# Rapport – Conception Logicielle

# Equipe J

**Software Architect :** Adote Adjo Emmanuelle
**Quality Assurance Engineer :** Chloé Vandenbrulle
**Continuous Integration and Repository Manager :** Nicolas Zanin
**Business Analyst and Product Owner :** Arrigoni Guillaume

Année scolaire 2023-2024

## I - Périmètre fonctionnel : Hypothèses, Limites, Extensions, Points Forts et Points Faibles

# I - Périmètre fonctionnel : Hypothèses, Limites, Extensions, Points Forts et Points Faibles

## I.1 - Hypothèses de Travail (E.4)

On considère que :

- Les utilisateurs ont une connexion stable et fiable lors de l'utilisation de l'application.
- Un nombre de livreurs toujours suffisant quel que soit le nombre de commandes.
- Les restaurateurs sont fiables au niveau de la gestion de leur établissement que ce soit pour les horaires, les menus, et la disponibilité.
- Les adresses de livraisons sont prédéterminées par l'application afin d'éviter une adresse inexistante.
- Chaque commande formulée, chaque préparation de commande et chaque livraison s'effectue selon un créneau de 10 minutes.
- Les API de paiement sont simples, faciles d'utilisation et que leurs intégration soit similaire d'une API à une autre.
- Chaque sous-commande appartient à un seul utilisateur.
- Le temps de préparation de chaque menu est identique.

## Limites identifiées :

Le projet ne couvre pas les cas suivants :

- L'optimisation des livraisons en fonction du trafic routier (l'optimisation est couverte uniquement dans le cas idéal d'un trafic fluide).
- L'interface graphique de l'application.
- La gestion des paiements hormis la validation du paiement (l'utilisateur valide sa commande).
- Résolution des conflits liés à l'échec de la préparation d'une commande par un restaurant.

## I.2 - Extensions choisies et éléments spécifiques

Parmi les extensions proposées, voici celles que nous avons implémentées.

- EX1 (requise) : Diversité des commandes
- EX2 : Ristournes des restaurateurs
- EX7 : Système de recommandations
  Le système de recommandations fonctionne de la même manière dans les 3 classes Restaurant, CompteLivreur et CompteUtilisateur. Ils ont, en attribut, une liste de notes qui leur ont été attribuées (pour CompteUtilisateur c'est deux listes, une pour le retard et une pour l'amabilité). A partir de cette liste, la méthode getNote calcule la moyenne de toutes ces notes. Enfin les méthodes pour attribuer les notes sont dans les classes respectives en fonction de qui peut noter qui. C'est-à-dire :
    - L'utilisateur du Campus peut noter le livreur et le restaurant
    - Le livreur peut noter l'utilisateur du campus.

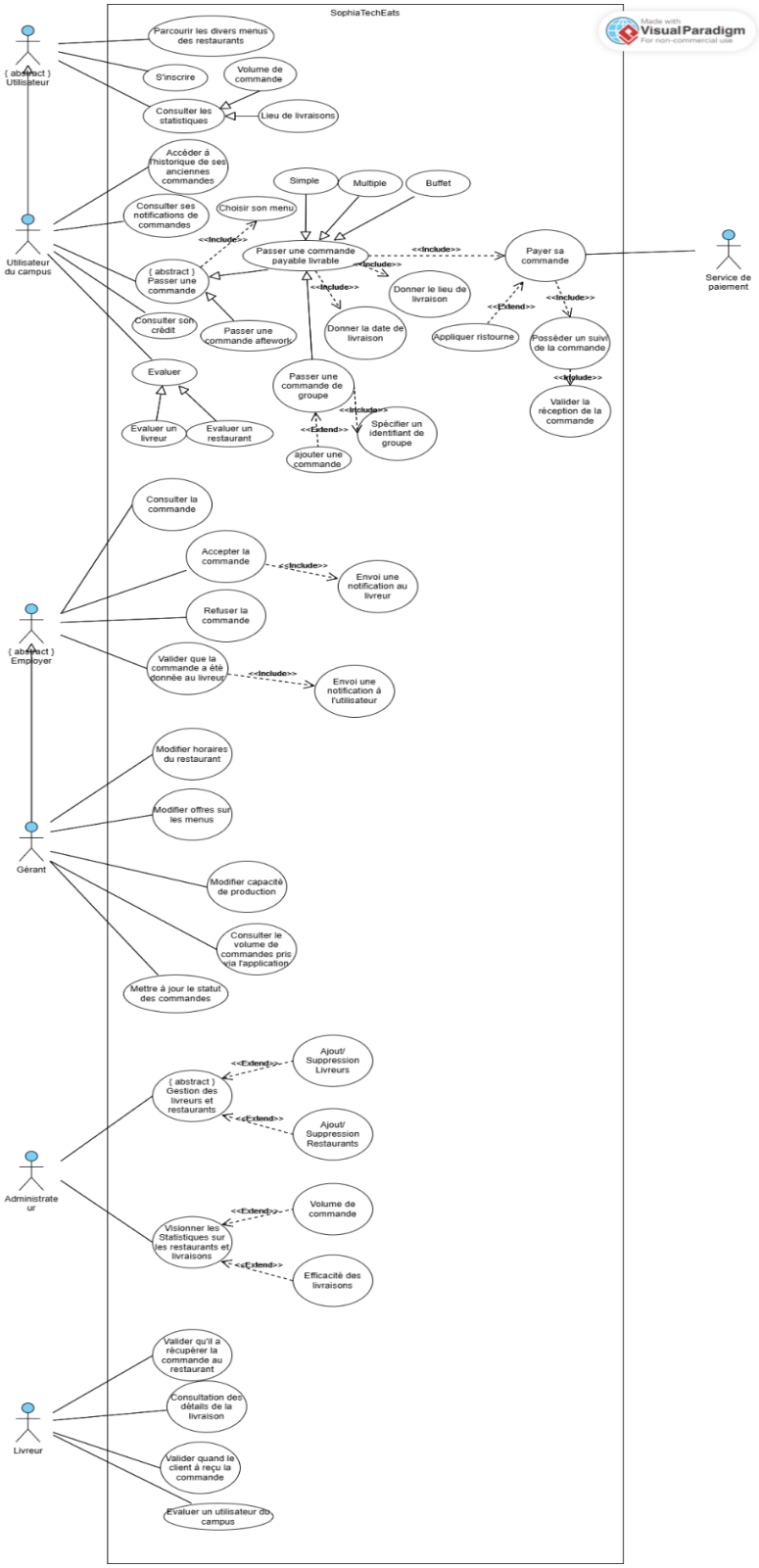
## I.3 - Points non implémentés relativement la spécification et des extensions choisies

Une partie non implémentée est de donner des informations précises sur les notifications.
De plus, nous avons remarqué un manque de tests pour quelques parties du code :

- Les notifications : seulement un scénario pour la notification du côté Utilisateur et un scénario également pour la notification du côté Livreur.
- Les statistiques : de même, seulement un scénario pour les statistiques sur les utilisateurs et un scénario pour les statistiques sur les restaurants.
- La vérification des statuts de l'utilisateur.

# II - Conception

## II.1 - Le diagramme de cas d'utilisation et User Stories



Créer une commande simple #23

- **En tant qu'**utilisateur enregistré, **Je veux** pouvoir créer et ajouter une commande simple **afin d'** être livré.

- Créer une commande multiple #150
  - **En tant que** utilisateur, **Je veux** créer une commande Multiple **afin de** prendre une commande dans plusieurs restaurant
- Créer une commande de groupe #88
  - **En tant que** utilisateur, **Je veux** créer une commande groupe **afin de** faire des commandes avec d'autres personnes
- Créer une commande buffet #144
  - **En tant que** utilisateur, **Je veux** créer une commande buffet **afin de** faire des commandes plus diversifiés
- Créer une commande AfterWork #126
  - **En tant que** usager commandeur, **Je veux** faire une commande afterwork **afin d'**avoir un choix plus varié.
- Evaluer un restaurant (*par un usager*) #167
  - **En tant** qu'utilisateur, **je veux** évaluer le ou les restaurants m'ayant servi **afin de** permettre aux autres utilisateurs d'être renseigné sur la qualité des commandes.
- Evaluer un livreur (*par un usager*) #168
  - **En tant** qu'utilisateur, **je veux** évaluer le livreur m'ayant servi **afin de** permettre aux autres utilisateurs d'être renseigné sur la qualité de la livraison
- Evaluer un utilisateur du campus (*par un livreur*) #169
  - **En tant** que livreur, **je veux** évaluer le ou les utilisateurs que j'ai servi **afin de** permettre aux autres livreurs d'être renseigné sur le type de client auquel ils doivent faire face.
- Réduction restaurant en fonction de la fidélité du client #44
  - **En tant que** restaurateur, **Je veux** que les clients ayant effectué plus de X commandes bénéficient d'une réduction temporaire de Y% pendant Z jours **afin d**'être plus attractif

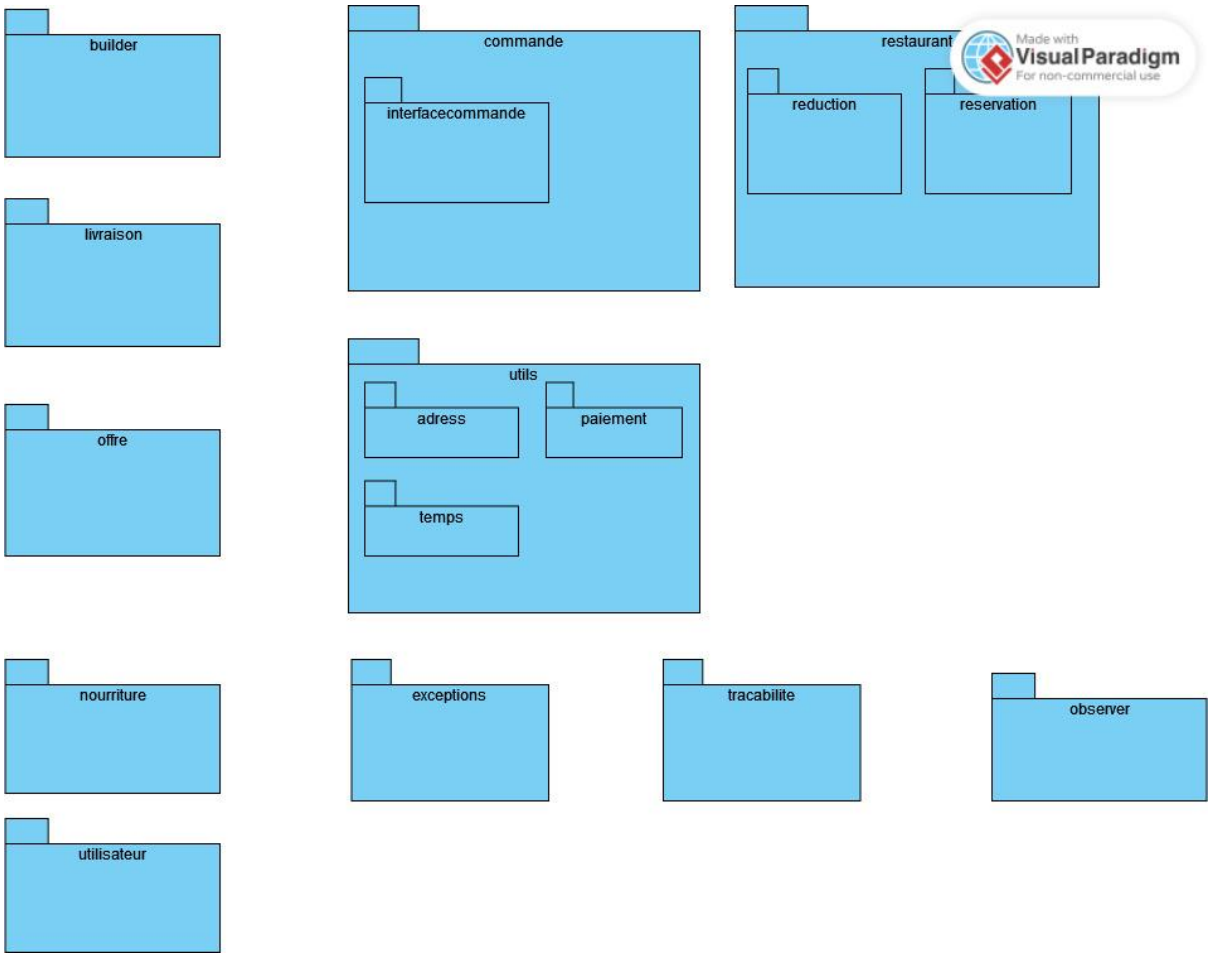
Extension non obligatoire : Réduction restaurant en fonction de la fidélité du client : (#44)

*Critère d'acceptation :*

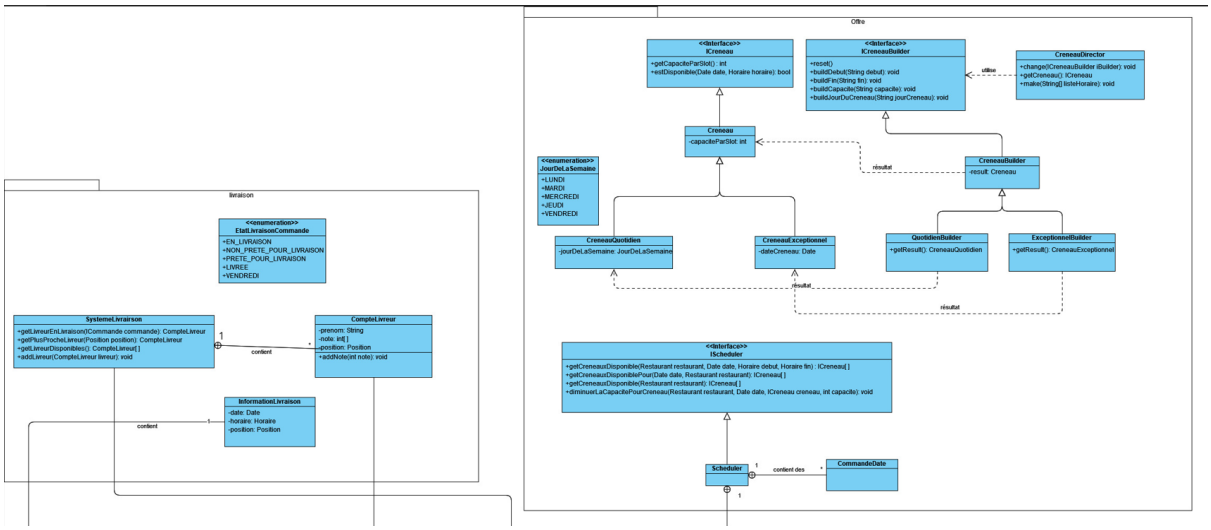
- **Etant donné** que je suis connecté
- **Et que** le restaurateur ADS propose une réduction de 5% pendant 10 jours au bout de 20 commandes effectuées dans le restaurant ADS
- **Et que** j'ai déjà effectué 19 commandes chez un restaurateur
- **Quand** je valide une nouvelle commande chez ADS
- **Alors** mes prochaines commandes chez ADS dans les 10 prochains jours auront une réduction de 5%

## II.2 - Le diagramme de classes

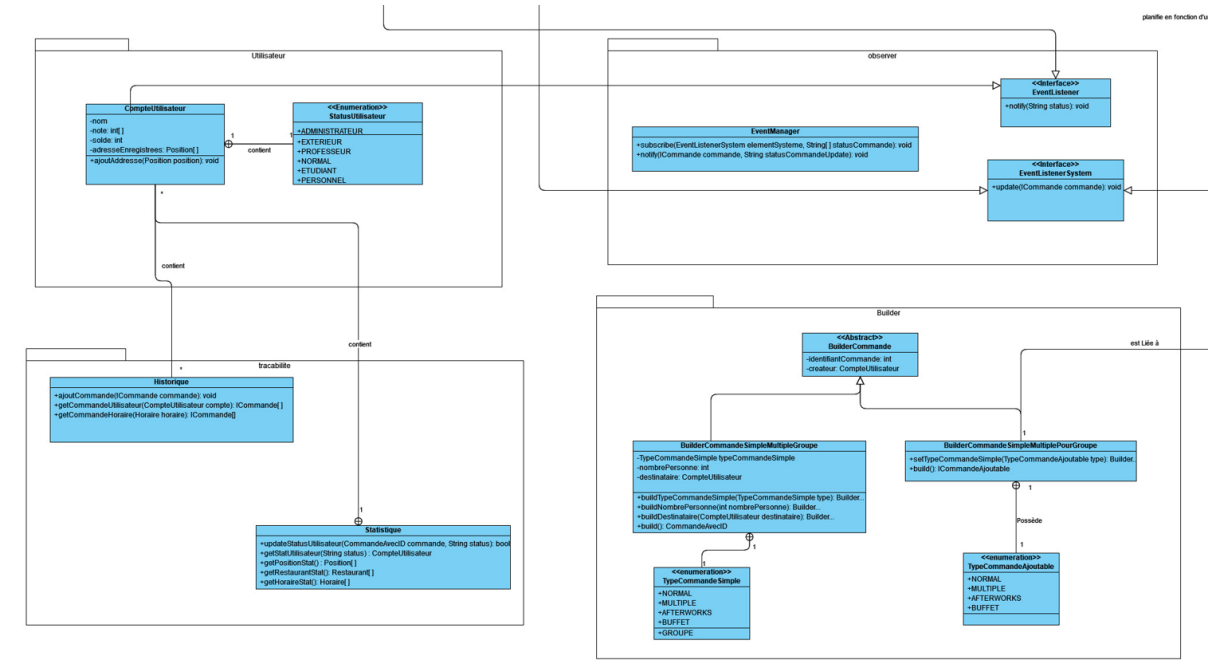
Voici la vue par package:




Voici les packages livraison et offres:

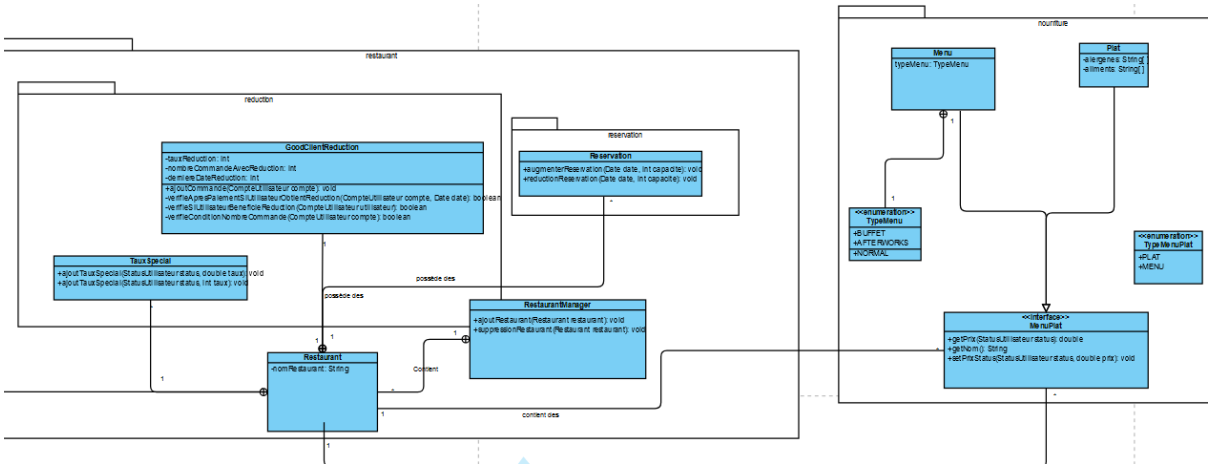


Voici les packages utilisateur, observer, builder et tracabilite:

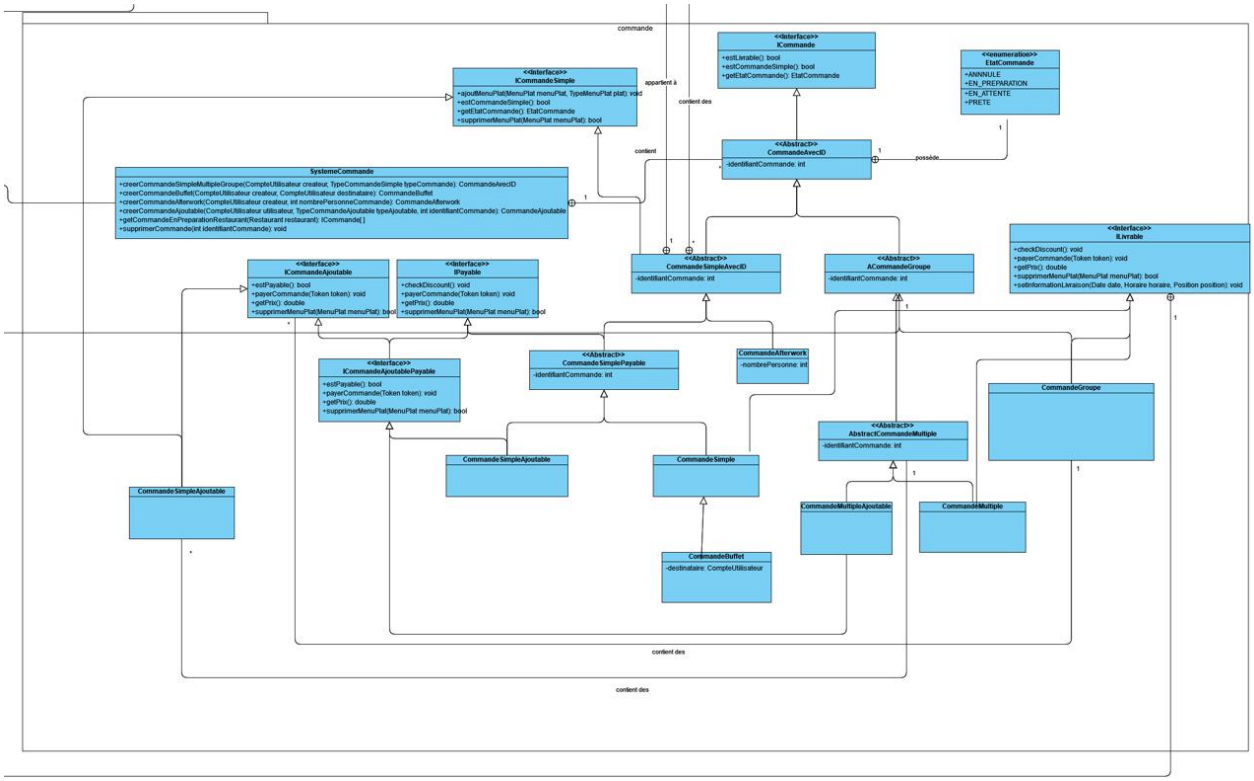



---

Voici les packages restaurant et nourriture:

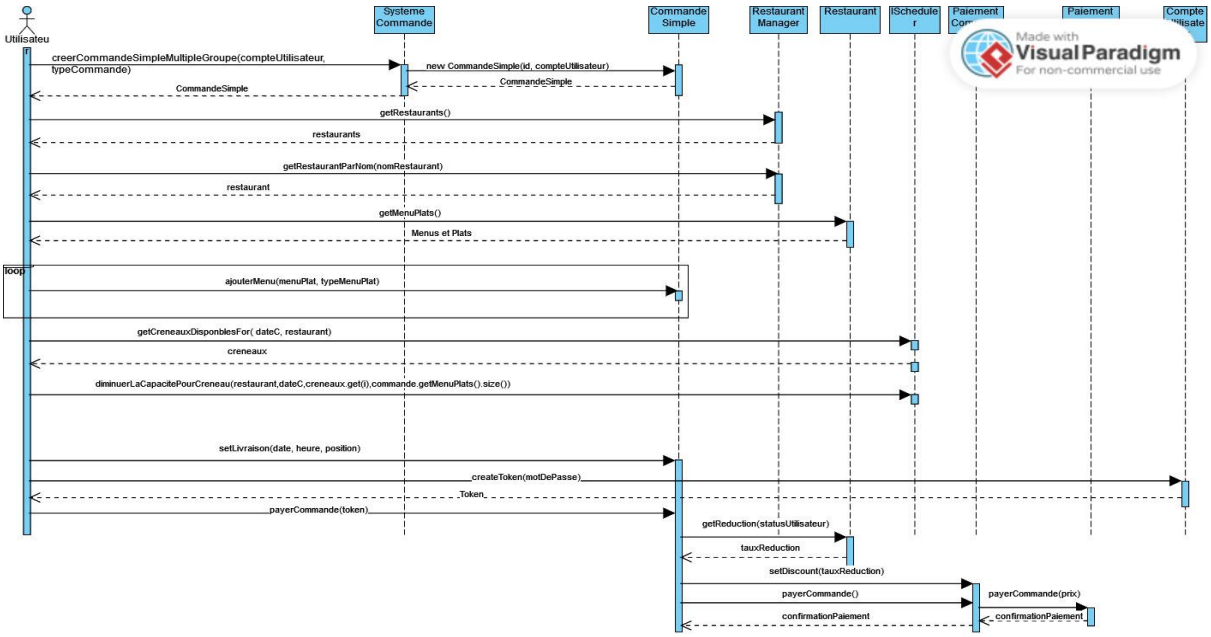


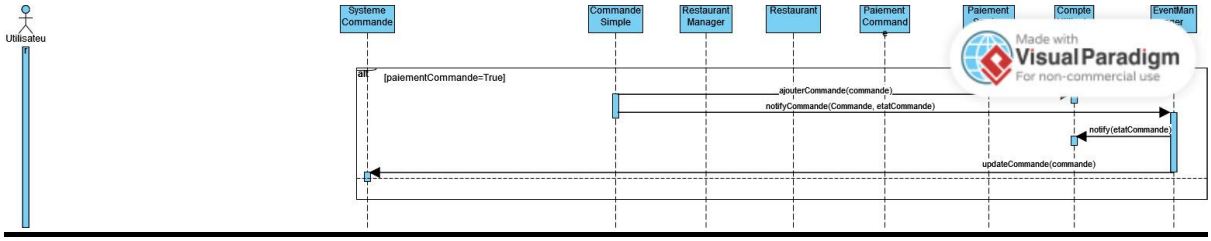

Voici le package commande:


commande
<<Interface>>
ICommande
+estLivrable(): bool
+estCommandeSimple(): bool
+getEtatCommande(): EtatCommande
<<enumeration>>
EtatCommande
+ANNNULE
+EN_PREPARATION
+EN_ATTENTE
+PRETE
<<Interface>>
ICommandeSimple
+ajoutMenuPlat(MenuPlat menuPlat, TypeMenuPlat plat): void
+estCommandeSimple(): bool
+getEtatCommande(): EtatCommande
+supprimerMenuPlat(MenuPlat menuPlat): bool
appartient à
contient des
1
contient
<<Abstract>>
CommandeAvecID
-identifiantCommande: int
possède
SystemeCommande
+creerCommandeSimpleMultipleGroupe(CompteUtilisateur createur, TypeCommandeSimple typeCommande): CommandeAvecID
+creerCommandeBuffet(CompteUtilisateur createur, CompteUtilisateur destinataire): CommandeBuffet
+creerCommandeAfterwork(CompteUtilisateur createur, int nombrePersonneCommande): CommandeAfterwork
+creerCommandeAjoutable(CompteUtilisateur utilisateur, TypeCommandeAjoutable typeAjoutable, int identifiantCommande): CommandeAjoutable
+getCommandeEnPreparationRestaurant(Restaurant restaurant): ICommande[ ]
+supprimerCommande(int identifiantCommande): void
<<Interface>>
ICommandeAjoutable
+estPayable(): bool
+payerCommande(Token token): void
+getPrix(): double
+supprimerMenuPlat(MenuPlat menuPlat): bool
<<Interface>>
IPayable
+checkDiscount(): void
+payerCommande(Token token): void
+getPrix(): double
+supprimerMenuPlat(MenuPlat menuPlat): bool
<<Abstract>>
CommandeSimpleAvecID
-identifiantCommande: int
<<Abstract>>
ACommandeGroupe
-identifiantCommande: int
<<Interface>>
ILivrable
+checkDiscount(): void
+payerCommande(Token token): void
+getPrix(): double
+supprimerMenuPlat(MenuPlat menuPlat): bool
+setInformation(Livraison Date date, Horaire horaire, Position position): void
<<Interface>>
ICommandeAjoutablePayable
+estPayable(): bool
+payerCommande(Token token): void
+getPrix(): double
+supprimerMenuPlat(MenuPlat menuPlat): bool
<<Abstract>>
CommandeSimplePayable
-identifiantCommande: int
CommandeAfterwork
-nombrePersonne: int
CommandeGroupe
<<Abstract>>
AbstractCommandeMultiple
-identifiantCommande: int
CommandeSimpleAjoutable
CommandeSimple
CommandeMultipleAjoutable
CommandeMultiple
CommandeBuffet
-destinataire: CompteUtilisateur
contient des
contient des

## II.3 - Le diagramme de séquence

Les trois éléments systèmes manquant sont Paiement Commande puis Paiement Service et CompteUtilisateur



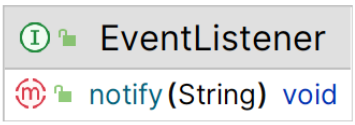

Ici il manque Paiement Commande, CompteUtilisateur et EventManager.

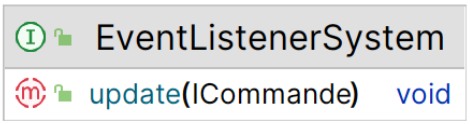
# III - Design Patterns appliqués ou pas

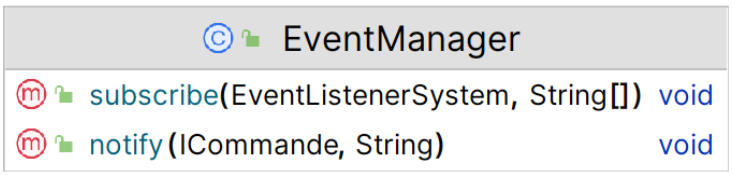
## III.1 - DPs en détail

2 design patterns ont été appliqué à ce projet, la design pattern builder et observer.

Le patron observer a été appliqué dans ce projet pour permettre soit de notifier les utilisateurs ou les livreurs soit de notifier le système comme le système de commande ou le système de livraison.
Cela permet que quand une commande change de statut, cela notifie soit le système de commande, soit le système de livraison en fonction de l'état de la commande et notifie toujours l'utilisateur du changement de statut de sa commande pour qu'il puisse savoir le statut de sa commande.



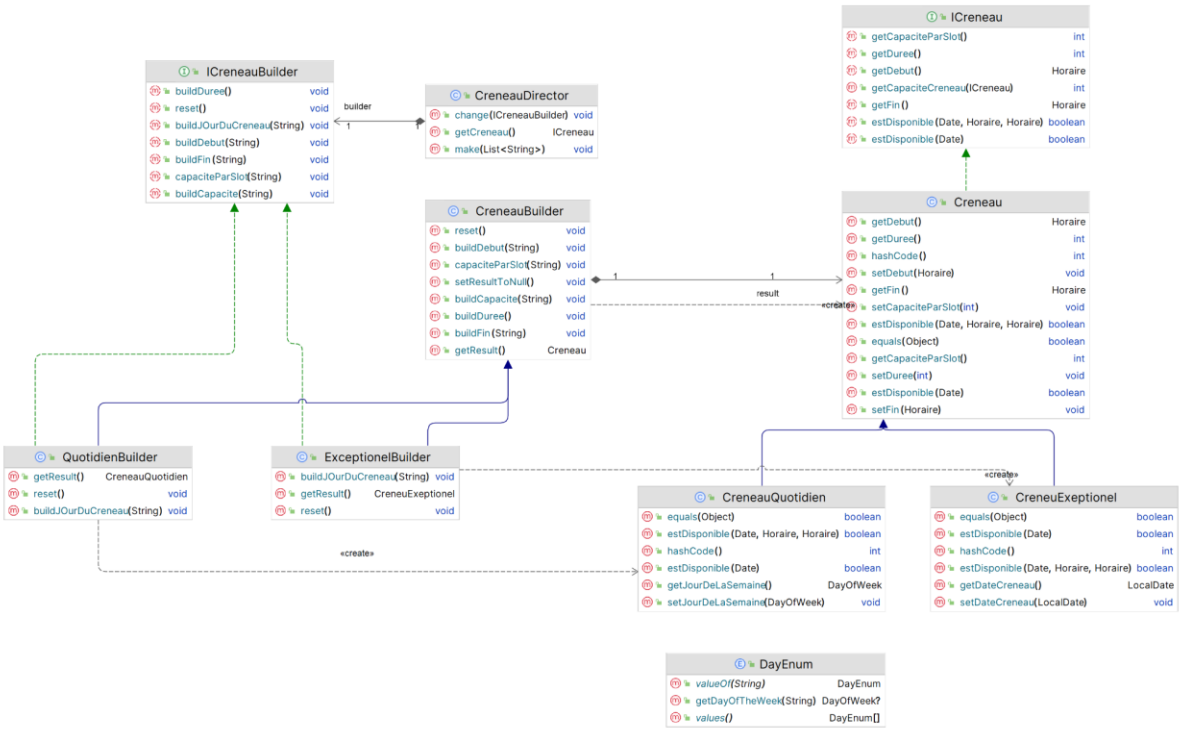




Ici, nous avons les classes EventListener et EventListenerSystem qui sont les observables, et EventManager l'observer.
Dans la méthode notify, on notifie les éléments du système en fonction de l'état de la commande, et notifie l'utilisateur ou le livreur du statut de la commande.
Les éléments du système seront notifiés en fonction des états de commandes spécifiés dans la méthode subscribe.

Le patron builder à été appliqué dans ce projet pour les créneaux d'une commande.
On utilise ce builder pour créer soit des Créneaux quotidiens, soit des créneaux exceptionnels. Un créneau quotidien est en fonction d'un jour dans la semaine, on aura une capacité du nombre de menus possible précise tout au long de la journée alors que le créneau exceptionnel permet de créer une capacité spécifique à une tranche horaire choisie.
On a décidé d'utiliser ce patron de conception car les deux créneaux ont une horaire de début et de fin, ainsi qu'une capacité, mais le créneau quotidien sera pour un jour de la semaine, alors qu'un créneau exceptionnel est pour une date précise.

Voici une représentation de notre patron de conception builder:




Donc dans ce patron, on a le créneau director qui permet de choisir un builder de créneau par la méthode change et de créer un créneau via la méthode make.

L'interface ICreneauBuilder permet de déclarer les étapes de construction d'un créneau donc avec un début, une fin et une capacité.
Et les deux différents produits QuotidienBuilder et ExceptionnelBuilder qui donneront des résultats différent, CreneauQuotidien pour QuotidienBuilder et CreneauExceptionnel pour ExceptionnelBuilder.

## III.2 - Autre DPs

Un  patron de conception qui nous paraît pertinent mais qu'on a pas appliqué est le patron state, ils nous parait pertinent de l'ajouter dans ce projet puisqu'il nous aurait permis de mieux implémenter les états d'une commande et donc d'avoir de meilleurs vérification lors des changements d'états d'une commande au lieu de le faire uniquement dans la classe Commande.

Nous n'avons pas implémenté ce patron puisque nous avons préféré mettre en priorité les deux patrons expliqués en III.1 car ce sont des patrons qui nous ont servi lors de l'implémentation du code alors que le patron state nous aurait permis.

# IV - Qualité des codes et gestion de projets

## Gestion du dépot Github:

En ce qui concerne la stratégie de branching, on a fait 2 branches “principales” main et développement, ainsi qu’une branche par fonctionnalité.
La branche main nous permet de donner les codes livrables et la branche développement nous permet d’ajouter le code pour chaque fonctionnalité implémentée.

Pour l’automatisation, nous avons le github action qui exécute le fichier maven.yml et qui vérifie si le projet compile bien et que les tests passent.
Ce fichier est exécuté lors des push ou pull request sur développement et main pour effectuer une intégration continue du projet.

De plus, nous avons découpé notre projet en quelques milestones pour respecter certaines fonctionnalités à implémenter en fonction d’un certain temps donné.

# V - Rétrospective et Auto-évaluation

## Rôle:

Ops: J'ai mis en place une intégration continue et la stratégie de branching sur le dépôt ainsi que la gestion des milestones et issues.
Pendant ce projet j'ai appris à mettre en place une intégration continue et donc appris comment l'utiliser (je me suis créer un dépôt personnel pour tester ceci avant de le faire sur le dépôt de ce projet)
Mes erreurs ont été de ne pas avoir mis en place plus vite la vérification des tests pour la CI car nous avons eu un problème lors d'une pull request qui ne passait pas alors que les tests étaient passés localement mais pas sur le dépôt.
Et de ne pas avoir appliqué la stratégie de branching assez vite, ce qui à provoqué quelques duplication et quelques conflits dans le dépôt lors du début de notre projet.

Archi: Depuis la première phase de conception, j'ai avec l'aide de mes camarades réalisé les différents diagrammes. J'ai vérifié la cohérence de l'architecture avec des outils comme CodeMR et MetricsTree. J'ai tout au long du développement fait attention à certaines métriques comme le manque de cohésion et l'index de maintenabilité de notre code. J'ai veillé à ce que les interfaces soient les plus petites possibles, privilégié l'extension à la modification des classes. Au moment de l'intégration des patterns j'ai proposé plusieurs possibilités de modélisations avec les diagrammes correspondants. Avec une seconde validation de mes camarades, les patterns validés ont été implémentés.

QA :
Au fur et à mesure que nous avancions dans l'implémentation des fonctionnalités, j'ai pu vérifier le pourcentage de coverage que nous couvrions avec nos tests grâce à l'outil SonarCube. Avec ces données, nous avons alors pu savoir quelles parties du code n'avaient pas été testées lors des tests Cucumber et analyser quels tests étaient importants, cohérents et prioritaires à ajouter afin d'améliorer notre pourcentage de coverage et donc de faire en sorte que notre code soit testé un maximum. A la fin du projet, nous avons un coverage de 71%. Une leçon que j'ai apprise est l'importance des tests scénarios qui permettent de vérifier le comportement du code sous forme de cas d'utilisation concrets. En effet, grâce aux scénarios nous pouvons réaliser des simulations de fonctionnalités, de situations dans lesquelles les usagers pourraient se trouver et vérifier ce qu'il se passe tout au long du scénario.

PO :

J'ai veillé tout au long du projet à ce que chaque implémentation soit accompagnée de code métier. Pour ce faire, l'une des méthodes que nous avons mises en place consiste à relier chaque fonctionnalité à une issue sur GitHub, taguée comme "user story". Avec l'aide de mes camarades, nous avons créé en priorité les user stories que nous jugions les plus importantes sur GitHub. Cela permettait à chacun de s'en attribuer une et de progresser sur le projet. Ces user stories ont également contribué à hiérarchiser l'importance de chaque fonctionnalité, favorisant ainsi une approche productive pour couvrir rapidement les fonctionnalités clés. En outre, l'estimation temporelle de chaque user story a facilité la répartition du travail entre les membres de l'équipe, évitant ainsi une surcharge. Il semble toutefois que certaines estimations aient été mal évaluées et ont dû être révisées à la hausse, en particulier pour l'ajout de nouvelles fonctionnalités de type "commande". Cela explique pourquoi cette dernière fonctionnalité a pris plus de temps à être mise en place malgré son importance élevée.

## Bilan de fonctionnement :

Au début du projet, l'équipe était assez coordonnée et communiquait assez correctement, ce qui nous permettait d'avancer correctement.
Puis par manque de communication et d'organisation, nous avons eu quelques retards sur certaines fonctionnalités, une fonctionnalité faite en double ainsi qu'une baisse de qualité de notre code et donc ce qui nous fait une grande différence sur les statistiques github entre les membres du groupe.

## Auto-Évaluation et Implication :

Pour la répartition des points entre les membres des équipes, on pense répartir 75 points pour la QA puisque malgré son implication sur les tests et par un problème de communication et un malentendu sur qui devait réaliser quelles fonctionnalités, des parties de ses codes n'ont pas été intégrés au projet
Ensuite 100 points pour le SA car elle nous as bien aidé pour effectuer nos architectures ou regarder les architectures faites par les autres membres du groupe.
Pour le PO, on lui réparti 125 points, puisqu'il nous as aider à produire quelques users story ainsi que pas mal de production de code.
Et le OPS, on lui a réparti les 200 points restants, car il s'est impliqué tout le long du projet de manière assez constante et as produit pas mal de code en suivant l'architecture réfléchi ensemble ainsi qu'une bonne gestion du git.