RAPPORT FINAL

PROJET CONCEPTION S7

EQUIPE O

Wassim BARATLI : Rôle PO

Aubin RAHMANI : Rôle QA

Valentin PODDA : Rôle SA

1. Périmètre fonctionnel : Hypothèses, Limites, Extensions, Points Forts et Points Faibles

1.1. Nos hypothèses de travail (E.4)

- Diversité des restaurants : Nous supposons que notre application englobe une variété de restaurants, offrant une large gamme de cuisines pour répondre aux préférences diverses des clients.

- Interactivité et convivialité : Nous considérons que l'interface utilisateur de l'application sera intuitive et conviviale, permettant aux utilisateurs de naviguer facilement, de placer des commandes rapidement et de suivre leurs livraisons en temps réel.

- Fiabilité du service de livraison : Nous partons de l'idée que le service de livraison sera efficace et fiable, garantissant que les repas arrivent à temps et dans des conditions optimales.

1.2. les limites identifiées (G.6)

- Dépendance aux services tiers : Notre application pourrait dépendre de services tiers (par exemple, des services de cartographie ou des API de paiement), ce qui expose notre application à des risques potentiels en cas de changements, de pannes ou de limitations imposées par ces services.

- Gestion de données volumineuses : La gestion efficace de grandes quantités de données, notamment les menus des restaurants, les profils des utilisateurs et les historiques de commandes, représente un défi majeur en termes de performance, de stockage et d'accès rapide aux informations.

1.3. Extensions choisies (E.4)

- [EX1] Diversité des commandes. Le projet a un tel succès que l'on voudrait diversifier les types de commandes : une commande simple (un restaurant, un usager), une commande multiple (des restaurants, un usager), des commandes de groupe (des restaurants, des usagers), des commandes buffets (un restaurant qui offre des commandes buffets, un staff université, un usager destinataire), des commandes afterWorks (un restaurant qui peut recevoir et qui expose des « menus » afterworks, un usager qui passe commande, un nombre de participants prévu : pas de livraison, pas de paiement, …). On doit pouvoir ajouter d'autres types de commandes ultérieurement.
- [EX2] Ristournes des restaurateurs : Les restaurants, en plus de diversifier leur prix en fonction du statut des types d'utilisateurs [09], peuvent proposer des réductions en fonction du nombre de commandes directes (i.e., pas au niveau du groupe). Par exemple : au bout de 10 commandes dans un restaurant A, vous bénéficiez de 5% de ristourne sur toute commande pour une durée de 15 jours. Si dans cette période vous avez encore passé 10 commandes, la réduction se prolonge.
- [EX7] Système de recommandations. On souhaite pouvoir évaluer les restaurants (usagers), les livreurs (usagers livrés), les usagers (livreur (retard, aimable)).

1.4. Points non étudiés.

- Optimisation des algorithmes de recommandation : Nous n'avons pas encore exploré en profondeur l'optimisation des algorithmes de recommandation pour personnaliser les choix des menus en fonction des préférences et des habitudes des utilisateurs.

- Tests de charge et de performance : Nous n'avons pas encore réalisé des tests approfondis de charge et de performance pour évaluer la réactivité de l'application et son efficacité lors de périodes d'utilisation intensive.

2. Conception

2.1. Le glossaire :

Des définitions claires et précises de tout le vocabulaire propre au domaine d'application, y compris les termes techniques, les mots du langage courant utilisés dans un sens particulier et les acronymes. Il présente la terminologie du projet ; non seulement de l'environnement au sens strict, mais de toutes ses composantes.[1]

Campus: Correspond à une zone centrée sur PNS, qui se situe à 10 minutes à pied du centre. Cela inclut PNS, les IUT, l'INRIA, les Algorithmes, et s'étend à Luciole et Saint-Philippe.

User : Désigne les individus au sein de la communauté du campus, notamment les étudiants, le personnel et les membres du corps professoral. Seuls les utilisateurs enregistrés du campus peuvent accéder au système SophiaTech Eats.

Restaurant : Désigne le personnel du restaurant qui interagit avec le système.

Livreur : Désigne le personnel de livraison chargé de livrer les commandes de nourriture aux endroits désignés sur le campus.

Administrateurs de campus : Représenter les personnes occupant des postes administratifs responsables de la supervision et de la gestion des activités du campus. Par exemple, ils peuvent collaborer pour répartir les heures de cours sur l'ensemble du campus afin d'éviter la congestion au restaurant universitaire. RGPD (Règlement

Général sur la Protection des Données : Désigne le règlement de l'Union européenne régissant la protection des données personnelles et de la vie privée.

API (interface de programmation d'applications) : Un ensemble de protocoles et d'outils qui permettent à différentes applications logicielles de communiquer et d'interagir entre elles.

Systèmes de paiement externes : Désigne les plateformes de paiement tierces, telles que PayPal et Google Pay, utilisées pour traiter les transactions financières.

Commande/Order : Une demande faite par un utilisateur du campus pour des menus alimentaires spécifiques d'un ou plusieurs restaurants.

Livraison: Processus de transport des aliments commandés depuis les restaurants vers des emplacements désignés sur le campus.

Un menu : Fait référence à une liste de produits alimentaires proposés par un restaurant sur commande. Nous travaillons uniquement sur la granularité des menus. Il appartient aux restaurants de définir les menus qu'ils proposent.

Temps de préparation: Le temps estimé nécessaire à un restaurant pour préparer un menu. Nous considérons que le temps de préparation d'un menu dans un restaurant est indépendant du menu.

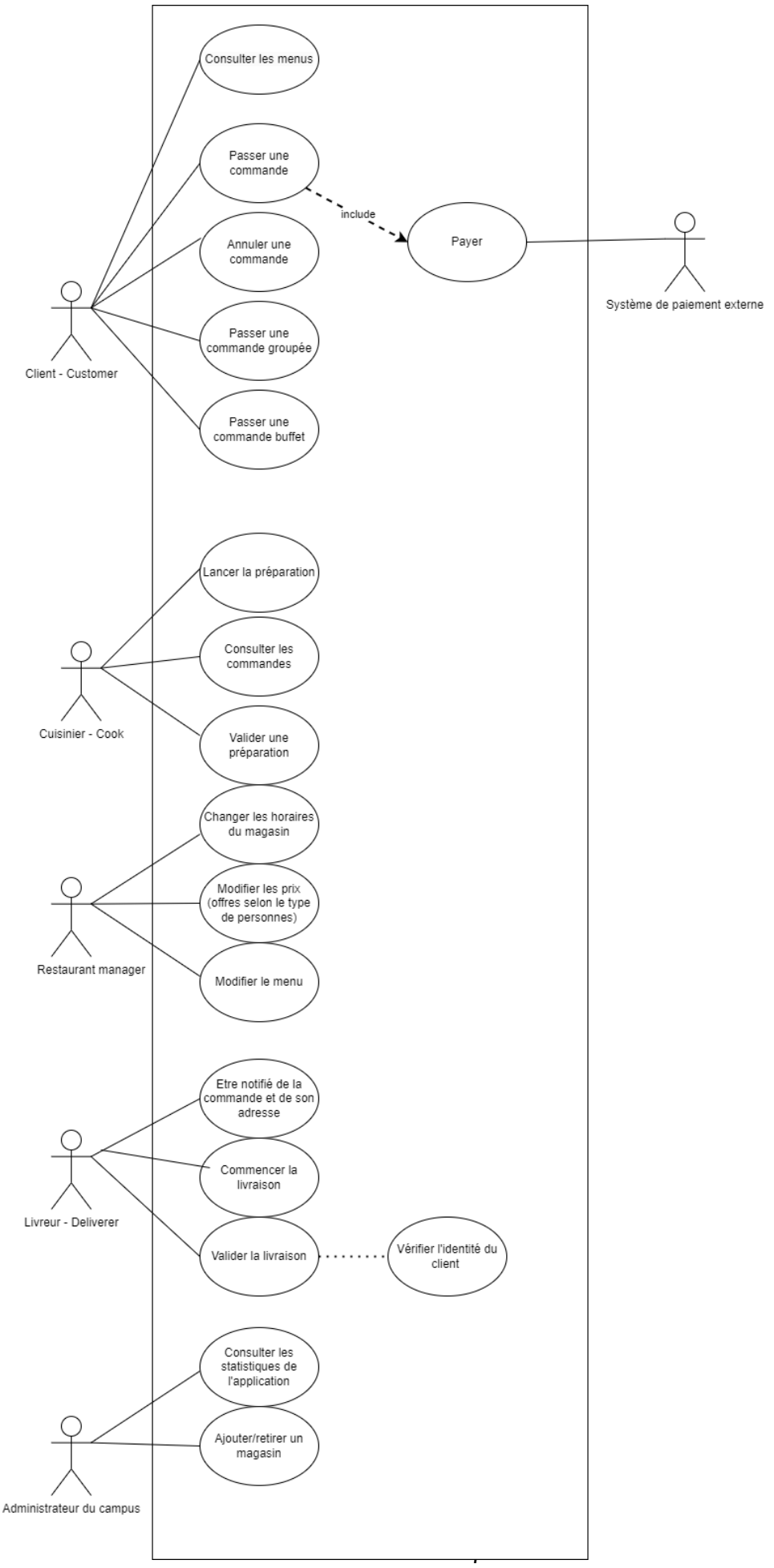
Le débit d'un restaurant correspond au nombre de menus qu'il peut réaliser sur une période donnée, par exemple 100 menus par heure.

Interface utilisateur : Composants visuels et interactifs qui permettent aux utilisateurs d'interagir avec le système. Cependant, aucune implémentation d'interface utilisateur n'est demandée dans ce projet.

Module : Désigne l'unité pédagogique au sein de laquelle s'inscrit ce projet (COO).

Ce glossaire fournit des définitions claires du vocabulaire et des termes spécifiques utilisés dans le contexte du projet. Il garantit une compréhension partagée des concepts et de la terminologie clés entre les parties prenantes et les membres de l’équipe.

2.2. Le diagramme de cas d’utilisation et User Stories :


Consulter les menus
Passer une commande
include
Payer
Système de paiement externe
Annuler une commande
Client - Customer
Passer une commande groupée
Passer une commande buffet
Lancer la préparation
Consulter les commandes
Cuisinier - Cook
Valider une préparation
Changer les horaires du magasin
Modifier les prix (offres selon le type de personnes)
Restaurant manager
Modifier le menu
Etre notifié de la commande et de son adresse
Commencer la livraison
Livreur - Deliverer
Valider la livraison
Vérifier l'identité du client
Consulter les statistiques de l'application
Ajouter/retirer un magasin
Administrateur du campus

Choisir un restaurant :
As a a customer “Toto” , I want“Toto” choose the restaurant “Nice” so that he will be able to add dishes from the restaurant “Nice”

GIVEN a customer “Toto”
WHEN “Toto” choose the restaurant “Nice”
THEN he will be able to add dishes from the restaurant “Nice”

Choisir un plat :
As a a customer “Toto” who has chosen the restaurant “Nice”, I want“Toto” add a dish “Pizza” so that the order price will increase by “Pizza” ‘s price

GIVEN a customer “Toto” who has chosen the restaurant “Nice”
WHEN “Toto” add a dish “Pizza”
THEN the order price will increase by “Pizza” ‘s price

Consulter les menus :
As a customer “Toto”, I want “Toto” want to see the available menu of the restaurant “KebabDelice” so that available menus of the restaurant “KebabDelice” is displayed

GIVEN a customer “Toto”
WHEN “Toto” want to see the available menu of the restaurant “Nice”
THEN available menus of the restaurant “Nice” are displayed

Payer :
As a customer “Toto”, I want “Toto” pay my order so that my order is ready to cook

GIVEN a customer “Toto” who has an order with one dish “Pizza” in the restaurant “Nice”
WHEN “Toto” pays the order
THEN the order is validated and the restaurant is notified of the order for preparation

Annuler une commande :
As a user who has passed an order, I want to delete it so that i can restart it again

GIVEN user "Toto" with a pending order at restaurant "Nice"
WHEN "Toto" selects the option to cancel the order
THEN the order at restaurant "Nice" is canceled, and "Toto" and the the Restaurant “Nice” are notified of the cancellation

Passer une commande groupée :
As a customer, I want to create a group order so that I can add simple orders of my friends

Given a connected user "Alice"
When "Alice" create an order simple with that dishes: "kebab"
And "Alice" create another order simple for user "bob" with that dishes : "fries"
Then the order grouped is created

Passer une commande buffet :
As a collective Polytech, I want to order a buffet to students so that all my students are satisfied

Given public "Polytech" order a buffet for his 100 students at the restaurant "KebabDelice"
When 50 "kebab", 50 "pizza" and 100 "tiramisuDelice" are ordered to 21:0 pm
Then The order Buffet can be delivered only at 21:0

Préparer une commande :
As a cook, I want to launch the preparation of the order's "Toto" so that the statement of the order is now on "IN PROGRESS"

GIVEN restaurant "Nice" with a paid order for user "Toto"
WHEN restaurant "Nice" begins preparing the order for "Toto"
THEN the status of "Toto's" order is updated to "preparation in progress"

Valider une préparation :
As a cook, I want to valid the preparation of an order so that the order statement is changed into "ready to deliver"

GIVEN restaurant "Nice" with an order from "Toto" in the "preparation in progress" status
WHEN the restaurant completes the preparation of "Toto's" order
THEN the order status for "Toto's" order is updated to "ready to deliver," and "Toto" is notified

Changer les horaires du magasin :
As a Restaurant Manager, I want to change my restaurant hours so that i could change my schedule

GIVEN restaurant "Nice" logged into the application
WHEN the restaurant accesses their profile settings
THEN they can modify the operating hours of their store and update them accordingly

Modifier les prix :
As a Restaurant Manager, I want to change restaurant information so that I can update them with new informations

GIVEN restaurant "Nice" logged into the application
WHEN the restaurant accesses their menu settings
THEN they can modify prices based on different user types, offering special discounts or deals, including adjustments for user "Toto"

Modifier le menu :
As a Restaurant Manager, I want to change restaurant information so that I can update them with new informations

GIVEN restaurant "Nice" logged into the application
WHEN the restaurant accesses their menu settings
THEN they can add, remove, or modify dishes on the menu for restaurant "Nice," updating it to reflect the latest offerings

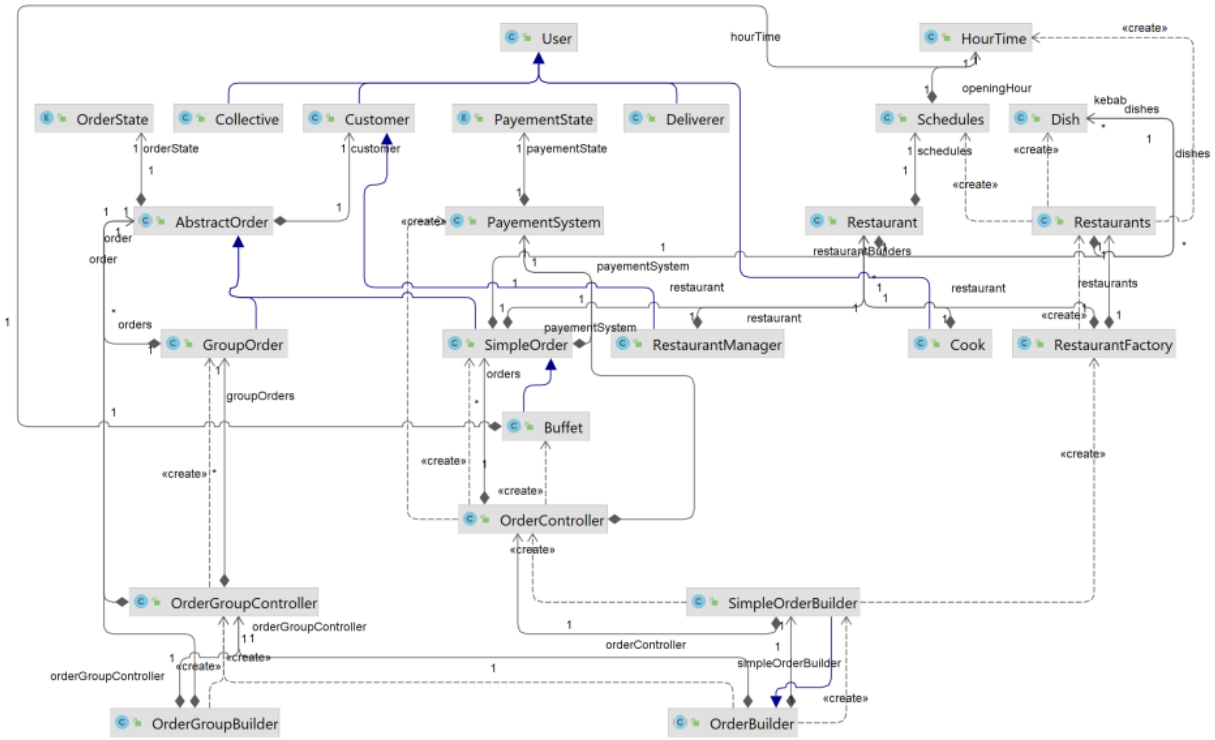
Prendre en charge une livraison
As a deliverer "Jacky", I want to deliver an order so that it will be marked delivered

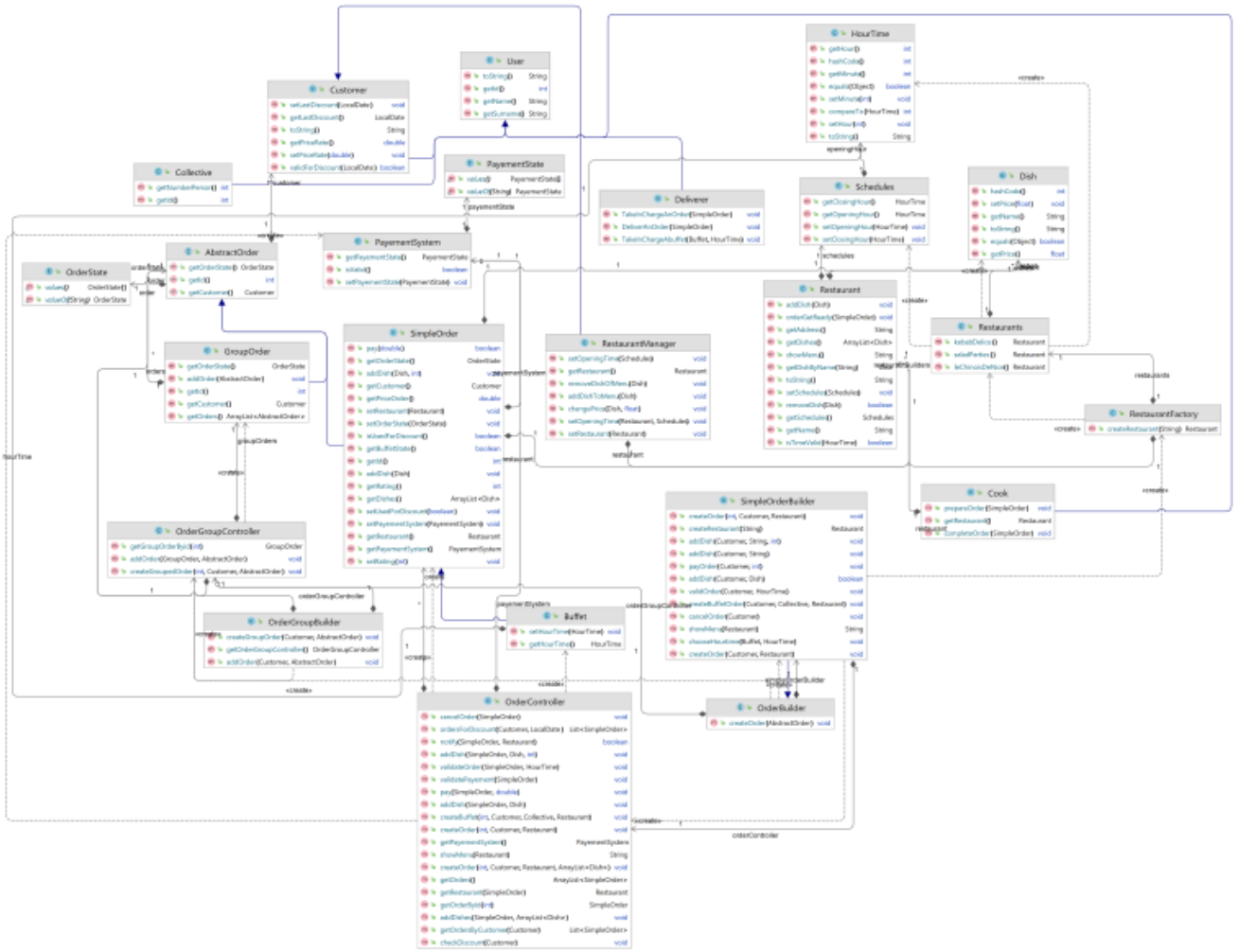
GIVEN a deliverer "Jacky",
WHEN the deliverer "Jacky" selects to take charge of a delivery,
THEN the delivery is assigned to the deliverer "Jacky." and is updated “In process delivery”

Livrer une commande
As a deliverer Jacky, I want to take a charge an order so that i can deliver it

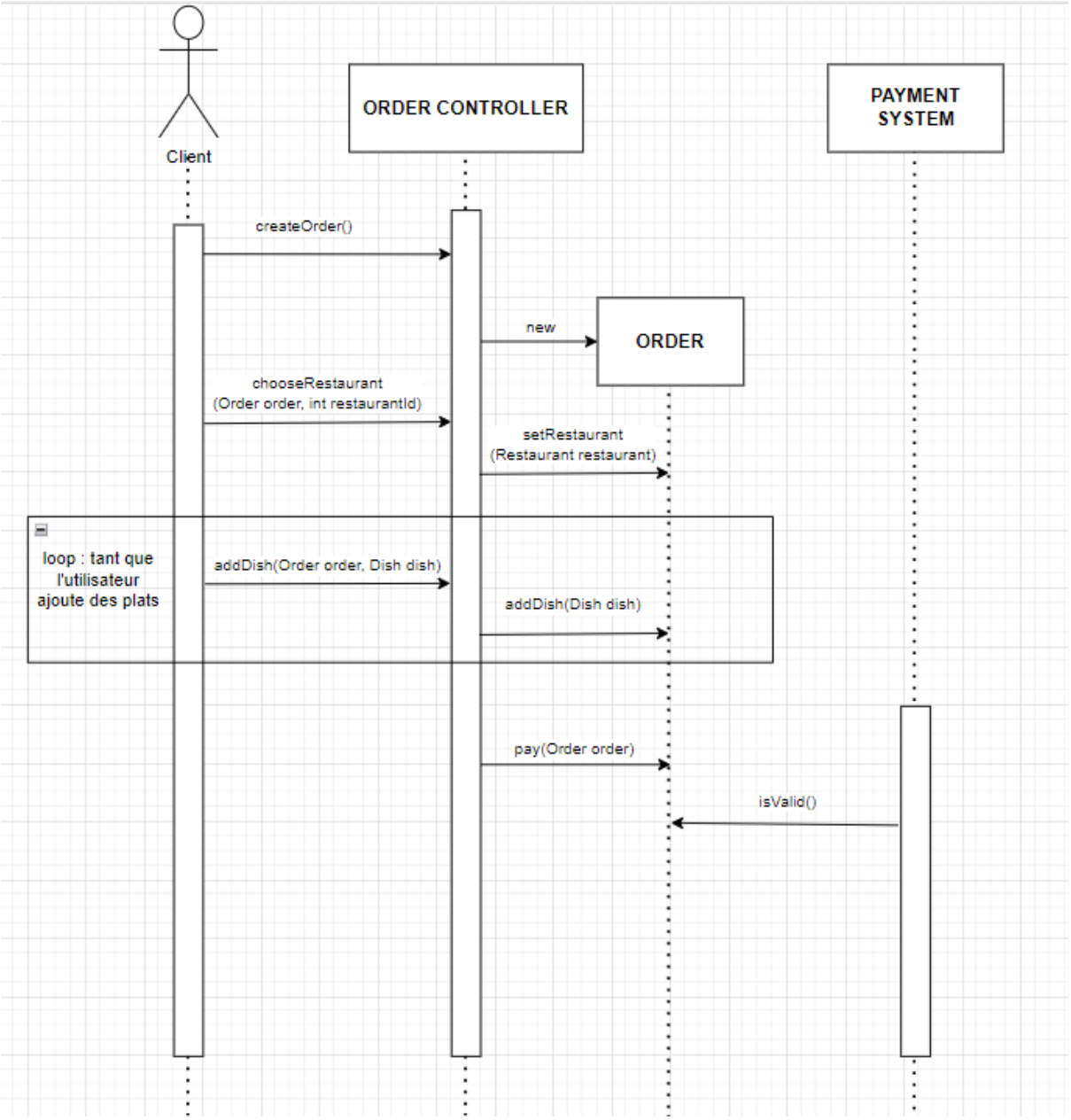
GIVEN a deliverer "Jacky" has retrieved the order for delivery,
WHEN the deliverer "Jacky" successfully delivers the order to the specified destination,
THEN the delivery is marked as "delivered" by the system.

2.3. Le diagramme de classe :

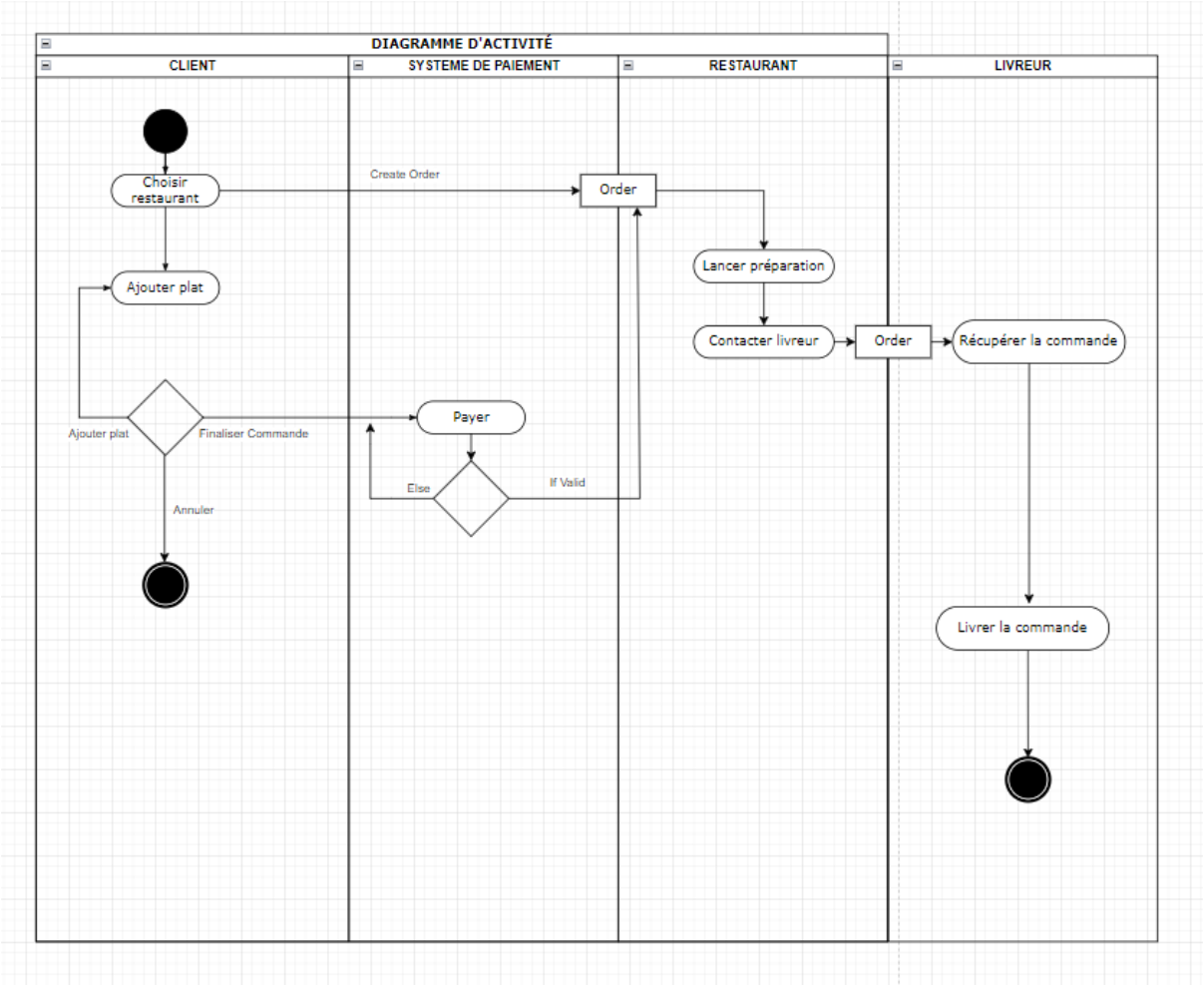



HourTime
User
Customer
Collective
PaymentState
Deliverer
Schedules
Dish
PaymentSystem
AbstractOrder
OrderState
Restaurant
Restaurants
SimpleOrder
RestaurantManager
RestaurantFactory
GroupOrder
SimpleOrderBuilder
Cook
OrderGroupController
Buffet
OrderGroupBuilder
OrderController
OrderBuilder

2.4. Le diagramme de séquence :



2.5. Le diagramme d’activité :



3. Designs patterns

Au cours du processus d'implémentation de notre projet, nous avons été confrontés à la gestion de la complexité croissante de l'architecture et de l'implémentation. C'est à ce moment-là que nous avons pris conscience de l'existence et de l'utilité des Design Patterns. Ces derniers sont des solutions éprouvées pour résoudre des problèmes récurrents de conception logicielle. Dans notre cas, nous avons choisi d'intégrer certains de ces patterns pour simplifier la construction de composants spécifiques du projet et améliorer la flexibilité de notre code. Voici les patterns que nous avons utilisés.

- Builder.

  Nous avons utilisé le pattern de création Builder pour créer les commandes, ce qui permet de créer les commandes étapes par étape. Nous avons créé une classe SimpleOrderBuilder et OrderGroupBuilder, qui permettent de créer les orders. La classe SimpleOrderBuilder a été conçue pour la construction d'objets de type SimpleOrder. Ce builder offre une méthode pour chaque attribut nécessaire à la création d'une commande simple. En utilisant cette classe, nous pouvons étape par étape définir les caractéristiques d'une commande simple.

  De manière similaire, la classe OrderGroupBuilder a été implémentée pour faciliter la création d'objets de type GroupOrder. Ce builder offre également des méthodes pour chaque attribut requis pour construire une commande de groupe. Il nous permet de procéder progressivement à la définition des informations liées à une commande de groupe, telles que la liste des commandes simples incluses dans le groupe.

  En amont de ces builders, nous avons également utilisé une structure de classes comprenant AbstractOrder, SimpleOrder et GroupOrder. La classe AbstractOrder établit la structure de base pour les commandes, tandis que SimpleOrder représente une commande individuelle avec des détails spécifiques : le client, le restaurant et les dishes. D'autre part, GroupOrder est conçue pour regrouper plusieurs commandes simples au sein d'une seule entité de commande.

Nous avons utilisé le pattern Builder car, de tous les patterns appris, c'est le seul qui offre une approche spécifique pour la construction étape par étape d'objets complexes sans surcharger les constructeurs ou exposer les détails de leur implémentation interne.

- Factory.

  Nous avons utilisé le pattern de création Factory pour créer les restaurants de différents types (maitre kebabier, restaurant asiatique, restaurant diététique…), ce qui permet de simplifier la création de ces restaurants. Au lieu de créer des instances de restaurant directement, le code client peut demander à la factory de le faire (ici dans les tests et le simpleOrderBuilder qui appelle la factory). De plus, notre méthode : createRestaurant nous permet de créer un restaurant avec tous ses attributs juste en renseignant son nom, ce qui simplifie grandement le code. On utilise donc une hasmap afin d'associer ces noms (string) avec leur objet correspondant, leur objet restaurant.

  Comparativement à d'autres patterns tels que le Singleton, Builder ou Prototype, le pattern Factory se concentre spécifiquement sur la création d'objets sans exposer les détails de leur création directement au code client. Contrairement au pattern Builder qui vise à construire des objets étape par étape, le pattern Factory se focalise sur la délégation de la création à une méthode dédiée, simplifiant ainsi le code et permettant une meilleure gestion des instances créées.

4. Qualité des codes et gestion de projets

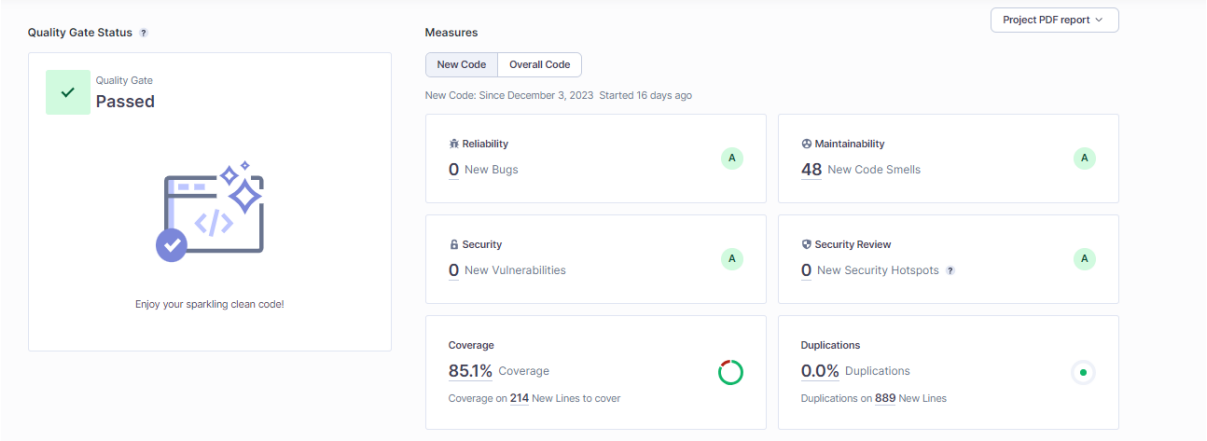
Dans le cadre de la gestion de la qualité de notre code et de notre projet, nous avons intégré SonarQube en tant qu'outil de mesure pour évaluer la qualité de notre base de code. Cette approche nous a permis d'obtenir des données significatives et des métriques précises pour évaluer notre travail. Les statistiques résultantes de cette analyse sont les suivantes : nous avons développé un total de 44 tests unitaires et écrit 19 scénarios Gherkin.

Nombre de tests Unitaires : 44

Nombre de scénarios gherkin : 19

Pourcentage de coverage : 85%

Pourcentage de duplication de codes : 0%



5. Rétrospective

Finalement, grâce à ce projet, nous sommes tous montés en compétences. Nous avons appris les bases de la conception logicielle, à travers l'élaboration des diagrammes UML : Diagramme de cas d'utilisation, diagramme de classe, diagramme de séquence. Nous avons appris les spécificités des designs pattern et surtout à bien les utiliser dans notre projet. Nous avons également eu l'occasion d'expérimenter les scénarios gherkin et les tests cucumber à travers l'écriture des User Story et des scénarios. En somme, tout le travail effectué et réalisé se rapproche de ce que nous verrons bientôt en entreprise.

Wassim - PO : En tant que PO, j'étais en charge de l'avancement et de la livraison des features dans le temps imparti, j'ai réalisé un diagramme de gant en début de projet et j'ai eu la charge de hiérarchiser les features à réaliser dans le but d'avoir le plus de valeur possible.

Valentin - SA : En tant qu'architecte, j'ai eu pour mission de définir les grandes lignes de l'architecture du projet. Cela n'a pas été facile lorsqu'il a fallu changer l'architecture et le faire évoluer. Nous avons su utiliser des design patterns pour simplifier l'implémentation et avoir une architecture cohérente lisible et compréhensible.

Aubin - QA : Concernant la qualité du code, nous sommes fiers d'avoir plus de 80% de coverage. Nous disposons d'un code fonctionnel et couvert par des tests unitaires et des scénarios cucumber. Nous avons une faible dette technique, aucun bug et aucune duplication de code.

OPS : Étant une équipe de 3 dans notre projet, nous avons tous les 3 plus ou moins géré cette partie. Nous avons créé des issues suivant le template mis en place, nous avons créer des branches afin de pouvoir avancer chacun de notre coté sans nous empiéter les uns sur les autres.

6. Auto-évaluation

En ce qui concerne l'auto-évaluation, nous estimons avoir bien assimilé les concepts fondamentaux de la conception logicielle. Cependant, notre projet a pris du retard, et ce malgré nos efforts, nous en sommes conscients. Ceci est en partie attribuable au fait que notre équipe se compose uniquement de trois membres. Nous avons identifié plusieurs points à améliorer :

- **La communication**. Une meilleure communication nous aurait aidé à mieux gérer les conflits lors de la fusion du code, mais aussi à mieux avancer.
- **L'organisation.** notamment dans la gestion du temps et des ressources. En effet, nous avons mal analysé le temps à notre disposition. Nous pensions au départ avoir jusqu'à décembre pour implémenter les fonctionnalités de la V1, et avons été pris de cours lors de l'annonce des nouvelles fonctionnalités.
- **La gestion du Repo** notamment le lien entre les commits et les issues

Malgré ces défis, nous avons tout de même redoublé d'efforts pour apporter le plus de valeur possible. Pour cette raison, nous avons préféré concentrer nos efforts sur un nombre limité de fonctionnalités à forte valeur ajoutée plutôt que de disperser nos ressources sur un grand nombre de fonctionnalités à faible valeur ajoutée. Cette approche visait à maximiser l'impact dans le cadre des délais impartis.