POLYTECH | UNIVERSITÉ CÔTE D'AZUR

RAPPORT DE CONCEPTION LOGICIELLE

Equipe K

Année scolaire 2023-2024

Clervie Causer : OPS

Benjamin Neuville-Bruguière : SA

Romain Pellegrini : QA

Julien Soto : QA

Carla Wagschal : PO

Sommaire

1. PÉRIMÈTRE FONCTIONNEL : HYPOTHÈSES, LIMITES, EXTENSIONS, POINTS FORTS ET POINTS FAIBLES

1.1 Hypothèses de travail

Hypothèse 1 : Le restaurant propose des menus qui sont directement associés à un créneau défini (par le restaurant).

Hypothèse 2 : Lorsqu'une note est donnée à un utilisateur, un restaurant ou un livreur, elle est supposée comprise entre 0 et 5.

1.2 Extensions choisies et éléments spécifiques

Nous avons choisi deux extensions.

Pour l'extension moins simple, nous avons choisi la diversité des menus [EX3]. Pour une meilleure compréhension et un exemple visuel de ce qui va suivre, nous vous invitons à examiner l'annexe 1. Dans notre implémentation, lorsqu'un manager de restaurant crée un menu, il se décompose en plusieurs choix d'éléments. Il peut proposer plusieurs boissons, plusieurs plats et plusieurs desserts par exemple et donnera un nombre qui sera le nombre d'éléments maximum possible que l'utilisateur pourra sélectionner. Par exemple, si le menu est composé d'un choix entre salade ou soupe pour l'entrée avec une seule sélection possible et d'un choix entre steak et frites pour le plat avec deux sélections possibles, l'utilisateur pourra prendre en entrée soit la salade, soit la soupe et en plat soit le steak, soit les frites, soit les deux, soit deux de l'un ou l'autre. Dans la même idée, le manager de restaurant va indiquer les choix pour les composants de ses éléments avec par exemple plusieurs ingrédients et plusieurs sauces avec encore un nombre maximal de choix. Reprenons l'exemple précédent, le manager peut indiquer que la salade se compose d'une sauce au choix entre vinaigrette et sauce César avec une sélection possible, de crudités au choix entre tomates, salade verte et concombre avec trois sélections possibles. L'utilisateur pourra alors choisir une sauce soit la vinaigrette soit la sauce César et des crudités soit les tomates, soit la salade verte, soit le concombre, soit seulement deux, soit les trois.

Le manager de restaurant peut également proposer des éléments supplémentaires dans chacun de ses menus. Ces suppléments sont payants et l'utilisateur peut en choisir autant qu'il souhaite dans le menu. Dans notre exemple, le manager de restaurant peut donner la possibilité de prendre une glace et/ou une salade de fruits et l'utilisateur pourra alors les sélectionner. Au même titre, le manager de restaurant peut proposer des composants supplémentaires à ses éléments qui sont payants et sans quantité limitée. Dans la salade de notre exemple, il peut proposer des croûtons, des dés de jambons, des olives noires et l'utilisateur pourra alors sélectionner autant de suppléments qu'il souhaite sans être limité. Il faut savoir que les éléments supplémentaires ont eux aussi des composants proposés par le manager et donc également des

éléments supplémentaires.

Cette implémentation laisse à l'utilisateur la possibilité de composer son menu à son goût dans la limite des propositions du menu. Tous les choix d'éléments (ou de composants) doivent avoir une quantité sélectionnée entre 1 et le nombre choisi par le manager de restaurant. Si l'utilisateur ne choisit pas, il aura le premier élément (ou composant) proposé par défaut, il ne peut donc pas choisir de ne rien prendre. Une limite de cette implémentation est le fait que lorsque le manager de restaurant laisse plus d'une sélection possible dans un choix d'élément ou un choix de composant, l'utilisateur peut prendre plusieurs fois la même chose. Dans notre exemple, si le manager de restaurant ne veut pas qu'un utilisateur prenne deux steaks, il faudra alors qu'il sépare le choix d'élément en deux avec par exemple un choix d'élément viande avec un steak et une sélection possible, un choix d'élément accompagnement avec les frites et une sélection possible. Cela réduit les possibilités et l'utilisateur sera obligé d'avoir un steak et des frites puisque lorsqu'un choix d'élément ne contient qu'un seul élément, il est choisi par défaut. Il est donc de la responsabilité du manager de restaurant de mettre un élément qui correspondrait à sans viande ou sans accompagnement.

Pour l'extension simple, nous avons choisi le système de recommandation. Les utilisateurs ont la possibilité de donner un avis avec une note au restaurant dans lequel la commande a été passée et le livreur. Le livreur peut noter l'utilisateur qu'il vient de livrer. Dans notre implémentation, il n'y a pas de limite de valeur de note. C'est-à-dire que plus le chiffre est bas, plus l'avis est considéré comme défavorable sinon, plus la valeur est haute, plus la personne l'avis donné est favorable.

1.3 Points non implémentés

Il y a deux éléments manquants dans notre projet.

Le premier est la possibilité pour les restaurants de proposer des prix différents en fonction de l'utilisateur [O9]. Nous ne l'avons pas implémenté car, lorsque nous nous sommes rendu compte que cet élément était manquant, le projet était déjà très avancé et nous aurions dû modifier une très grande partie de notre code qui n'était pas préparé à un tel changement ce qui nous aurait pris trop de temps dans le temps imparti.

Le deuxième concerne l'extension obligatoire. Nous n'avons pas implémenté les commandes AW [EX1]. Cette partie aurait pu être implémentée si on avait eu plus de temps. En effet, son implémentation est similaire à celle que l'on a utilisée pour les commandes buffets et ne nous aurait pas demandés de modification de code.

2. CONCEPTION

2.1 Le glossaire

Buffet : type de commande directement proposé par le restaurant, dont les menus sont prédéfinis et dont seule la quantité de chacun peut être modifiée.

Choix élément : (noté ChoixElement dans le code) Partie du menu contenant la liste des éléments que l'utilisateur peut choisir pour une section de son menu (Exemple : Entrée : Salade, Assortiment de fruits de mer ; Plat : Steak, Frites)

Choix composant : (noté ChoixComposant dans le code) Partie du menu contenant la liste des composants que l'utilisateur peut choisir dans son élément pour une catégorie donnée (Exemple : Catégorie Sauce : Ketchup, Mayonnaise, Moutarde ; Catégorie Parfum: Vanille, Fraise, Chocolat)

Composant : Partie du menu tout en bas de l'échelle. Correspond à un ingrédient d'un élément, une préférence pour un élément, etc. (Exemple : Une tomate dans une salade, saignant pour une viande)

Elément : Partie du menu concernant un produit. (Exemple : Une salade, une viande)

Supplément élément : Élément supplémentaire payant que l'utilisateur peut ajouter à son menu

Supplément composant : Composant supplémentaire payant que l'utilisateur peut ajouter à un élément

Statut de commande : dépendant des statuts de Commandable (cf. 3.1 DPs en détails).
(définition des statuts particuliers seulement, les autres sont explicites)

- EN_PREPARATION : Lorsqu'un Commandable passe EN_PREPARATION, le statut de la commande passe EN_PREPARATION. Tant qu'un menu n'est pas encore en préparation, la commande reste en préparation
- PRETE : Lorsque tous les Commandables sont en statut "PRET".
- ANNULEE : Si tous les Commandables sont en statut "ANNULE".

2.2 Le diagramme de cas d'utilisation et User Stories

Voici notre diagramme de cas d'utilisation modifié en intégrant les extensions.

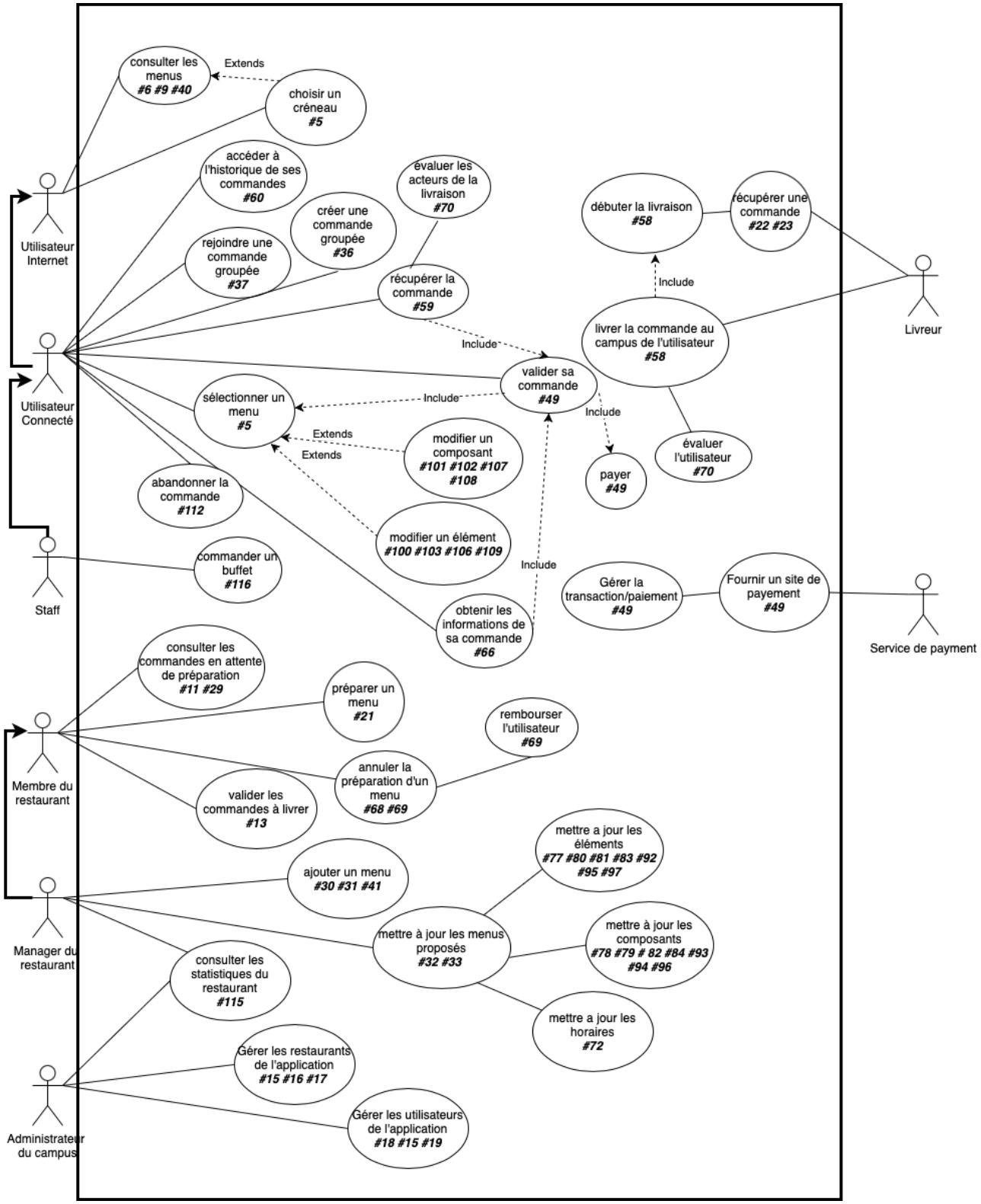


Figure 1 : Diagramme de cas d'utilisation

Référence des cas d'utilisation correspondant à une extension :

- **Acteur : Utilisateur connecté**
    - évaluer les acteurs de la livraison [EX7]
        - **#70** : US17 système d'évaluation
        - User Story :
          En tant qu'utilisateur,
          Je veux pouvoir noter le restaurant dans lequel j'ai commandé
          afin que les autres utilisateurs puissent voir mon avis
          Critère d'acceptation : Un autre utilisateur peut voir la note qui vient d'être ajoutée au restaurant

          En tant qu'utilisateur livré,
          Je veux pouvoir noter mon livreur
          afin que les autres utilisateurs puissent avoir un retour sur l'efficacité du livreur
          Critère d'acceptation : Un autre utilisateur peut voir la note qui vient d'être ajoutée au livreur

    - modifier un composant [EX3]
        - **#101** : Supprimer le choix d'un composant pour un utilisateur
          #102 : Supprimer le choix d'un supplément composant pour un utilisateur
          #107 : Modifier le choix d'un composant pour un utilisateur
          #108 : Modifier le choix d'un supplément composant pour un utilisateur
        - User Story :
          En tant qu'utilisateur,
          Je veux modifier le choix d'un composant (ou supplément composant) d'un menu
          afin de mettre à jour mon menu
          Critère d'acceptation : Le composant (ou supplément composant) du menu a été modifié

          En tant qu'utilisateur,
          Je veux supprimer le choix d'un composant d'un menu
          afin de mettre à jour mon menu
          Critère d'acceptation : Le composant a été supprimé et remplacé par le composant par défaut

          En tant qu'utilisateur,
          Je veux supprimer le choix d'un supplément composant d'un menu
          afin de mettre à jour mon menu
          Critère d'acceptation : Le supplément composant a été supprimé

- modifier un élément [EX3]
  - **#100** : Supprimer le choix d'un élément
    #103 : Supprimer le choix d'un supplément élément pour un utilisateur
    #106 : Modifier le choix d'un élément d'un menu pour un utilisateur
    #109 : Modifier le choix d'un supplément élément pour un utilisateur
  - User Story :
    En tant qu'utilisateur,
    Je veux modifier le choix d'un élément (ou supplément élément) d'un menu
    afin de mettre à jour mon menu
    Critère d'acceptation : L'élément (ou supplément élément) du menu a été modifié

    En tant qu'utilisateur,
    Je veux supprimer le choix d'un élément d'un menu
    afin de mettre à jour mon menu
    Critère d'acceptation : L'élément a été supprimé et remplacé par l'élément par défaut

    En tant qu'utilisateur,
    Je veux supprimer le choix d'un supplément composant d'un menu
    afin de mettre à jour mon menu
    Critère d'acceptation : Le supplément de l'élément a été supprimé

- **Acteur : Staff**
  - commander un buffet [EX1]
    - **#116** : Buffet partie Métier
    - User Story :
      En tant que membre du staff,
      Je veux pouvoir commander un buffet dans un restaurant
      afin de fêter un évènement
      Critère d'acceptation : La commande du staff est une commande buffet

- **Acteur : Manager du restaurant**
  - ajouter un menu [EX3]
    - **#41** : Création des classes qui composent les menus
    - User Story :
      En tant que manager du restaurant,
      Je veux créer un menu avec des options
      afin de laisser le choix à l'utilisateur
      Critère d'acceptation : Le menu propose un choix à l'utilisateur avec des aliments définis

      En tant que manager du restaurant,

Je veux créer un menu avec des suppléments
afin de laisser du choix supplémentaire à l'utilisateur
Critère d'acceptation : Le menu propose des suppléments sur certains aliments définis

- <u>mettre à jour les éléments [EX3]</u>
  - **#77** : Modification d'un élément
    #80 : Modification d'un supplément élément
    #81 : Modification d'un choix élément
    #83 : Modification du nombre de choix d'un élément
    #92 : Supprimer un élément d'un menu
    #95 : Supprimer un supplément élément
    #97 : Supprimer un choix élément d'un menu
  - User Story :
    En tant que manager du restaurant ,
    Je veux modifier un élément (ou un supplément élément ou le choix élément ou le nombre de choix possible pour un choix élément) d'un menu
    afin de mettre à jour les propositions
    Critère d'acceptation : Le menu se voit modifié

    En tant que manager du restaurant ,
    Je veux supprimer un élément (ou un supplément élément ou un choix élément) d'un menu
    afin de mettre à jour les propositions
    Critère d'acceptation : Le menu se voit modifié

- <u>mettre à jour les composants [EX3]</u>
  - **#78** : Modification d'un composant d'un menu
    #79 : Modification d'un supplément composant
    #82 : Modification d'un choix composant
    #84 : Modification du nombre de choix d'un choix composant
    #93 : Supprimer un composant d'un menu
    #94 : Supprimer un supplément composant d'un menu
    #96 : Supprimer un choix composant d'un menu
  - User Story :
    En tant que manager du restaurant ,
    Je veux modifier un composant (ou un supplément composant ou un choix élément ou le nombre de choix possible pour un choix composant) d'un menu
    afin de mettre à jour les propositions
    Critère d'acceptation : Le menu se voit modifié

En tant que manager du restaurant ,
Je veux supprimer un composant (ou un supplément composant ou un choix élément) d'un menu
afin de mettre à jour les propositions
Critère d'acceptation : Le menu se voit modifié

- **Acteur : Livreur**
  - commander un buffet [EX7]
    - **#70** : US17 système d'évaluation
    - User Story :
      En tant que livreur,
      Je veux pouvoir noter l'usager
      afin de le signaler à l'administrateur du campus
      Critère d'acceptation : L'administrateur du campus peut voir la note ajoutée à l'utilisateur

2.3 Le diagramme de classe

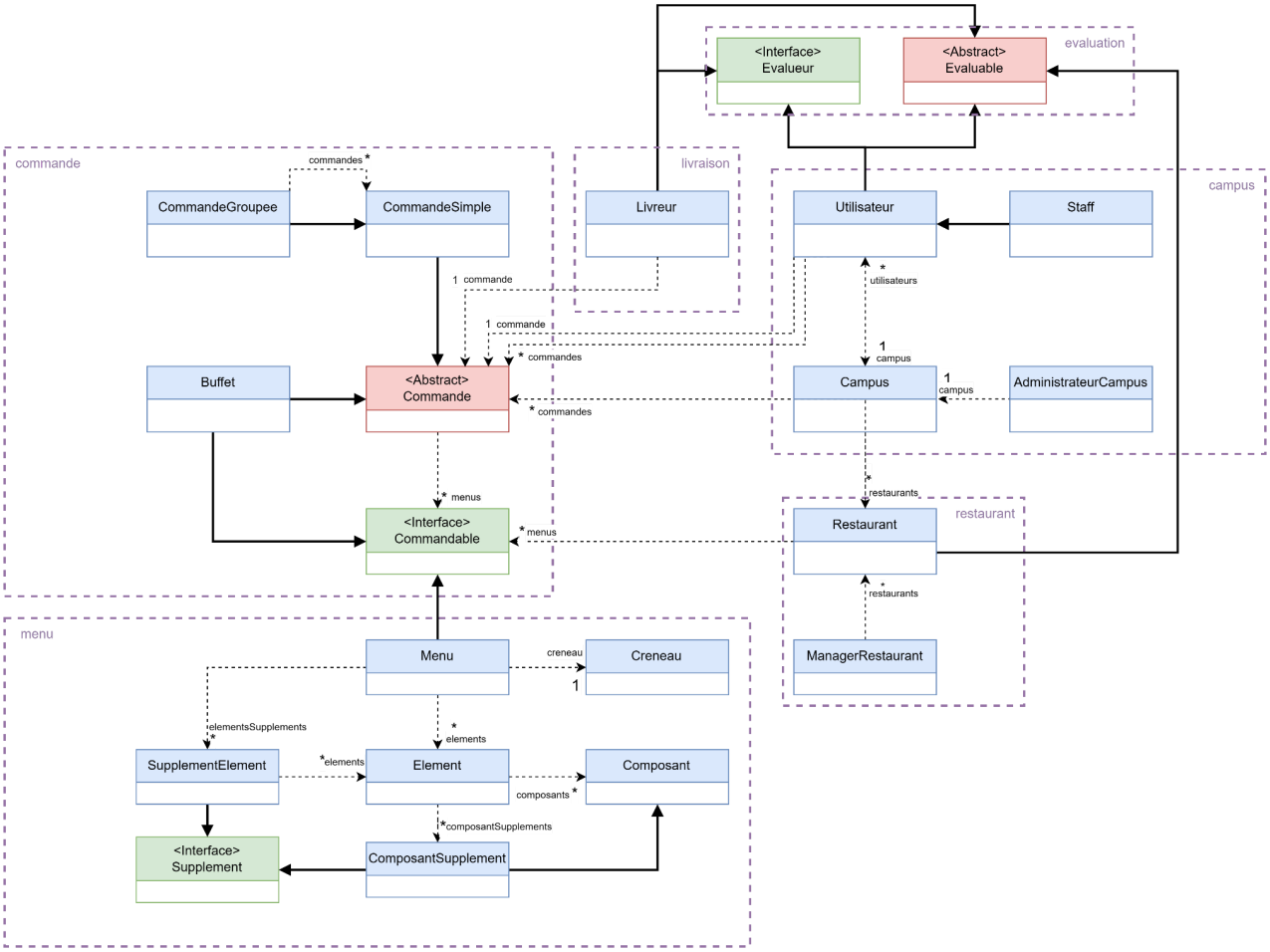


Figure 2 : diagramme de classe global simplifié

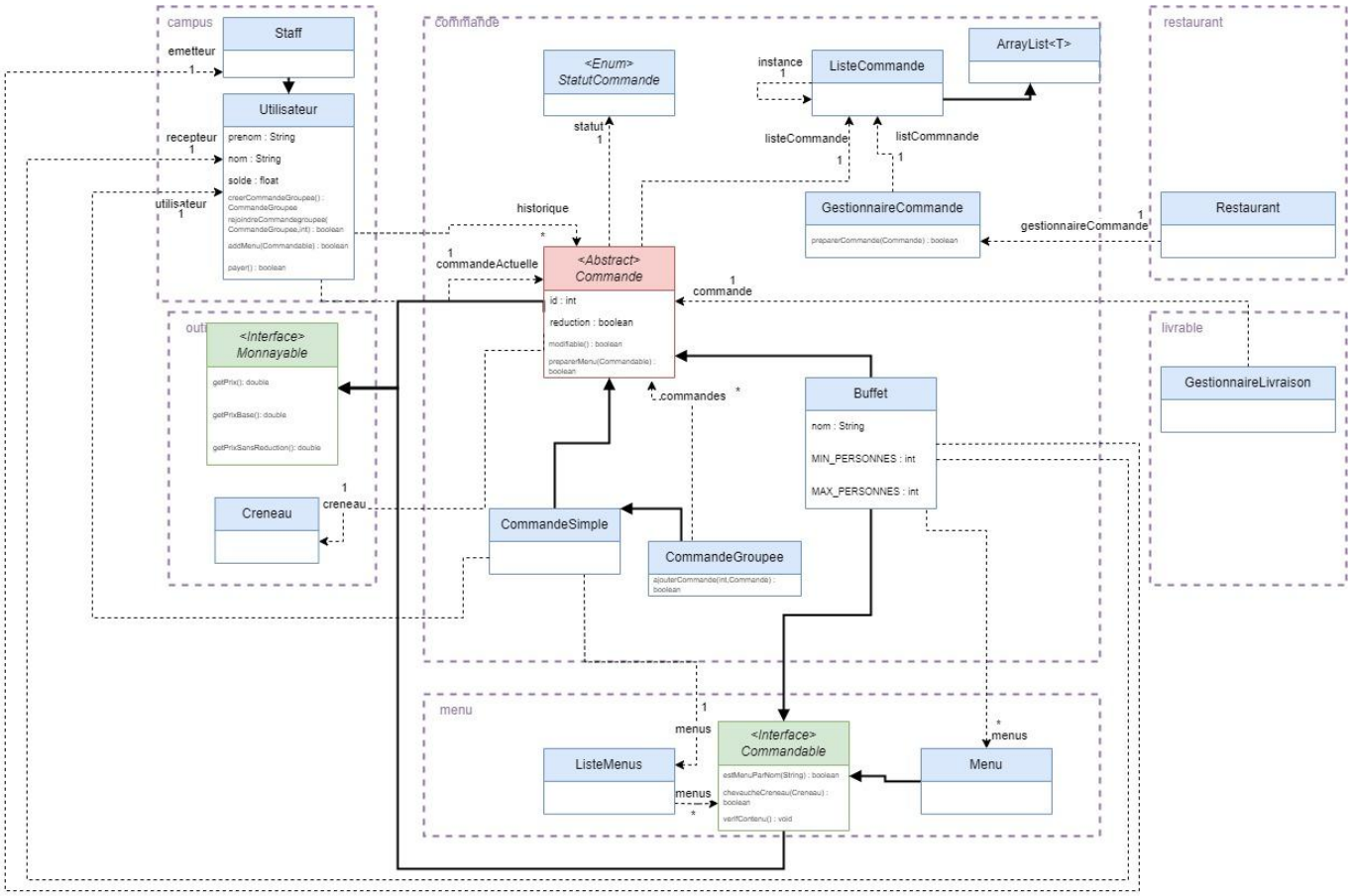


Figure 3 : package commande et relations avec autres packages

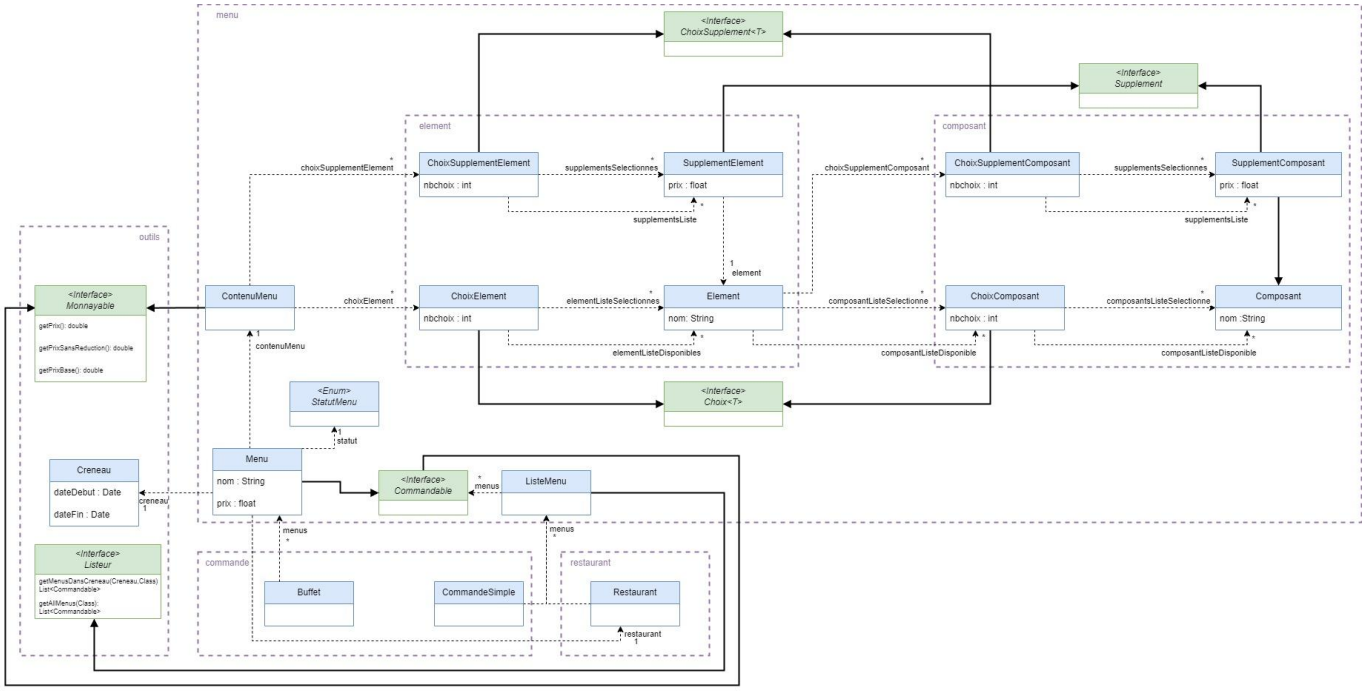


Figure 4 : package menu et relations avec les autres packages

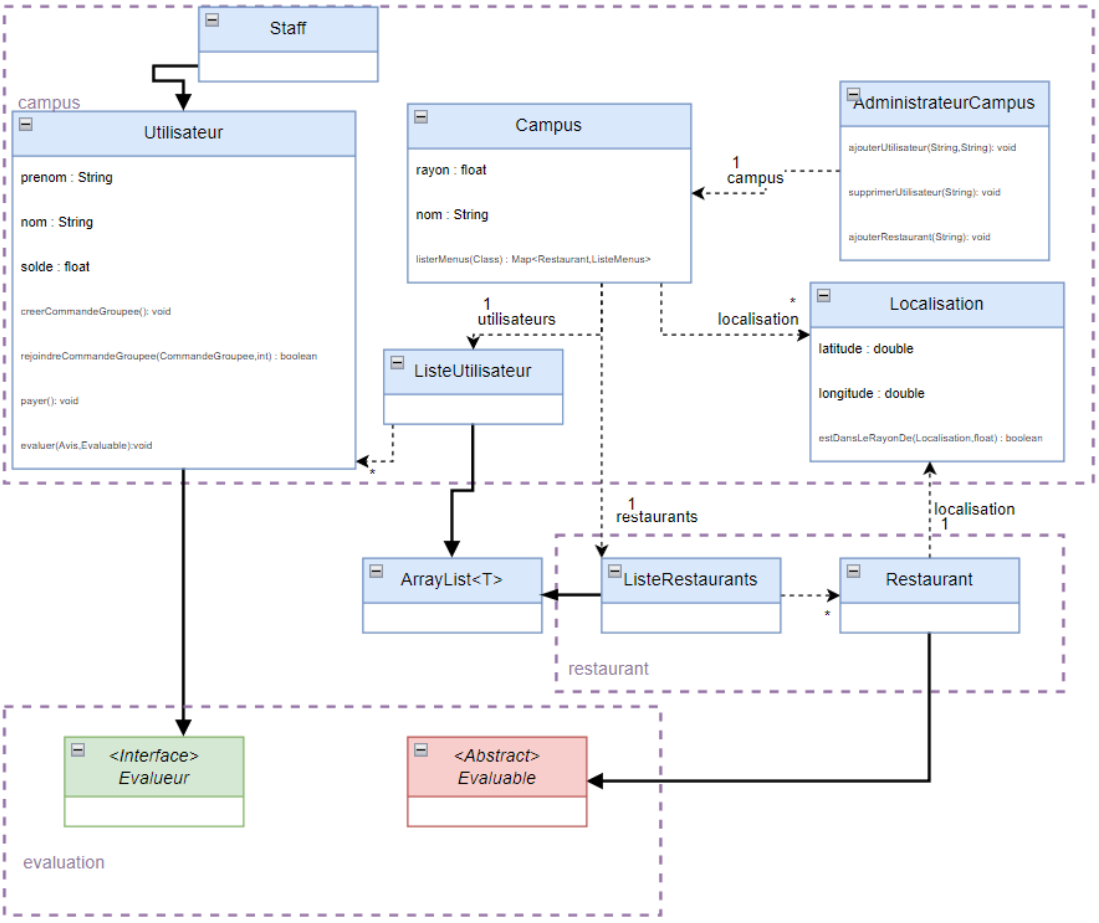


Figure 5 : package campus et relations avec les autres packages

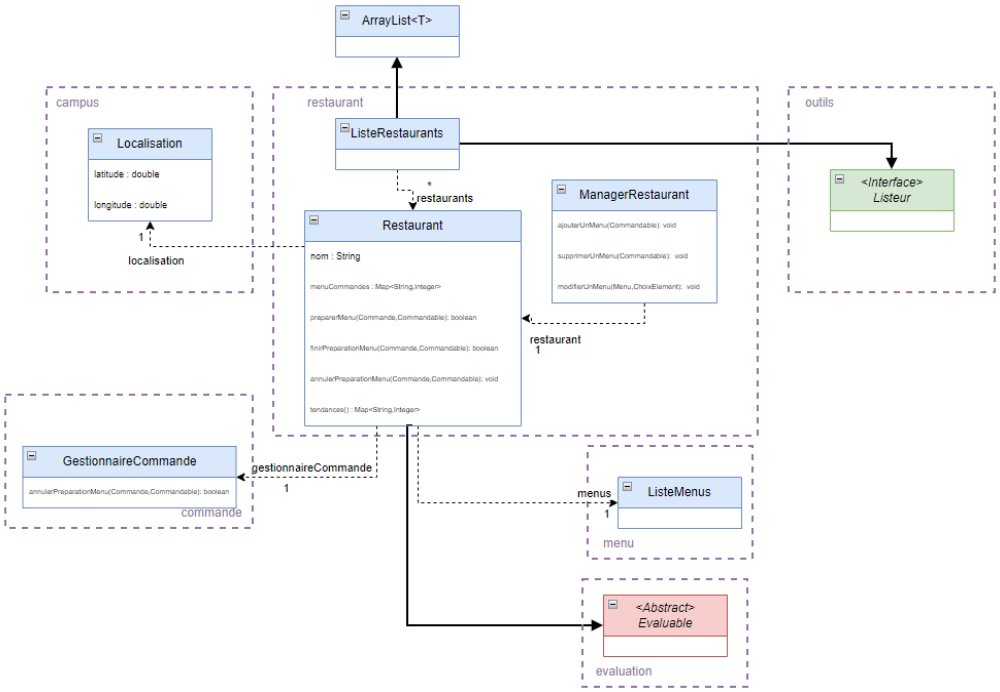


Figure 6 : package restaurant et relations avec les autres packages

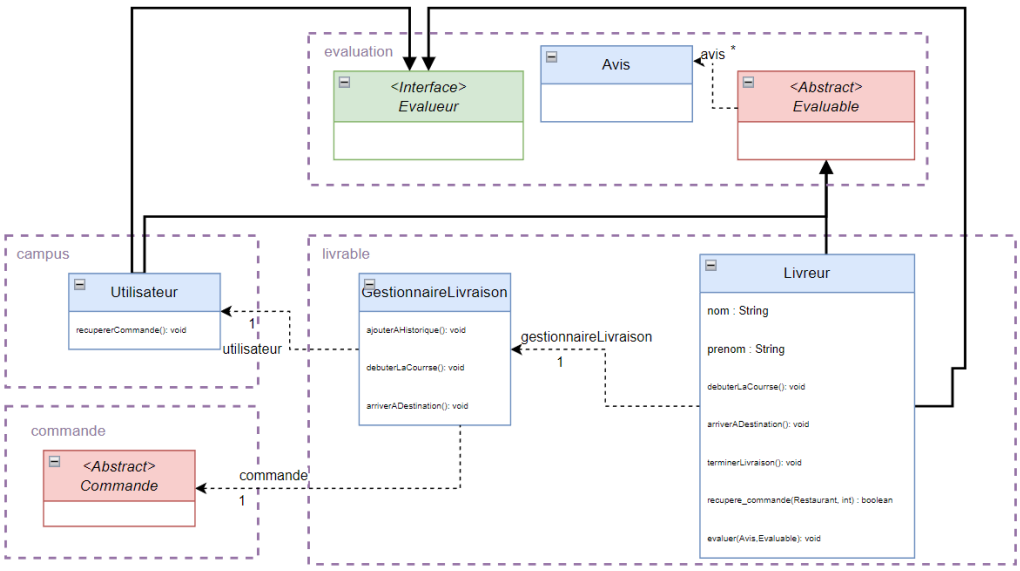


Figure 7 : package livrable et relations avec les autres packages

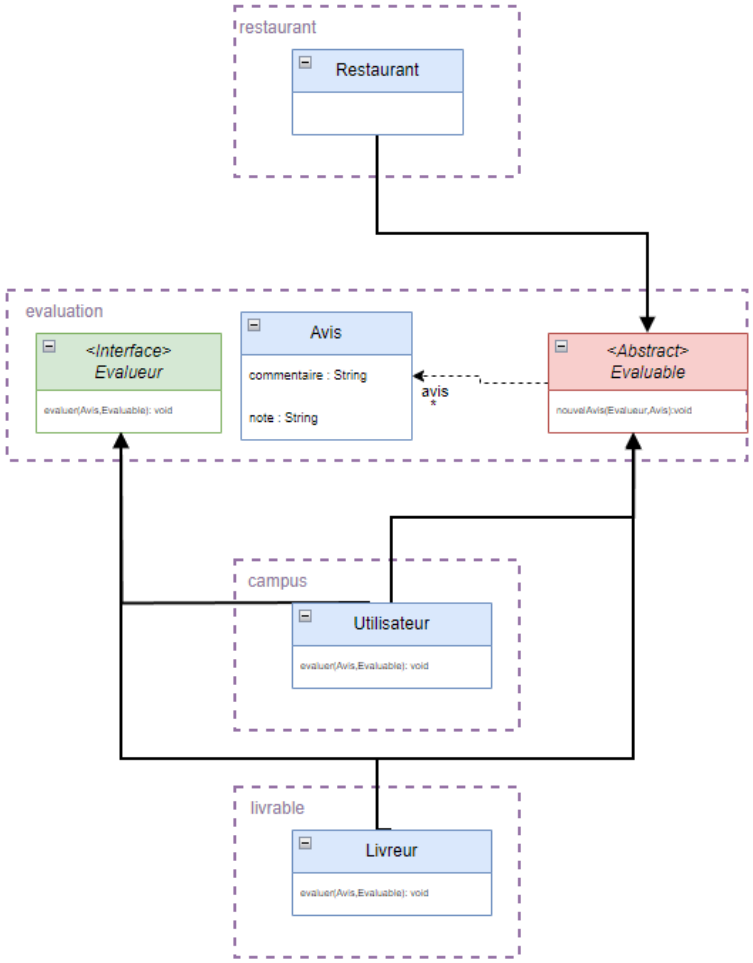


Figure 8 : package evaluation et relations avec les autres packages

2.4 Un diagramme de séquence

Voici notre diagramme de séquence représentant en globalité notre architecture. Nous ne mettons pas en avant l'extension sur la diversité des menus pour que le diagramme reste lisible.

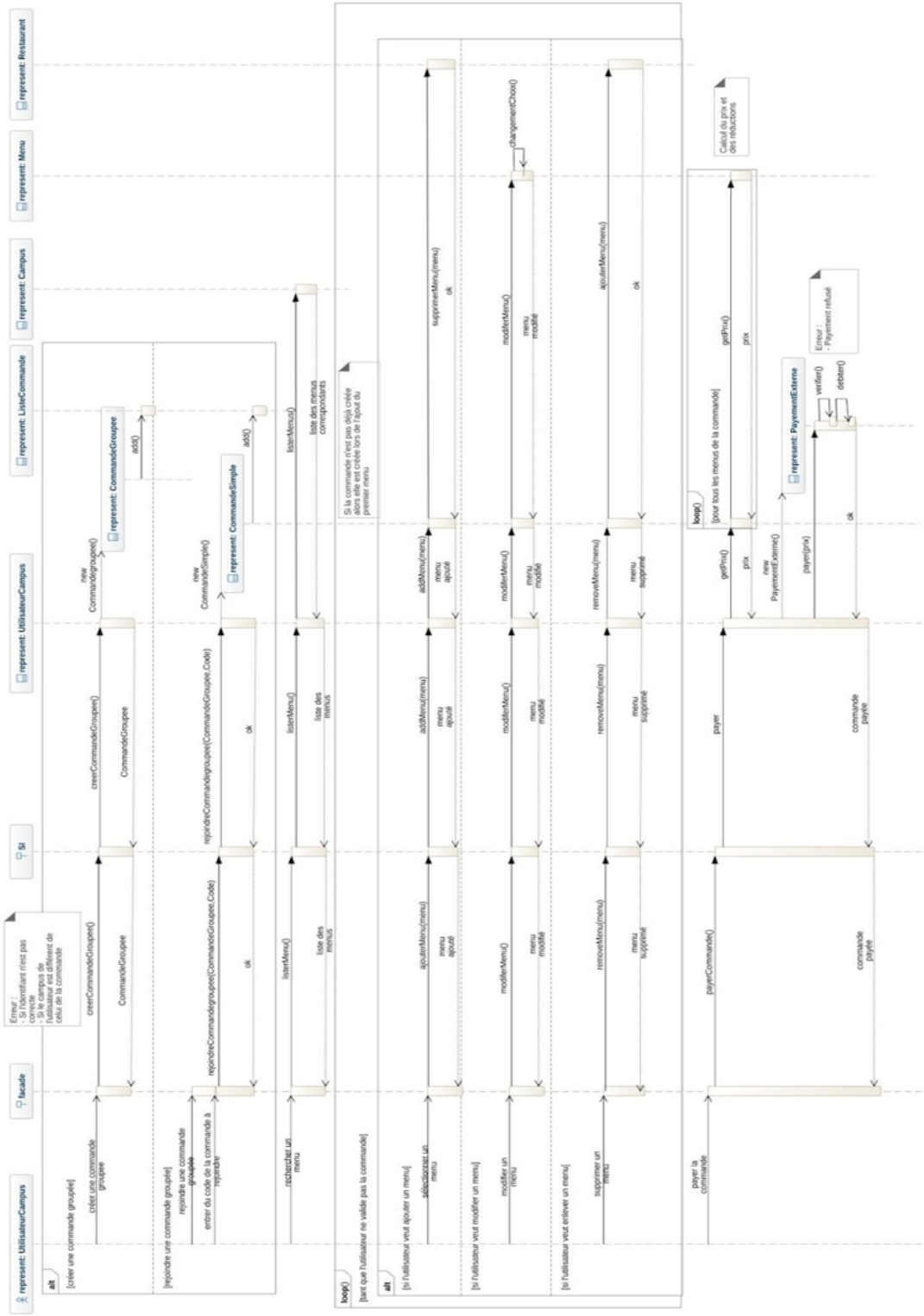


Figure 9 : Diagramme de séquence

2.5 Une esquisse du diagramme de composants

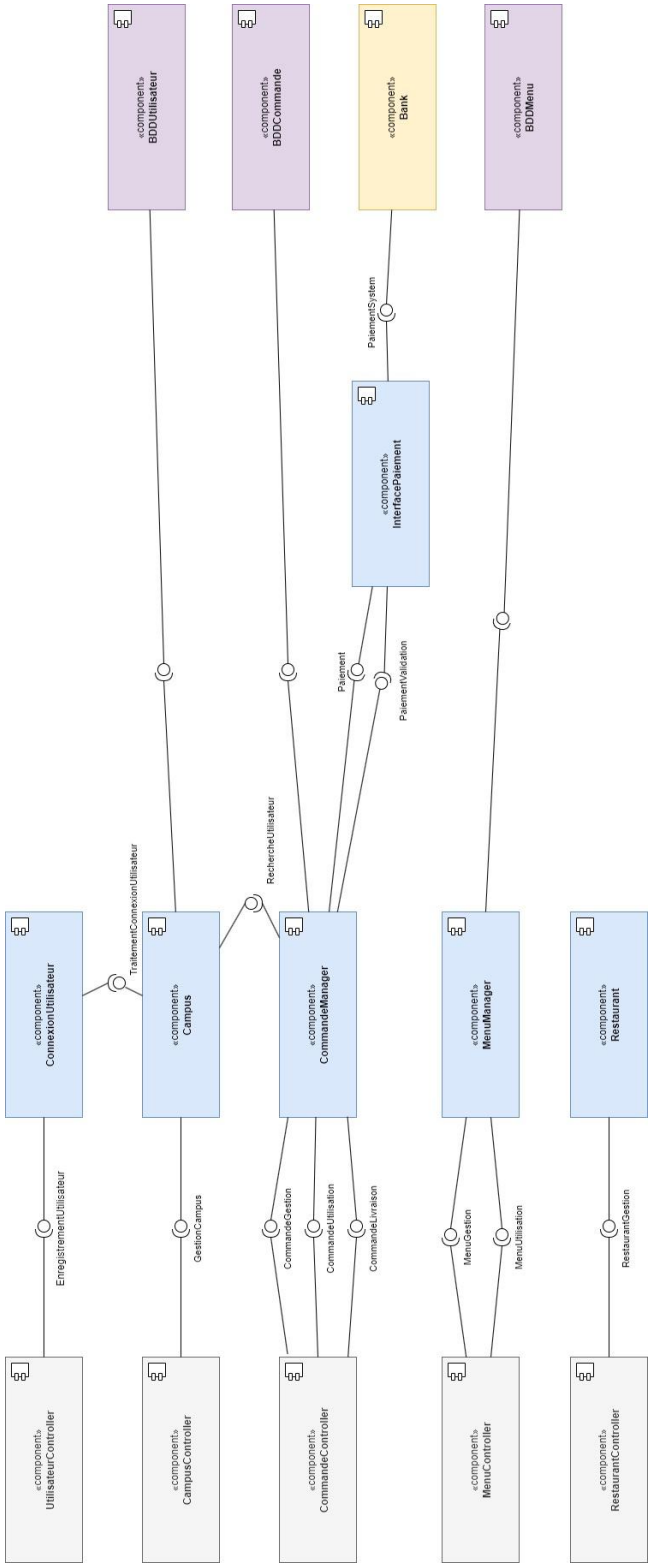


Figure 10 : Diagramme de composants

3. DESIGN PATTERN (DP) APPLIQUÉS OU PAS

3.1 DPs en détails

- **Les Observers**
  - Commande et Menu

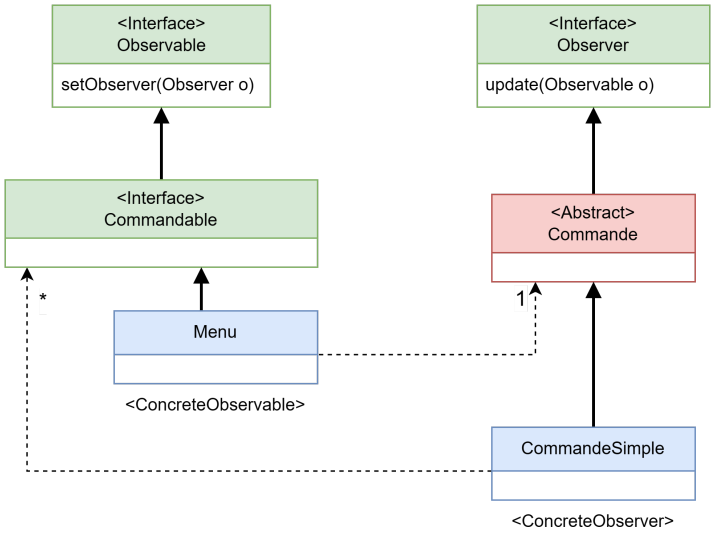


Figure 11 : DP Observer avec Commande / Menu

On implémente les interfaces Observable (pour les composantes de notre commande) et Observer (pour notre abstraction de classe Commande).

Problématique : L'objectif est que lorsque le statut d'un élément Commandable change, la Commande qui le contient doit être informée de ce nouveau statut afin que la Commande puisse mettre à jour son propre statut.

Grâce à l'application de ce Design Pattern Observer, nous allons pouvoir répondre à la problématique précédente. Lorsqu'un restaurant va mettre à jour le statut d'un Commandable (PRET, EN_PREPARATION, ANNULE), il va permettre de mettre à jour le statut de la Commande qui correspond (cf. 2.1 Le glossaire).

  - CommandeGroupee et CommandeSimple

Le principe est le même, c'est une application du DP nous permettant d'avoir le statut de la CommandeGroupee en fonction des CommandeSimple qui la composent.

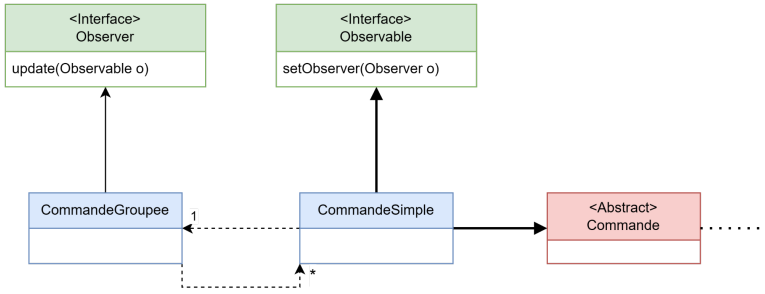


Figure 12 : DP Observer avec CommandeGroupee / CommandeSimple

3.2 Autres DPs

3.2.1 - DP implémenté abîmé

- **Les builders**

Ils vont s'appliquer sur la partie construction d'un Menu.

Problématique : Puisque chaque Menu est prédéfini sur un Créneau particulier, cela rend donc difficile l'ajout de Menu similaires.

Le but était donc de rendre plus simple l'ajout de Menu sans avoir à renseigner tout le contenu de celui-ci (éléments, composants, suppléments…) mais en pouvant renseigner seulement le Créneau auquel on l'associe.

Pourquoi abîmé ? Tout d'abord, l'état actuel du DP ne permet pas d'utiliser le Builder de Menu correctement. Il faut tout lui renseigner pour pouvoir lui faire faire générer un Menu, ce qui ne correspond pas à ce qu'on veut ici. C'est aussi une raison pour laquelle nous ne l'avons pas implémenté avec le reste du code. Avec plus de temps, le builder aurait été mis correctement en place, et donc ajouté au reste du code.

3.2.2 - DP non-implémenté

3.2.2.1 - Aligné avec notre implémentation

- **Décorateurs**

Il s'appliquerait à la partie concernant les Réductions.

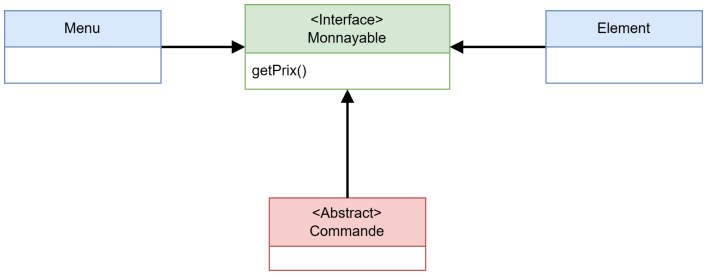


Figure 13 : Représentation actuelle des prix, sans réduction

Problématique : le but est de pouvoir appliquer différentes réductions soit à des commandes, soit à des menus, soit à des éléments… sans impacter la maintenabilité de notre code.

Pour cela, le DP Réduction s'appliquerait parfaitement à notre cas. Nous pouvons relever 3 composants différents dans la représentation actuelle du prix.

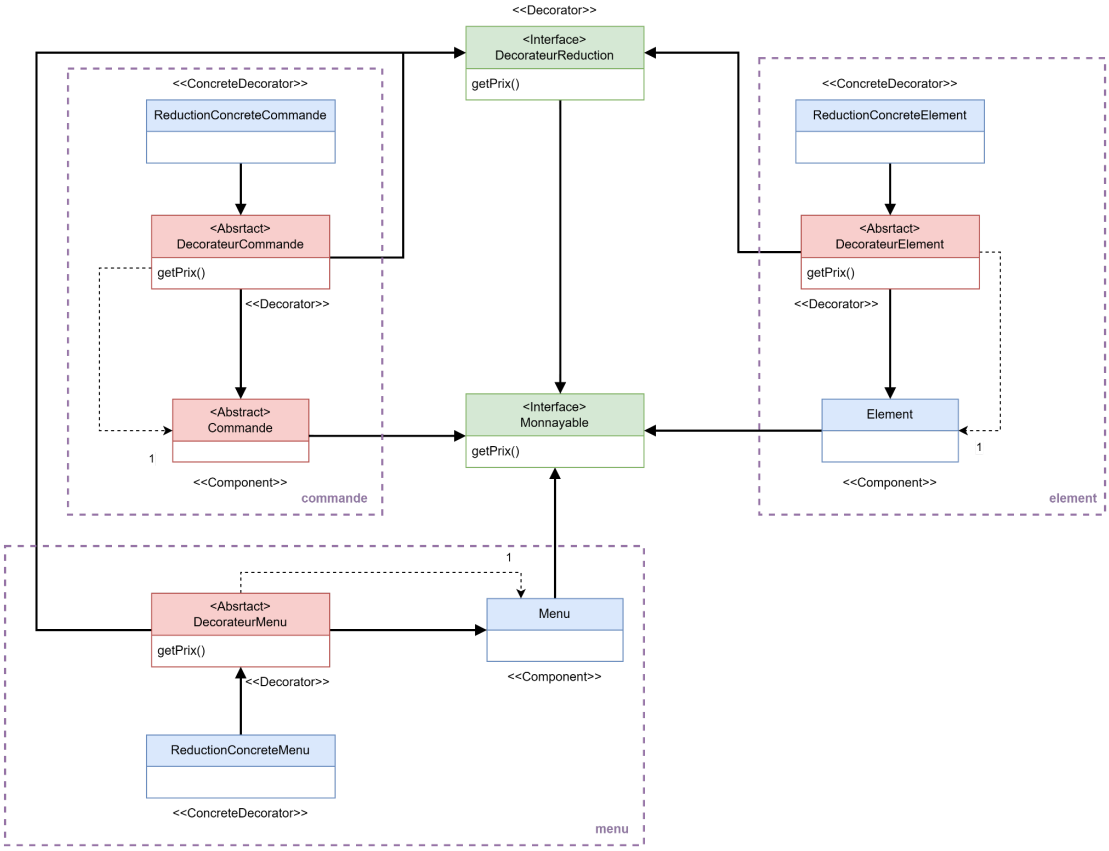


Figure 14 : Application du DP Décorateur

Malheureusement, par manque de temps, il a été impossible de le mettre en place.

- **Visiteur**

Il s'appliquerait à la partie Paiement.

Problématique : le but est d'avoir différents types de paiement selon le type de commande sans avoir à impacter la maintenabilité et permettre facilement l'ajout de nouveau type de commande.

En appliquant ce DP, cela permettrait donc de résoudre la problématique mais aussi de réduire le couplage entre les classes dépendant du paiement, et du prix associé aux différents utilisateurs.
Actuellement, lors du paiement, nous avons, dans l'utilisateur, la méthode appelante pour le paiement. Seulement, cela rend l'extensibilité compliquée (nouvelles conditions appliquées à certains types de commande, commande payée par facturation…). Actuellement, s'il y a facturation, le prix de la commande vaut 0, ce qui ne correspond pas à une bonne

implémentation de la facturation.

Le visiteur permet, pour chaque commande, d'appliquer le paiement associé. Il calcule aussi le prix en fonction de l'utilisateur associé à la commande.

3.2.2.2 - Non aligné avec notre implémentation

- **Composite**

Il s'appliquerait à l'extension "Menu composable".

Problématique : le but est d'avoir une structure du ContenuMenu plus intelligente et surtout plus compréhensible.

En analysant notre problème (cf. extensions choisies) de plus près, nous avons relevé que les imbrications d'objets avaient une limite et étaient ordonnées (ChoixElements comporte des Elements, Element comporte des ChoixComposants, ChoixComposant comporte des Composants). C'est pourquoi notre DP Composite n'améliore pas la structure du ContenuMenu.

- **Facade**

Il s'appliquerait à la partie Paiement.

Problématique : étant donné qu'il existe plusieurs façons de payer une commande (selon le type de commande), le but est de réduire le couplage du code sur cette partie et simplifier la maintenabilité du code, permettre l'extensibilité des commandes.

Le but était donc de réduire le couplage actuel du paiement d'une commande, qui est dépendante de son type, mais aussi des différents prix, selon le type d'utilisateur et enfin appliquer les différentes réductions. Seulement, en appliquant le paterne Décorateur pour les réductions sur les différents composants (cf. DP Decorateur), nous n'avons pas de réels calculs à effectuer pour les différents utilisateurs, juste une récupération de valeur de prix associés et d'autres DP sont plus pertinents.
Après une analyse plus poussée du DP Facade, nous nous sommes rendu compte qu'il ne correspondait pas à la résolution de notre problématique. Principalement car il n'y a pas de réelle complexité entre les sous-classes présentées précédemment, et surtout parce qu'il y a d'autres DP plus adaptés.

4. QUALITÉS DES CODES ET GESTION DE PROJETS

4.1 Tests

Les tests pris en charge dans notre processus de développement suivent une méthodologie bien définie.

Tout commence par la rédaction d'une User Story détaillant une ou plusieurs fonctionnalités à implémenter pour répondre aux besoins de l'application. Ensuite, les scénarios sont rédigés sous le format Gherkin, décrivant de manière précise les étapes à suivre. Ces scénarios sont ensuite implémentés en Java pour créer les tests automatisés. Chaque test d'intégration est conçu pour évaluer les fonctionnalités implémentées à travers des assertions, garantissant ainsi la conformité avec les spécifications définies dans la User Story. Par souci de clarté et de lisibilité, nous avons fait le choix de ne pas utiliser des picocontainers, car ils offraient une visualisation limitée de nos actions. À la place, nous avons privilégié l'utilisation de déclarations statiques pour une meilleure compréhension et une gestion plus efficace de nos tests.

Notre processus de tests, bien que rigoureux, présente actuellement une couverture de 67,2% selon l'évaluation par SonarQube (annexe_2). Cette valeur est influencée par la présence de nombreux setters et getters ainsi que les classes builders qui n'ont pas été inclus dans nos tests automatisés. Bien que ces composants soient cruciaux pour la construction et la manipulation de nos objets, leur exclusion a impacté notre taux de couverture.

4.2 Qualité du code

De manière générale, nous sommes satisfaits de la qualité de notre code. En effet, nos classes et nos tests sont organisés dans des répertoires, rendant ainsi le projet facile à explorer. Le code est clair, lisible et compréhensible malgré quelques conventions de nommage.

Grâce à la mise en place d'un processus de contrôle automatisé avec SonarQube, nous pouvons facilement repérer et corriger les potentiels bugs, vulnérabilités et security hotspots dans le code.

4.3 Gestion du projet

Afin d'assurer une bonne gestion de notre projet, nous avons décidé d'opter pour une politique de git feature branch workflow. C'est-à-dire que nous avions toujours la branche du main qui était stable et nous faisions des branches sur lesquelles coder.

Nous avons décidé qu'une branche serait dédiée à une milestone qui elle-même serait une User Story. Celle-ci pouvait prendre en compte plusieurs scénarios.

Par exemple, notre milestone 17 correspond à notre User Story 18 qui a elle-même 8 scénarios différents. Chaque scénario correspond à au moins un test Gherkin. Comme vous pouvez le voir

sur l'annexe 3, nos milestones ont une nomenclature spécifique : Milestone *numéro de la milestone* : US *numéro de la User Story* : *le nom de la User Story traitée*. Nous avons fait plusieurs issues dans une milestone, chacune correspondant à des aspects spécifiques d'implémentation pour une feature. Chaque commit est lié à une issue par le #, qui est elle-même liée à une milestone.

Cette nomenclature peut se revoir aussi dans le nommage des branches. Comme vous pouvez le voir ci-dessous sur la figure 15, chaque branche est tirée depuis la branche main et réinsérée par la suite après avoir développé la nouvelle feature.

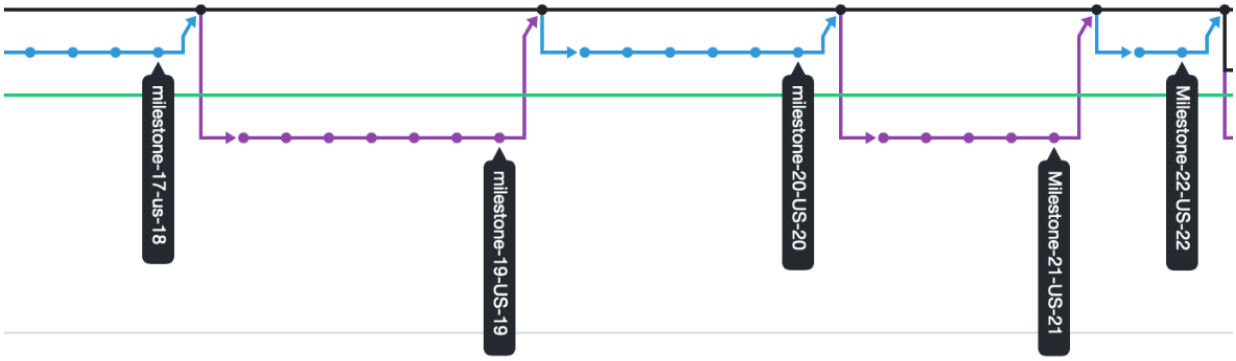


Figure 15 : Extrait de notre politique de branche

Après avoir fini avec le code et les tests sur la branche, nous faisions une Pull Request (PR). Celle-ci lançait la CI (continuous integration) qui consistait à lancer les tests avec la commande "maven clean install". Les tests étaient lancés 10 fois à la suite pour s'assurer que le test ne passait pas qu'une seule fois mais bien plusieurs fois. Si les tests passaient, une personne autre que celle ayant développé la feature, pouvait accepter la PR et donc intégrer la branche sur la branche main stable. Si les tests ne passaient pas ou bien qu'il y avait des conflits avec la branche main, il fallait régler le souci avant de pouvoir merge la branche.

Ce système mis en place a été respecté tout du long du projet. Il nous a permis de vérifier que notre code fonctionnait toujours au niveau des tests, grâce à la CI qui les lançait une fois la PR de faite.

5. RÉTROSPECTIVE ET AUTO-ÉVALUATION

L'équipe Kaleaté a su exploiter les différentes qualités de ces cinq membres à travers quatre rôles distincts pour implémenter au mieux le projet.

5.1 Rôle de Ops - Clervie Causer

En tant que Ops, j'ai dû mettre en place le git et la structure de notre dépôt. Nous nous sommes tout de suite mis d'accord sur quelle politique de branche utilisée et nous avons décidé d'une nomenclature claire. J'ai demandé à ce qu'une Pull Request soit faite à chaque fois qu'une branche devait être intégrée dans la branche main. Les tests devaient tous passer et il ne devait pas y avoir de conflits. Les commits devaient être reliés à une issue qui était reliée à une milestone.

Cette façon de procéder a été suivie et respectée mais a parfois été compliquée pour avancer tous ensemble. En effet, nous avions des parties très connectées entre elles, et nous devions donc attendre que la branche soit stable afin de continuer les features demandées. Cependant, pour la majorité du projet, nous avons réussi à avancer tous en même temps sur des branches différentes et en mettant en commun quand quelqu'un avait merge sur la branche main, afin d'éviter des conflits sur la branche par la suite.

5.2 Rôle d'Architecte Logiciel - Benjamin Neuville-Burguière

En tant qu'Architecte Logiciel, je me suis assuré de l'intégrité de ce qui est implémenté, de vérifier que les modélisations correspondent encore à l'état actuel du code et de la pertinence de l'utilisation des design patterns.

Je pense m'être penché sur l'extensibilité de notre code que trop tardivement, ce qui a entravé la progression, demandant souvent de refactoriser le code lors de l'implémentation des extensions (ex : extension obligatoire sur les types de commande) ou de contraintes manquantes relevées plus tard dans l'implémentation (ex : différents prix pour chaque utilisateur).

Le manque d'implémentation de DP est dû principalement au manque de temps. Il aurait donc fallu penser à leur utilisation plus tôt dans le code, ce qui aurait facilité la progression et l'ajout de DP à notre implémentation.

5.3 Rôle de QA - Romain Pellegrini & Julien Soto

En tant que QAs, notre responsabilité était d'assurer la qualité globale du projet. Pour cela, nous avons vérifié que nos tests et l'intégration de nos fonctionnalités respectent la méthodologie structurée que nous avions définie. Parallèlement, nous avons mis en place un processus de contrôle automatisé à l'aide de SonarQube pour signaler rapidement les potentiels bugs, vulnérabilités et security hotspots dans le code. Nous pouvions ainsi prendre des mesures pour

remédier à ces problèmes, assurant ainsi la stabilité et la sécurité du code source.

5.4 Rôle de Product Owner - Carla Wagschal

En tant que Product Owner, j'ai supervisé les User Stories en m'assurant que celles-ci soient le plus ciblées et le plus détaillées possible afin de couvrir un maximum les besoins utilisateurs. J'ai suivi l'évolution du projet en tenant à jour un récapitulatif des différentes User Stories qui avait été implémentées. Cela a permis d'avoir une vision globale pour l'équipe sur l'avancement du projet et de pouvoir coder les tests des User Stories de façon précise.

5.5 Bilan du fonctionnement de notre équipe

Pendant ce projet, l'équipe a su communiquer et s'entraider ce qui a permis d'avancer de façon régulière et de progresser chacun à notre niveau. Nous avons pu être efficaces grâce à une bonne répartition des tâches à implémenter et chaque personne a su comprendre et appliquer son rôle.

5.6 Auto-évaluation

Benjamin	Carla	Clervie	Julien	Romain
100	100	100	100	100

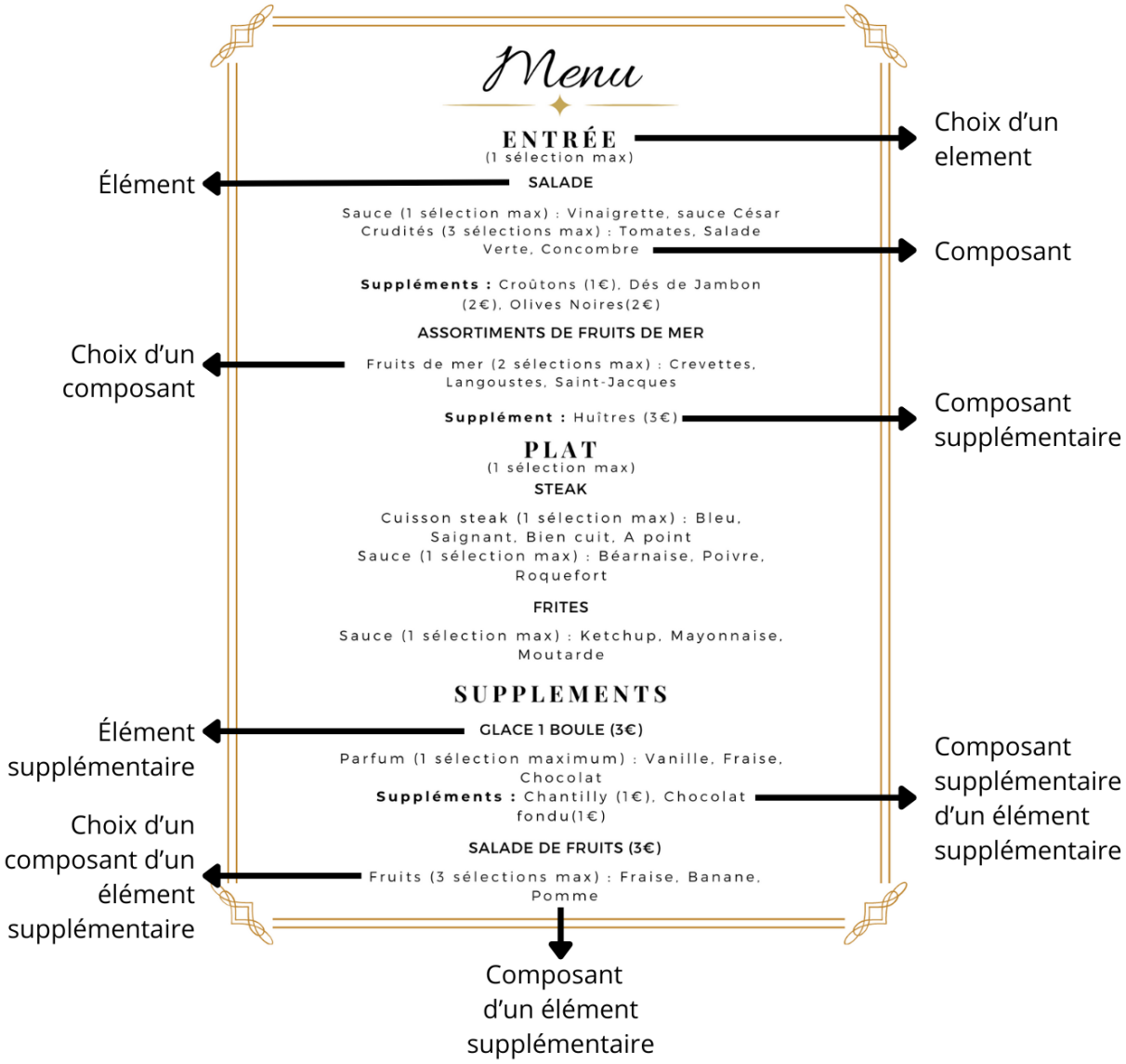
Nous avons décidé de nous attribuer la même note, car le travail que nous nous sommes assignés individuellement a été réalisé de manière efficace et autonome, sans entraver l'avancée du projet. Cette collaboration a démontré notre capacité à contribuer de manière égale au succès de l'ensemble du projet, d'où notre choix de partager une évaluation commune.

CONCLUSION

Nous sommes particulièrement satisfaits du travail accompli et des fonctionnalités implémentées, qui répondent aux besoins spécifiques définis dans nos User Stories. Ce qui a marqué notre expérience a été l'adoption d'une nouvelle approche dans la conception logicielle. En accordant une importance particulière à la rédaction préalable des tests avant l'implémentation des fonctionnalités, nous avons découvert une méthodologie qui a non seulement amélioré la qualité de notre code, mais a également facilité la détection de problèmes. Cette pratique nous a permis d'accroître notre efficacité et de minimiser les erreurs tout au long du processus de développement.

Au-delà des compétences techniques acquises, notre collaboration au sein de l'équipe a été un élément clé de notre succès. La communication, l'entraide et la compréhension mutuelle des rôles ont contribué à une dynamique de travail positive.

Annexes



Annexe 1 : *Explication de la diversité des menus*

Annexe 2 : *Dernière évaluation par SonarQube*

Milestone 17 : US18 : Modification d'un menu par le Manager de Restaurant

Closed 2 weeks ago **100%** complete

User story 18.1
En tant que Manager de Menu
Je veux modifier un élément d'un menu
afin de mettre à jour mes propositions

User story 18.2
En tant que Manager de Menu
Je veux modifier un composant d'un menu
afin de mettre à jour mes propositions

User story 18.3
En tant que Manager de Menu
Je veux modifier un supplément composant d'un menu
afin de mettre à jour mes propositions

User story 18.4
En tant que Manager de Menu
Je veux modifier un supplément élément d'un menu
afin de mettre à jour mes propositions

User story 18.5
En tant que Manager de Menu
Je veux modifier un choix composant d'un menu
afin de mettre à jour mes propositions

User story 18.6
En tant que Manager de Menu
Je veux modifier un choix élément d'un menu
afin de mettre à jour mes propositions

User story 18.7
En tant que Manager de Menu
Je veux modifier le nombre de choix d'un choix élément d'un menu
afin de mettre à jour mes propositions

User story 18.8
En tant que Manager de Menu
Je veux modifier le nombre de choix d'un choix composant d'un menu
afin de mettre à jour mes propositions

Annexe 3 : Détails d'une milestone