RAPPORT - STEATS

Equipe-R

Rapport UML : diagrammes et glossaire illustrant la conception et fonctionnalités

Equipe-R

AKHYATE Brahim (SA)

BEIDOURI Saad (Ops)

EL BETIOUI Jamel Eddine (PO)

MESAOURI DEBOUN Bilal (QA)

Sommaire

1. Périmètre fonctionnel

Hypothèses de Travail (E.4)

Nous avons fait des suppositions pour simplifier la création du système. L'enregistrement des utilisateurs sera libre et immédiat sans processus de vérification ou d'approbation. Les utilisateurs pourront commencer à passer des commandes dès la création de leur compte **US [S1].** Le personnel du restaurant recevra des notifications des commandes à préparer une fois le paiement validé. Il n'y aura pas de limite de temps pour la validation des commandes prêtes à être préparer **US [R1].** Les mises à jour des heures d'ouverture et des menus par les gérants de restaurant seront effectuées de manière autonome sans contraintes de fréquence ou de temps réel imposées par le système **US [R2].** Le système intégrera un mécanisme de paiement généralisé qui retournera une valeur booléenne indiquant si le paiement a été réalisé avec succès **US [P1].** Les commandes seront attribuées aux livreurs disponibles sur la base de la disponibilité et de la préparation des commandes, en s'assurant qu'une seule commande est attribuée par livreur à la fois **US [D1].** Les utilisateurs s'abonneront aux notifications et recevront des alertes comprenant l'ID de la commande, la date de livraison et le lieu via le système de notification intégré **US[D2].**

Extensions

EX1 - Diversité des commandes :

Nous avons choisi de la mettre en œuvre à l'aide de la pattern Strategie pour modulariser les différents types de traitement de commandes requis, en fonction de divers facteurs et des actions possibles, telles que le paiement ou la livraison.

EX3 - Diversité des menus :

L'extension "Personnalisation de Menu" offre aux utilisateurs la possibilité de personnaliser leurs menus en fonction de leurs préférences. Les utilisateurs peuvent choisir des sauces, desserts et add-ons spécifiques, ajustant ainsi le contenu de leur commande. La personnalisation inclut le calcul automatique du prix en fonction des choix effectués. Cette fonctionnalité vise à offrir une expérience culinaire plus personnalisée. Cependant, aucune information détaillée n'est actuellement disponible sur les choix de personnalisation, étant donné que chaque menu peut avoir des options différentes.

EX7 - Système de recommandations :

Permet aux utilisateurs d'évaluer les restaurants et les livreurs, ajoutant ainsi une dimension sociale et de qualité de service au système. La moyenne de ces notes sera calculée et présentée comme indicateur de la qualité de service. La limitation actuelle réside dans l'absence de commentaires détaillés qui pourraient

fournir un retour qualitatif plus nuancé et spécifique sur l'expérience des utilisateurs.

Limites Identifiées (G.6)

Notre projet ne s'occupera pas de l'optimisation des livraisons en fonction du trafic pour les restaurants hors campus, ni de la création de différentes interfaces graphiques. Il se concentrera uniquement sur la validation des paiements. Les scénarios où un restaurant ne peut pas préparer une commande après l'avoir acceptée ne sont pas non plus couverts par notre système.

Points Non Étudiés

Nous reconnaissons des défis tels que la gestion d'un grand nombre de commandes en même temps et la disponibilité changeante des restaurants. Des problèmes techniques possibles, comme les pannes de serveur ou les problèmes de connexion Internet, ne sont pas complètement explorés dans notre projet. Nous sommes conscients de ces limites et chercherons des moyens de les surmonter à mesure que le projet progresse.

2. Conception

Glossaire

En complément du glossaire fourni dans la description du projet, nous avons identifié les définitions suivantes afin de maintenir une compréhension commune et partagée du projet en cours, ainsi qu'avec les parties prenantes et les clients.

Termes supplémentaires et leurs définitions :

1. **Order:** Est-ce qu'un utilisateur sélectionne et demande à recevoir, comme un ensemble de plats à partir d'un menu. Il s'agit de la phase initiale où le client exprime son choix et le soumet pour traitement.
2. **Order Volume:** Désigne la quantité totale de commandes actuellement en traitement dans le système.
3. **Delivery:** Fait référence à la phase où l'ordre a été préparé conformément aux spécifications du client et est prêt à être transporté.
4. **Schedule:** Une représentation de l'agenda d'un restaurant, détaillant les heures d'ouverture et de fermeture ainsi que les créneaux horaires spécifiques durant lesquels des services peuvent être fournis.
5. **Cart :** Un composant qui contiendra une sélection de menus d'un restaurant spécifique, avec la quantité désirée pour chaque menu.

6. **Registry : U**n composant logiciel qui agit comme un référentiel centralisé permettant d'enregistrer, de récupérer et parfois de gérer des instances d'objets spécifiques au sein d'une application.
7. **Repository :** Un Repository est un élément logiciel qui sert de point central pour l'enregistrement, la récupération et parfois la gestion d'instances spécifiques d'objets au sein d'une application.
8. **Time slot :** Représente un créneau horaire pour un restaurant, agissant comme un référentiel pour enregistrer et récupérer des instances de créneaux horaires spécifiques au sein de l'application.

DIAGRAMME DE CAS D'UTILISATION ET US :

1. Diagramme de UC :

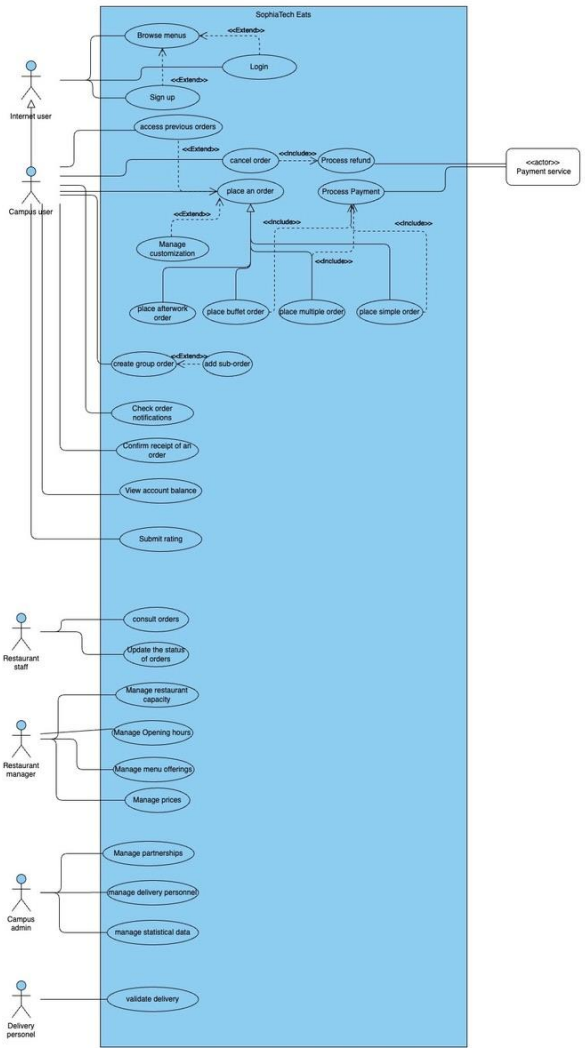
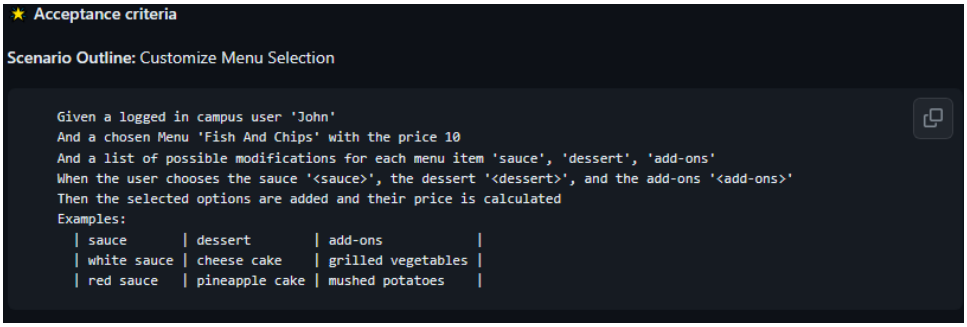
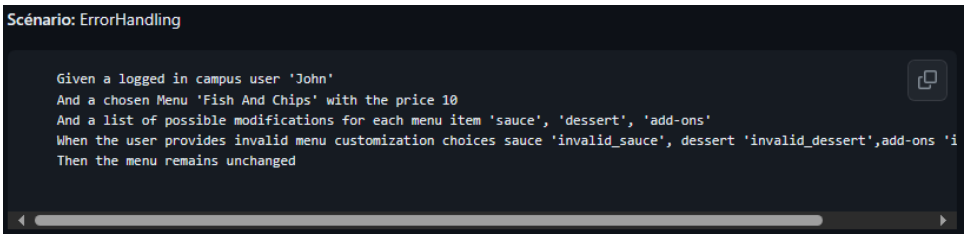


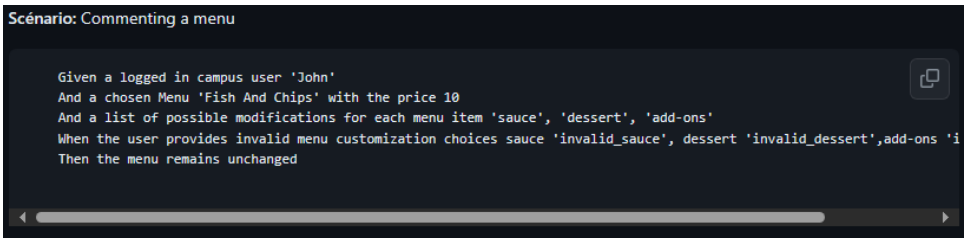
Figure 1 - Diagramme de cas d'utilisation

2.US (extension) :

- EX1 : diversité des commandes
    - **Nous avons implémenté tous les types de commandes requis, mais seulement un test Cucumber en tant qu'exemple représentatif, celui d'une commande de buffet, en raison de contraintes de temps. Voici l'histoire utilisateur pour la commande de buffet :**
        - **As** a Campus User
        - **I** want to choose and order a buffet menu from a restaurant on campus
        - **So that** I can have it delivered to a specific location at a designated time for group events.
- EX3 - personnalisation des menus :
    - **Description :**
      As a campus user, **I want** to modify some menu items **so that** the menu fit my preferences.
    - **Critères d'acceptation:**







Référence GitHub : https://github.com/PNS-Conception/ste-23-24-equipe-r/issues/22

- EX7 - Système de recommandations :

- Description :
  As a Campus user, **I want to** rate a restaurant or a delivery person **so that I** can provide feedback to help improve the overall dining and delivery experience.
- **Critères d' acceptation:**

  Feature: Rating and View Restaurant Rating
  Scenario: CampusUser1 rates Restaurant

  Given Restaurant A has a "<list>" of rating.
  When the CampusUser1 rates the restaurant 5 out of 5
  Then the rating of this restaurant should be 4.0 out of 5

  Scenario: CampusUser2 Checks Restaurant Rating

  Given Restaurant A has a "<list>" of rating.
  When the CampusUser2 checks the rating of the restaurant
  Then the rating of this restaurant should be 3.6 out of 5

  Feature: Rating and View Delivery Person Rating
  Scenario: CampusUser1 rates a Delivery Person

  Given Delivery Person DP1 has a "<list>" of rating.
  When the CampusUser1 rates the delivery person with 5 out of 5
  Then the rating of this delivery person should be 4.0 out of 5

  Scenario: CampusUser2 Checks Delivery Person Rating

  Given Delivery Person DP1 has a "<list>" of rating.
  When the CampusUser2 checks the rating of the delivery person
  Then the rating of this delivery person should be 3.6 out of 5

Référence GitHub : https://github.com/PNS-Conception/ste-23-24-equipe-r/issues/15

DIAGRAMME DE SÉQUENCE :

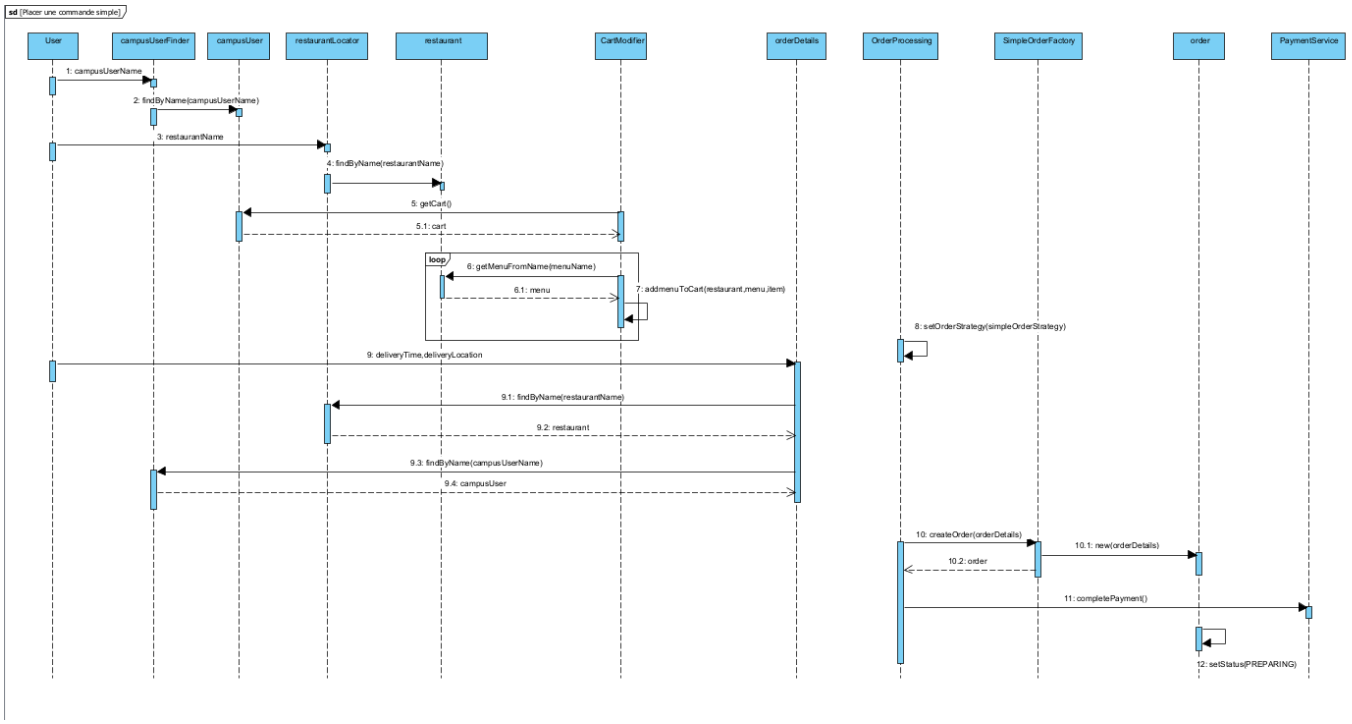
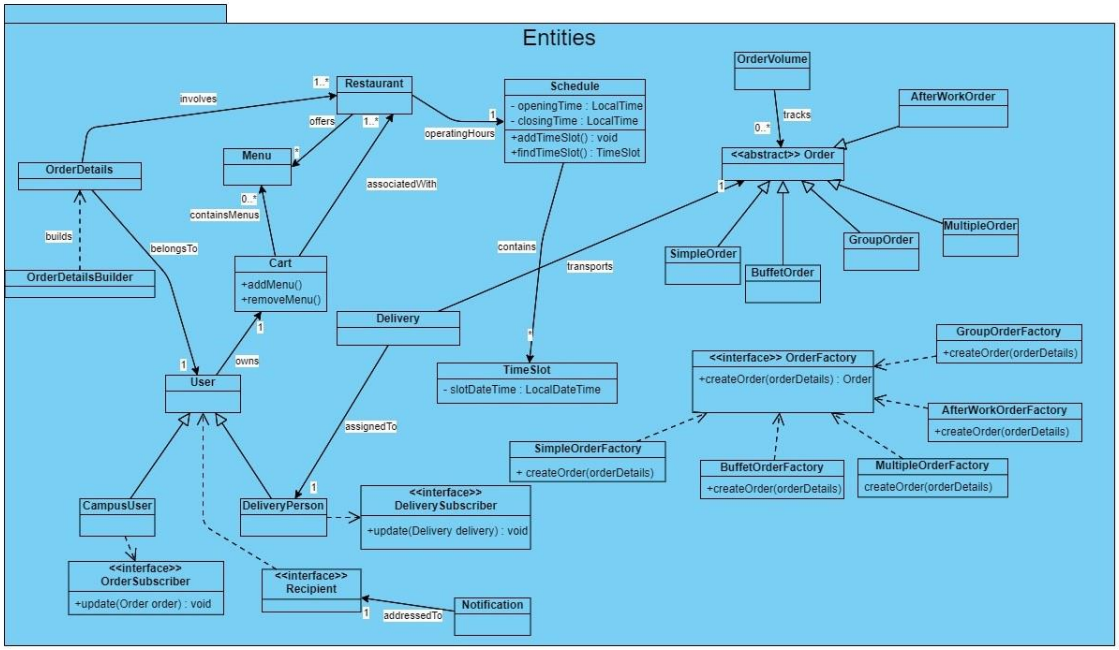


Figure 2 - Diagramme de séquence du cas d'utilisation : place a simple order

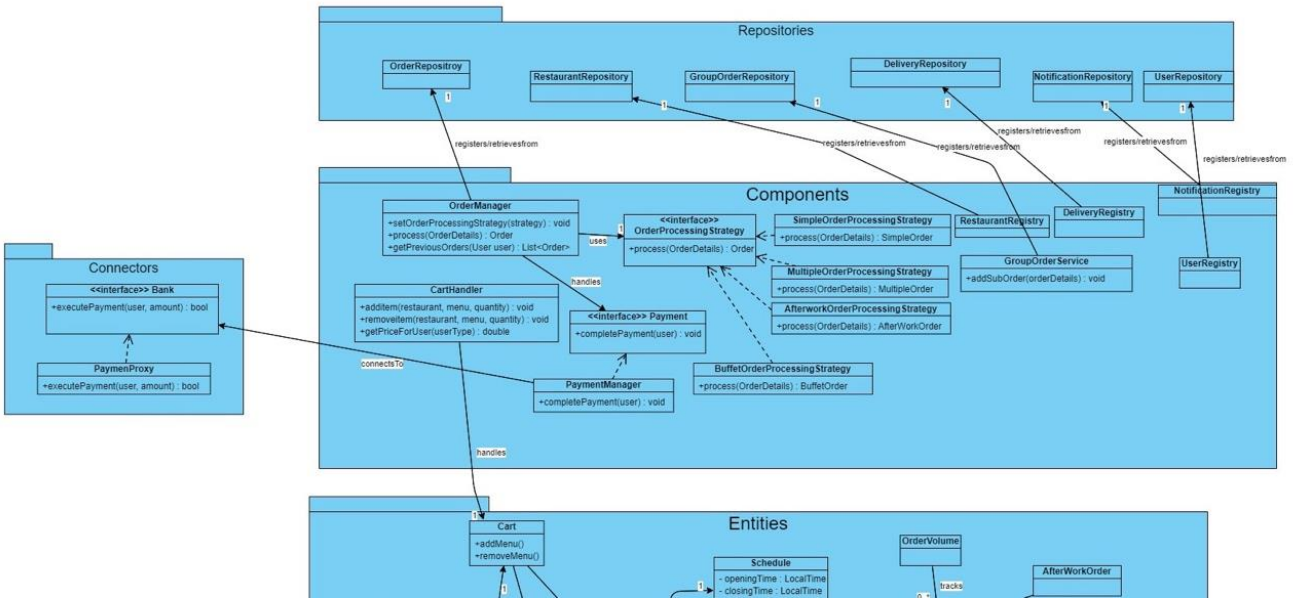
DIAGRAMME DE CLASSE :

Dans ce segment, nous exposons des éléments structuraux du diagramme de classe pour notre système. Ces détails fournissent un aperçu des mécanismes internes et des relations entre les différentes classes.

Ce premier extrait présente le diagramme de classe des entités de notre projet. La majorité de ces classes possèdent uniquement des accesseurs, et lorsqu'il existe une méthode exécutant un type de logique, cela est précisé.



et ensuite, ce second extrait concerne les composants du projet, les repository et les connecteurs, en mettant l'accent sur les interactions et les flux de données entre eux. Il détaille également comment ces éléments contribuent à la robustesse et à l'efficacité du système, illustrant la manière dont ils s'intègrent dans l'architecture globale.

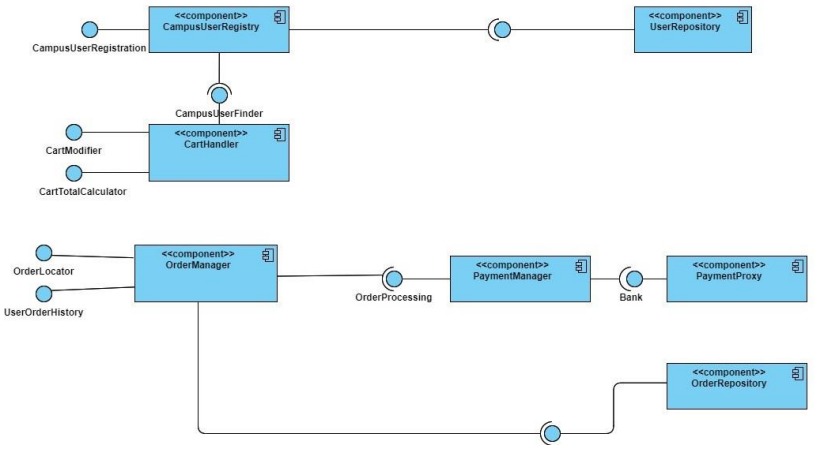


ESQUISSE DU DIAGRAMME DE COMPOSANT :

1. Les classes qui ont presque déjà le rôle de Composants :

Classe	Description
CartHandler	Gère le panier d'un CampusUser
OrderManager	Traitement des commandes
GroupOrderService	Traitement des commandes de groupe
PaymentManager	Gère les payements
RatingSystem	Gère les évaluations des restaurants et des livreurs
StatisticsManager	Gère les statistiques des commandes
CampusUserRegistry	Gère les clients
DeliveryPersonRegistry	Gère les livreurs
DeliveryRegistry	Traitement des livraisons
NotificationRegistry	Gère les notifications
RestaurantRegistry	Gère les restaurants

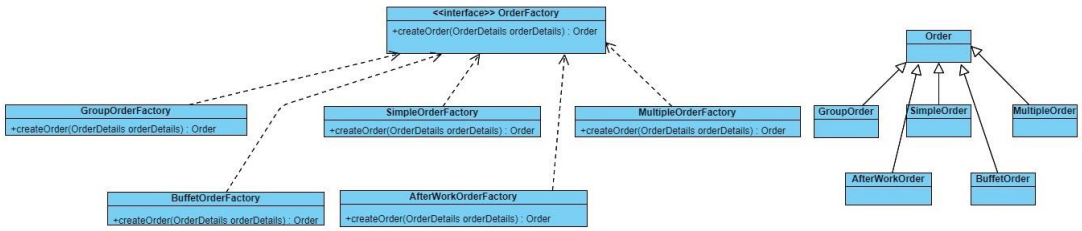
2. Esquisse du diagramme des composants:



3.DESIGN PATTERNS APPLIQUÉS OU PAS :

DPS EN DÉTAIL :

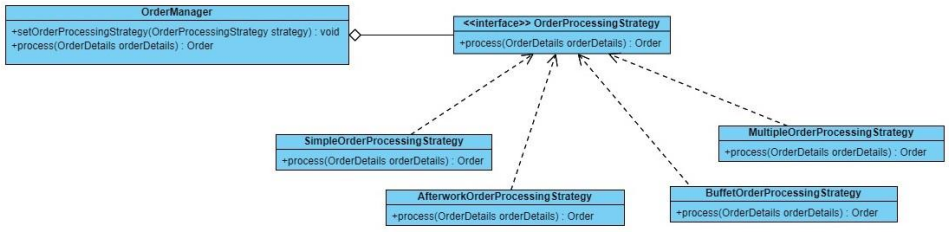
FACTORY METHOD:



La méthode Factory dans ce contexte sert à centraliser la création d'ordres pour favoriser l'évolutivité et la facilité de maintenance, en découplant le processus de création d'objet du code client. Cela permet d'ajouter de nouveaux types d'ordres sans impacter le code existant.

- AbstractFactory : OrderFactory
  - Déclare la méthode createOrder pour créer des ordres.
- Concrete Factories :
  - GroupOrderFactory, SimpleOrderFactory, MultipleOrderFactory, AfterWorkOrderFactory, BuffetOrderFactory
- AbstractProduct : Order
  - Définit la structure et les fonctionnalités de base d'un ordre.
- Concrete products :
  - GroupOrder, SimpleOrder, MultipleOrder, AfterWorkOrder, BuffetOrder

STRATEGY PATTERN:



Dans le OrderManager, la pattern Stratégie est utilisé pour gérer différents types de commandes, chacun ayant des exigences de traitement uniques. Pour les commandes simples, la stratégie couvre les procédures standard telles que l'allocation de créneaux horaires et le traitement des paiements. Les commandes impliquant plusieurs restaurants nécessitent une approche plus complexe, coordonnant les créneaux horaires, les paiements et les livraisons pour des commandes s'étendant sur plusieurs établissements. La stratégie pour les commandes afterwork est adaptée pour gérer les aspects logistiques des

événements sans nécessiter de paiement ou de livraison. En outre, les commandes de type buffet sont gérées en mettant l'accent sur la gestion de grandes quantités et la capacité d'accueil. Cette approche basée sur la stratégie assure que chaque type de commande est traité efficacement, en accord avec ses caractéristiques et exigences spécifiques, tout en maintenant une cohérence dans le flux de traitement des commandes.

AUTRES DPS :

- La pattern Observer est utilisé pour créer un système de notification fonctionnel. Ce modèle est adapté aux situations où un objet, nommé le sujet, doit notifier automatiquement une liste d'observateurs en cas de changements d'état. Dans notre application, cela est particulièrement pertinent pour informer les utilisateurs sur le statut de leurs commandes. L'avantage de ce modèle réside dans sa capacité à maintenir un faible couplage entre le sujet (gestionnaire de commande) et les observateurs (utilisateurs), ce qui favorise l'évolutivité et la maintenabilité du système.

4.QUALITÉ DES CODES ET GESTION DE PROJETS:

MÉTHODOLOGIE DE TESTS

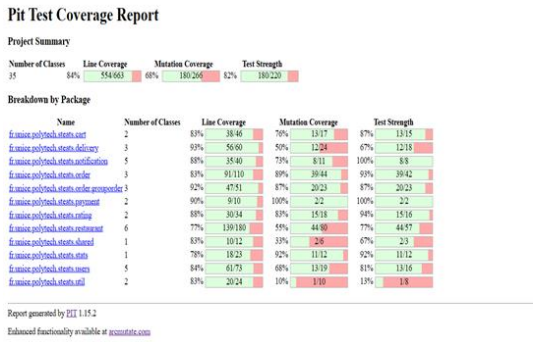
Tests Fonctionnels

Pour les tests fonctionnels, leur portée s'est concentrée principalement sur les cas de succès, couvrant efficacement divers scénarios de l'application tels que le placement d'une commande, la personnalisation d'une commande, etc. Ces tests ont joué un rôle crucial en garantissant que l'application répondait de manière adéquate aux besoins des utilisateurs dans des conditions normales d'utilisation. Il est important de noter toutefois que la couverture des cas d'erreurs dans ces tests fonctionnels pourrait être renforcée.

Tests unitaires

Pour les tests unitaires, une attention particulière a été accordée à certaines classes clés, telles que les classes Cart et Order. L'objectif était d'évaluer le bon fonctionnement de leurs composants individuels. Cette approche a été adoptée dans le but de faciliter la maintenance du code et de permettre une détection rapide d'éventuels problèmes, incluant subtilement l'utilisation de mocks pour renforcer l'isolement au cours de ces tests unitaires.

Tests de mutation



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
35	84% 554/663	68% 180/266	82% 180/220

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
fr.unice.polytech.steats.cart	2	83% 38/46	76% 13/17	87% 13/15
fr.unice.polytech.steats.delivery	3	93% 56/60	50% 12/24	67% 12/18
fr.unice.polytech.steats.notification	5	88% 35/40	73% 8/11	100% 8/8
fr.unice.polytech.steats.order	3	83% 91/110	89% 39/44	93% 39/42
fr.unice.polytech.steats.order.geosponder	3	92% 47/51	87% 20/23	87% 20/23
fr.unice.polytech.steats.payment	2	90% 9/10	100% 2/2	100% 2/2
fr.unice.polytech.steats.rating	2	88% 30/34	83% 15/18	94% 15/16
fr.unice.polytech.steats.restaurant	6	77% 139/180	55% 44/80	77% 44/57
fr.unice.polytech.steats.shared	1	83% 10/12	33% 2/6	67% 2/3
fr.unice.polytech.steats.stats	1	78% 18/23	92% 11/12	92% 11/12
fr.unice.polytech.steats.users	5	84% 61/73	68% 13/19	81% 13/16
fr.unice.polytech.steats.util	2	83% 20/24	10% 1/10	13% 1/8

Report generated by PIT 1.15.2

Enhanced functionality available at arcmutate.com

L'utilisation de Pitest a permis de réaliser des tests de mutation, identifiant ainsi les zones du code qui manquaient de couverture par les tests existants.Et aussi pour évaluer la qualité des suites de tests existantes. L'objectif principal des tests de mutation est d'identifier la capacité des tests à détecter des changements dans le code source. Ceci a été essentiel pour améliorer la robustesse de l'application en détectant des faiblesses potentielles dans le code.

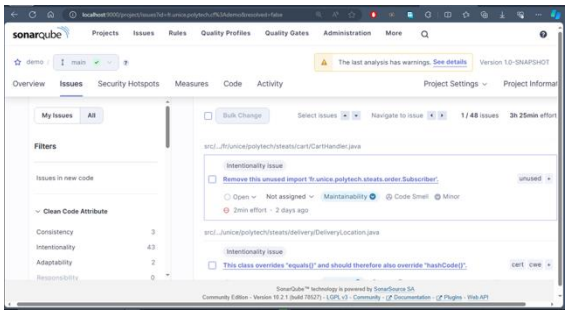
MESURES DE QUALITÉ

Inspection de Code

Des inspections de code ont été réalisées en utilisant les outils intégrés dans IntelliJ. Cela a permis de détecter et de corriger les violations des normes de codage, les erreurs potentielles et d'améliorer la lisibilité du code.

SonarQube

L'intégration de SonarQube a été ajoutée pour effectuer une analyse statique du code. Cela a permis d'identifier des problèmes de qualité du code tels que les duplications, les vulnérabilités de sécurité et les mauvaises pratiques de codage.



5. RÉTROSPECTIVE ET AUTO-ÉVALUATION :

RÉTROSPECTIVE :

El Betioui Jamal Eddine (PO) :

En tant que Product Owner, mon rôle a consisté à guider le développement du projet en veillant à ce que les besoins des utilisateurs soient correctement interprétés et traduits en fonctionnalités. Voici un aperçu de ma contribution :

- **Définition des Exigences :**

J'ai travaillé en étroite collaboration avec les parties prenantes pour définir clairement les exigences du projet, en m'assurant que les fonctionnalités développées répondaient aux attentes des utilisateurs.

- **Priorisation des Fonctionnalités :**

La priorisation des fonctionnalités a été une étape cruciale. J'ai hiérarchisé les exigences en fonction de leur importance stratégique et de la valeur qu'elles ajoutent à l'expérience utilisateur.

- **Communication Transparente :**

Maintenir une communication transparente avec l'équipe de développement a été essentiel. J'ai veillé à ce que l'équipe comprenne clairement les objectifs du produit et les exigences spécifiques de chaque itération.

- **Itérations et Feedback Utilisateur :**

J'ai participé activement aux réunions d'itération, fournissant des retours continus et travaillant en tandem avec l'équipe de développement pour s'assurer que les ajustements nécessaires étaient apportés en fonction des retours des utilisateurs.

- **Validation des Livraisons :**

J'ai validé les livraisons de fonctionnalités, m'assurant qu'elles répondaient aux critères définis et qu'elles étaient prêtes à être présentées aux utilisateurs.

- **Rétrospective :**

Dans une rétrospective, je reconnais l'importance de renforcer la collaboration avec l'équipe de tests pour garantir une couverture exhaustive des scénarios d'utilisation, en mettant particulièrement l'accent sur les cas d'erreur. Une amélioration future pourrait impliquer une intégration plus étroite des retours utilisateurs dans le processus de développement afin d'ajuster rapidement les priorités et les fonctionnalités en cours de route.

AKHYATE Brahim (SA) :

Globalement, je suis assez satisfait de l'architecture finale du projet. J'ai fait de mon meilleur pour diviser le projet en **trois couches principales (Entities, Components, Repositories)**, les classes en général ayant des responsabilités séparées, bien que je crois que certaines optimisations peuvent certainement être apportées pour un code encore plus propre.

Voici les principaux enseignements que je retire en tant qu'SA de mon groupe :

- Premièrement, c'était une expérience très enrichissante en ce qui concerne l'architecture d'un projet Java. Pour mon rôle, j'ai dû faire des recherches afin de définir la structure du projet, ce qui m'a beaucoup appris. La chose principale est l'architecture en couches (**layered architecture**), qui est une notion puissante pour construire des projets propres et maintenables.

- Grâce aux **design patterns** appris, nous avons fourni nos propres solutions aux problèmes rencontrés pendant la conception et le codage. Ils nous ont spécifiquement aidés à résoudre les problèmes survenus avec l'introduction des nouvelles extensions.

- À travers **l'injection de dépendances**, le principe final récemment acquis de **SOLID** et les principes SOLID précédemment vus ont aidé à rendre le **code moins couplé** et **plus dépendant des abstractions en utilisant des interfaces plutôt que des classes concrètes**.

- **Injection de dépendances à travers PicoContainer** pour les tests également, les tests Cucumber ont été structurés d'une manière où les steps sont partagées entre plusieurs fonctionnalités, avec **des classes steps déterminées selon leur responsabilité**, et **de multiples fonctionnalités peuvent utiliser les mêmes steps**, **évitant ainsi la duplication de code** grâce à un PicoContainer.

Des erreurs ont certainement été commises pendant le projet, les principales étant le fait que nous codions initialement sans une vision partagée, conduisant à du code inutile, répétitif, trop couplé et qui est inextensible. À un certain moment, nous avons dû tout jeter et recommencer à zéro. Avec les connaissances actuelles acquises après ce projet, je crois que nous pouvons éviter ces erreurs dans un projet futur, avec beaucoup plus de temps qui pourrait être consacré à la conception d'un code encore plus propre et à l'optimisation de la logique métier.

MESAOURI DEBOUN Bilal (QA) :

- **Tests Fonctionnels et unitaire :**

La mise en place de tests fonctionnels a évalué divers scénarios de l'application, assurant ainsi sa conformité aux attentes des utilisateurs. En parallèle, des tests unitaires ont été déployés sur des classes spécifiques, renforçant ainsi la robustesse du code.

- **Tests de mutation :**

La mise en œuvre des tests de mutation à l'aide de l'outil Pitest a été essentielle pour garantir le bon fonctionnement des tests existants et leur capacité à détecter des mutations.

- **Contrôle de Qualité :**

Nous avons instauré des pratiques de contrôle de qualité, comprenant la mesure de la couverture du code, des inspections de code à l'aide des outils intégrés en IntelliJ, et l'intégration de SonarQube pour une analyse statique approfondie.

- **Rétrospective :**

En rétrospective, bien que les tests fonctionnels aient principalement couvert les cas de succès, il serait bénéfique d'accorder davantage d'attention aux scénarios d'erreur pour renforcer la robustesse de l'application. De plus, une amélioration potentielle pourrait consister à intégrer davantage les tests unitaires dans le processus de développement pour garantir une couverture exhaustive du code.

BEIDOURI Saad (Ops) :

- **Milestones et Issues :**

Nous avons structuré notre gestion de projet sur GitHub en créant deux milestones principaux correspondant à nos sprints.

Le premier milestone, nommé "**Sprint 1 : Version 1**", englobe toutes les user stories (US) mises en œuvre dans notre projet. Pour chaque US, nous avons utilisé un template standardisé qui inclut des détails tels que la **description**, la **priorité**, l'**estimation** du temps, les **règles métier** et les **critères d'acceptation**. La visibilité et l'organisation de ces US ont été renforcées par l'emploi de **labels spécifiques** qui marquent la priorité et l'estimation de chaque US.

Le deuxième jalon, "**Sprint 2 : new functionalities**", suit le même processus mais est dédié aux issues relatives aux nouvelles fonctionnalités, qui représentent les extensions de notre projet initial.

- **Stratégie de Branching :**

Pour notre projet, nous avons adopté une stratégie de gestion de branches basée sur **le modèle GitFlow**. Cette approche consiste à maintenir une branche principale **main**, qui est réservée aux versions stables. Parallèlement à cela, nous

utilisons une branche de développement, nommée **dev**, qui sert de base pour l'intégration continue de nouvelles fonctionnalités.

Lorsqu'une nouvelle user story (US) est prête à être implémentée, nous créons une branche spécifique pour celle-ci à partir de la branche dev. Cela nous permet de travailler isolément sur chaque US sans perturber le code sur la branche de développement principale. Une fois que la fonctionnalité est terminée et que tous les tests sont passés, la branche de la US est fusionnée (merge) de retour dans la dev.

NB : J'ai mis en place un processus d'automatisation sur la branche **dev** qui utilise un fichier **maven.yml** pour s'assurer que chaque build est correctement effectué, grâce à GitHub Actions qui configure l'environnement, installe les dépendances nécessaires et construit le projet avec Maven à chaque push ou pull request sur les branches **dev** et **main**.

- **Rétrospective :**

En tant que responsable des opérations (OPS) dans notre projet, j'ai appris des leçons précieuses sur l'importance d'avoir des descriptions détaillées pour chaque User Story (US) ainsi que l'utilisation de templates standardisés. Lorsqu'un problème survient, nous pouvons nous référer directement à la description de l'US concernée pour comprendre le contexte et le comportement attendu. Cela réduit le temps passé à chercher dans le code où et pourquoi un problème se manifeste.

Malgré ma vigilance concernant la gestion des commits et des branches, j'ai commis certaines erreurs dans notre processus. Un exemple notable est l'absence de références GitHub dans tous les scénarios. En effet, certains commits ont été effectués sans associer explicitement le scénario auquel ils appartenaient à une issue ou une référence GitHub spécifique. De même, pour certains refactors effectués dans le code, je n'ai pas systématiquement attribué les modifications à une issue GitHub spécifique ni inclus de référence GitHub.

AUTO-ÉVALUATION :

	MESAOURI DEBOUN Bilal	EL BETIOUI Jamel Eddine	AKHYATE Brahim	BEIDOURI Saad
MESAOURI DEBOUN Bilal	------	100	100	100
EL BETIOUI Jamel Eddine	100	------	100	100

AKHYATE Brahim	100	100	------	100
BEIDOURI Saad	100	100	100	-----