en train de livrer d'autres clients. C'est pour cela qu'il a été ajouté la possibilité pour un livreur de signaler un usager en retard. Un client qui se voit être en retard perd 1 point, en sachant que tout client a 3 points de base. Lorsqu'il atteint un score de 0, il est banni et ne peut plus commander. Si un client fait 3 commandes sans être en retard, il récupère 1 point.

- Les restaurateurs ont la possibilité de proposer des réductions aux clients fidèles. Ils peuvent définir un nombre de commandes successives à effectuer pour bénéficier d'une réduction, en spécifiant le pourcentage de réduction ainsi que la durée de cette offre.

- Le but de cette extension (obligatoire) est de pouvoir proposer de nouveaux “types” de commandes au sein de notre application UberTech. Avec la V1, nous avions les commandes simples, les commandes groupées, et maintenant nous voulons ajouter des commandes comme des AfterWorks, ou alors des Buffets. Il est également prévu que cette extension puisse permettre d'ajouter d’autres types de commandes ultérieurement.

2.3- Points non implémentés relativement la spécification et des extensions choisies:

[D2] L’utilisateur est tenu au courant lorsque sa commande change d’état et peut consulter l’id de la commande, le jour de livraison et sa localisation dans son historique. Cependant, il ne reçoit pas de notification contenant toutes ces informations.

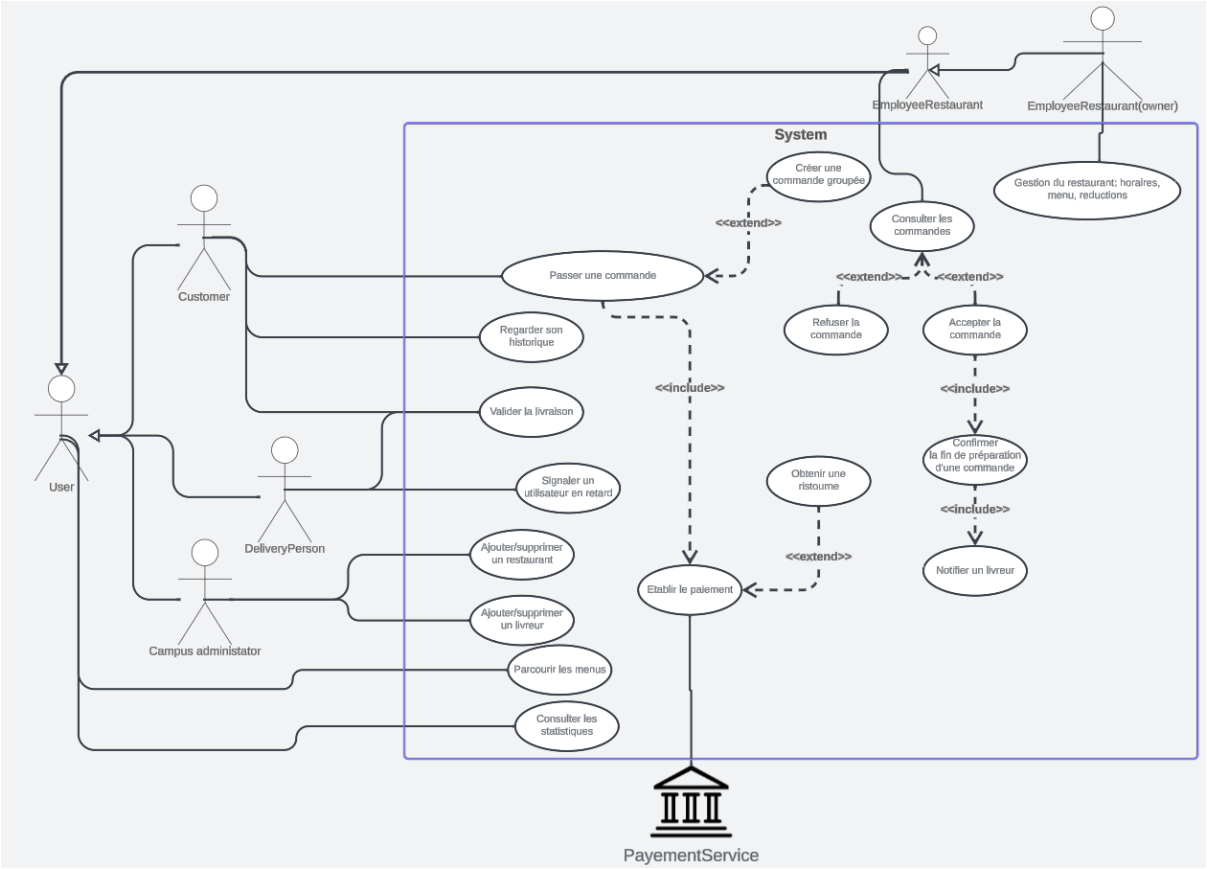
[S1] Concernant ce point, il nous manque deux spécifications :
- l’efficacité de la livraison
- les informations sur les volumes de commande

3- Conception :

3.1- Le glossaire:

Statistics: Ce sont des données qui sont propres à chaque utilisateur (Livreur, client, restaurateur). Ces données comprennent des caractéristiques différentes selon le type d'utilisateur. Livreur: total de commandes prises en charges, prix total des commandes prises en compte, prix moyen des commandes prises en comptes etc …
Capacity: C'est la capacité totale restante de commandes qu'un restaurant peut prendre en charge.
User Type: Ce sont les différents types d'utilisateurs: un élève, un personne faisant partie du campus, une personne du staff.
History: C'est l'historique d'un utilisateur où il peut retrouver les commandes qu'il a effectuées ainsi que leurs détails.

3.2- Le diagramme de cas d'utilisation et User Stories:



- Signaler un utilisateur en retard:
  - Url: https://github.com/PNS-Conception/ste-23-24-equipe-f-ubertech/issues/36
  - Issue #36 : "[US] Report a late user by Delivery Personnel #36"
  - **As a** delivery person, **I want to** report users who are late for their orders, **so that I** can ensure effective monitoring and management of deliveries.

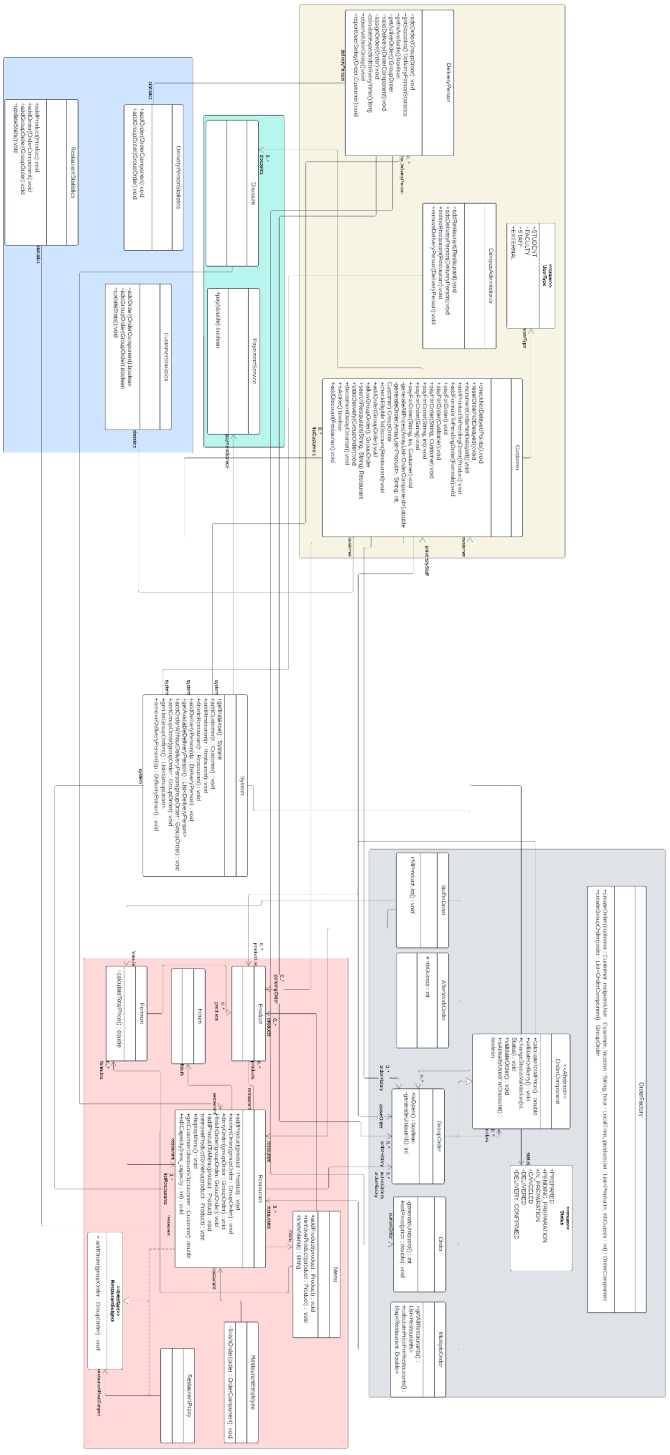
- Obtenir une ristourne:
    - Url: https://github.com/PNS-Conception/ste-23-24-equipe-f-ubertech/issues/44
    - Issue #44: “[US] [EX2] Restaurant discount #44”
    - **As a** restaurant owner **I want to** allow customers to get a discount when they purchase lots of orders and to modify the amount of the discount **so that** I can have more loyal customers.
    - Critère d’acceptation:
        - Un client ne peut pas cumuler les réductions
        - Seul la durée est augmentée lorsqu’il obtient à nouveau une réduction
        - Le restaurateur doit pouvoir choisir le montant de commandes à effectuer pour avoir une réduction, le montant de la réduction et sa durée
- Passer une commande
    - Url: https://github.com/PNS-Conception/ste-23-24-equipe-f-ubertech/issues/79
    - Issue #79 “[US] Multiple Order”
    - **As a** customer, **I want to** put an order with as many restaurants as I want **so that** I only have one order to check and to pay for

      https://github.com/PNS-Conception/ste-23-24-equipe-f-ubertech/issues/81
    - Issue #81 “[US] Ordering Buffet for a University Event #81”
    - **As a** member of the university staff, **I want to** be able to place a buffet order from a restaurant for a specific university event **so that** the buffet is delivered to a designated user for that event.

      https://github.com/PNS-Conception/ste-23-24-equipe-f-ubertech/issues/82
    - Issue #82 “[US] After Work Orders”
    - **As a** customer, **I want to** put an order for a specific type of product and for everyone that takes part in the after work **so that** I don't have to create an order for each and every participant

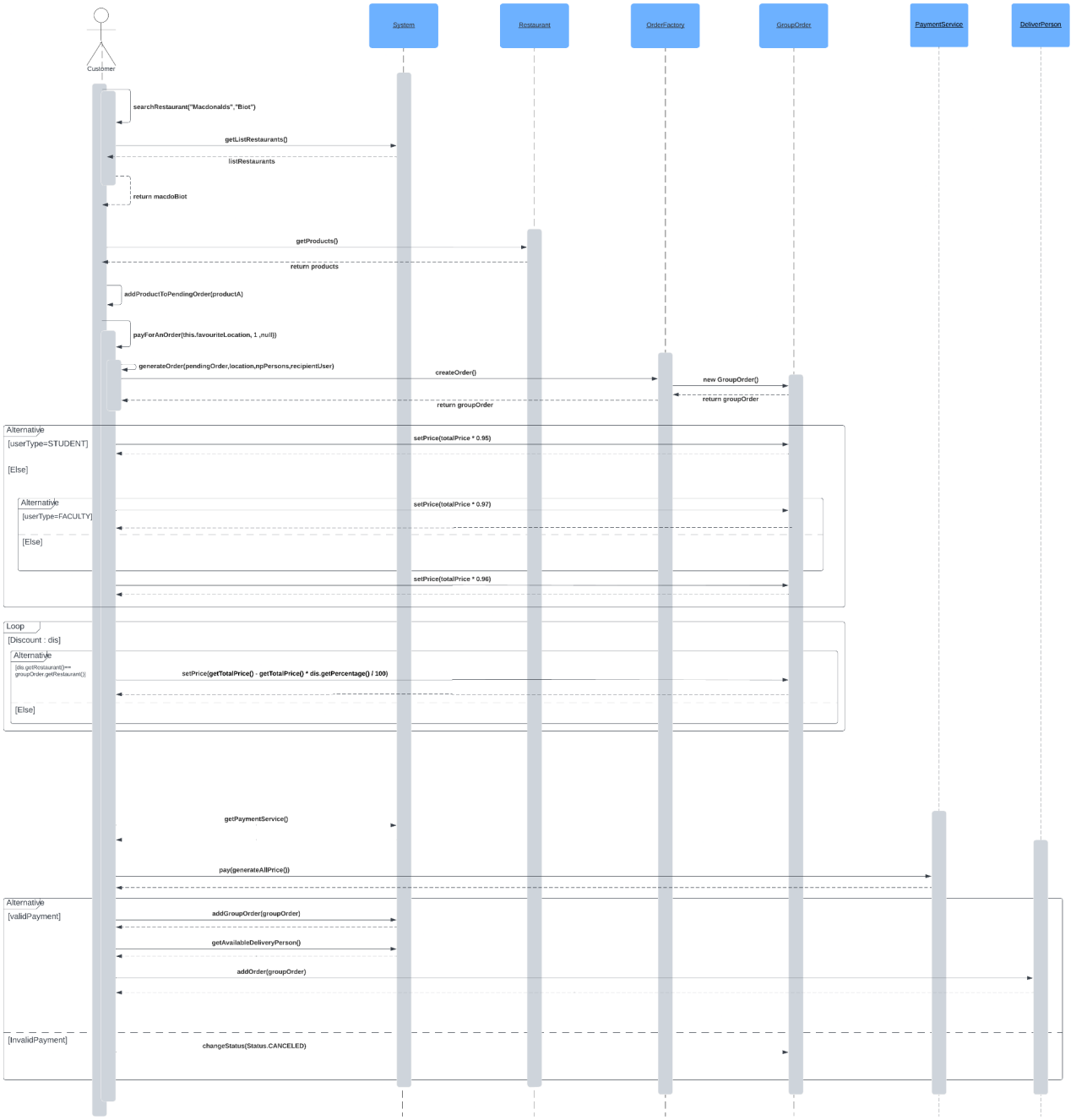
3.3- Le diagramme de classes:

Voici notre diagramme de classes. Cependant, si vous n’arrivez pas à le visualiser correctement, vous pouvez utiliser ce lien pour pouvoir l’ouvrir depuis votre navigateur pour pouvoir mieux distinguer les détails (besoin d’un compte comme un compte google pour visualiser)
:https://lucid.app/lucidchart/263c8ed9-79cd-45d0-9f63-c9585df0be50/edit?viewport_loc=-5025%2C-3075%2C10401%2C5019%2C0_0&invitationId=inv_a76db368-fb99-4a32-beef-04bf702b9c9b



3.4- Un diagramme de séquence:

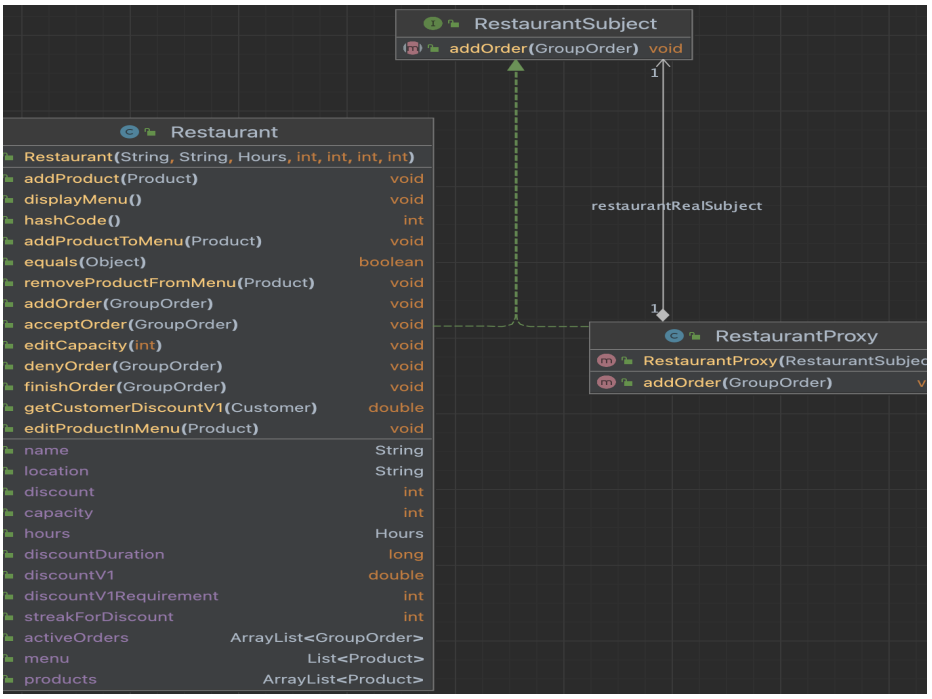
Voici notre diagramme de séquence. Comme pour le diagramme de classes présenté en amont, vous pouvez utiliser ce lien pour l'ouvrir depuis votre navigateur et mieux le visualiser (besoin d'un compte comme un compte google pour visualiser) : https://lucid.app/lucidchart/7c679c30-7378-4614-b2d7-1bfb7de93fa2/edit?viewport_loc=-1582%2C901%2C4992%2C2409%2C0_0&invitationId=inv_49ab7902-de7a-4681-9500-629dfe92253b



4- Design Patterns appliqués ou pas

4.1- Design Patterns en détail:

Le proxy: Nous avons choisi d'implémenter le design pattern Proxy pour une gestion efficace de la validation des commandes avant qu'elles ne soient traitées par un restaurant. Le choix du Proxy s'est avéré judicieux, car il agit comme un "gardien" entre le système de gestion des commandes et les restaurants réels. Avant de transmettre une nouvelle commande à un restaurant, le Proxy effectue une vérification, s'assurant que le restaurant est prêt à accepter de nouvelles commandes. Cette approche permet un contrôle d'accès précis, évitant tout impact direct sur les classes de restaurant existantes.



On peut voir sur l'image la représentation du proxy en diagramme de classe.

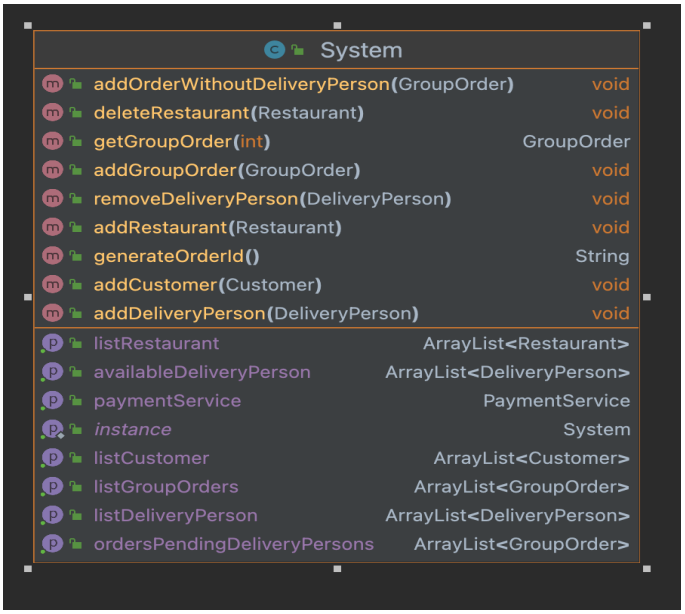
L'interface RestaurantSubject agit comme un contrat standardisé que tous les restaurants doivent respecter. Elle définit une norme en spécifiant que chaque restaurant doit être capable d'ajouter une commande à sa liste. Cela assure que tous les restaurants possèdent une méthode d'ajout de commande.

La classe RestaurantProxy est comme un assistant du restaurant. Son travail est de s'assurer que le restaurant n'est pas trop occupé avant d'accepter une nouvelle commande. Voici comment nous l'avons implémenté.

- Il regarde l'heure à laquelle la commande arrive et détermine si le restaurant peut la prendre.
- Il regarde combien de commandes le restaurant a déjà à cet instant précis

- Si le restaurant peut prendre plus de commandes, alors le Proxy dit au restaurant réel (le vrai restaurant) d'ajouter la commande.

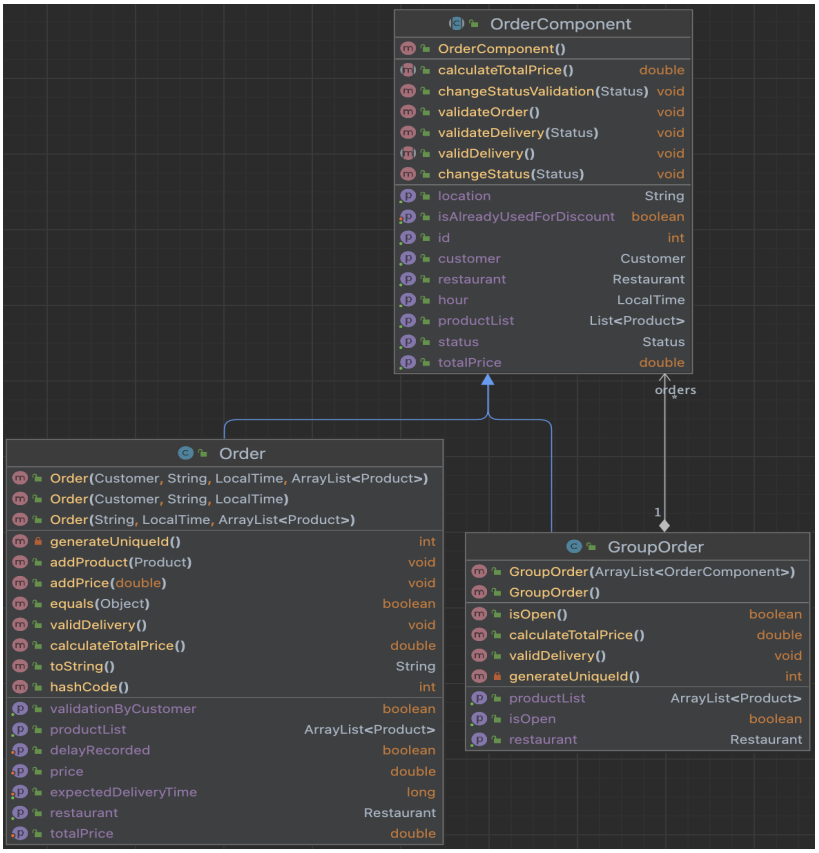
Le singleton: Le pattern Singleton a été adopté naturellement dans notre projet à travers l'objet “System” de la classe portant le même nom, lequel doit exister en une seule instance. Ce design pattern assure qu'une unique instance du système est créée pour garantir sa singularité et sa cohérence.



Ce singleton est simple se traduit par la déclaration d'une variable statique d’instance qui stocke l'unique instance de la classe. Le constructeur de System est défini en tant que privé, ce qui signifie qu'il ne peut être appelé que de l'intérieur de la classe elle-même. La méthode statique getInstance() est mise en place pour permettre l'accès à l'instance unique. Dans cette méthode, nous vérifions si l'instance existe déjà. Si c'est le cas, nous retournons l'instance existante. Sinon, nous créons une nouvelle instance en utilisant le constructeur privé. Cette approche garantit que, peu importe la partie de notre appli qui demande l'instance de System, elle obtiendra toujours la même instance qui est unique, cela évite de dupliquer inutilement l’objet System et assure une gestion cohérente du système.

4.2- Autres Design Patterns:

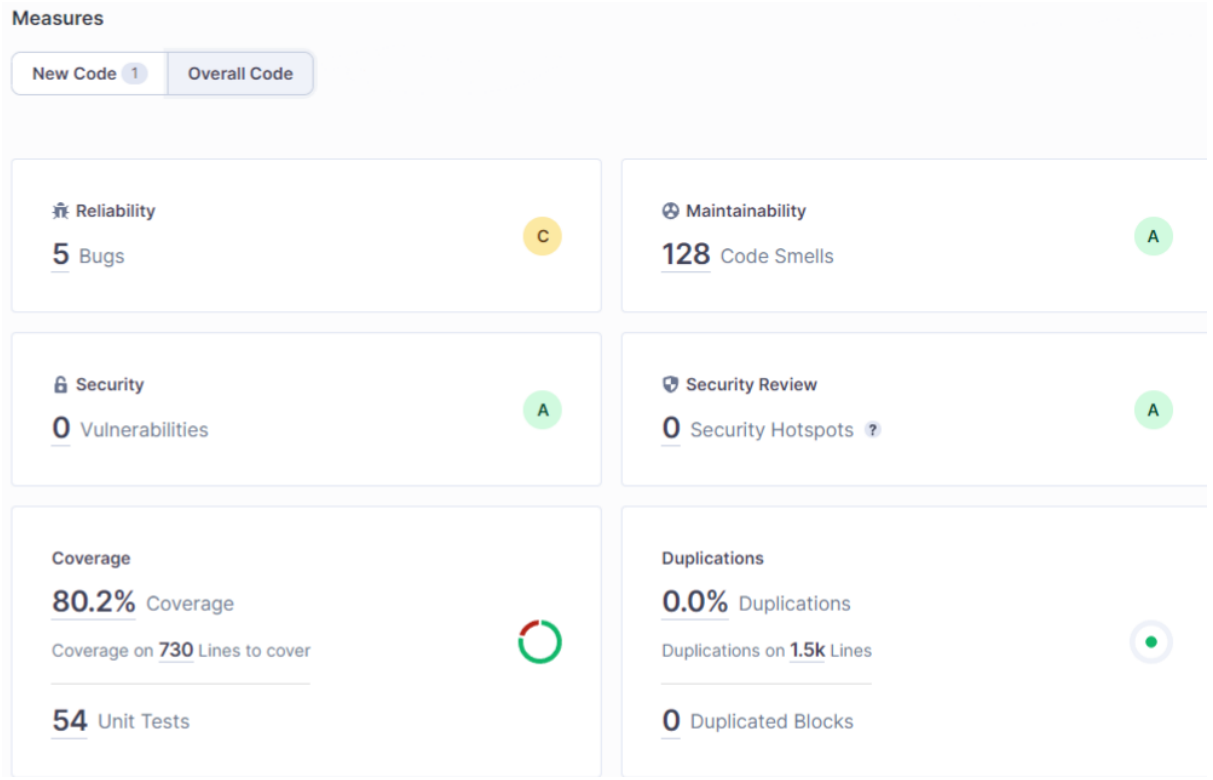
Dans la mise en place du patron Composite à travers l’ajout des Commandes (Order) au sein d'une Commande groupé (GroupOrder), cet ajout est effectué directement via le constructeur de la classe GroupOrder. Cette méthode ne s'aligne pas parfaitement avec le modèle du patron Composite tel qu’il a été étudié dans le cours.



Malgré notre volonté de procéder à un refactoring pour corriger ce patron, des contraintes de temps ont limité notre capacité à le réaliser. Nous avions néanmoins envisagé d'introduire une méthode "addOrder()" dans la classe GroupOrder, qui fait en sorte d'ajouter des Commandes dans la list<OrderComponent> au sein de GroupOrder. Cela permettra une structure plus dynamique et flexible, alignée sur le modèle de conception attendu.

5- Qualité des codes et gestion de projets:

- Tous les tests que nous avons implémentés dans notre projet sont des tests d'intégration. Ceux-ci sont essentiels pour assurer la qualité et la fiabilité de notre application du point de vue de l'utilisateur. Ils garantissent le bon fonctionnement entre les composants, permettent de simuler les scénarios que les utilisateurs vont rencontrer durant l'utilisation de notre application et nous permettent de vérifier le bon fonctionnement de ces scénarios à chaque ajout de code.



Le taux de coverage de 80.2% est un taux de couverture décent et suggère que ⅘ du code est couvert par nos 54 tests. Cependant cela ne garantit pas la qualité de nos tests.

- Nous pensons personnellement que la qualité de nos tests est bonne, cela grâce à la méthode de développement utilisée durant ce projet. Le fait d'avoir réfléchi à l'implémentation des tests au travers de scénarios gherkins nous a permis de bien penser aux parties de code que nous allions implémenter juste après à partir de ces tests. Nous savions donc quel scénario l'utilisateur allait rencontrer, ce que nous voulions tester et donc ce que nous devions implémenter.
- Pour la gestion du projet, notre DevOps avait mis en place une vérification des tests avant chaque merge sur une branche pour empêcher le fait qu'un test ne passant pas soit merge sur une branche. Pour ce qui est de la branche Main, celle-ci avait une règle supplémentaire: au moins 2 membres devaient approuver le merge avant que celui-ci soit possible. Cela poussait tout monde à rester à jour sur l'avancée des autres.
  Nos branches avaient une structure simple, nous avions la branche Main servant de présentation client, donc la branche la plus stable. La branche "develop" où chaque membre faisait un merge de sa propre branche de développement lorsque sa

fonctionnalité était terminée. Cette branche était donc stable mais était vérifiée par 3 membres (comme expliqué plus haut) avant d'être merge avec Main.
Pour finir, chaque membre devait créer une branche pour la fonctionnalité qu'il développait et se devait de lier ses commits aux issues ouvertes sur Github.
De plus, à chaque commits/pull-request, l'utilisation de GithubAction était déclenchée pour vérifier que le projet compilait parfaitement et arrivait à exécuter tous les tests.

6- Rétrospective et auto-évaluation:

6.1-Travail effectué selon le rôle:

- **Product Owner (MAUROIS Quentin):** En tant que product owner, mon rôle principal a été de définir les objectifs pratiques du projet. J'ai pris en charge la sélection des caractéristiques du produit, en m'assurant qu'elles étaient bien en accord avec le cahier des charges. J'ai aussi été responsable de rassembler les documents d'analyse et de conception créés par l'équipe, afin que tout le monde comprenne bien ce que nous devions faire et ce principalement au travers de l'application discord.

  Mon rôle me demandait également de suivre assidûment la progression du projet pour savoir à tout moment notre état de progression. Cela nous a permis de nous concentrer sur les obstacles potentiels et d'en parler ensemble pour trouver des stratégies pour les dépasser.

  Cependant, il est à noter que beaucoup de décisions ont été prises par l'ensemble de l'équipe sans prendre en compte spécifiquement les rôles et ce pour tous les points du projet.

- **Quality Assurance Engineer (FROMENT Lorenzo):** En tant qu'ingénieur en assurance qualité, j'ai été impliqué tout au long du projet, en particulier au début, pour soutenir mes collègues dans la rédaction et la mise en œuvre des histoires utilisateur. La transition vers la méthode de développement d'application avec Cucumber n'était pas évidente et semblait complexe au début. C'est pourquoi certaines erreurs étaient fréquentes, comme l'utilisation du pronom "I" dans les scénarios, même si cela avait été déconseillé lors des cours.

```
Given a Restaurant called "Le petit gueuleton"
When I specify my production's capacity
Then I can't accept more order than my production's capacity
```

```
Given I am logged in
When I delete a post on the blog
Then I should see a successful deleted message
```
 # BAD EXAMPLE! Do not copy.

En tant qu'ingénieur en assurance qualité, ma responsabilité principale était de surveiller et d'augmenter en permanence le pourcentage de couverture du projet par des tests d'intégration automatisés. Mon objectif était d'assurer une couverture des fonctionnalités tout en prévenant toute baisse lors des modifications du code. Cette approche a renforcé la stabilité du projet tout en garantissant une qualité constante du code tout au long du processus de développement.

- **Software Architect (AZIKI Tarik):** En tant qu'architecte du projet, ma mission principale était de superviser mon équipe lors de l'élaboration des diagrammes liés au cahier de charge et de s'assurer que ceux-ci respectent les règles de conception logicielle et s'alignent parfaitement à notre besoin. J'ai également surveillé l'implémentation lors de la phase de développement, garantissant ainsi que le code soit en conformité avec notre besoin tout en respectant les normes architecturales.

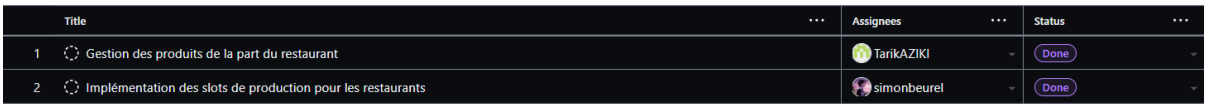
J’ai également participé au refactoring de certaines parties identifiées comme des axes d'amélioration. L’objectif est de réduire le couplage entre les classes du projet, et de renforcer la cohésion entre les fonctions au sein d’une même classe, garantissant ainsi de bonnes cohérence et scalabilité. Ceci ayant pour but de faciliter l’implémentation des nouvelles extensions et d’assurer une maintenance future optimale, en intégrant stratégiquement des patrons de conception pour structurer efficacement le code.

- **Responsable OPS (BEUREL Simon):** En tant que responsable de la partie OPS de ce projet, mon but était de faire en sorte que toute l’équipe travaille de manière cohérente et homogène sur le projet. Une de mes premières tâches était d’établir une stratégie de “branching” cohérente. Ensuite, une de mes deuxièmes tâches était de faire en sorte de modifier le fichier “workflow” qui correspond aux actions réalisées par GithubAction pour faire en sorte qu’à chaque push/pull-request sur le projet ce dernier se déclenche, pour vérifier si le projet compile bien et que tous les tests passent. Ma dernière tâche a été d’implémenter une analyse SonarQube sur notre projet. Pour cette dernière tâche, j’ai dû implémenter un script effectuant cette analyse toutes les 30 minutes sur un VPS. Malgré ce travail, je reste déçu car je n’ai pas pu effectuer un bon suivi de projet grâce aux Milestones sur Github.

6.2- Bilan du fonctionnement de l’équipe:

Au cours du développement du projet nous avons eu des phases de fonctionnement différentes.

- Au début, le temps de découvrir le sujet et les nouvelles technologies, nous étions tous ensemble.
- Très vite, pour commencer à développer un squelette du projet, nous nous sommes réparti le travail en deux groupes. Un groupe développant les histoires utilisateur nécessaires, et l’autre les implémentait.
- A l’approche de la version 2 du projet, notre DevOps a mis en place un tableau sur github (“Project”), nous permettant la répartition et le suivi des tâches restantes. Voici l’extrait d’un de ces tableaux:



	Title	Assignees	Status
1	Gestion des produits de la part du restaurant	TarikAZIKI	Done
2	Implémentation des slots de production pour les restaurants	simonbeurel	Done

6.3- Auto-évaluation:

Durant tout le long de ce projet, nous avons eu beaucoup de mal surtout lors du début de ce projet. Cependant, une des grosses forces de notre groupe est que tout le monde a “joué le jeu” et ainsi tout le monde s’est investi pleinement dans le projet pour essayer de le réaliser jusqu’au bout. C’est donc pour cela que nous avons décidé de donner la même note à tous les membres de notre groupe :

- **MAUROIS Quentin :** 100 points
- **AZIKI Tarik :** 100 points
- **FROMENT Lorenzo :** 100 points
- **BEUREL Simon :** 100 points