Rapport Conception Logiciel

SophiaTech Eats

Université côte d'azur
Polytech Nice Sophia

Année Scolaire :
2023/2024

Equipe G:

CHATELAIN Clément (QA)
FARHAT Melek (SA)
SIKA Dassou Ablam Kaleb (OPS)
ZENKRI Sarra (PO)

Sommaire

1) Périmètre fonctionnel

1.1) Hypothèses de Travail

- Un acteur User (Guest) qui est un visiteur qui ne possède pas de compte et qui a donc la possibilité de s'inscrire et de naviguer sur le système.
- Nous supposons que le personnel, le responsable du restaurant, l'administrateur du campus et le livreur sont préalablement connectés pour permettre le déroulement des actions ultérieures. L'omission de la spécification du lieu et de la date de livraison a été faite pour des raisons de simplicité.
- Le Staff c'est celui qui travaille en restaurant donc celui qui accepte ou refuse les commandes et celui qui a la possibilité de marquer la commande comme prête.
- Le Restaurant Manager c'est qui a la possibilité de générer son restaurant.
- Admin est responsable d'ajouter des restaurant et par la suite créer des Restaurant Manager et des Livreur
- L'admin a le droit de donner peu n'importe quel utilisateur un rôle sauf le rôle d'Admin pour la sécurité d'application
- Une commande ne peut être annulée qu'après avoir la payer.
- Le livreur peut signaler un utilisateur en le donnant des étoiles de 0 à 5

1.2) Extensions choisies et éléments spécifiques

Dans le cadre de notre projet, nous avons introduit plusieurs extensions et fonctionnalités innovantes, allant au-delà des spécifications initiales.

- Diversité des Types de Commandes: L'originalité réside dans la flexibilité et la variété des options de commande. En offrant des commandes multiples, de groupe, des buffets et des afterworks, nous adressons des besoins spécifiques et variés des utilisateurs, allant au-delà des services traditionnels de livraison. Cette approche est particulièrement novatrice pour les commandes afterWorks, où nous supprimons la livraison et le paiement, créant une nouvelle catégorie de service. La limite ici pourrait être la complexité de la gestion de ces divers formats de commandes et la nécessité d'une interface utilisateur intuitive pour les gérer efficacement.

- Diversité des Menus: En permettant la personnalisation des menus, notre proposition se distingue par son attention portée à l'expérience culinaire personnalisée de l'utilisateur. Cette extension va au-delà de la simple sélection de plats, en permettant aux utilisateurs de modifier les composants individuels des menus. Toutefois, cette personnalisation peut entraîner des défis logistiques pour les restaurateurs et nécessiter des systèmes de commande plus sophistiqués pour gérer efficacement les préférences des utilisateurs.

- Signalement des Usagers en Retard: Cette fonctionnalité est originale car elle vise à responsabiliser les utilisateurs pour leurs retards, ce qui est rare dans les applications de livraison. Cela peut améliorer l'efficacité de la livraison et la satisfaction des livreurs. La limite ici réside dans le potentiel de conflits ou de mécontentements des clients signalés, nécessitant une gestion délicate des cas signalés pour maintenir une bonne relation client.

Chacune de ces extensions est conçue pour améliorer l'expérience utilisateur tout en reconnaissant les défis et limites inhérents à leur mise en œuvre.

1.3) Points non implémentés relativement la spécification et des extensions choisies

Dans notre rapport, il est crucial de reconnaître certaines déviations par rapport au plan initial en raison de défis inattendus.
D'accord, tentons une nouvelle formulation avec un ton positif et prospectif :

Notre projet a accompli des progrès significatifs, bien que certaines fonctionnalités, telles que C1 et O3 n'aient été que partiellement implémentées. C1 a été élaborée pour améliorer la gestion des partenariats avec les restaurants, et bien que nous ayons jeté les bases nécessaires, quelques ajustements sont encore à prévoir pour optimiser le contrôle et l'efficacité de cette gestion.

Pour la création de commandes représentée par O3, nous avons établi un système fonctionnel qui, même s'il nécessite actuellement une intervention manuelle, pose les fondations pour une future automatisation complète. Cela représente une étape intermédiaire vers une plateforme plus autonome et intuitive.

Bien que ces fonctionnalités ne soient pas encore pleinement réalisées, leur état actuel pose une base solide pour l'expansion future. L'achèvement de ces éléments représentera une amélioration notable de notre service, renforçant à la fois l'expérience utilisateur et l'efficacité opérationnelle des partenaires. C'est une promesse d'évolution vers la vision complète de notre projet.
Quant au "Système de recommandations" [EX7], il fonctionne à capacité réduite, délivrant uniquement des conseils généraux plutôt que des suggestions hautement personnalisées. Cette limitation restreint l'engagement utilisateur et, par extension, la fidélisation qui est vitale pour la croissance du projet.

Ces écarts ont un impact modéré sur notre projet : ils limitent une certaine souplesse opérationnelle pour les restaurateurs, augmentent la possibilité d'erreurs financières manuelles et ne capturent pas pleinement l'expérience utilisateur enrichie que nous avions anticipée. Toutefois, il est à noter que l'implémentation intégrale de ces fonctionnalités pourrait considérablement renforcer la valeur ajoutée de notre plateforme. Dans l'éventualité d'une extension ou d'une reprise du projet, l'achèvement de ces composants deviendrait une

priorité pour parfaire notre opérationnalité et pour cultiver l'engagement de nos clients selon les normes les plus élevées initialement prévues.

2) Conception

2.1) Le glossaire

menu: c’est la carte du restaurant ou il met tous les plats qui offre.

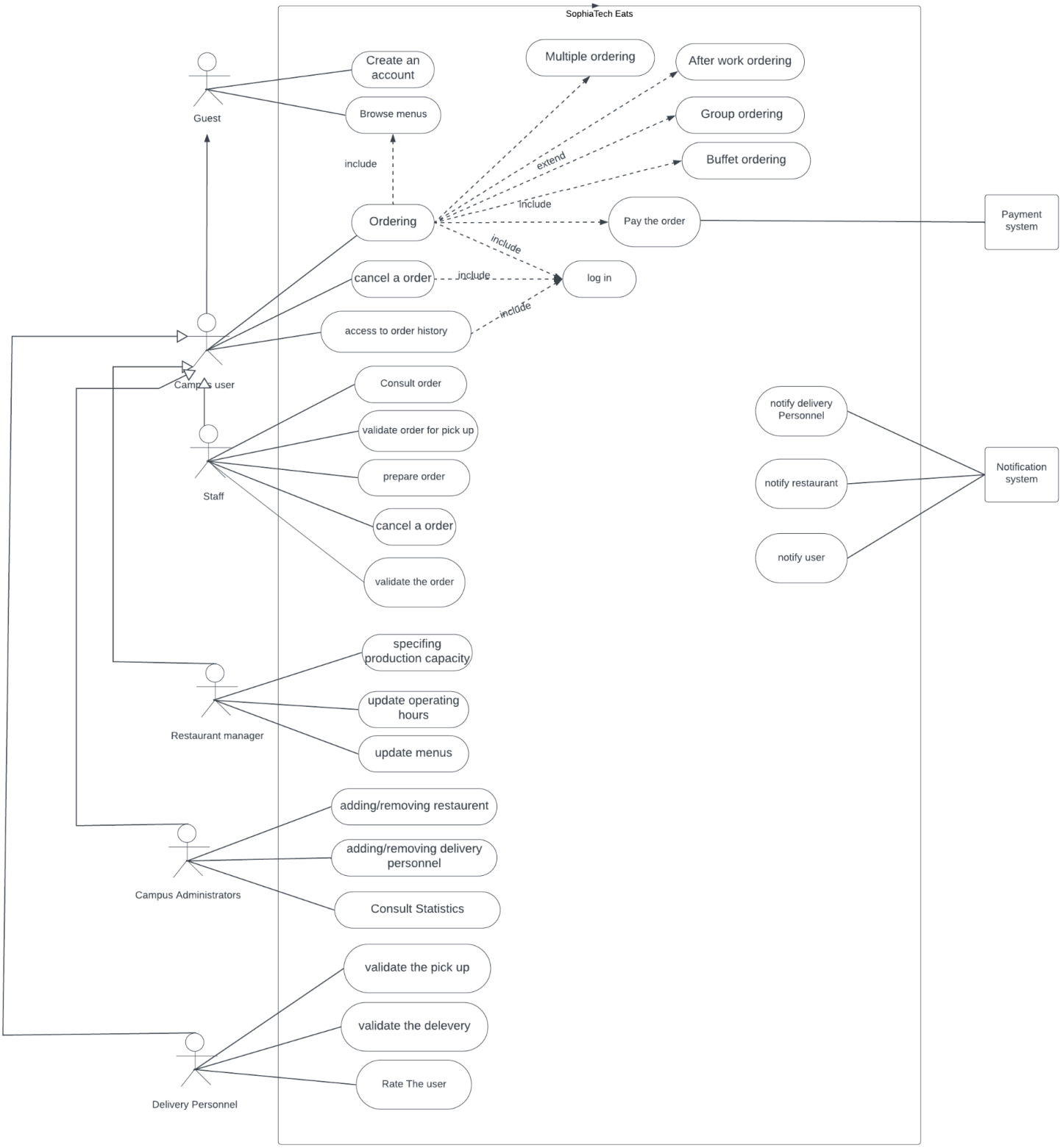
validate: marquer commande comme prête.

Item: Terme utilisé pour désigner un plat spécifique ainsi que la quantité commandée de ce plat.

Pick up: Référence au fait que le livreur vienne chercher la commande.

Operating hours: Heures d'ouverture du restaurant.

2.2) Le diagramme de cas d'utilisation:



2.3) User Stories :

1. Création d'ordres pour les événements Afterwork
   - User Story :
     - Référence GitHub : https://github.com/PNS-Conception/ste-23-24-equipe-g/issues/25
       - Formulation : "En tant qu'utilisateur souhaitant organiser un événement Afterwork, je veux avoir la possibilité de créer des ordres pour ces événements, afin de pouvoir organiser des événements Afterwork sans processus de paiement immédiat ni de livraison."

2. Gestion de commandes multiples
   - User Story :
     - Référence GitHub : https://github.com/PNS-Conception/ste-23-24-equipe-g/issues/24
       - Formulation : "En tant qu'utilisateur, je veux pouvoir créer, ajouter, annuler et gérer plusieurs commandes en une seule fois afin de simplifier le processus de commande groupée."

3. Gestion des Commandes de Buffet
   - User Story :
     - Référence GitHub : #28
       - Formulation : En tant que membre du personnel universitaire (par exemple, une secrétaire), je veux pouvoir commander des buffets pour divers événements universitaires, afin d'assurer une organisation efficace et une expérience satisfaisante pour les participants.

4. Extension Signalement des Usagers
   - User Story :
     - Référence GitHub : #27
     - Formulation : "En tant que livreur, je veux pouvoir noter l'utilisateur.

Critère d'acceptation/Acceptance criteria
Scénario: Le livreur attribue sa première note au client.

Etant donné que le client commande pour la première fois
Et que le livreur met une note de 5 au client
Alors le client a une moyenne de 5.
Et le client possède 1 note.

Scénario: Le livreur met une mauvaise note au client.

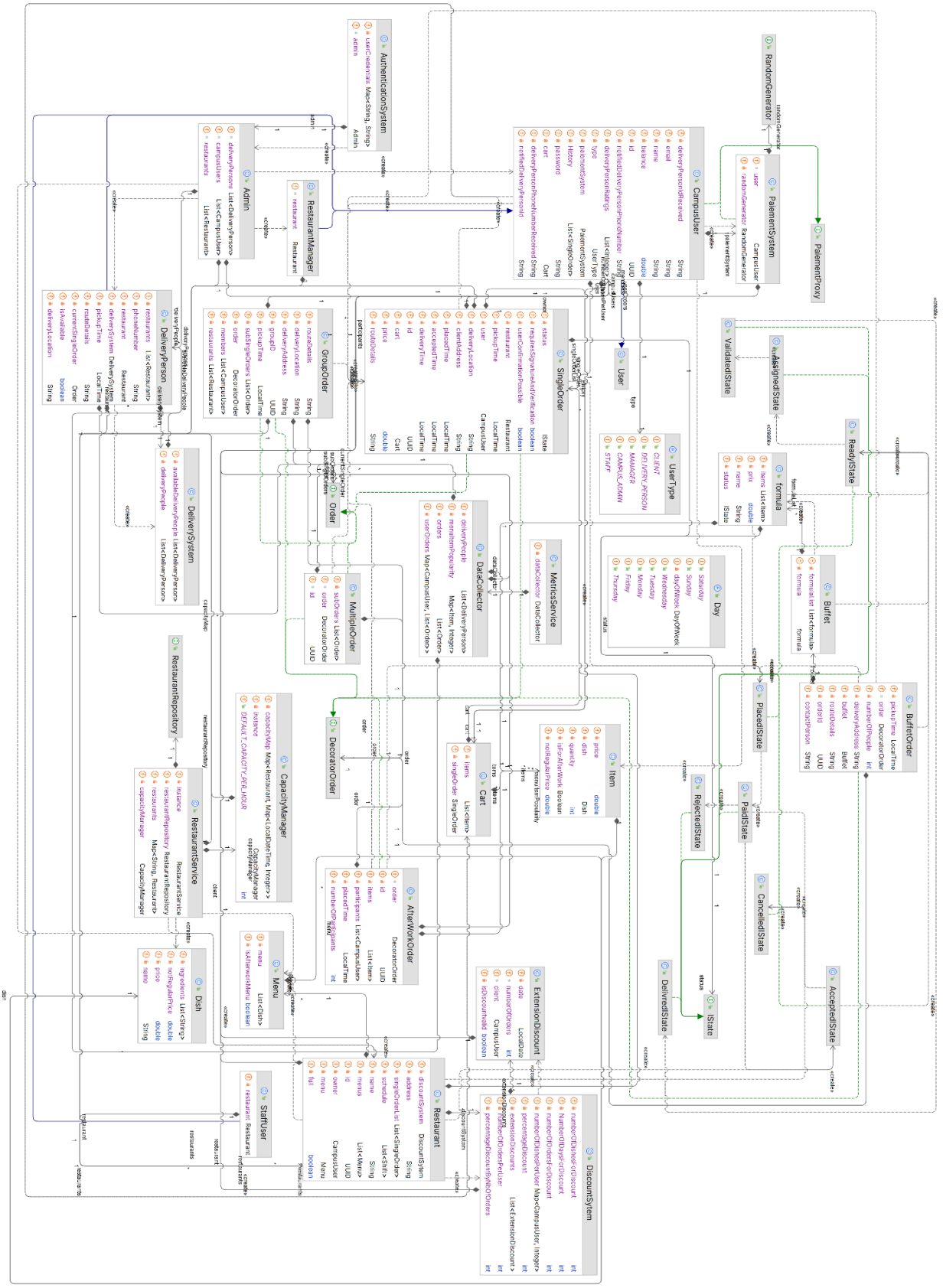
Etant donné que le client a déjà une note de 3
Et que le livreur met une note de 1 au client
Alors le client a une moyenne de 2.

Scénario: Le livreur met une bonne note au client.

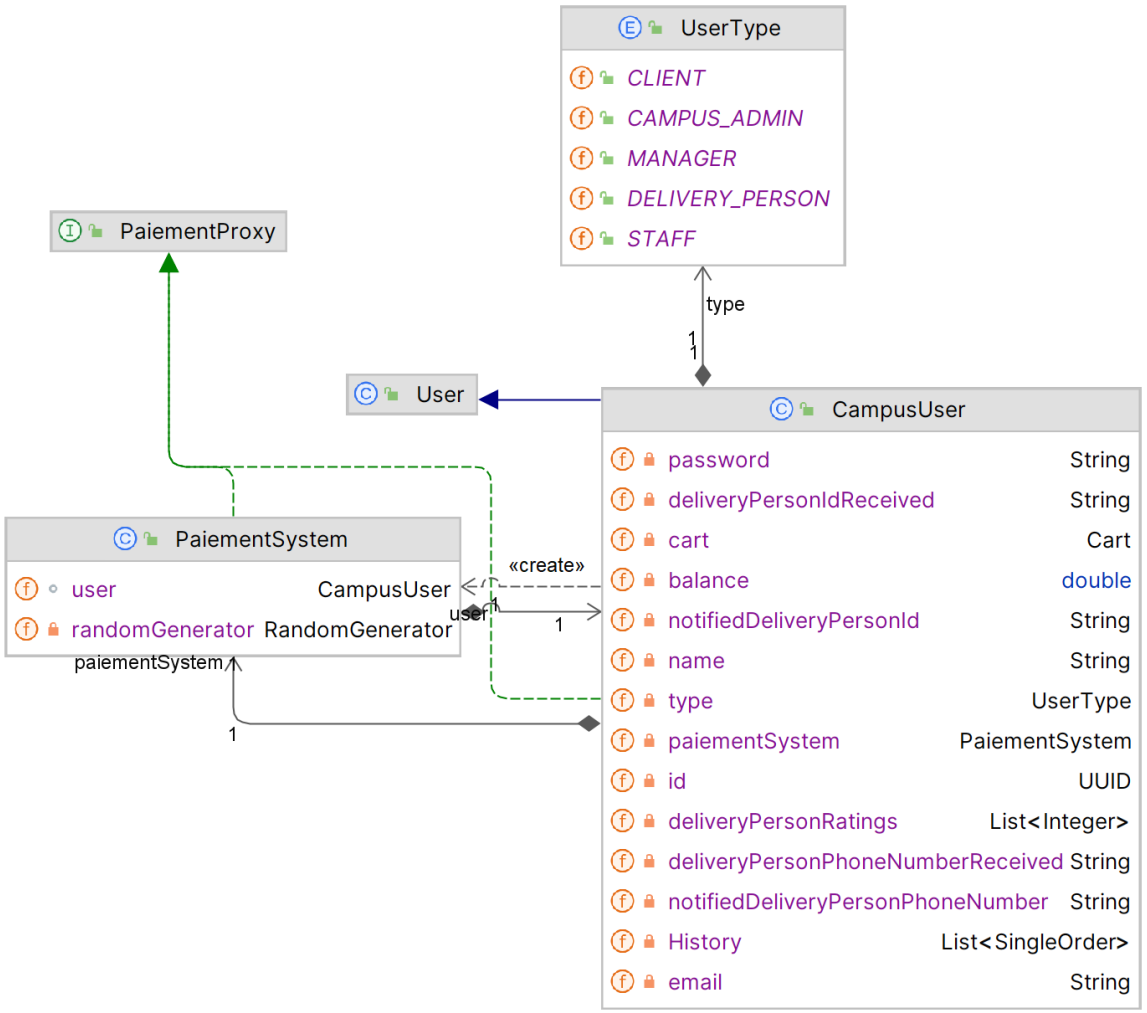
Etant donné que le client a déjà 3 évaluations avec une note de 4
Lorsque le livreur évalue le client avec une note de 5
Alors la moyenne des notes du client devient 4,25

2.3) Le diagramme de classes :

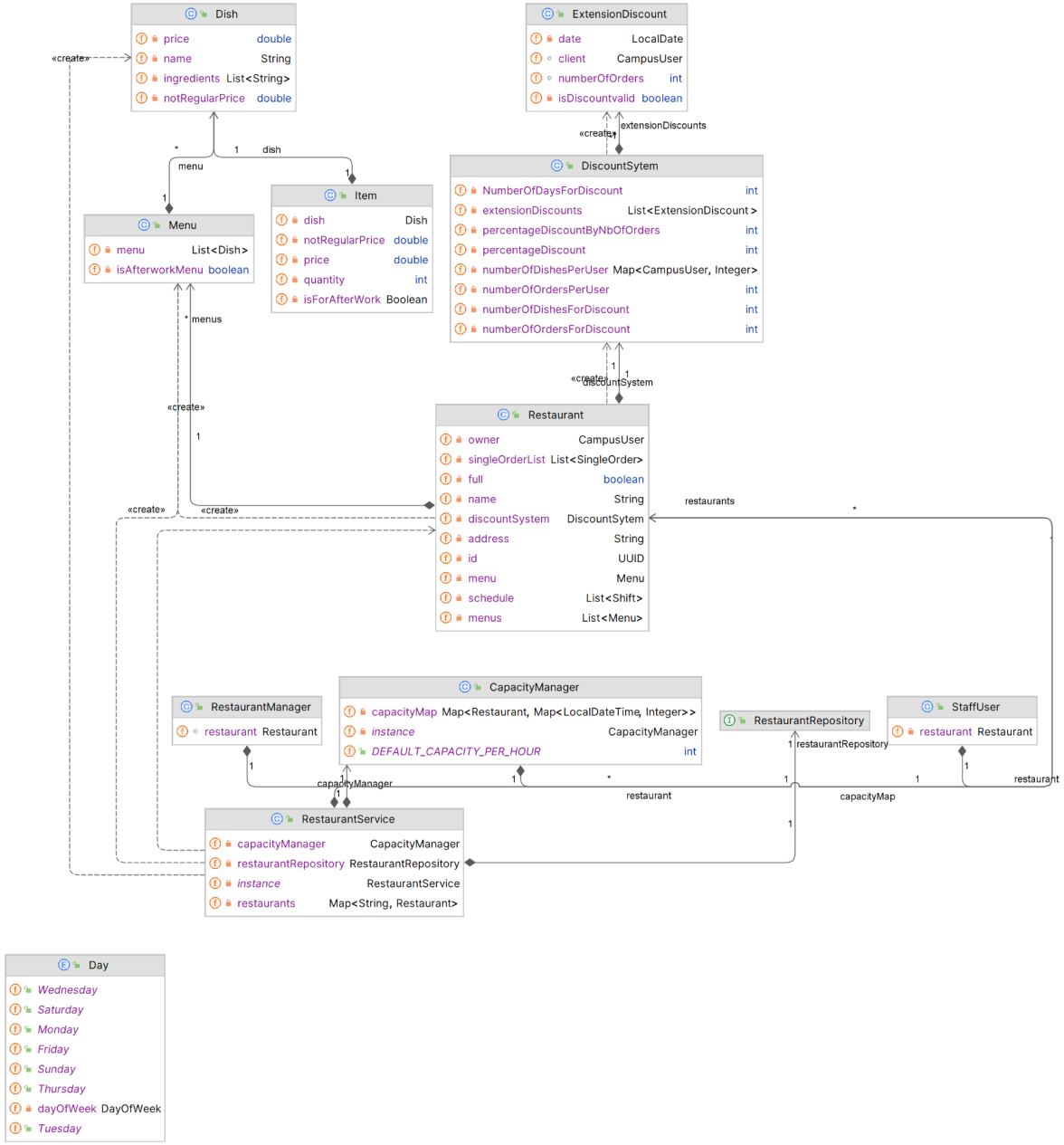
2.3.1) Diagramme globale:



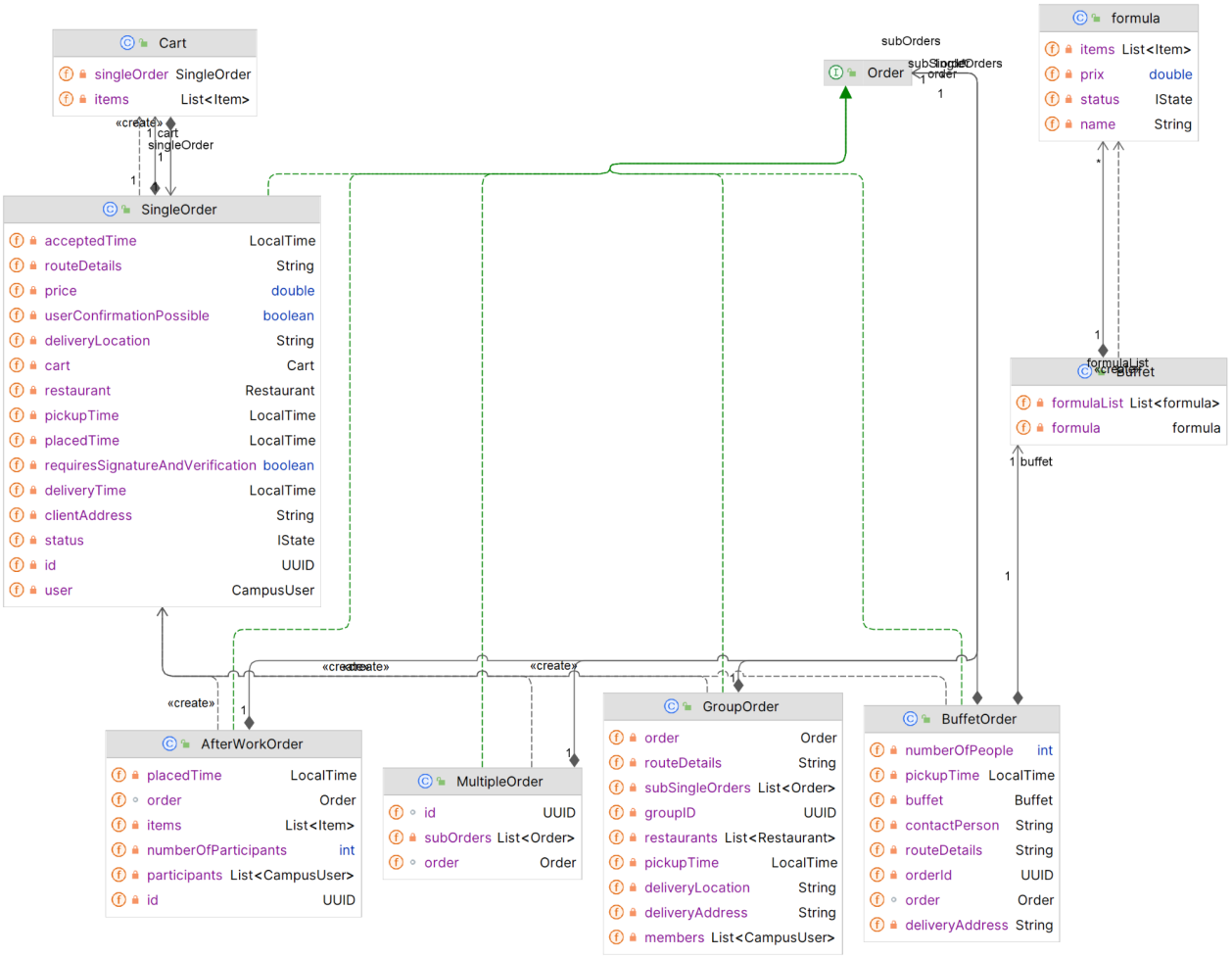
3.3.2) Diagramme de classes par package :



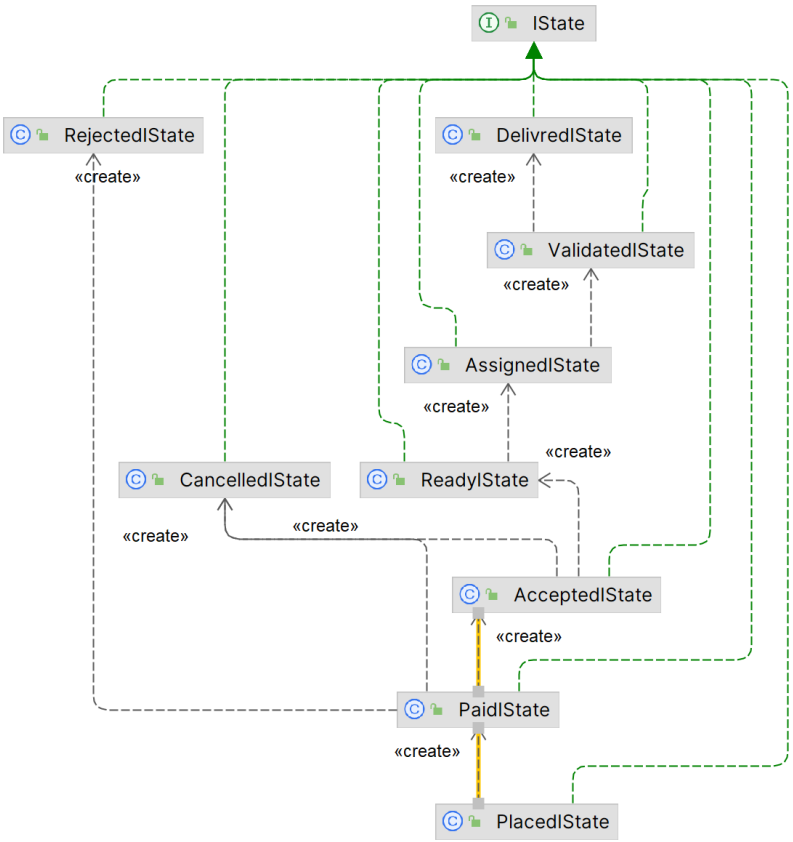
User package diagramme



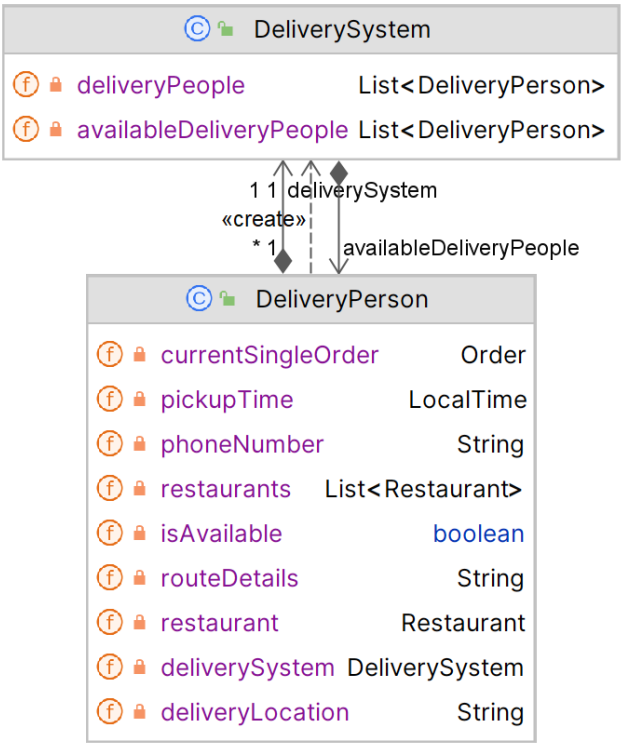
Restaurant package diagramme



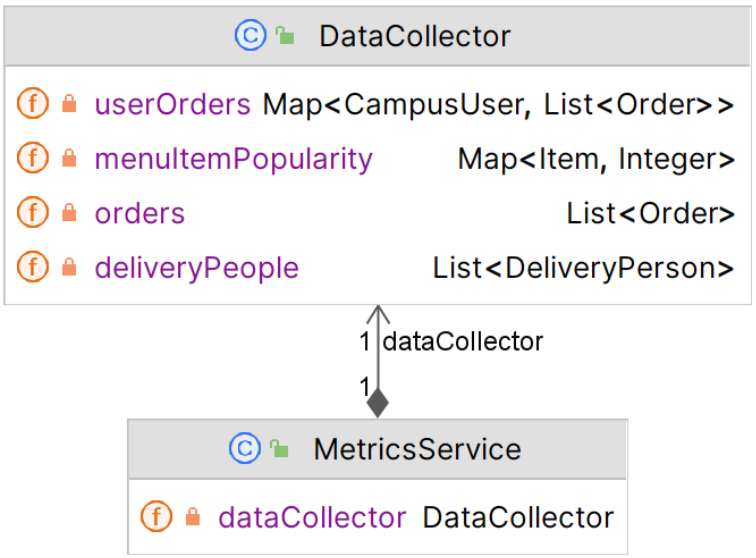
Order package diagramme



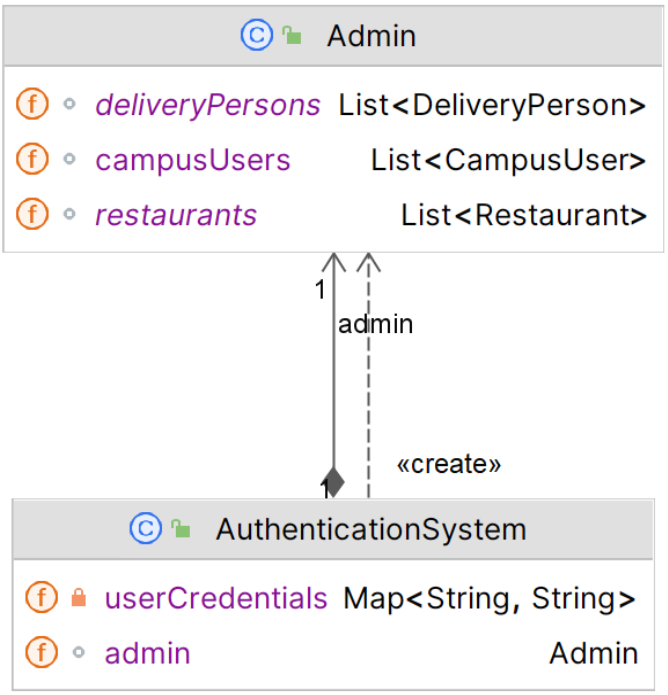
State package diagramme



Delivery package diagramme

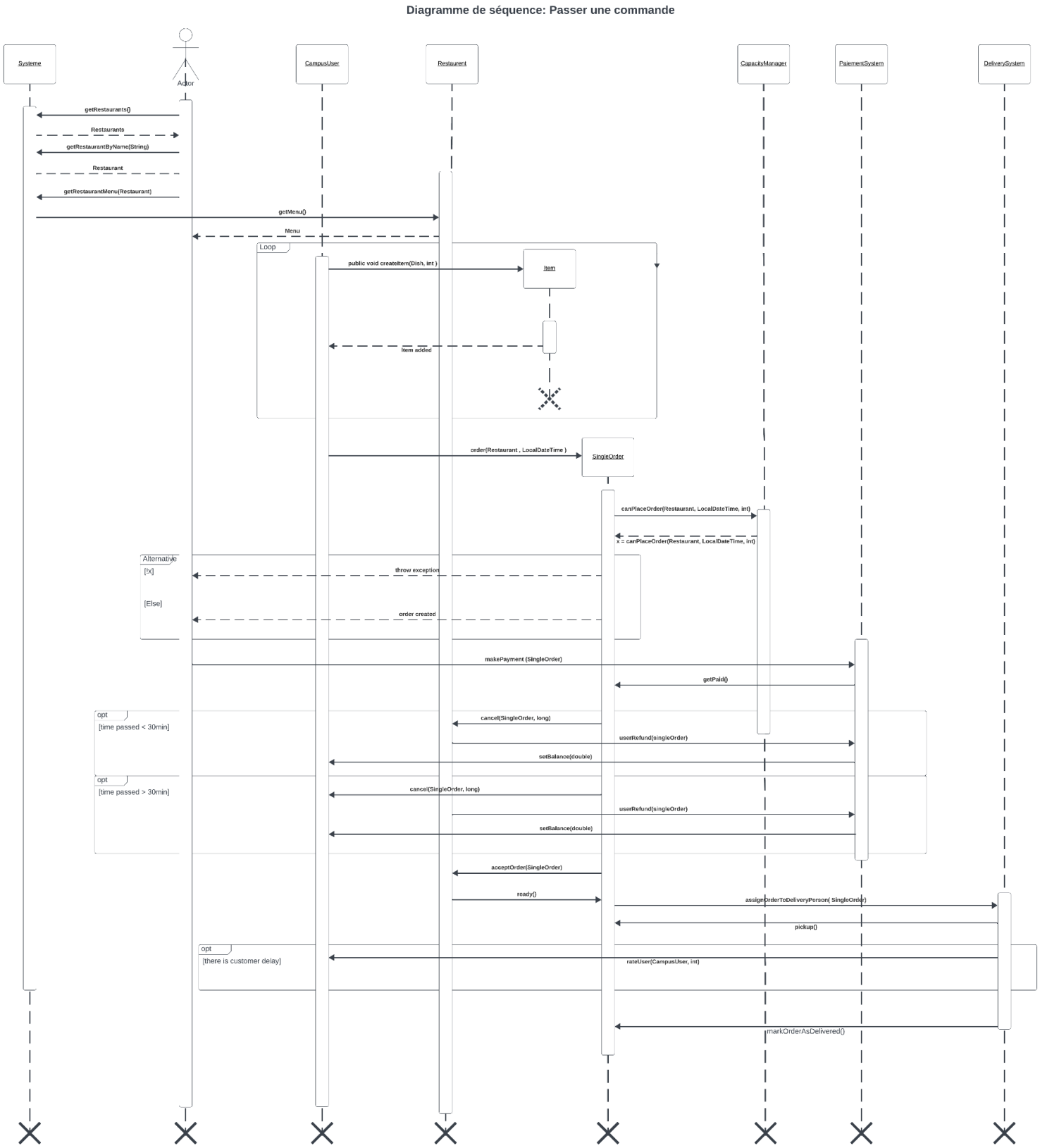


Statics package diagramme



Statics package diagramme

2.4) Un diagramme de séquence



4) Design Patterns

Nous avons un projet en conception logicielle de grande envergure. Pour structurer et faciliter la maintenance du code, nous avons décidé d'adopter quelques design pattern. Ces petits outils améliorent non seulement la lisibilité du code, mais ils le rendent aussi plus compréhensible. Voilà les patrons qu'on a intégré:

4.1) DP State:

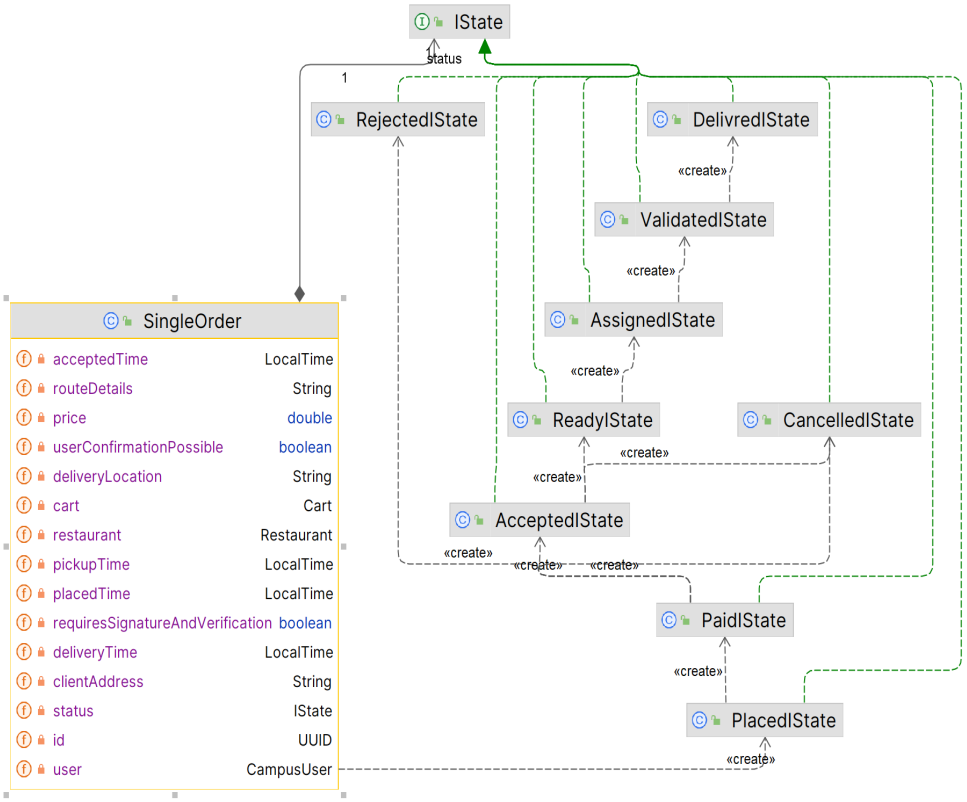
Dans notre implémentation du patron State, chaque état spécifique de la commande, tel que PlacedIState, PaidIState, AcceptedIState Etc., est représenté par une classe distincte. Chacune de ces classes concrètes implémente l'interface IState, définissant ainsi les actions spécifiques associées à chaque état. Par exemple, dans la classe PlacedIState, nous avons défini les actions nécessaires lorsqu'une commande est placée par exemple dans PlacedIState, lorsqu'une commande est placé nous pouvons que la payer

```
public void pay(Order order) throws Exception{
   order.setStatus(new PaidIState());
}
```

et pour les autres fonctions nous lançons une exception par exemple dans la même classe PlacedIState nous avons:

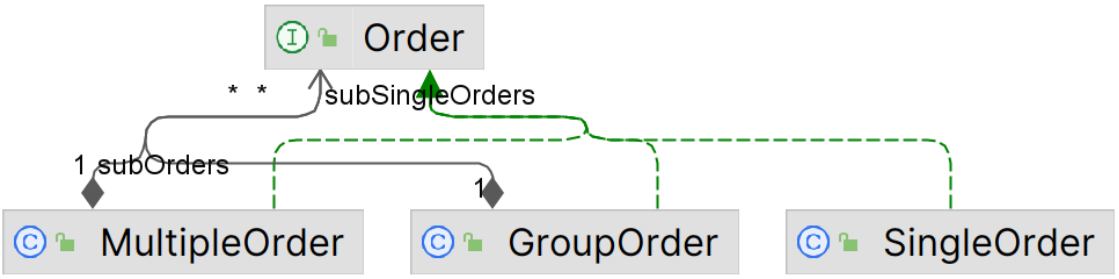
```
public void placeOrder(Order order) throws Exception {
   throw new Exception("Order already placed");
}
```

De même, les autres classes décrivent les actions à entreprendre lorsque nous appliquons une action sur une commande. Pour faciliter les transitions entre ces états, la classe SingleOrder détient un attribut de type IState et donc dans le constructeur de Single order on initialise cet attribut par PlaceOrder, représentant l'état actuel de la commande. Les fonctions de transition entre les états sont définies dans l'interface IState, assurant ainsi une gestion cohérente et fluide des changements d'états.En résumé, notre implémentation du patron State garantit une structure claire et extensible, permettant l'ajout aisé de nouveaux états et simplifiant la compréhension du comportement dynamique de l'objet "commande".
pour faire une analogie entre notre diagramme et le cours :
“IState” présente Etat. PlacedISate, PaidIState, AcceptedIState Ect. présentent les EtatsConcrets. la classe SingleOder a un attribut de type IState pour l’etat de l'order



4.2) DP Composite :

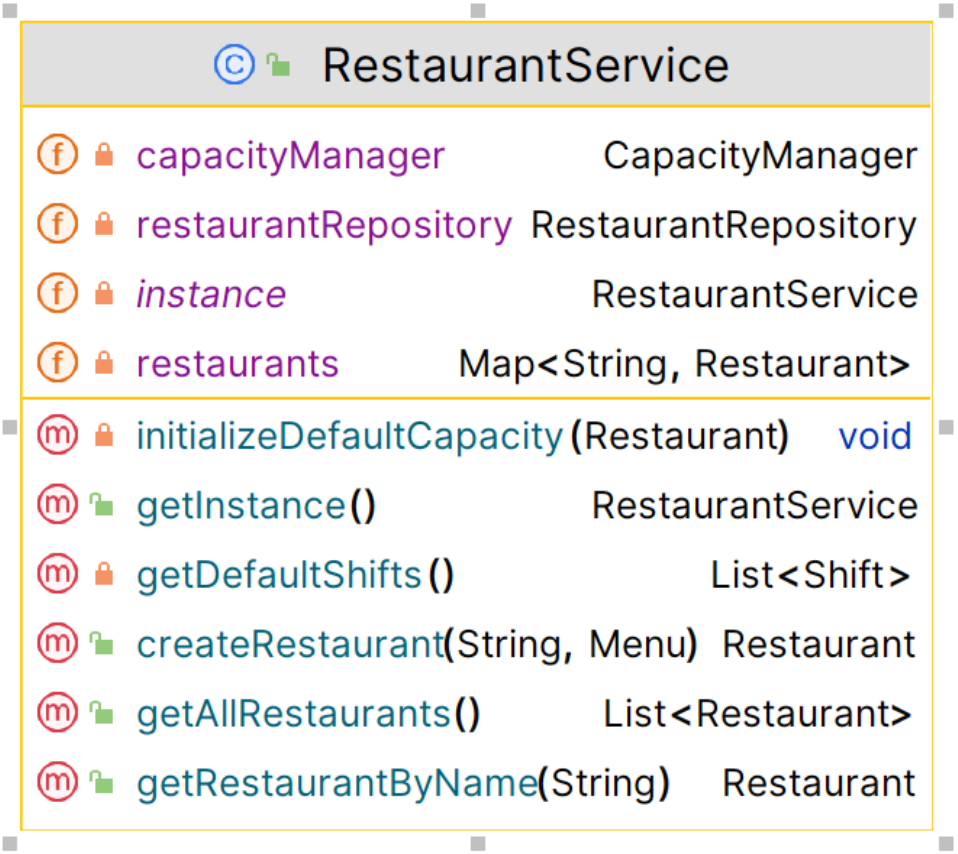
Dans notre application, l'utilisation du patron Composite s'avère cruciale pour la gestion efficace des commandes. Considérant la complexité croissante des structures de commandes, avec des commandes simples (SingleOrder) et des regroupements de commandes (GroupOrder, MultipleOrder), le patron Composite offre une approche unifiée pour les traiter tous comme des objets simples. La classe Order agit en tant que composant de base (Component), définissant les opérations communes à toutes les commandes, qu'elles soient simples ou groupées. Les classes SingleOrder représentent les feuilles (Leaf) de notre structure, chacune contenant des détails spécifiques d'une commande individuelle.D'autre part, les classes GroupOrder et MultipleOrder agissent en tant que composites, pouvant contenir à la fois des SingleOrder et d'autres groupes d'ordres. Ainsi, le patron Composite facilite la manipulation homogène de ces structures arborescentes, que ce soit pour effectuer des opérations sur des commandes individuelles ou sur des ensembles plus vastes de commandes regroupées. En résumé, le patron Composite se révèle être un choix pertinent pour notre architecture, offrant flexibilité, clarté et extensibilité.



4.3) Autres patrons appliqués :

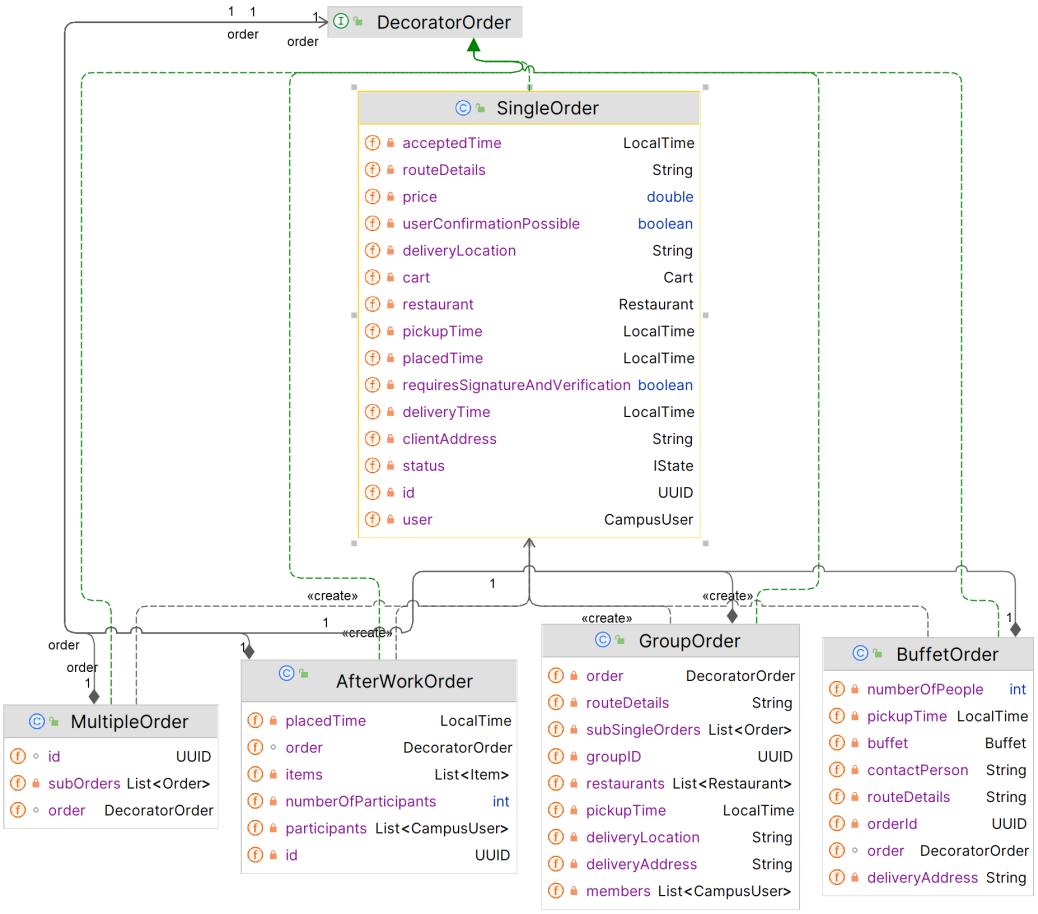
4.2.1) Singleton :

Le Singleton est un patron de conception garantissant qu'une classe n'a qu'une seule instance et fournissant un point d'accès global à cette instance. donc notre architecture nous avons utilisé pour centraliser le contrôle sur la classe RestaurantService, afin d'éviter la création excessive d'instances et de partager un état commun entre les différents composants du système



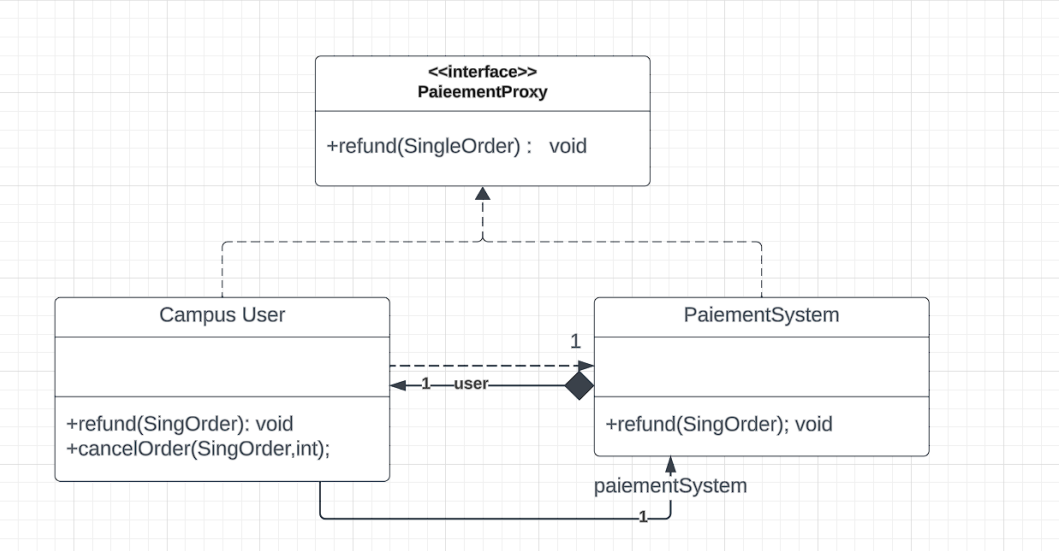
4.2.2) Decorator :

Le patron Decorator offre la possibilité d'ajouter dynamiquement des fonctionnalités supplémentaires à un objet en utilisant des classes de décorateurs. Nous l'avons appliqué pour unifier la logique commune de tous les types de commandes tout en permettant l'ajout flexible de fonctionnalités spécifiques à chaque type de commande



4.2.3) Proxy :

Le patron Proxy permet de contrôler l'accès à un objet en agissant comme une interface intermédiaire. Il est utilisé pour des raisons telles que le contrôle d'accès, la gestion de la mémoire ou la mise en cache, offrant ainsi une couche d'abstraction supplémentaire entre le client et l'objet réel.



4.4) patrons peuvent être intégré :

Factory nous aurions pu appliquer ce patron puisque le patron Factory permet de déléguer la création d'objets à des sous-classes. puisque notre utilisateur peut être de types différents et peut créer des commandes de types différentes. Dans notre situation, ce choix s'explique par la simplicité des types d'utilisateurs et de commandes dans notre application. Intégré le patron Factory aurait ajouté une couche de complexité qui n'était pas nécessaire pour le projet.

5) Qualité des codes et gestion de projets

Types de Tests, Qualité et Couverture

Dans notre projet, nous avons adopté une approche rigoureuse en matière de tests, en privilégiant les tests Gherkin. Pour assurer un suivi précis de la qualité de nos tests, nous utilisons SonarQube, un outil d'analyse de qualité de code renommé. Actuellement, notre projet atteint 72% de couverture de tests, une légère baisse par rapport à un pic précédent de 82%. Cette diminution est principalement due à l'intégration de divers design patterns qui, tout en enrichissant l'architecture de notre application, ont temporairement réduit notre couverture de tests. Nous évaluons la qualité de nos tests non seulement par leur couverture mais aussi par leur capacité à détecter des régressions et à valider de nouvelles fonctionnalités de manière fiable. Notre objectif est de maintenir et d'améliorer cette couverture, visant constamment un seuil de 80% pour garantir une base de code robuste et fiable. Nous sommes particulièrement fiers des tests concernant le paiement ou encore la gestion du panier qui fonctionne bien et qui couvre un grand nombre de scénarios.

Vision de la Qualité du Code

Nous adhérons à des standards élevés en matière de qualité du code. Cela implique l'adhésion à des principes de programmation propre, l'utilisation de commentaires pertinents pour une meilleure compréhension et maintenance, et le respect strict des conventions de codage. Nous nous assurons que le code maintient un haut niveau de qualité en effectuant des revues de code régulières et en encourageant une culture de feedback constructif au sein de l'équipe. De plus, l'intégration de l'analyse statique du code via SonarQube nous aide à identifier et à corriger rapidement les vulnérabilités potentielles, les bugs et les mauvaises pratiques de codage.

Gestion du Projet et Automatisation

Notre cycle de vie du développement logiciel est géré de manière agile, avec une intégration continue et un déploiement continu (CI/CD) via GitHub Actions. Chaque push ou pull request sur la branche `development` déclenche la CI, qui comprend la récupération du code, la configuration de JDK 17, la compilation avec Maven, l'exécution des tests, et une analyse SonarQube. Initialement, nous avions intégré la validation de SonarQube dans le CI pour la branche `development`, mais un bug avec SonarQube nous a poussés à simplifier la CI, en déplaçant cette étape sur la branche `main`.

Concernant la gestion des branches, nous suivons la méthodologie Git Flow. Chaque développeur travaille sur sa propre branche (par exemple, `dev-kaleb`, `dev-sarra`), et les fonctionnalités sont fusionnées dans `development` après une phase de test complète et une revue de code approfondie. La branche `main` est réservée au code stable, avec des tags pour marquer chaque version publiée. Cette approche nous permet de maintenir un flux de travail organisé, de minimiser les conflits de fusion, et de garantir la stabilité de notre base de code.

6) Rétrospective et Auto-évaluation:

6.1)Rétrospective:

Architecte (FAHRAT Melek):

Mon rôle en tant qu'architecte a été crucial pour assurer la cohérence et la robustesse de la structure logicielle. J'ai supervisé, en collaboration avec mon équipe, l'établissement de l'architecture en élaborant les diagrammes de classes, d'activité, de Use Case, et de séquence. J'ai également suivi de près l'évolution de l'architecture et de la structure du projet, en particulier avec l'introduction des patrons de conception que j'ai personnellement veillé à mettre en place. Cette expérience a été enrichissante, me permettant de découvrir les responsabilités variées d'un architecte logiciel. J'ai tiré des leçons précieuses de la gestion dynamique d'une application, de son initiation à sa réalisation, et du fait que j'ai assuré que l'architecture mise en place était solide et adaptée aux besoins du projet. La mise en place des patrons a été une réussite, simplifiant la

maintenance et améliorant la lisibilité du code, tout en mettant en lumière l'importance d'une architecture solide pour la pérennité du projet. Cependant, j'ai noté que la communication continue avec les membres de l'équipe est cruciale. Des échanges réguliers ont facilité la compréhension mutuelle des besoins et ont permis d'ajuster la conception en fonction des retours des développeurs.

Product Owner (ZENKRI Sarra):

En tant que Product Owner, j'ai assumé la responsabilité de cristalliser la vision du produit et de transmettre cette vision à l'équipe de développement. J'ai veillé à ce que l'application soit gérée avec une précision qui garantit que chaque User Story soit non seulement bien formulée et priorisée mais aussi parfaitement intégrée par l'équipe. Mon rôle s'est étendu à être le médiateur essentiel entre les attentes des parties prenantes et la capacité d'exécution de nos développeurs, m'assurant ainsi que les fonctionnalités réalisées soient en adéquation totale avec les besoins métiers.

Ce rôle m'a permis d'affiner mes compétences décisionnelles, spécialement dans la sélection des User Stories les plus pertinentes. L'adaptabilité et la réactivité face aux changements imprévus ont été des compétences clés que j'ai développées, me permettant de garder le cap sur notre stratégie globale sans perdre de vue les besoins immédiats.

J'ai également reconnu l'importance vitale d'une communication proactive avec l'équipe de développement. Les briefings réguliers et les sessions de clarification, essentiels à notre progression, ont été d'autant plus efficaces grâce à l'accompagnement de nos professeurs. Leur guidance avisée a été cruciale pour nous garder alignés avec nos objectifs et pour prévenir tout écart significatif dans le développement du produit. Ces interactions ont renforcé la nécessité d'une écoute active et d'une collaboration étroite pour atteindre les objectifs fixés. Cette expérience a été un véritable renforcement de la fonction de PO, affirmant mon rôle dans la maximisation de la valeur du produit et dans le soutien de la vision à long terme du projet.

Ops(SIKA Kaleb Dassou Ablam) :

En tant que l'Ops de l'équipe, ma mission s'est concentrée sur l'optimisation de notre pipeline CI/CD et la maintenance des environnements de développement et de production. Voici les points clés :

Réussites :

- Mise en place de la CI/CD: J'ai efficacement intégré GitHub Actions, améliorant notre automatisation.

- Maintenance des Environnements : J'ai assuré la sécurité et la performance des environnements, facilitant un développement fluide.

Leçons Apprises :

- Adaptabilité aux Outils: Face aux problèmes avec SonarQube, j'ai appris à m'adapter rapidement et à trouver des solutions alternatives.

- Communication Inter-Équipes : La collaboration avec les développeurs a souligné l'importance d'une communication efficace pour aligner les opérations avec les besoins de développement.

Erreurs et Améliorations :

- Gestion du Changement : Le retrait de SonarQube de la CI a été un défi, révélant le besoin d'une meilleure évaluation des alternatives.

Mon rôle a été crucial pour la gestion technique du projet, avec des leçons importantes pour l'avenir.

QA (Clément CHATELAIN) :

En tant que Quality Assurance Engineer de l'équipe, ma mission était centrée sur la qualité du projet en termes de tests de fonctionnalité, de performance et de qualité du code.

Réussites : J'ai pu organiser la mise en place des tests et ainsi effectuer de nombreux tests mais permettre aussi à l'équipe de bien répartir les tests et que notre réflexion au niveau de la qualité et de la performance du code soit cohérente au sein de l'équipe. Cela a abouti à la mise en place de nombreux tests (133) et de nombreux scénarios associés.

Leçons Apprises : Évidemment, la découverte des tests Cucumber avec les scénarios/features Gherkin était nouveau dans ce projet ainsi qu'appliquer réellement le test des fonctionnalités que l'on souhaite dans un premier temps et ensuite le code métier associé. De plus, avoir une responsabilité sur un aspect du code était assez nouveau. Cela force la prise de recul et la compréhension globale du fonctionnement de l'équipe dans cet aspect et de mettre en place une réflexion commune. J'ai aussi beaucoup appris, et l'équipe au global aussi, au respect d'un schéma type pour le développement du code en commençant par la création d'une User Story à laquelle on associe un ou plusieurs scénarios que nous viendrons ensuite tester à l'aide de Gherkin/Cucumber et finir par implémenter le code métier qui valide ces tests.

Erreurs et Améliorations : Au début du projet, l'équipe s'est précipitée à écrire des tests sans suivre le schéma type que j'ai énoncé plus haut et j'ai mal géré cet aspect de conception. Cela nous a impacté sur le temps mis à adapter notre code déjà implémenté après l'écriture des User Story et le fait qu'on se rende compte de leur importance.

Mon rôle était vraiment important pour la gestion des tests et de la performance/qualité du code. Les leçons apprises me permettent une prise de recul plus importante ainsi que d'avoir une vision plus globale sur l'ensemble de la partie code/tests du projet.

6.2) Bilan :

En utilisant la méthodologie Git Flow pour notre gestion de projet, nous avons adopté une approche structurée pour la répartition et la gestion des tâches. Chaque semaine, nous avons effectué une planification prévisionnelle à l'oral pour définir les tâches de la semaine suivante, chacune correspondant à une ou plusieurs issues, que nous considérons comme des User Stories (US) dans notre contexte. Au début de chaque séance de travail, nous faisions le point sur l'avancement des tâches assignées la semaine précédente. Cette réunion nous permettait également de résoudre collectivement les bugs et problèmes techniques rencontrés par les membres de l'équipe. Concernant la rédaction de notre rapport, les responsabilités ont été clairement définies selon les rôles de chacun dans l'équipe :

- L'Architecte a pris en charge la création des diagrammes et ainsi la présentation des design pattern.
- L'OPS (Opérations) a rédigé la section concernant la gestion du projet et l'utilisation de Git, en soulignant comment Git Flow a facilité notre processus de développement.
- Le QA (Assurance Qualité) a été responsable de tout ce qui concerne les tests et la qualité du code, assurant que nos pratiques de développement respectaient les normes de qualité élevées.
- La PO (Product Owner) a élaboré la partie dédiée aux User Stories, détaillant comment elles ont été définies et implémentées.

Pour les sections telles que le périmètre du projet ou la conclusion, nous avons travaillé de manière collaborative, chaque membre apportant son expertise pour enrichir ces parties du rapport. Cette collaboration a permis de garantir que toutes les perspectives étaient prises en compte et que le rapport reflétait fidèlement notre travail et nos réalisations en tant qu'équipe.

6.3) Auto-évaluation:

CHATELAIN Clément	Dassou Ablam Kaleb SIKA	FARHAT Melek	ZENKRI Sarra
95	95	100	95

7) Conclusion :

Nous sommes dans l'ensemble satisfaits du travail accompli. Grâce à ce projet, nous avons pu découvrir de nombreux aspects de conception et ainsi approfondir nos compétences dans l'utilisation des patrons qui nous ont permis de définir un langage commun pour aider notre équipe à communiquer plus efficacement. Nous avons pu ainsi justifier les choix de conception d'une application, et ainsi connaître mieux l'architecture de son projet avant de coder grâce aux normes et diagrammes UML.

De plus, ce projet nous a permis de découvrir des nouvelles méthodes agiles telles que : T-shirt-sizing et MoSCoW qui nous a permis de pondérer nos user stories et ainsi bien répartir les tâches entre les différents membres de l'équipe, pour permettre de développer les fonctionnalités nécessaires au cours des sprints.

Avec du recul, ce projet nous a bien fait mûrir, le fait de coder tous ensemble a été une nouvelle expérience dans l'organisation du travail. Les nouvelles fonctionnalités découvertes nous seront d'une grande aide pour nos futurs projets et nous saurons surtout mieux les utiliser pour développer de manière plus saine. Ce projet nous a permis une grande prise de recul chacun selon son rôle et de comprendre l'intérêt de chaque rôle au sein d'une équipe.

Nous tenons également à exprimer notre gratitude envers nos professeurs pour leur soutien et leurs conseils tout au long de ce projet. Leur aide a été un atout précieux dans notre parcours d'apprentissage et de développement.