DOCUMENT DE CONCEPTION DU PROJET

Rédigé par :

Groupe Projet 10

Supervisé par :

M. Domingo PALAO

M. Bruce JOUGUEM YOUMBI

ANNEE ACADEMIQUE 2024 - 2025

TABLE DES MATIERES

Membres du groupe

Les membres du groupe projet 10 sont :

- NGANDO David Lance Legrand
- NJENGWES Johana Aricie
- NKOULOU Joseph Emmanuel
- NZALI Schilt Erwan Joachim

Introduction

Ce projet vise à concevoir une application informatique pour une chaîne internationale de restaurants afin de résoudre des problématiques opérationnelles critiques telles que l'organisation inefficace des salles et de la cuisine, le mauvais accueil des clients, les longs délais de service, et une gestion inadéquate des ressources. Ces lacunes ont conduit à une baisse de satisfaction client et, par conséquent, à une détérioration de la réputation des restaurants.

L'objectif est de développer une solution complète et modulable permettant :

1. Une gestion optimale des réservations et des affectations de tables.
2. Une coordination fluide entre la salle de restauration et la cuisine.
3. Une simulation et une supervision en temps réel des opérations du restaurant pour maximiser l'efficacité.

Cette application repose sur une architecture modulaire et doit intégrer des fonctionnalités avancées telles que la gestion des stocks en temps réel, des notifications d'alerte sur les ressources insuffisantes, une interface utilisateur graphique intuitive, et des mécanismes robustes de synchronisation. Le développement sera réalisé en **C++** avec l'utilisation de **QT** pour l'interface graphique, des bases de données comme **MySQL** ou **MariaDB**, et des mécanismes de communication interprocessus (IPC) comme les threads et les sockets.

Les différents rôles dans le restaurant

➢ La salle de restauration

- **Maître d'Hôtel**

- **Responsabilité principale** : Gérer l'accueil des clients et leur assignation à des tables adaptées, tout en maximisant l'utilisation des capacités du restaurant.
- **Tâches spécifiques** :
    - Évaluer la disponibilité des tables en temps réel.
    - Minimiser les placements inefficaces (par ex., éviter de placer 2 personnes à une table de 10, sauf si nécessaire).
    - Gérer les paiements à la fin du repas.
    - Objectif : Maximiser le chiffre d'affaires en optimisant le remplissage des tables.

Chefs de Rang

- **Responsabilité principale** : Superviser un carré de tables et transmettre les commandes à la cuisine.
- **Tâches spécifiques** :
    - Prendre les commandes après avoir présenté le menu.
    - Assurer la coordination avec les serveurs pour un service fluide.
    - Rester disponible pour aider les autres chefs de rang si besoin.

Serveurs

- **Responsabilité principale** : Assurer la livraison des plats aux tables et maintenir la satisfaction des clients.
- **Tâches spécifiques** :
    - Apporter pain et boissons immédiatement après la commande.
    - Synchroniser le service des plats pour qu'ils soient servis simultanément à une table.
    - Débarrasser les tables progressivement (par exemple, après les entrées avant de servir le plat principal).

Commis de Salle

- **Responsabilité principale** : Compléter le service en fournissant des ressources aux clients.

- **Tâches spécifiques** :
    - Fournir du pain et des boissons supplémentaires à la demande.
    - Remplacer temporairement un serveur si celui-ci est surchargé.

➢ La cuisine

Chef de Cuisine

- **Responsabilité principale** : Organiser et superviser toutes les activités en cuisine pour minimiser les délais de préparation.
- **Tâches spécifiques** :
    - Prioriser les commandes pour réduire les temps d'attente.
    - Dispatcher les tâches parmi les chefs de partie.
    - Gérer les stocks en temps réel, décider des recettes disponibles.

Chefs de Partie

- **Responsabilité principale** : Préparer les plats selon leurs spécialités.
- **Tâches spécifiques** :
    - Réaliser les préparations en suivant les instructions du chef de cuisine.
    - Optimiser les tâches en parallèle (par ex., cuisiner plusieurs portions simultanément si possible).

Commis de Cuisine

- **Responsabilité principale** : Soutenir les chefs de partie en effectuant des tâches de base.
- **Tâches spécifiques** :
    - Éplucher et couper les légumes.
    - Transporter les plats prêts vers le comptoir de service.

Plongeur

- **Responsabilité principale** : Maintenir la propreté des ustensiles et de la vaisselle.
- **Tâches spécifiques** :
    - Charger les machines à laver régulièrement.
    - Stocker la vaisselle propre pour réutilisation.

Explication des diagrammes UML

➢ Les digrammes d'activité

Un **diagramme d'activité** est un type de diagramme UML (Unified Modeling Language) qui représente le déroulement d'un processus ou d'une activité sous la forme d'un flux de travail. Il décrit les étapes successives d'une activité, les choix possibles, les parallélismes, et les interactions avec d'autres entités. Chaque étape est représentée par des **nœuds** (actions) et les connexions entre ces nœuds par des **flèches** indiquant le flux de contrôle ou de données.

Ces diagrammes sont particulièrement utiles pour visualiser et analyser les processus métier complexes, tels que ceux d'un restaurant, en identifiant les séquences, dépendances, et responsabilités.

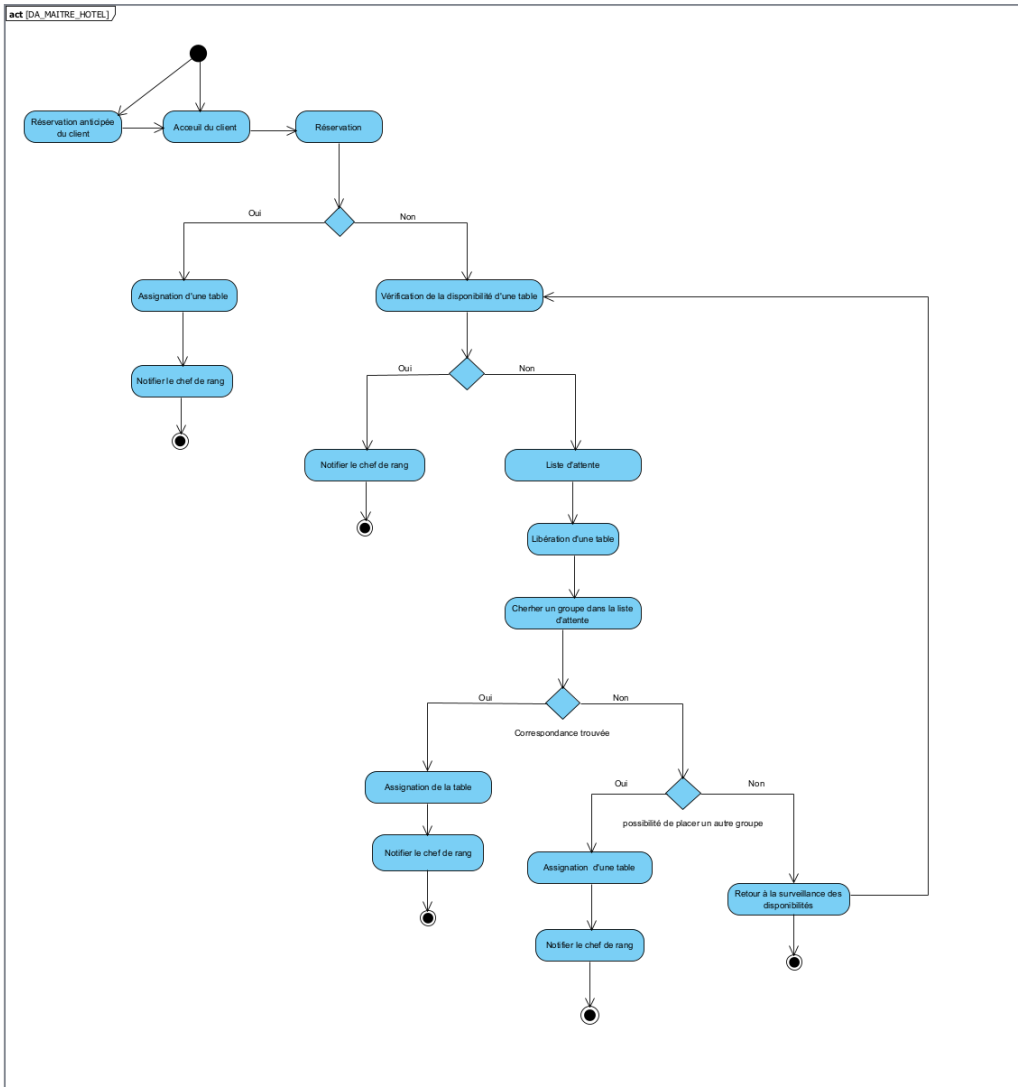
Diagrammes d'Activité pour chacun des postes du restaurant :

1. Maître d'Hôtel

Le maître d'hôtel gère l'accueil des clients et leur placement dans la salle. Voici les étapes clés :

1. **Début** : Un client arrive.
2. **Vérification des Réservations** : Le maître d'hôtel vérifie si une réservation existe.
    - Si oui, il attribue une table réservée.
    - Sinon, il cherche une table disponible.
3. **Assignation de Table** : Il assigne une table adaptée au nombre de clients.

4. **Mise en Attente (si nécessaire)** : Si aucune table n'est disponible, les clients sont placés en liste d'attente.
5. **Validation et Transmission** : Une fois la table assignée, il informe le chef de rang.
6. **Fin**.

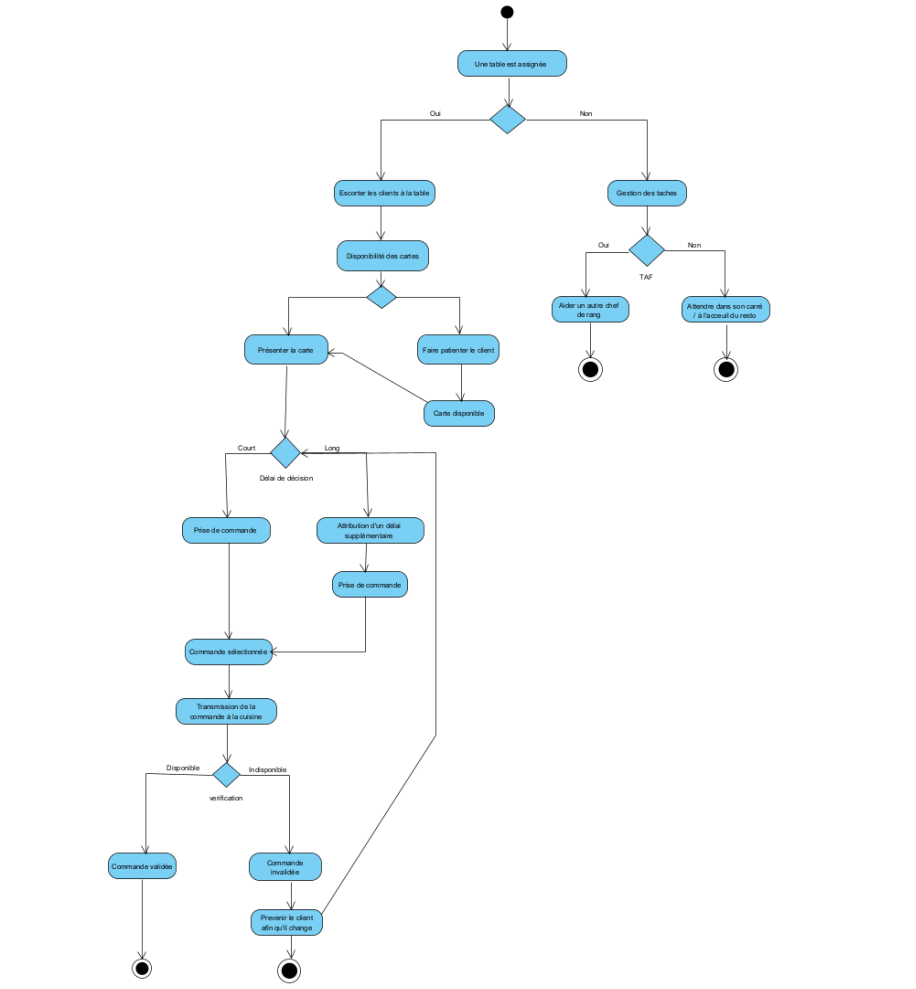


2. Chef de Rang

Le chef de rang supervise un carré de tables et prend les commandes des clients :

1. **Début** : Notification du maître d'hôtel pour accueillir les clients.

2. **Placement des Clients** : Accompagne les clients à leur table.
3. **Présentation de la Carte** : Présente les menus et répond aux questions.
4. **Prise de Commande** : Note la commande et vérifie qu'elle est complète.
5. **Transmission à la Cuisine** : Envoie la commande au chef de cuisine.
6. **Suivi** : S'assure que le service progresse correctement.
7. **Fin**.

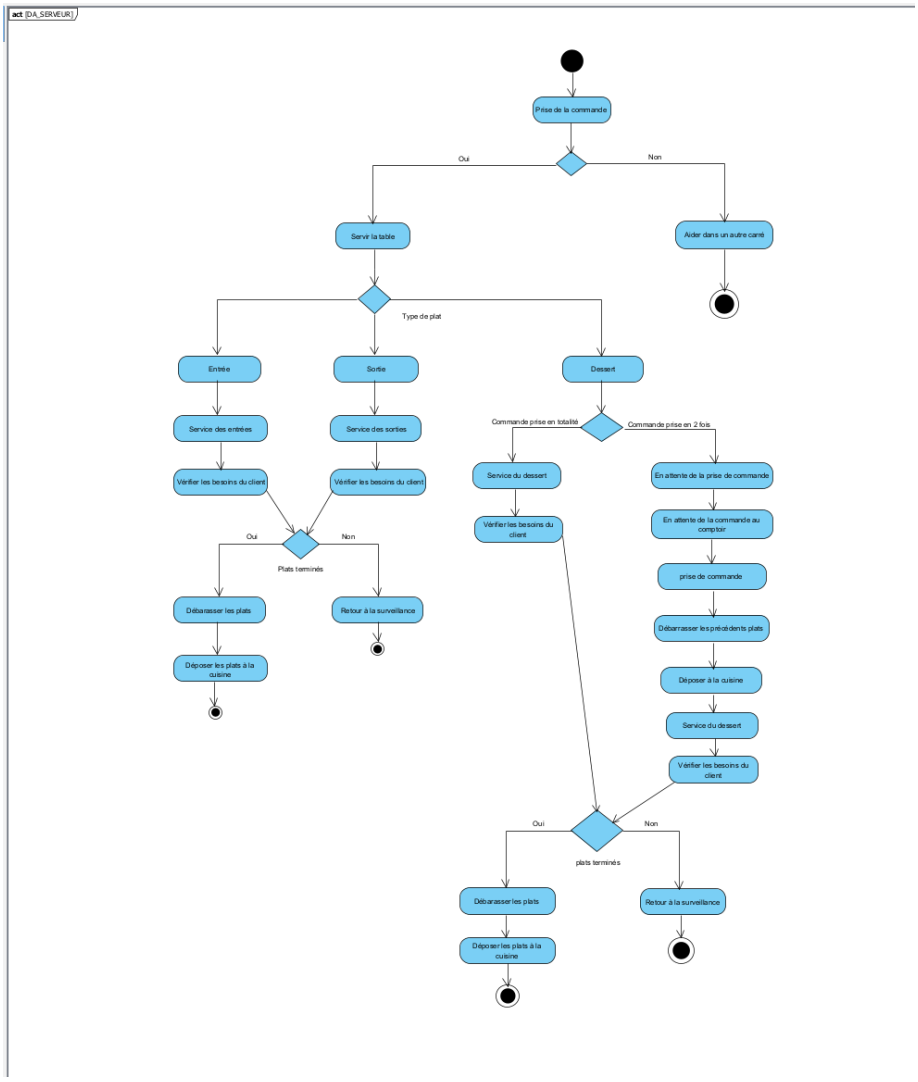


3. Serveur

Le serveur est chargé de servir les plats et d'assurer la satisfaction des clients :

1. **Début** : Reçoit une notification de préparation terminée de la cuisine.
2. **Vérification des Commandes** : S’assure que toutes les assiettes pour une table sont prêtes.
3. **Service des Plats** : Transporte les plats aux tables correspondantes.
4. **Débarrassage des Tables** : Retire les assiettes terminées et les apporte au comptoir de lavage.

5. **Gestion des Demandes Spéciales** : Fournit du pain, de l'eau ou d'autres demandes des clients.
6. **Fin**.

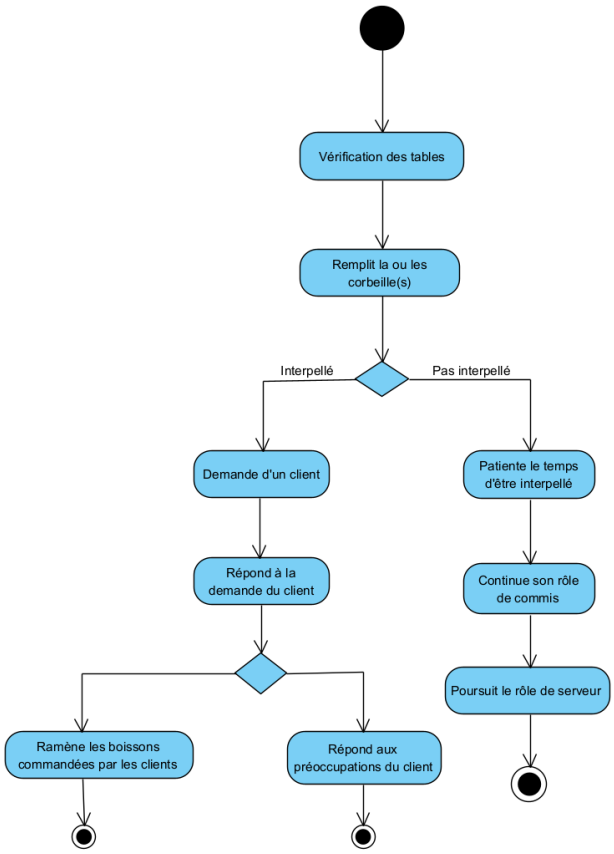


4. Commis de Salle

Le commis de salle soutient le serveur pour les besoins supplémentaires des clients :

1. **Début** : Vérifie les tables et s'assure qu'il ne manque rien.
2. **Fourniture de Pain et d'Eau** : Apporte ces éléments selon les besoins ou demandes.
3. **Remplacement du Serveur** : Intervient si un serveur est occupé.

4. **Nettoyage de la Salle** : S’assure que les zones communes sont propres et ordonnées.
5. **Fin**.

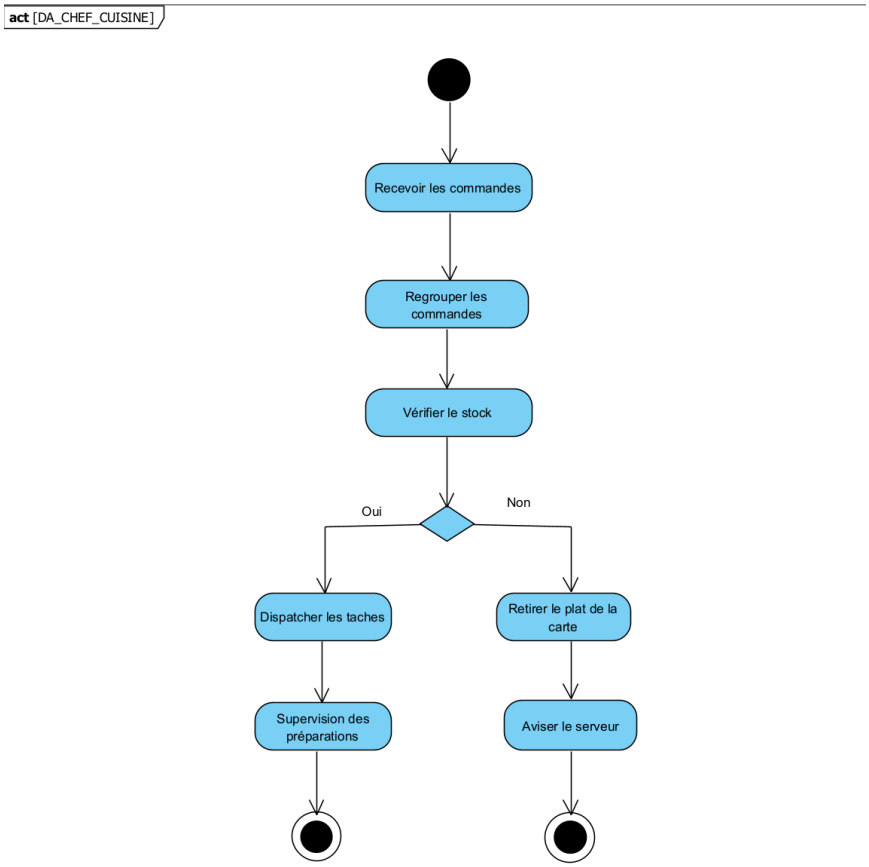


5. Chef de Cuisine

Le chef de cuisine coordonne les activités en cuisine :

1. **Début** : Reçoit une commande.
2. **Organisation des Tâches** : Divise les tâches entre les chefs de partie.
3. **Supervision** : Suit l’avancée des préparations et ajuste les priorités si nécessaire.
4. **Validation** : S’assure que les plats sont prêts pour le service.
5. **Gestion des Stocks** : Supprime des plats de la carte si des ingrédients manquent.

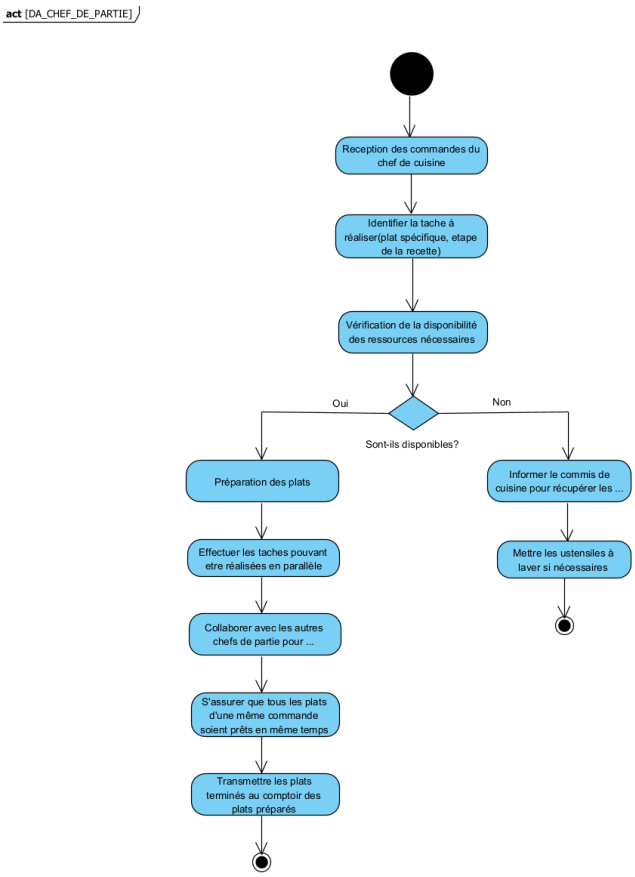
6. **Fin**.



6. Chefs de Partie

Les chefs de partie réalisent les préparations des plats :

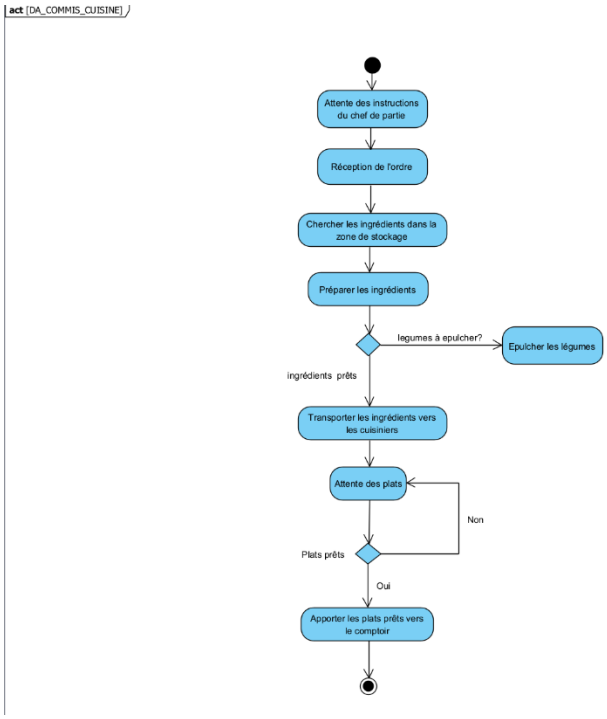
1. **Début** : Reçoivent des instructions du chef de cuisine.
2. **Préparation des Ingrédients** : Mesurent et préparent les ingrédients nécessaires.
3. **Cuisson/Assemblage** : Effectuent les étapes spécifiques aux recettes.
4. **Validation** : Confirment que le plat est prêt pour le comptoir.
5. **Nettoyage** : Nettoient leurs outils avant de commencer une nouvelle tâche.
6. **Fin**.



7. Commis de Cuisine

Les commis de cuisine assistent les chefs de partie :

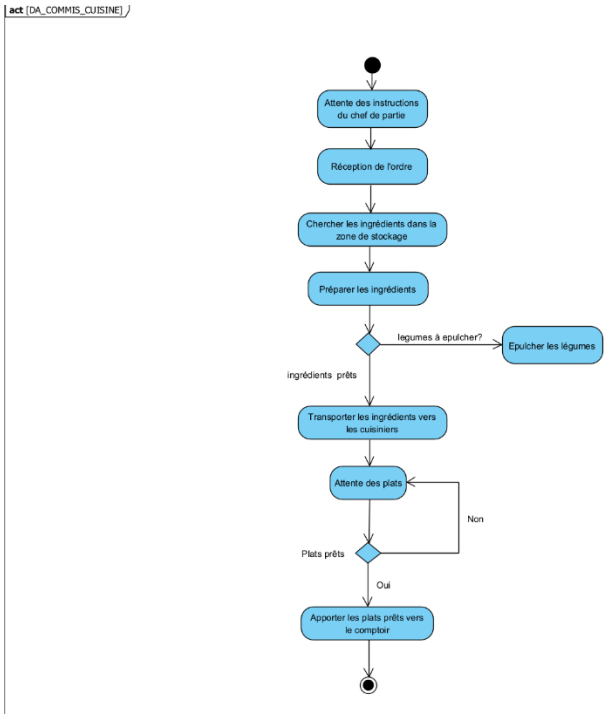
1. **Début** : Reçoivent des tâches du chef de partie.
2. **Préparation Initiale** : Épluchent et découpent les légumes, ou apportent les ingrédients depuis les stocks.
3. **Soutien** : Aident les chefs de partie à assembler les plats.
4. **Transport des Plats** : Apportent les plats finis au comptoir de service.
5. **Fin**.



8. Plongeur

Le plongeur gère le nettoyage des ustensiles et de la vaisselle :

1. **Début** : Reçoit des assiettes et ustensiles sales.
2. **Chargement de la Machine à Laver** : Place les objets sales dans le lave-vaisselle.
3. **Gestion des Ressources** : Transfère les objets propres vers les zones de stockage.
4. **Nettoyage Manuel** : Lave à la main les objets non pris en charge par la machine.
5. **Fin**.



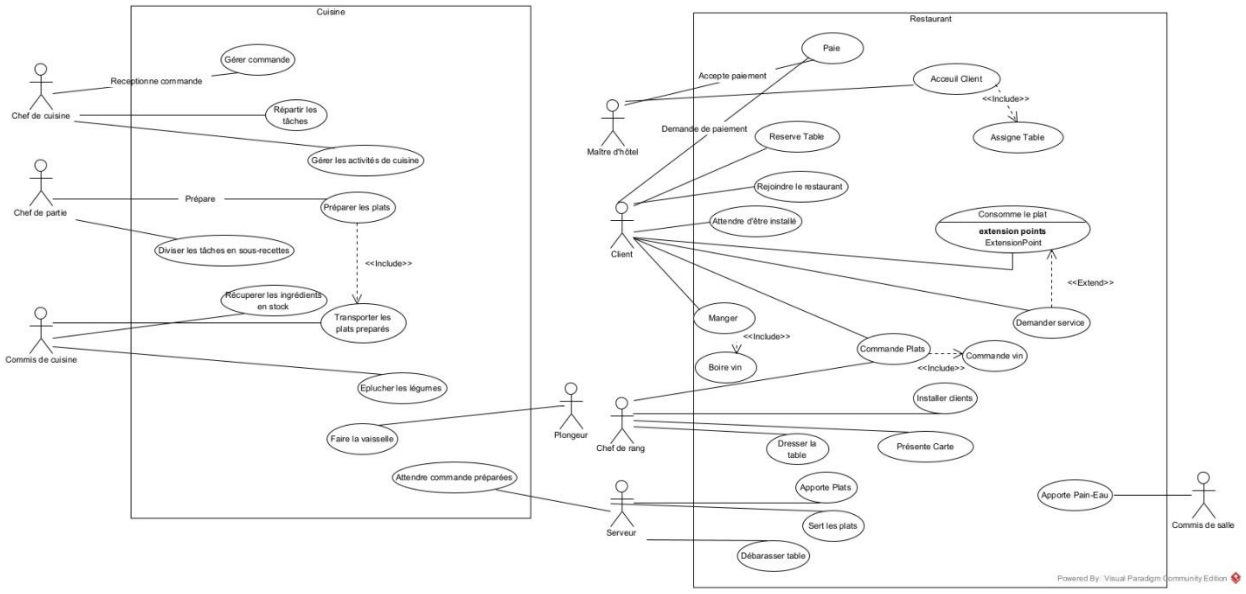
➢ Les digrammes de cas d'utilisation

Un **diagramme de cas d'utilisation** est une représentation graphique qui illustre les interactions entre les acteurs externes (comme les employés ou les clients) et le système étudié.

Il met en évidence les fonctionnalités principales du système sous forme de cas d'utilisation, chaque ellipse représentant une action ou un service offert par le système.

Dans ce projet, le diagramme de cas d'utilisation montre clairement les processus fondamentaux tels que l'accueil des clients, la prise et la gestion des commandes, la préparation des plats en cuisine, la gestion des stocks, et le paiement des clients.

Ce diagramme joue un rôle essentiel dans la définition des besoins du système et des responsabilités des différentes parties prenantes.



➢ Le diagramme de classe

Un diagramme de classe est une représentation graphique des classes, de leurs attributs, de leurs méthodes et des relations entre elles dans un système. Il est utilisé en génie logiciel pour modéliser la structure statique d'une application et fournit une vue d'ensemble des composants principaux.

Classes Principales

1. **MainWindow**
    - **Rôle** : Contrôle principal de l'application. Gère les interactions utilisateur et les connexions entre les composants.
    - **Attributs** :
        - diningScene, kitchenScene : Scènes graphiques pour afficher la salle et la cuisine.
        - clientController, tableController, fridgeController : Contrôleurs pour gérer les clients, les tables et les réfrigérateurs.
        - menuController : Gère les menus disponibles dans l'application.
        - db : Base de données connectée.

        - timer : Gestion du temps de simulation.
    - **Méthodes** :
        - connectToDatabase() : Connexion à la base de données.
        - loadMenuData(), loadStockInfo(), loadInventoryData() : Chargement des données de la BD.
        - startSimulation(), pauseSimulation(), stopSimulation() : Gestion de la simulation.

2. **MenuController**
    - **Rôle** : Gère les menus en les récupérant depuis la base de données et en fournissant un menu aléatoire.
    - **Attributs** :
        - menus : Liste des menus (id et nom).
    - **Méthodes** :
        - loadMenusFromDatabase() : Charge les menus depuis la BD.
        - getRandomMenu() : Sélectionne un menu aléatoire.
        - getMenus() : Retourne tous les menus.
3. **Client**
    - **Rôle** : Représente un client dans l'application.
    - **Attributs** :
        - graphicsItem : Représentation graphique.
        - id : Identifiant unique du client.
        - orderPopup : Affichage du menu choisi.
    - **Méthodes** :
        - setPosition() : Positionne le client dans la scène.
        - showOrderPopup() : Affiche un popup pour la commande.
4. **ClientController**
    - **Rôle** : Gère les clients et leurs interactions.
    - **Attributs** :
        - clients : Liste des clients.
    - **Méthodes** :

- addClient() : Ajoute un client.
- findClientById() : Trouve un client via son identifiant.

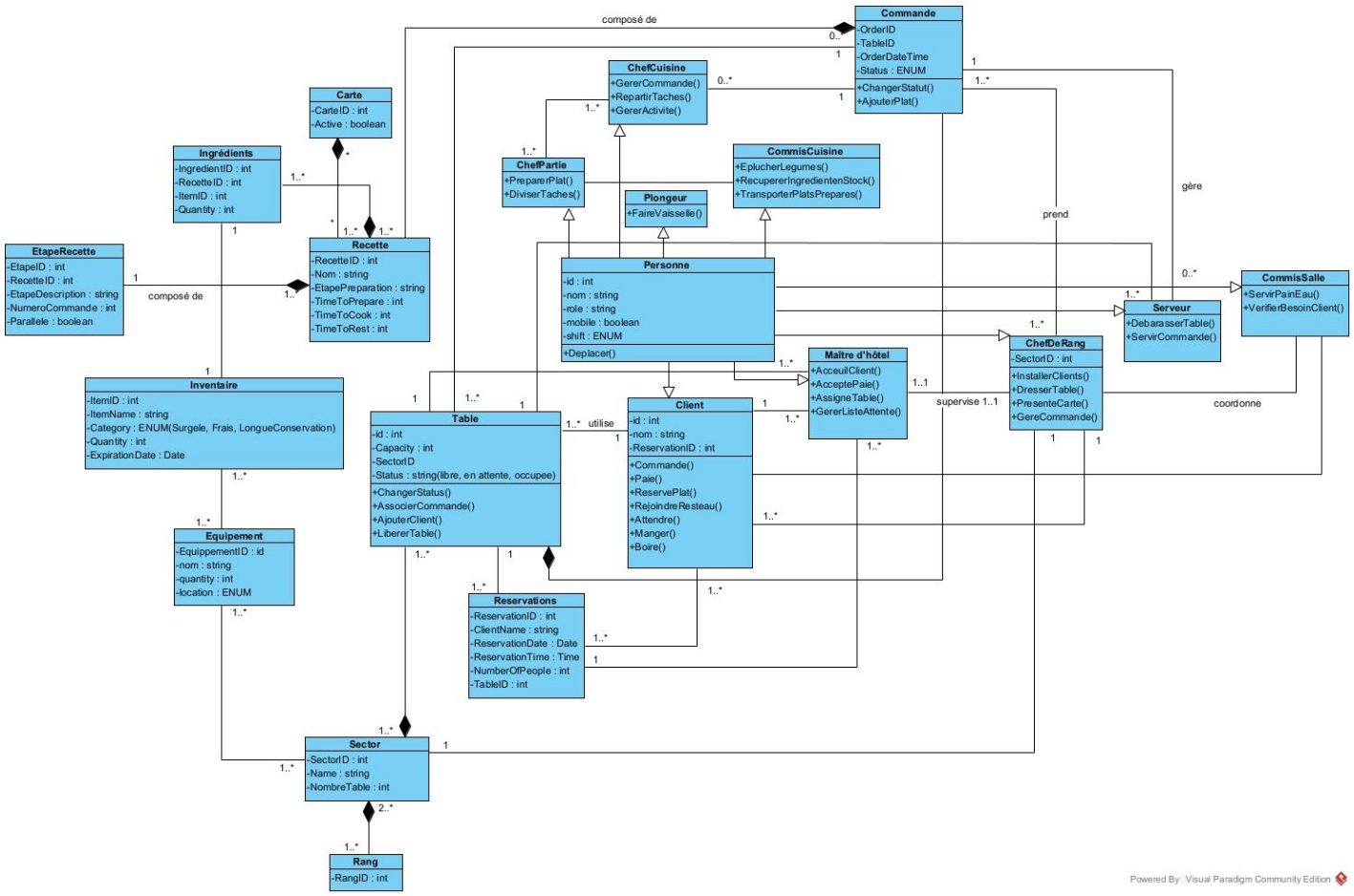
5. **Table**
   - **Rôle** : Représente une table dans la salle de restauration.
   - **Attributs** :
     - id : Identifiant unique.
     - capacity : Capacité maximale de la table.
     - occupied : État de la table (libre/occupée).
   - **Méthodes** :
     - setPosition() : Définit la position de la table.
     - setOccupied() : Change l'état de la table.
6. **TableController**
   - **Rôle** : Gère les tables de la salle.
   - **Attributs** :
     - tables : Liste des tables.
   - **Méthodes** :
     - addTable() : Ajoute une table.
     - findTableById() : Trouve une table par ID.
7. **Fridge**
   - **Rôle** : Représente un réfrigérateur dans l'application.
   - **Attributs** :
     - id : Identifiant unique.
     - capacity : Capacité en produits.
     - imagePath : Chemin vers l'image représentant le réfrigérateur.
   - **Méthodes** :
     - setPosition() : Positionne le réfrigérateur dans la scène.
8. **FridgeController**
   - **Rôle** : Gère les réfrigérateurs.
   - **Attributs** :
     - fridges : Liste des réfrigérateurs.

- **Méthodes** :
  - addFridge() : Ajoute un réfrigérateur.
  - findFridgeById() : Trouve un réfrigérateur via son identifiant.

Relations Entre les Classes

1. **MainWindow**
   - **Associe** avec MenuController, ClientController, TableController, FridgeController pour gérer les différents composants.
   - **Manipule** des instances de Client, Table, Fridge pour les afficher et interagir avec eux.
2. **MenuController**
   - **Dépend de la BD** pour charger les menus.
3. **ClientController**
   - **Manipule** des instances de Client.
4. **TableController**
   - **Manipule** des instances de Table.
5. **FridgeController**
   - **Manipule** des instances de Fridge.

Ci-joint notre diagramme de classes.



➢ Le diagramme de composants

Designs Patterns utilisés

➢ Factory Method

Emplacement identifié : IngredientFactory

Description :

La classe IngredientFactory implémente le pattern Factory Method, permettant de créer des objets Ingredient de manière centralisée. Cela offre plusieurs avantages :

Encapsulation de la logique de création : Si la manière de créer un Ingredient devait changer (par exemple, en fonction de certains critères ou en utilisant des sous-classes), cette logique serait contenue dans la factory.

Facilité de maintenance : Les développeurs peuvent utiliser createIngredient sans se soucier des détails de l'initialisation.

Pourquoi un design pattern ? Le pattern Factory Method permet de déléguer la responsabilité de l'instanciation d'objets à une classe spécifique, ce qui améliore la flexibilité et réduit le couplage entre les composants.

➢ MVC (Model-View-Controller)

Emplacement identifié : Architecture globale

Implémentation spécifique :

Modèle : La classe Ingredient représente le modèle des données (attributs, getters, setters).

Vue : Les classes comme mainWindow, controlDialog, stockWindow agissent en tant que vue, fournissant une interface utilisateur avec des champs pour saisir des informations.

Contrôleur : Les méthodes de stockWindow : onCommandButtonClicked) sert de contrôleur, manipulant le modèle et mettant à jour la vue en réponse aux événements.

Explications :

Le pattern MVC permet de séparer les préoccupations :

Le modèle contient les données brutes et la logique métier.

La vue affiche ces données et recueille les entrées utilisateur.

Le contrôleur agit comme un intermédiaire qui lie la vue et le modèle.

Cette séparation facilite la maintenance, les tests unitaires et l'évolution de l'application.

➢ Singleton

Emplacement identifié : Utilisation implicite de QSqlDatabase

Description :

Dans Qt, l'accès à la base de données se fait généralement via une instance unique gérée par QSqlDatabase::addDatabase et QSqlDatabase::database.

Cette approche suit le pattern Singleton, qui garantit qu'une seule instance d'une ressource partagée (la connexion à la base de données, ici) est utilisée dans l'application.

Avantages :

Évite les collisions ou les problèmes liés à l'ouverture multiple de connexions.

Simplifie la gestion des ressources partagées.

➢ Observer

Emplacement identifié : Système de signaux et slots dans Qt

connect(ui->commandButton, &QPushButton::clicked, this, &stockWindow::onCommandButtonClicked);

Description du pattern : Le pattern Observer consiste à maintenir une liste d'observateurs (ou écouteurs) qui sont notifiés automatiquement lorsqu'un état change.

Dans Qt, les signaux sont émis par un objet lorsqu'un événement se produit (par exemple, un clic sur un bouton). Les slots sont des méthodes qui réagissent à ces signaux. Qt connecte les deux dynamiquement.

Pourquoi les signaux et slots implémentent le pattern Observer :

Les objets émetteurs (comme QPushButton) ne connaissent pas les objets qui réagissent aux signaux. Cela garantit un faible couplage.

Plusieurs slots peuvent être connectés à un même signal, ce qui permet une programmation flexible.

Cas concret dans Qt :

Signal : Le bouton "Commander" émet un signal lorsqu'il est cliqué.

Observateur : La méthode onCommandButtonClicked agit comme un observateur, répondant à cet événement en exécutant la logique métier.

➢ DAO (Data Access Object)

Emplacement identifié : Gestion de la base de données (CommandeManager, Ingredient)

Description : Les classes et méthodes qui encapsulent les interactions avec la base de données (comme getAllIngredients, insertIngredientIntoDatabase) illustrent le pattern DAO. Ce pattern centralise les opérations de persistance et d'accès aux données.

Avantages :

Encapsulation des détails techniques : La logique SQL est isolée des autres parties de l'application.

Réutilisabilité : Les méthodes DAO peuvent être réutilisées dans plusieurs contextes.

Testabilité : Les DAO peuvent être testés indépendamment en simulant les bases de données.

1.6. Composite (Implication potentielle)

Emplacement identifié : Gestion des commandes (CommandeManager, OrderItem)

Description : Bien que votre structure actuelle des commandes reste simple, le fait de représenter une commande avec des détails imbriqués (liste de plats et quantités) évoque une structure composite. Si cette structure évolue pour permettre des sous-commandes ou des combinaisons complexes, le pattern Composite pourrait être utilisé explicitement.

2. Signaux et Slots comme Observer

Fonctionnement :

Signal : Émis lorsqu'un événement spécifique se produit.

Slot : Réagit au signal, exécutant une logique appropriée.

Connexion : L'association entre le signal et le slot est établie dynamiquement via QObject::connect.

Pourquoi c'est un Observer ?

Propagation des événements :

Les slots sont automatiquement notifiés lorsqu'un signal est émis. Cela reflète directement le comportement attendu dans le pattern Observer.

Dynamisme :

Les connexions peuvent être établies ou supprimées à l'exécution, permettant une grande flexibilité dans la gestion des événements.

Cas concret :

Le clic sur un bouton ("Commander") émet un signal. Ce signal est observé par le slot onCommandButtonClicked, qui exécute la logique pour insérer une commande.

Conclusion

- Factory Method pour la création d'objets (IngredientFactory).
- MVC pour structurer l'architecture.
- Singleton pour la gestion des connexions à la base de données.
- Observer pour la gestion des événements via les signaux et slots.
- DAO pour encapsuler les opérations liées à la base de données.