Rapport Backend

Table des matières:

I) Contexte

1.1) Rôles dans l'équipe A :

- Kayma-kcilar Nora : PO
- Mazzi Yohan : SA
- Santos Reis Mathias : Ops
- Abbonato Romain : Ops
- Marill Matice : QA

1.2) a) Hypothèses de travail :

Commande : une commande, qu'elle soit au sein d'un groupe ou non, est faite dans un seul restaurant.

Rejoindre une commande groupée : pour rejoindre un groupe, il est nécessaire de créer et de payer sa commande. Il est ensuite possible de passer une nouvelle commande qui remplacera la précédente. Tout utilisateur ayant passé commande peut valider le groupe et ainsi envoyer les commandes du groupe en préparation.

Restaurant : un restaurant possède des articles et des menus qui sont un ensemble d'articles, ils sont définis par les restaurateurs qui peuvent également mettre à jour leurs cartes.

Livraison : les livraisons sont admises comme réalisable dans les temps donnés, la gestion des livreurs et de leurs disponibilités ne sont pas gérées par l'application.

1.2) b) Limites identifiées :

Système de paiement : le système de paiement utilisé est un service externe à l'application, les potentielles erreurs de paiements, les remboursements ou autres ne sont pas directement gérés par notre système.

Livraisons : le nombre de livreurs nécessaires et la répartition des commandes à livrer entre ces derniers n'est pas géré au moment de la validation d'une commande.

Restaurants : la gestion des commandes par les restaurants, comme l'utilisation de bons n'est pas intégré à l'application.

1.2) c) Stratégies choisies et éléments spécifiques :

EX1 : Trois stratégies ont été mises en place, elles sont appliquées aux commandes groupées pour inciter les utilisateurs à utiliser cette fonctionnalité et ainsi faciliter la préparation et la livraison des commandes.

- deux stratégies pour récompenser directement les commandes de groupes :
  - le nombre de personne dans un groupe (réduction sur toutes les commandes)
  - le nombre d'articles commandés (réduction sur les commandes concernées)

- une stratégie pour récompenser les commandes individuelles
  - le nombre de commandes individuelles passées dans le restaurant de la commande groupée (réduction sur les commandes des utilisateurs concernés)

Avantages : ajouts simples de nouvelles stratégies

Limites : un restaurant possède une seule stratégie, elles ne sont donc pas cumulables.

EX2 : En ce qui concerne l'extension E2 nous avons pris en compte la capacité d'un restaurant en fonction du nombre de commandes dans le groupe, mais cette gestion n'a pas encore été étendue aux commandes individuelles ou globales du restaurant. Donc ce sera une prochaine étape d'amélioration à implémenter pour s'assurer que le restaurant ne prend pas plus de commandes que sa capacité ne le permet sur une période donnée.

II) Conception UML

2.1) Glossaire :

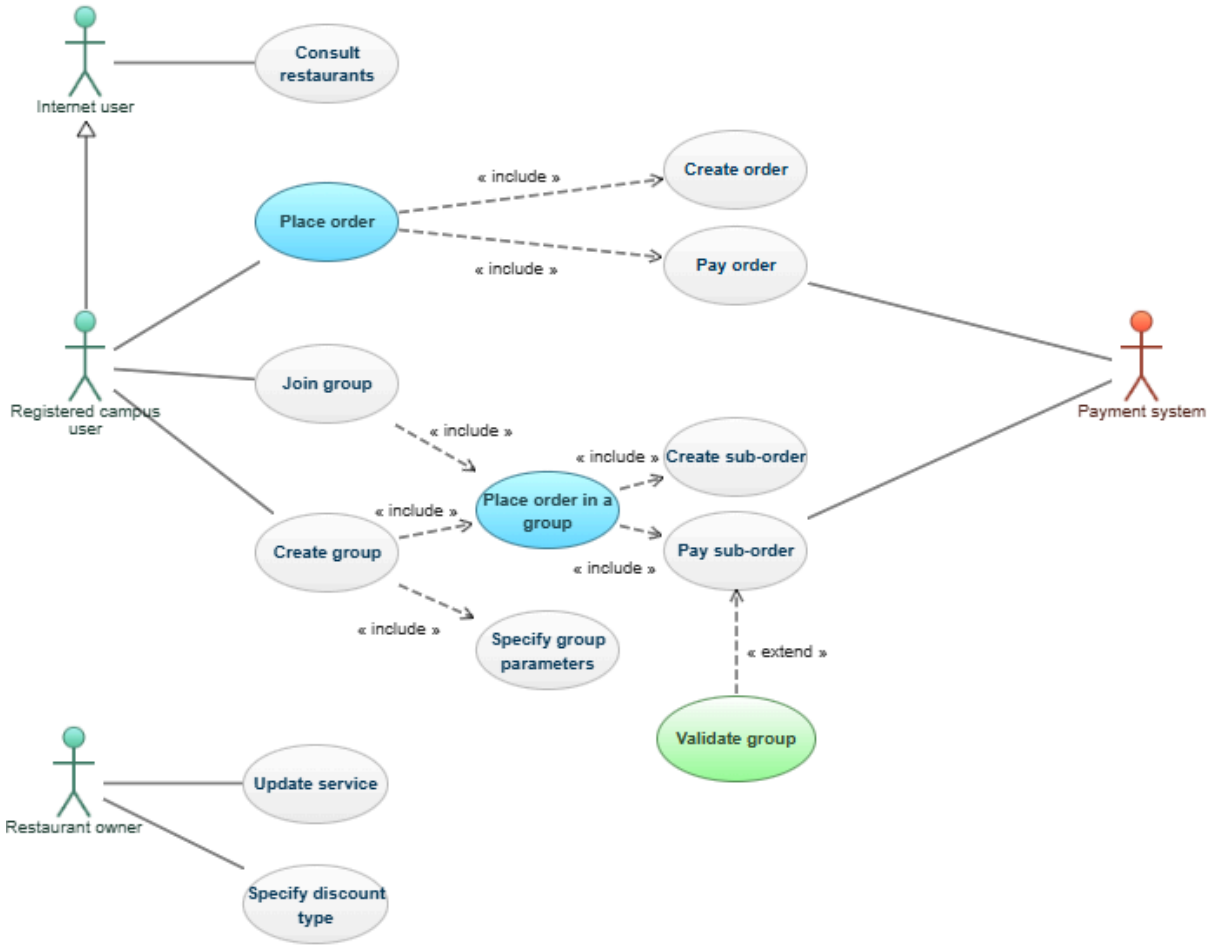
Acteurs :

- Internet users
- Registered campus users
- Restaurant owners
- Campus administrators
- Delivery person
- Payment system

Actions :

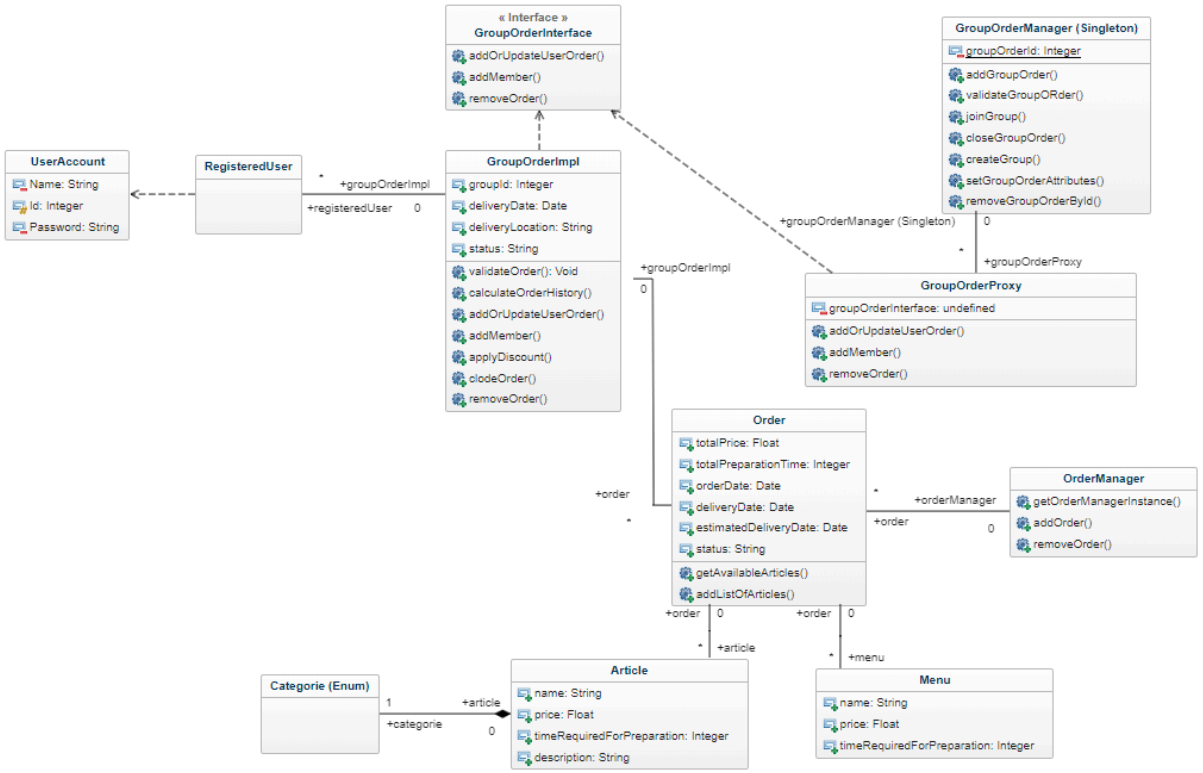
- N1 : browse menus (browseMenus)
- N2 : search restaurant (searchRestaurant)
  → N1 + N2 : consult restaurants
- O1 : place order (placeOrder)
- R2 : update service (updateService)
- O2 : create group order (createGroupOrder)
- O3 : create order (createOrder)
- O4 : update delivery date (updateDeliveryDate)
- P1 : pay order (payOrder)
- P2 : register order (registerOrder)
- P3 : skip payment for group order (skipPaymentForGroupOrder)
- O5 : update restaurant capacity (updateRestaurantCapacity)
- E1 : apply discount (applyDiscount)
- O6 : validate group order (validateGroupOrder)
- O7 : allow group order validation (allowGroupOrderValidation)

2.2) Diagramme de cas d'utilisation :

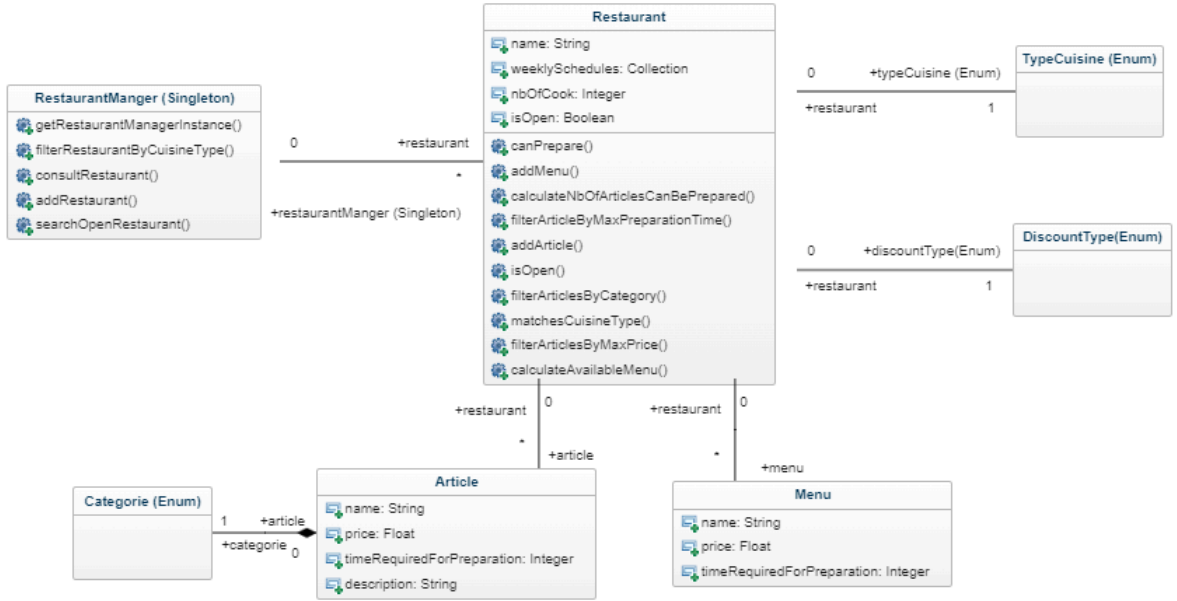


2.3) Diagrammes de classes :

Order :

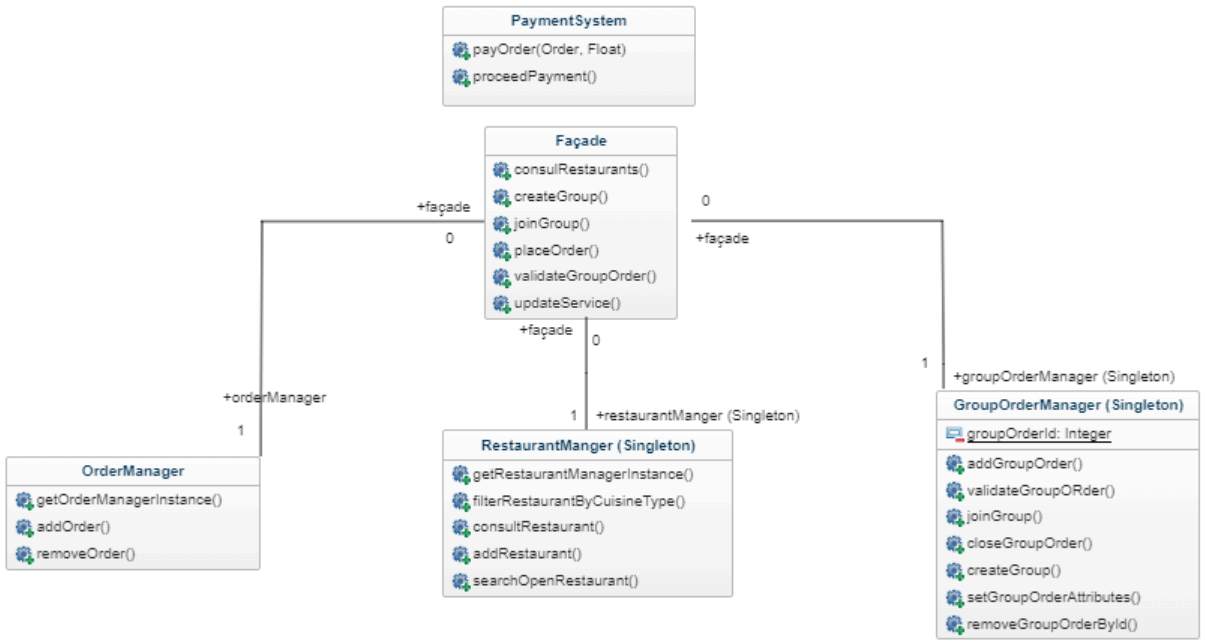


Restaurant :



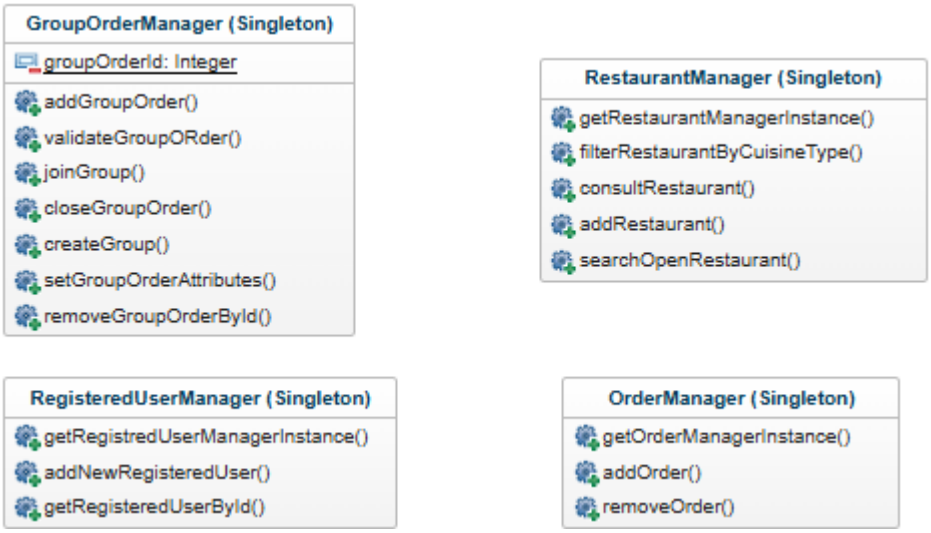
Discount :

System :



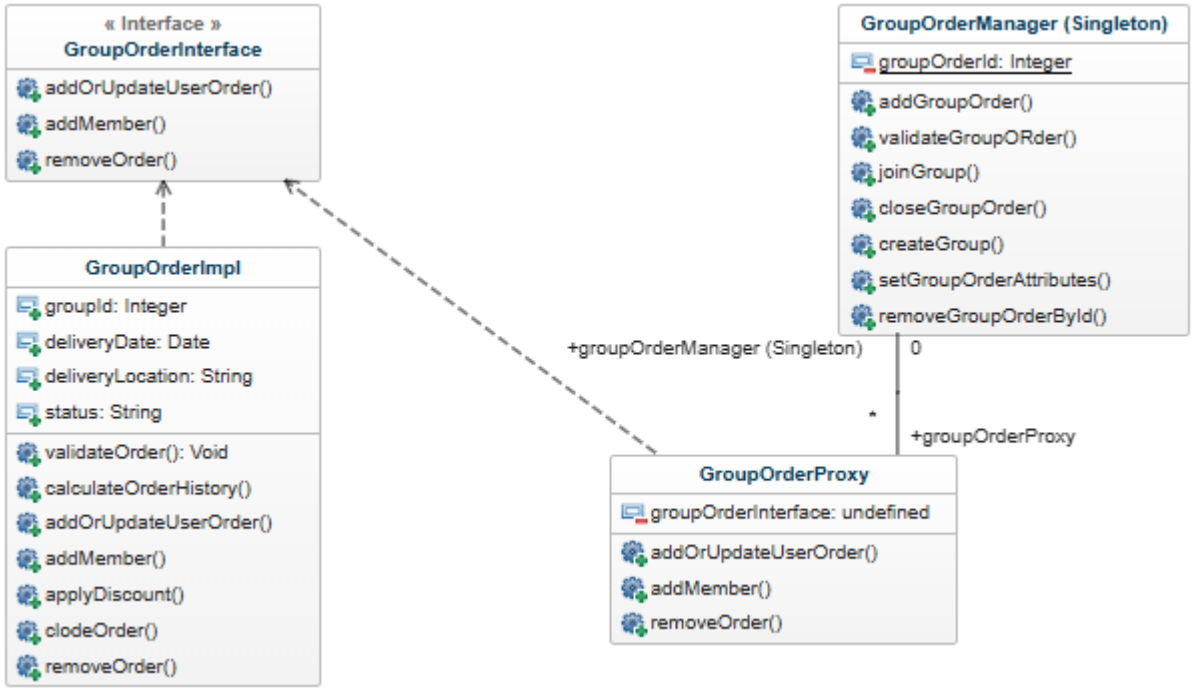
2.4) Design patterns :

- **Singleton :**



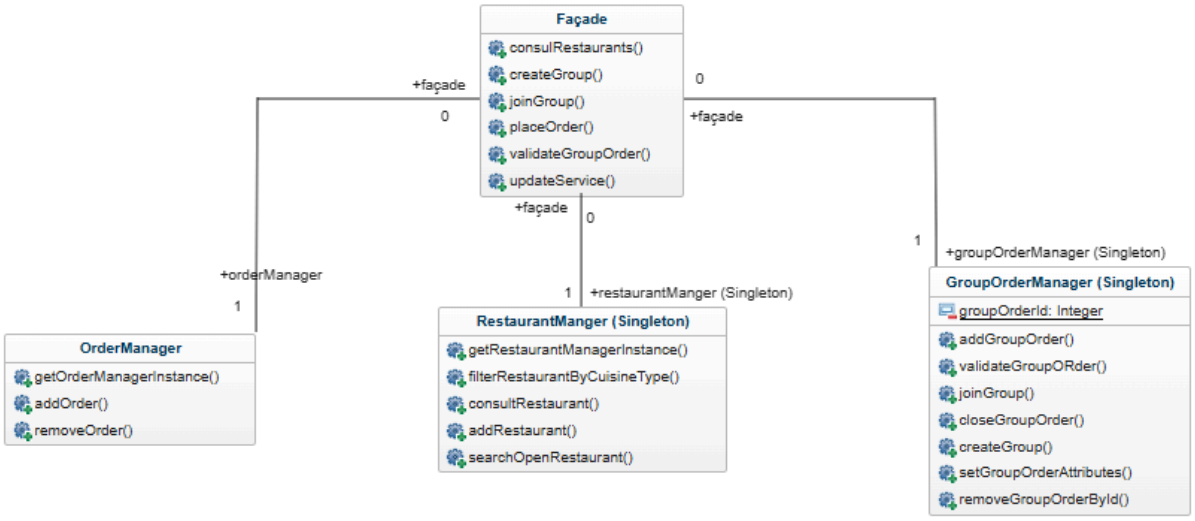
Les singletons permettent de centraliser pour tout le système les instances sauvegardées (commandes, groupes, restaurants existants, utilisateurs) et de les gérer.

- **Proxy :**



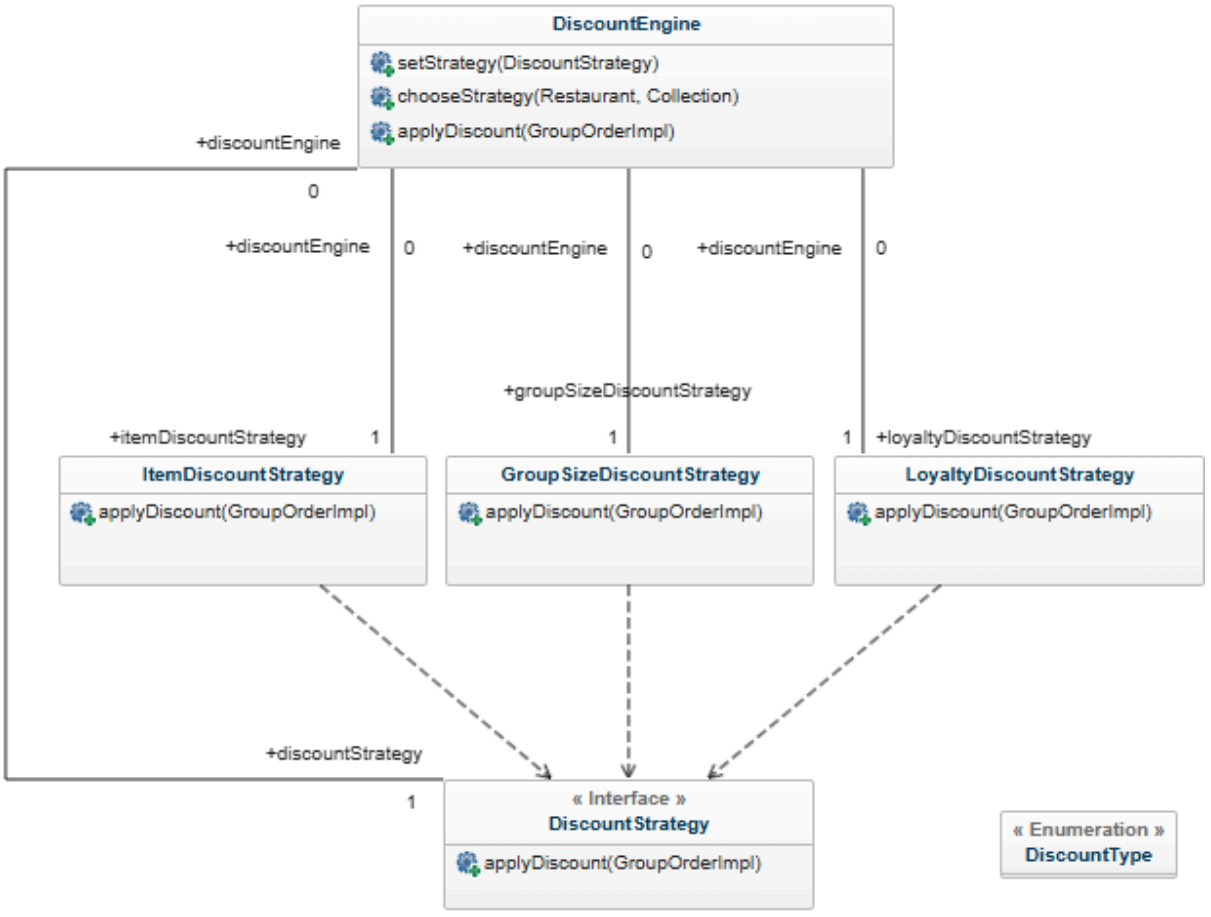
L'utilisation du proxy pour les méthodes liées à la gestion des groupes permet d'effectuer diverses vérifications (validité de la commande ajoutée dans le groupe, possibilité d'ajouter un utilisateur au groupe).

- **Façade :**



La façade permet de séparer la logique métier (contenu dans les managers) de l'accès simplifié aux fonctionnalités (méthodes des façades) de l'application.

- **Stratégie :**



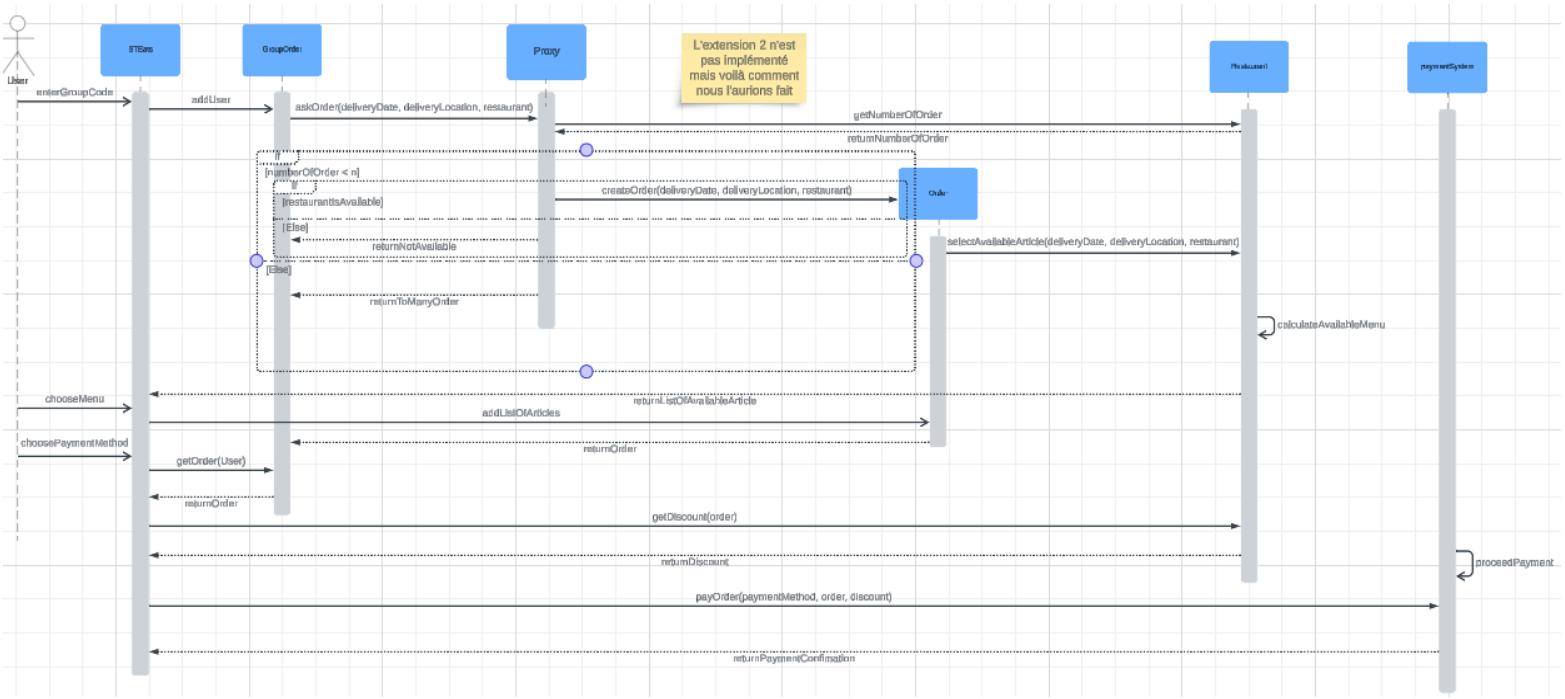
Le pattern strategy permet d'être flexible sur l'affectation des stratégies pour les restaurants et permet l'ajout facile de nouvelles stratégies.

- **Patterns manquants :**

  Pour les managers, l'utilisation d'un pattern **"repository"** aurait permis de mieux séparer l'accès aux données de leur utilisation. Par manque de temps nous avons combiné ces deux aspects.

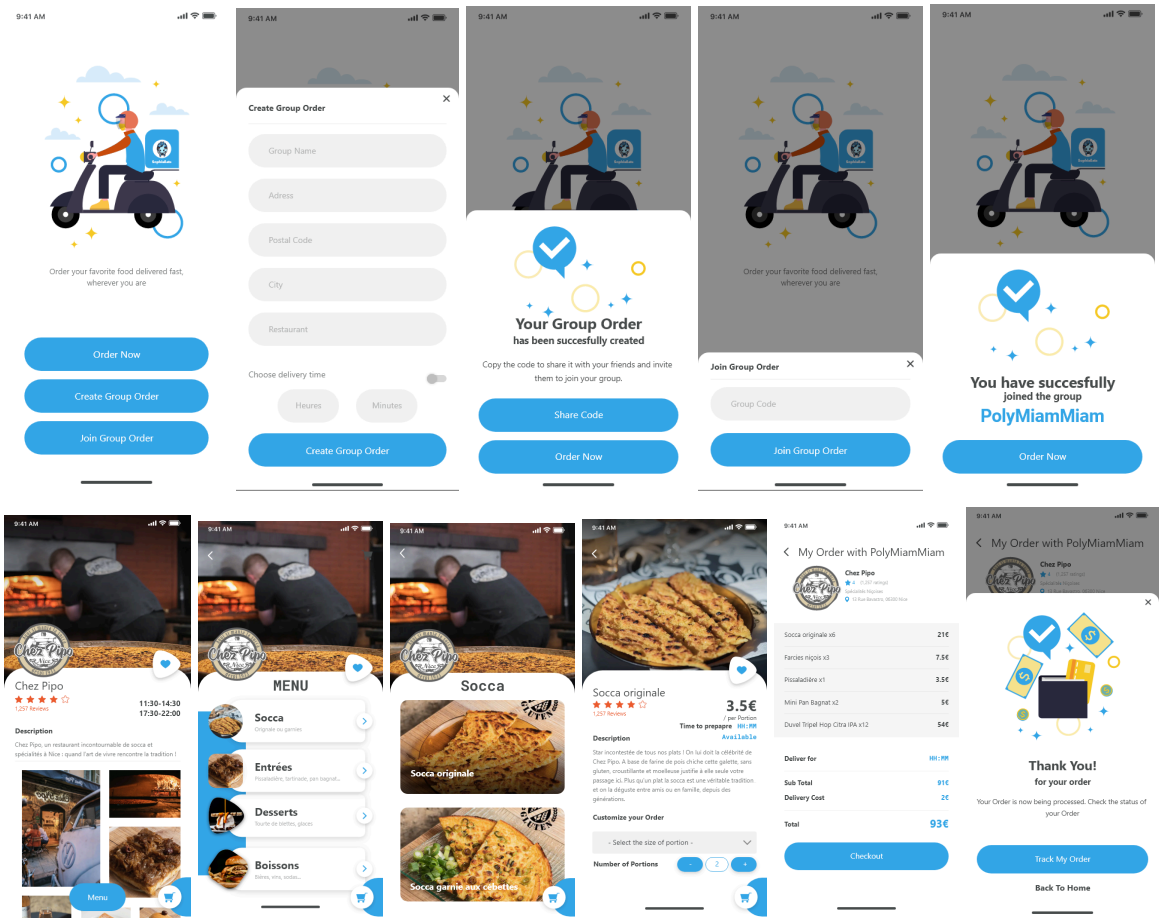
  Nous avions réfléchi à l'utilisation d'un **"builder"**, ce qui aurait pu permettre de construire plus facilement une commande car certaines commandes ne spécifient pas forcément de date de livraison (certaines commandes individuelles), d'autres n'ont pas à spécifier le restaurant, la date et lieu de livraison (sous-commandes dans les groupes de commandes) et d'autres ont besoin de tout spécifier (certaines commandes individuelles). Le rapport entre la valeur ajoutée à notre système et le temps que cela aurait pu prendre nous a finalement décidé à ne pas utiliser ce pattern.

2.5) Diagramme de séquence :
https://lucid.app/lucidchart/36cb9c8f-1a42-4272-b629-6f848c2d5656/edit?invitationId=inv_1881387b-d263-49e4-b827-5c061b723c7c&page=0_0#



III) Maquette

Maquette pour une prise de commande dans le contexte d'une commande groupée :



IV) Qualités des codes et gestions de projets

Dans notre projet, nous avons pris en charge divers types de tests pour assurer la qualité et la robustesse de notre code. Nous avons implémenté des tests unitaires à l'aide de JUnit, ce qui nous a permis de valider chaque composant individuellement, garantissant ainsi que chaque fonction et méthode se comportait comme prévu. De plus, nous avons utilisé des scénarios Gherkin pour définir des tests de comportement couvrant des cas d'usage et des cas d'erreurs. Grâce à cette approche, nous avons atteint une couverture de tests de 75 % couvrant ainsi une part significative du code et garantissant que les fonctionnalités principales sont bien testées.(Les 25% restants étant des méthodes des package System et Users). L'analyse avec SonarQube a confirmé la bonne qualité du code, avec peu de "code smells"

(la plupart étant des variables ou attributs non utilisés) et une base de code propre.

Pour la gestion du projet, nous avons adopté le "GitHub Flow", avec une gestion stricte des branches. Cette méthodologie nous a permis de travailler sur des branches de fonctionnalités séparées, d'effectuer des pull requests pour la revue de code.

V) Rétrospective et auto-évaluation

- Retours sur notre projet :
    - **Ce qui a bien été mené :**
      L'architecture du code a bien été respectée et l'étude de cas (définitions, termes et points importants) a été discutée en détail au sein de l'équipe, ce qui a permis d'avancer dans la même direction.

    - **Leçons apprises :**
      Une compréhension précise du sujet est primordiale, définir un glossaire permet de définir les termes principaux et éviter des interprétations différentes au sein d'une équipe.
      De plus, lorsque l'on conçoit l'architecture il est important de se baser sur les spécifications détaillées au préalable et de se demander si un design pattern déjà existant peut être utilisé et adapté à notre problème.

    - **Les erreurs :**
      Nous n'avons pas suivi un plan bien établi et détaillé en ce qui concerne la livraison de valeur sur une période de temps définie (chaque semaine…).
      En ce qui concerne notre premier rendu, il était incomplet et une grosse partie de l'architecture (diagramme de classe) a été à revoir, ce qui nous a coûté beaucoup de temps pour redéfinir les rendus à effectué et la répartition des tâches au sein de l'équipe.

- Missions remplies par les membres du groupe :

  Yohan Mazzi (**SA**) :

En tant que **Software Architect**, ma mission a été de m'assurer de la cohérence entre notre développement et les diagrammes que nous avions initialement produits.
J'ai conceptualisé une première version du diagramme de classe de cas d'utilisations. J'ai fait évoluer au fur et à mesure du développement les diagrammes, afin qu'ils respectent nos changements de point de vue de conception et les nouvelles contraintes que le développement et le projet nous a imposé.
Au niveau du développement, j'ai pu mettre en place les squelettes des singletons pour les managers ainsi que celui de la façade. Je me suis par la suite concentré sur les fonctionnalités (méthodes, tests, scénarios…) permettant de créer et rejoindre un groupe de commande.

Nora Kayma-kcilar (**PO**) :
En tant que **Product Owner**, j'ai été responsable de m'assurer que les fonctionnalités développées répondent aux attentes des utilisateurs. Au niveau du développement je me suis concentrée sur la logique de **validation des commandes de groupe** et l'implémentation des **stratégies de réduction**. J'ai veillé à garantir que chaque commande individuelle respecte les paramètres communs (date, lieu de livraison, restaurant) et qu'elle ne puisse plus être modifiée une fois validée, déclenchant ainsi la préparation des repas. J'ai également intégré les stratégies de réduction, basées sur la taille du groupe, le nombre d'articles commandés et la fidélité des clients, validées par des tests. L'extension E2 n'a pas encore été implémentée, car nous avons priorisé les fonctionnalités essentielles au processus de commande de groupe, ce qui sera une future amélioration pour rendre le système encore plus performant et conforme aux attentes des utilisateurs.

Mathias Santos Reis (**OPS**) :
En tant que **Continuous Integration and Repository Manager** sur ce projet, mon rôle a été de garantir la stabilité en veillant à l'intégration en continu du code. La gestion de l'état du repository a été également l'une de mes missions. Pour le développement je me suis concentré sur l'implémentation des horaires de restaurant et du temps de préparations d'articles.

Romain Abbonato **(OPS) :**

En tant qu'Operations Manager **(OPS)** sur ce projet, ma mission principale a été de garantir l'efficacité et la qualité des processus de développement, ainsi que la gestion des commandes.

J'ai commencé par créer les premières classes du package **order**, notamment celles liées à **GroupOrder** et à **Order**.

Par la suite, j'ai travaillé sur le **Group Order Proxy**, une fonctionnalité essentielle pour gérer les commandes groupées. J'ai veillé à ce que ce proxy vérifie la disponibilité des restaurants avant de finaliser une commande, assurant ainsi une expérience utilisateur fluide.

J'ai également élaboré des **user stories** concernant les filtres des restaurants et des menus.

Dans le cadre de mes responsabilités en tant qu'OPS, je me suis assuré de la bonne organisation des différentes issues dans notre système de gestion de projet. J'ai joué un rôle actif dans la gestion de notre dépôt Git.

Matice Marill **(QA) :**

En tant que **Quality Assurance** sur ce projet, mon rôle consiste à garantir la qualité et la fiabilité des produits avant leur mise en production. Un **QA** s'assure que le logiciel fonctionne comme prévu, en identifiant les défauts potentiels et en vérifiant que toutes les fonctionnalités répondent aux exigences spécifiées. Je me suis donc concentré sur les **user stories**, en veillant à leur cohérence en collaboration avec mes collègues, sur les différents diagrammes ainsi que sur les **tests unitaires**.

- Auto-évaluation : 500 points à répartir

  Nora Kayma-kcilar : 100
  Yohan Mazzi : 100
  Mathias Santos Reis : 100
  Romain Abbonato : 100
  Matice Marill : 100