

DOCUMENT DE CONCEPTION DU PROJET PROGRAMMATION SYSTEME

Rédigé par le :

Groupe Projet 10

Supervisé par :

M. Domingo PALAO

M. Bruce JOUGUEM YOUNBI

ANNEE ACADEMIQUE 2024 - 2025

TABLE DES MATIERES

Membres du groupe	4
Introduction	5
Les différents rôles dans le restaurant	5
➤ La salle de restauration	5
➤ La cuisine	7
Explication des diagrammes UML	8
➤ Les digrammes d'activité	8
➤ Les digrammes de cas d'utilisation	17
➤ Le diagramme de classe	18
➤ Le diagramme de composants	22
➤ Le diagramme de séquences	23
Designs Patterns Utilisés	27
Base de données et MCD	31
Conclusion	35

TABLE DES FIGURES

Fig 1 : Diagramme d'activité du maitre d'hôtel.....	9
Fig 2 : Diagramme d'activité du chef de rang	10
Fig 3 : Diagramme d'activité du serveur	12
Fig 4 : Diagramme d'activité du commis de salle	13
Fig 5 : Diagramme d'activité du chef de cuisine	14
Fig 6 : Diagramme d'activité du chef de partie	15
Fig 7 : Diagramme d'activité du commis de cuisine	16
Fig 8 : Diagramme d'activité du plongeur	17
Fig 9 : Diagramme de cas d'utilisation	18
Fig 10 : Diagramme de composants.....	23
Fig 11 : Diagramme de séquence (Passer commande)	24
Fig 12 : Diagramme de séquence (Payer addition)	25
Fig 13 : Diagramme de séquence (Préparer commandes).....	26
Fig 14 : Diagramme de séquence (Réserver table)	27
Fig 15 : Modèle conceptuel de données (MCD)	35

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. **Assignment 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.**

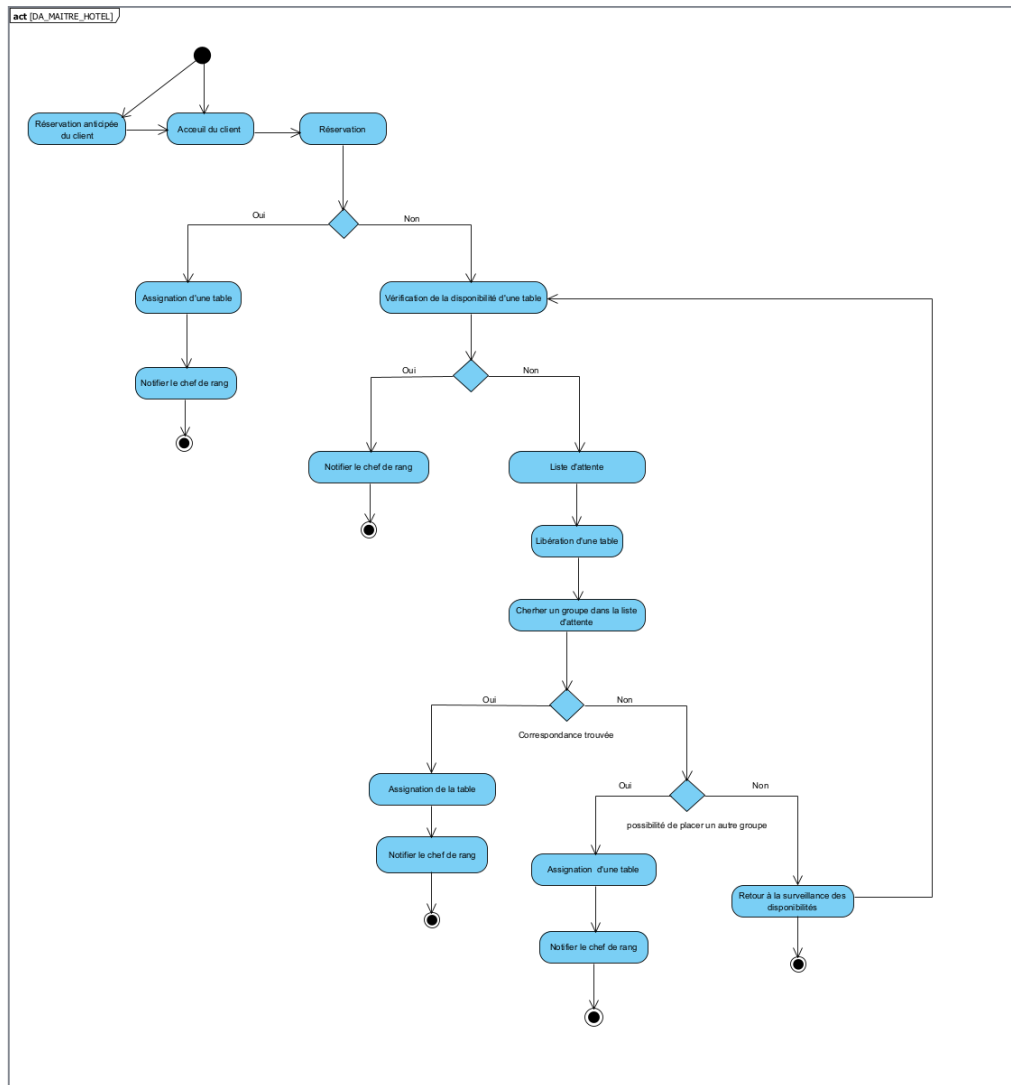


Fig 1 : Diagramme d'activité du maître d'hôtel

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.**

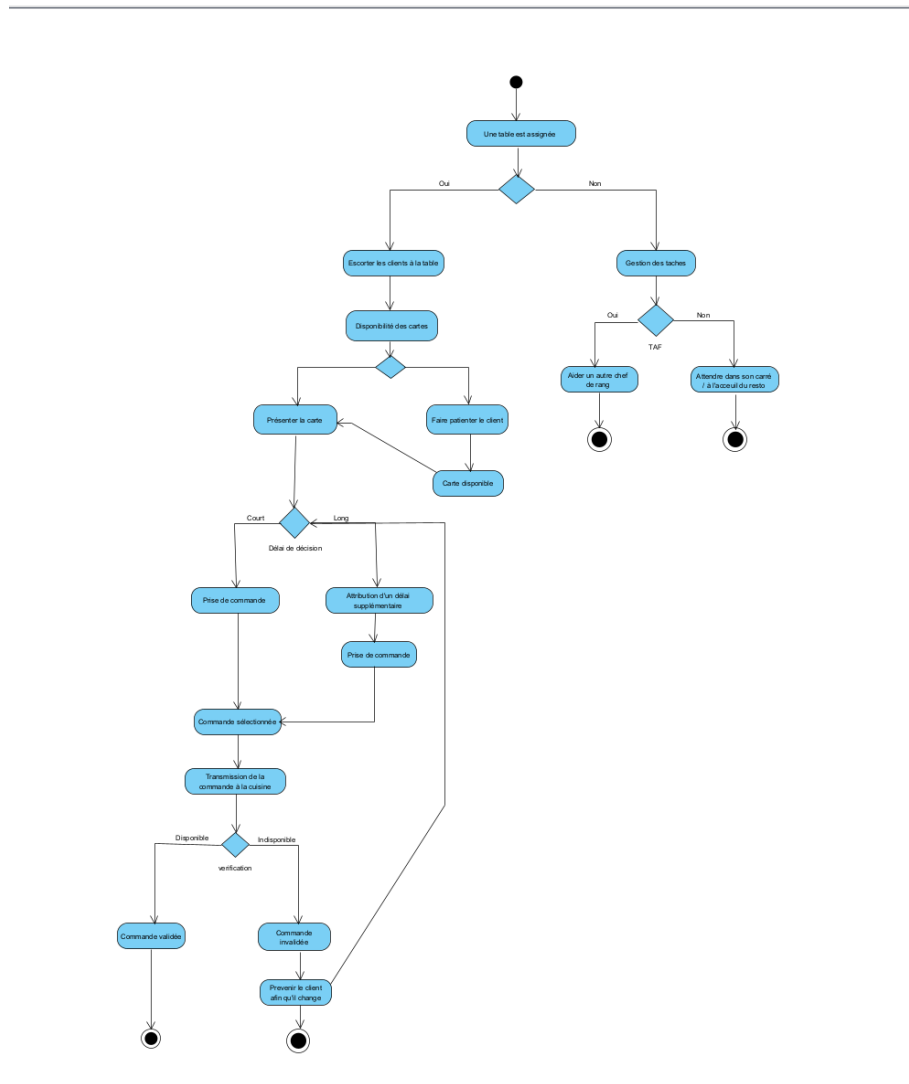


Fig 2 : Diagramme d'activité du chef de rang

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**.

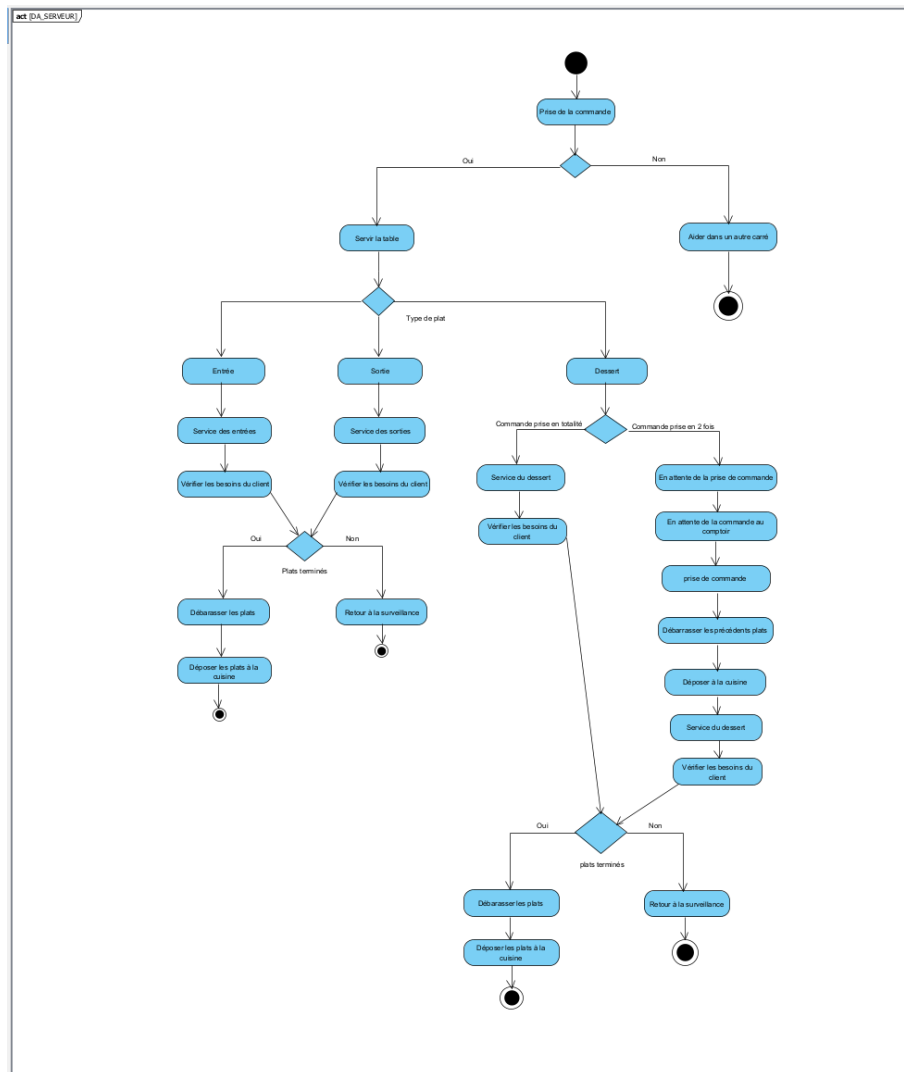


Fig 3 : Diagramme d'activité du serveur

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.**

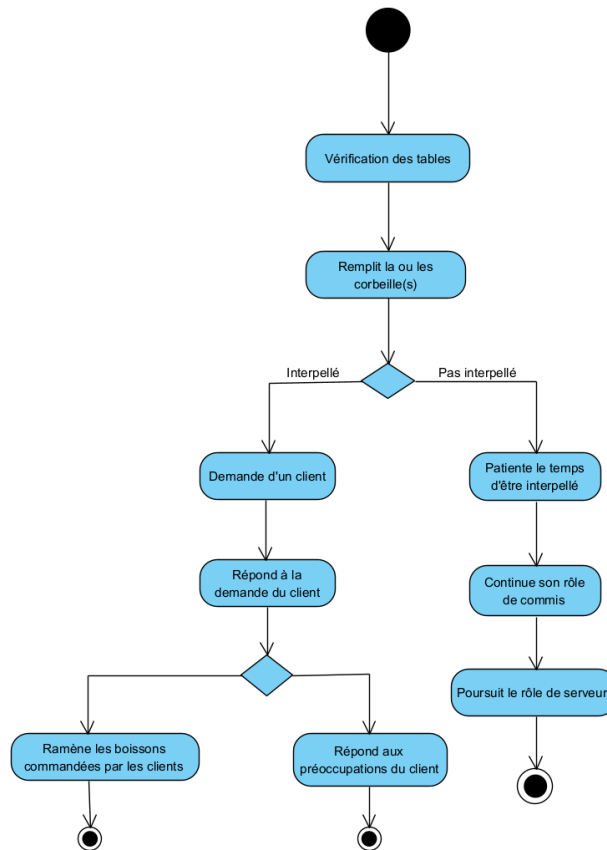


Fig 4 : Diagramme d'activité du commis de salle

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.**

act [DA_CHEF_CUISINE]

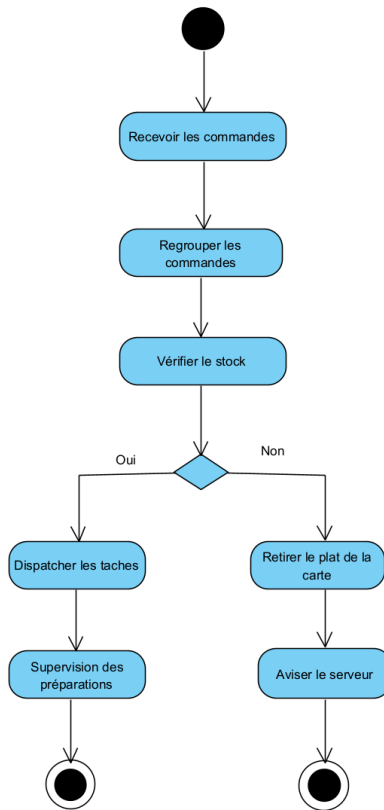


Fig 5 : Diagramme d'activité du chef de cuisine

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.**

act [DA_CHEF_DE_PARTIE]

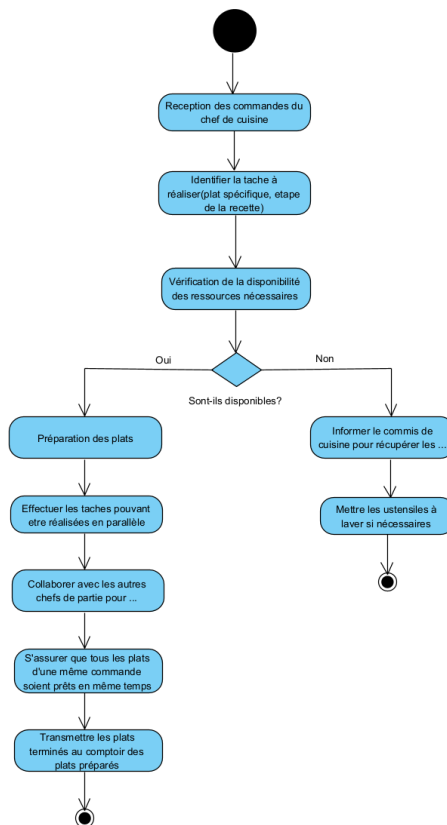


Fig 6 : Diagramme d'activité du chef de partie

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.**

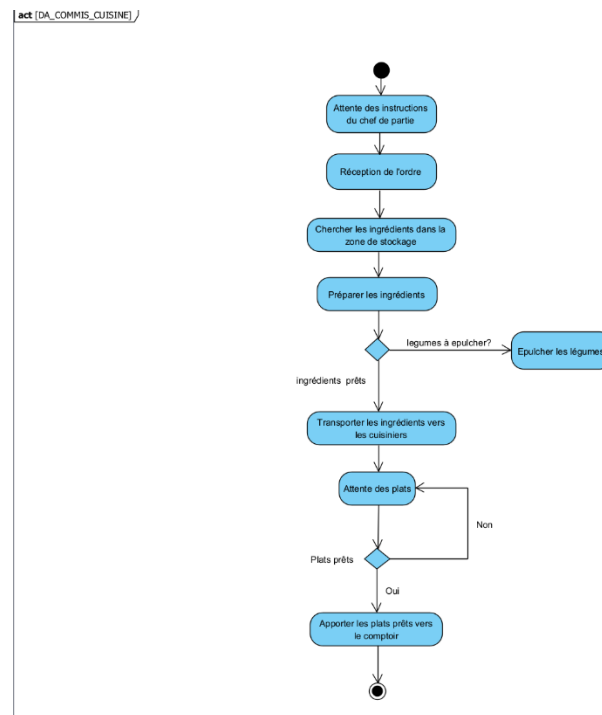


Fig 7 : Diagramme d'activité du commis de cuisine

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.

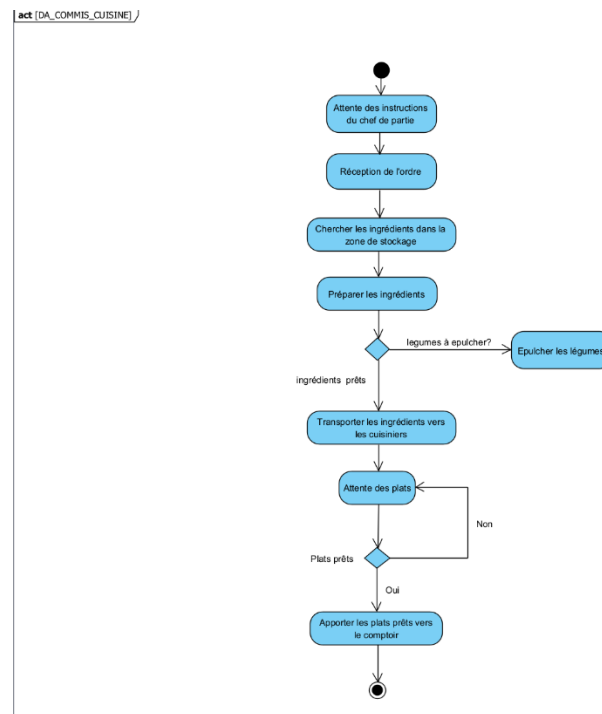


Fig 8 : Diagramme d'activité du plongeur

➤ 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.

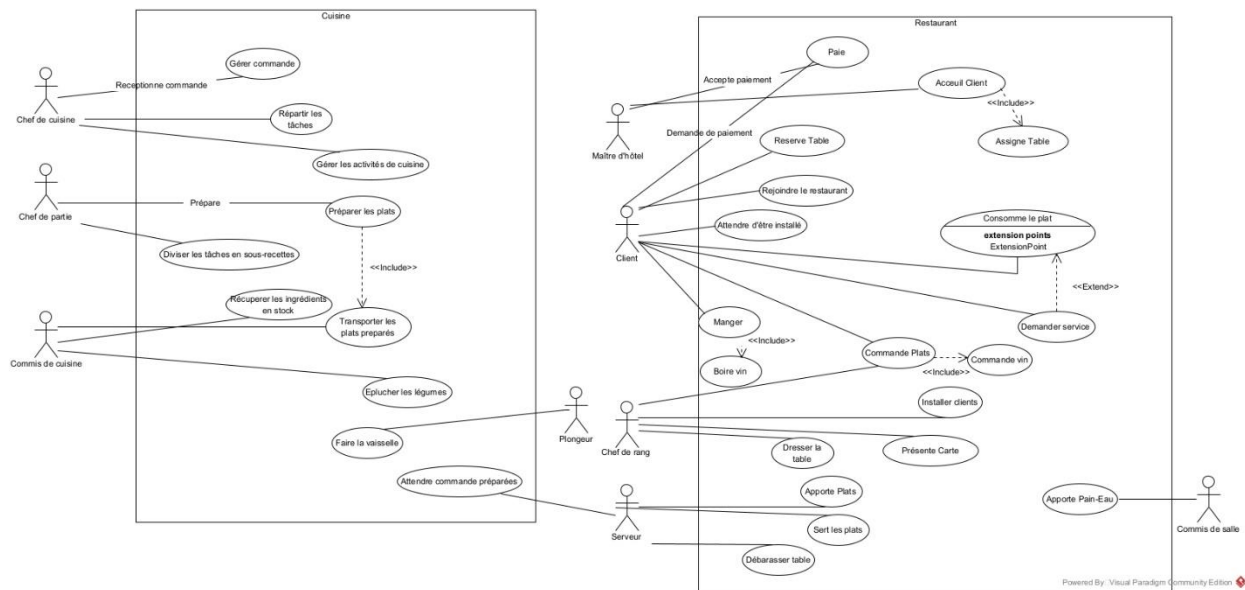


Fig 9 : Diagramme de cas d'utilisation

➤ 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`.

➤ Le diagramme de composants

Dans le cadre de notre projet, nous avons choisi d'adopter une architecture basée sur le modèle **MVC (Modèle-Vue-Contrôleur)**, une approche qui permet de séparer clairement la logique métier, l'affichage et la gestion des interactions avec l'utilisateur. Le **diagramme de composants** illustre cette séparation et montre comment les différentes parties du système interagissent.

Vue

La **vue** est responsable de l'affichage des données et de l'interaction avec l'utilisateur. Elle permet de visualiser les informations pertinentes du système, telles que les menus disponibles, les états des tables, les réservations, et les commandes. Elle est également responsable de la gestion de l'interface utilisateur, incluant la présentation dynamique de l'état du système en temps réel.

Contrôleur

Le **contrôleur** agit comme un médiateur entre la vue et le modèle. Il capte les actions de l'utilisateur et effectue les traitements nécessaires, que ce soit la gestion des événements ou l'interaction avec la logique métier. En fonction des actions de l'utilisateur, il met à jour la vue ou le modèle. Le contrôleur assure ainsi une séparation claire des préoccupations et veille à ce que les différentes couches du système puissent interagir de manière fluide.

Modèle

Le **modèle** est la couche qui gère la logique métier du système. Il est responsable de la manipulation des données et de leur persistance dans la base de données. Toutes les informations nécessaires, telles que les menus, les clients, les tables, et les commandes, sont gérées par cette couche. Le modèle interagit directement avec la base de données pour stocker et récupérer les informations nécessaires au bon fonctionnement de l'appli

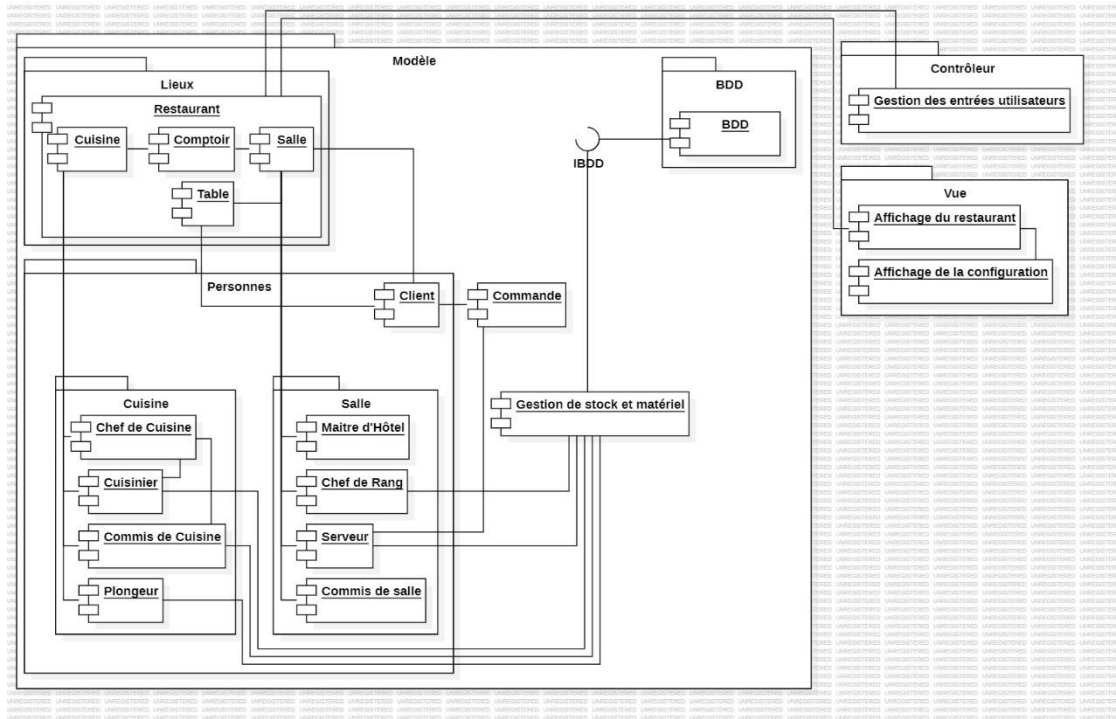


Fig 10 : Diagramme de composants

➤ Le diagramme de séquences

Un **diagramme de séquences** est un diagramme UML qui représente l'ordre chronologique des interactions entre différents objets ou acteurs d'un système. Il illustre comment les messages ou les actions sont échangés entre les entités au fil du temps pour accomplir un scénario particulier.

Chaque objet ou acteur est représenté par une **ligne de vie verticale**, et les messages échangés sont représentés par des **flèches horizontales**. Ce type de diagramme est particulièrement utile pour visualiser le flux des opérations et identifier les dépendances entre les différentes parties d'un système, ainsi que les délais de traitement.

Nous allons à présent présenter quelques-uns des diagrammes de séquences réalisés :

1. Passer Commande

Le diagramme de séquence pour passer une commande montre le processus par lequel un client passe sa commande, depuis la réception du client jusqu'à la transmission des informations à la cuisine :

- **Client** : Sélectionne les plats et transmet la commande.
- **Chef de Rang** : Prend la commande et l'enregistre.
- **Système** : Vérifie la commande et la transmet à la cuisine.
- **Cuisine** : Prépare les plats, puis la commande est prête à être servie. Ce diagramme met en évidence la séquence d'actions entre le client, le personnel de la salle et la cuisine pour assurer un service fluide.

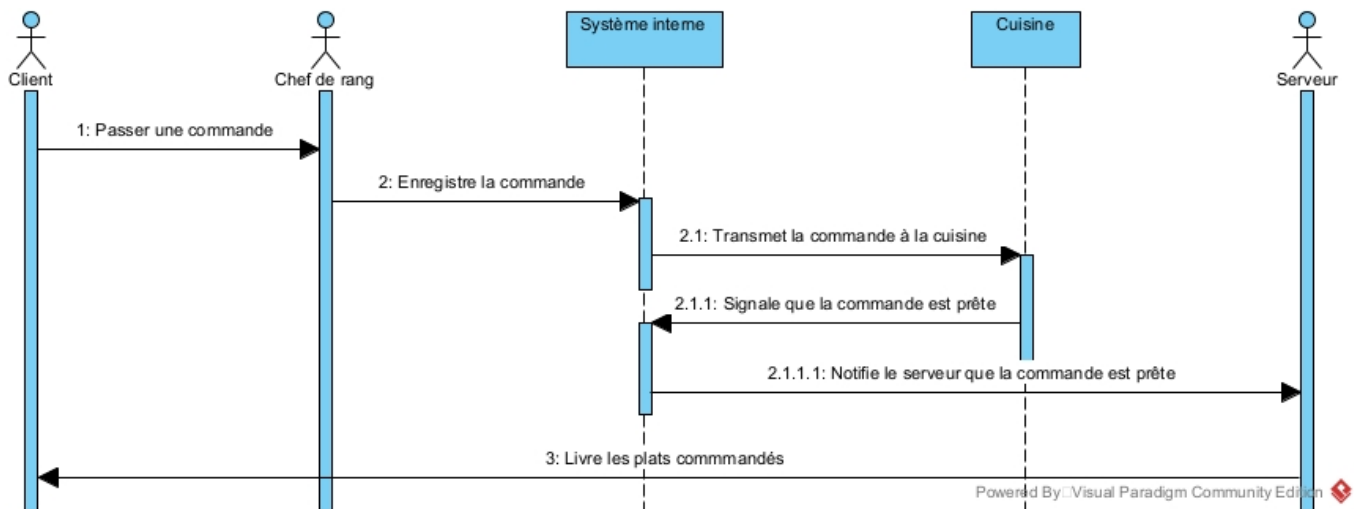


Fig 11 : Diagramme de séquence (Passer commande)

2. Payer l'Addition

Le diagramme de séquence pour le paiement montre comment l'addition est gérée lorsque le client est prêt à quitter :

- **Client** : Demande l'addition.
- **Serveur** : Prépare l'addition en fonction de la commande.
- **Système** : Affiche le montant total.
- **Client** : Effectue le paiement.
- **Maître d'Hôtel** : Confirme la transaction et libère la table pour le prochain client. Ce diagramme illustre l'échange de messages nécessaires pour finaliser une transaction, garantissant que le paiement soit pris en compte de manière transparente.

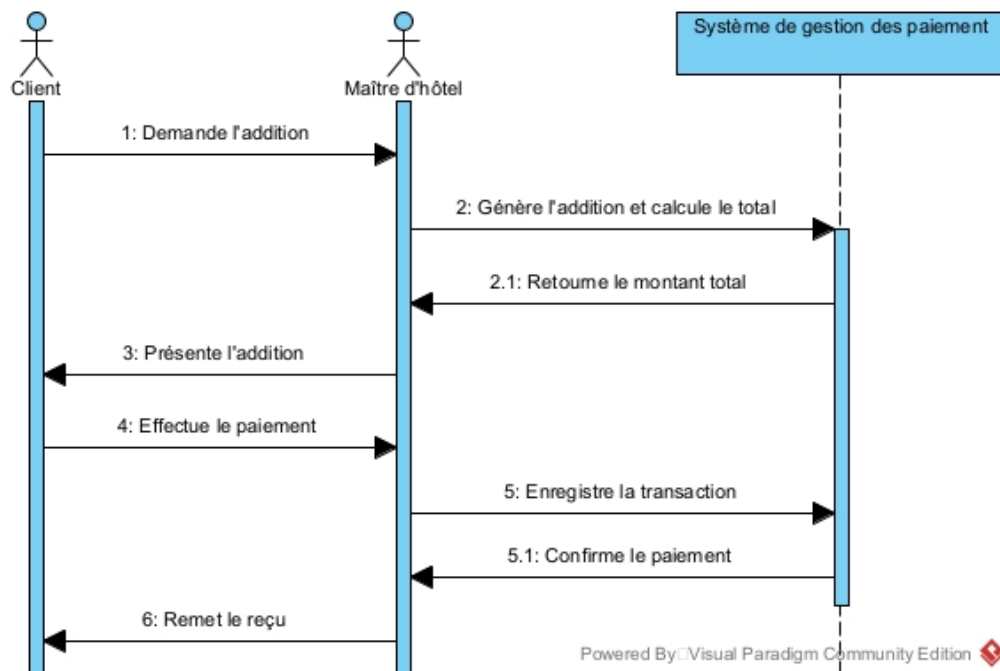


Fig 12 : Diagramme de séquence (Payer addition)

3. Préparer Commandes

Ce diagramme de séquence montre les interactions en cuisine pour préparer les plats commandés :

- **Chef de Cuisine** : Reçoit la commande et répartit les tâches entre les chefs de partie.

- **Chef de Partie** : Prépare les plats selon les instructions.
- **Commis de Cuisine** : Aide à la préparation et transporte les plats vers le comptoir de service. Ce diagramme permet de visualiser les rôles de coordination en cuisine pour assurer une préparation efficace des commandes.

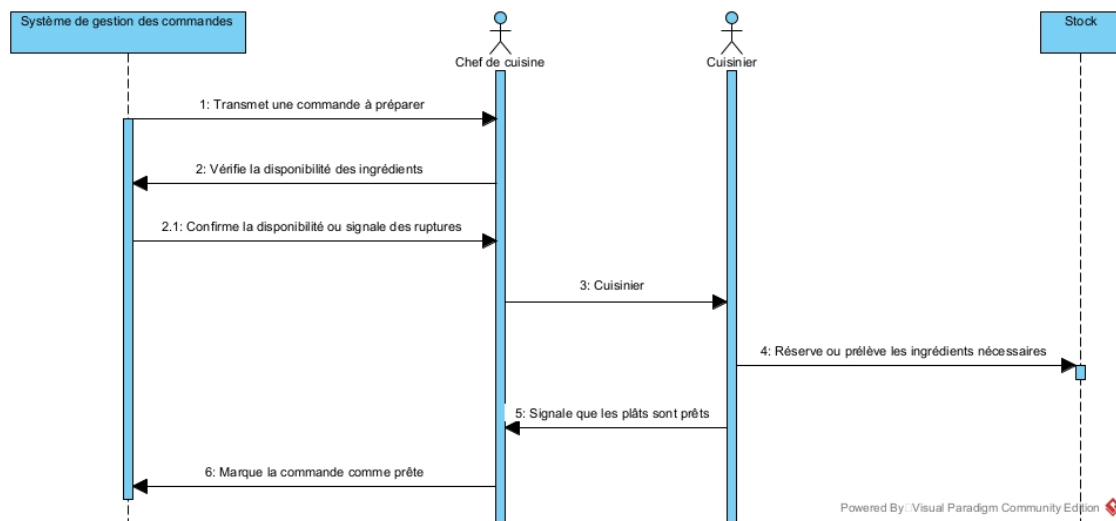


Fig 13 : Diagramme de séquence (Préparer commandes)

4. Réserver Table

Le diagramme de séquence pour la réservation de table montre le processus par lequel un client réserve une table à l'avance :

- **Client** : Contacte le restaurant pour réserver une table.
- **Maître d'Hôtel** : Vérifie la disponibilité des tables et enregistre la réservation.
- **Système** : Met à jour la base de données de réservation. Ce diagramme reflète l'interaction entre le client, le maître d'hôtel et le système de gestion des réservations, garantissant que la table soit réservée correctement et que le système soit mis à jour en temps réel.

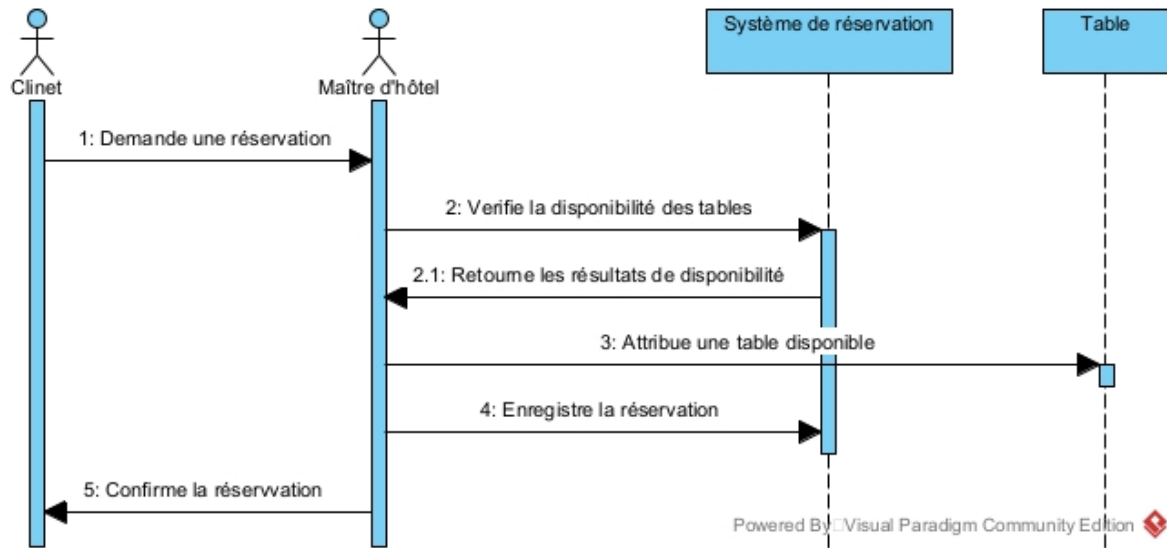


Fig 14 : Diagramme de séquence (Réserver table)

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.

➤ 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.

➤ Signaux et Slots comme Observer

✚ **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 est-ce 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.

En 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.

Base de données et MCD

Dans le cadre de ce projet, une base de données relationnelle a été conçue pour gérer les différentes facettes du restaurant, telles que la gestion des recettes, des commandes, des réservations, des stocks, des employés et de la salle.

Bien que MySQL ou MariaDB aient été spécifiés initialement, nous avons opté pour l'utilisation de **SQLite** en raison de sa simplicité d'intégration avec QT, tout en restant parfaitement adapté pour une application de cette envergure.

Architecture Générale

La base de données est organisée en plusieurs tables interconnectées, permettant de gérer de manière cohérente les données relatives à chaque aspect du fonctionnement du restaurant. Les relations entre les tables sont établies à l'aide de clés primaires et étrangères, et des contraintes telles que CHECK, FOREIGN KEY, et DEFAULT sont utilisées pour assurer la validité des données.

Rôle des Tables

a) Gestion des Recettes et des Catégories

- **CategoriesRecettes** : Cette table contient les différentes catégories de recettes, comme "Entrée", "Plat" et "Dessert". Elle permet de classer les recettes pour faciliter leur gestion dans le restaurant.
- **Recettes** : Cette table regroupe les informations relatives aux recettes du restaurant, telles que leur nom, le nombre de personnes qu'elles servent et les temps de préparation, cuisson et repos. Elle est liée à la table CategoriesRecettes, ce qui permet de classer chaque recette sous une catégorie spécifique.
- **IngredientsRecettes** : Elle répertorie les ingrédients nécessaires à la préparation de chaque recette. Chaque ingrédient est associé à une recette dans la table Recettes.
- **EtapasRecettes** : Cette table détaille les étapes de la préparation des recettes. Elle permet de suivre l'ordre et la description de chaque étape, facilitant ainsi la préparation des plats en cuisine.

b) Gestion de la Salle

- **Secteurs** : Cette table détermine les différentes zones ou secteurs de la salle du restaurant. Chaque secteur peut contenir plusieurs tables.

- **Tables** : Chaque entrée de cette table correspond à une table du restaurant, et contient des informations comme la capacité, l'état (libre, occupée, réservée) et le secteur auquel elle appartient. La gestion de l'état des tables est essentielle pour le bon déroulement des réservations et des commandes.
- **Clients** : Cette table gère les clients du restaurant, permettant de suivre leur réservation et leur identification dans le système.

c) Gestion des Réservations

- **Reservations** : Cette table enregistre les réservations effectuées par les clients. Elle contient des informations sur la date et l'heure de la réservation, le nombre de personnes et la table réservée. Cette table est liée à la table Tables pour associer une réservation à une table spécifique.

d) Gestion des Commandes

- **Commandes** : Enregistre chaque commande effectuée dans le restaurant. Elle est liée à la table Tables pour indiquer à quelle table la commande est associée. La table suit également l'état de la commande, qui peut être "EnAttente", "EnPréparation" ou "Servie".
- **DetailsCommandes** : Cette table contient les détails des plats commandés dans chaque commande. Elle relie chaque plat à une recette spécifique et inclut la quantité demandée.

e) Gestion des Stocks et des Équipements

- **Inventaire** : Regroupe les produits disponibles en cuisine, classés par catégorie (surgelés, frais, longue conservation). Cette table est essentielle pour le suivi des stocks de produits utilisés dans la préparation des recettes.

- **IngredientsInventaire** : Elle relie les produits de l'inventaire aux recettes qui les utilisent. Cette table permet de calculer les besoins en produits pour chaque préparation et commande.
- **Equipements** : Contient les équipements utilisés dans le restaurant, qu'ils soient dans la cuisine ou la salle. Elle permet de suivre la quantité et la localisation des équipements.

f) Gestion des Employés et des Tâches

- **Employes** : Cette table contient les informations sur les employés du restaurant, y compris leur rôle (par exemple, Maître d'Hôtel, Serveur, Chef, etc.) et leur secteur d'affectation dans le restaurant. Elle est liée à la table Secteurs pour associer chaque employé à un secteur spécifique de la salle.
- **LogsTaches** : Journalise les tâches effectuées par chaque employé. Elle permet de garder un historique des activités des employés, contribuant à une meilleure gestion du restaurant.

Le MCD (Modèle Conceptuel de Données)

Le **MCD** est un diagramme qui permet de représenter de manière abstraite les données d'un système, leurs relations et les règles d'intégrité qui les gouvernent. Il est utilisé pour décrire les entités, leurs attributs et les associations entre ces entités dans une base de données.

Le MCD est utile pour :

- **Modélisation des données** : Il offre une vue d'ensemble du système en identifiant les entités principales et leurs relations.
- **Conception de la base de données** : Le MCD sert de base pour la création de la structure de la base de données (tables, clés, etc.).

Ci-joint notre MCD :

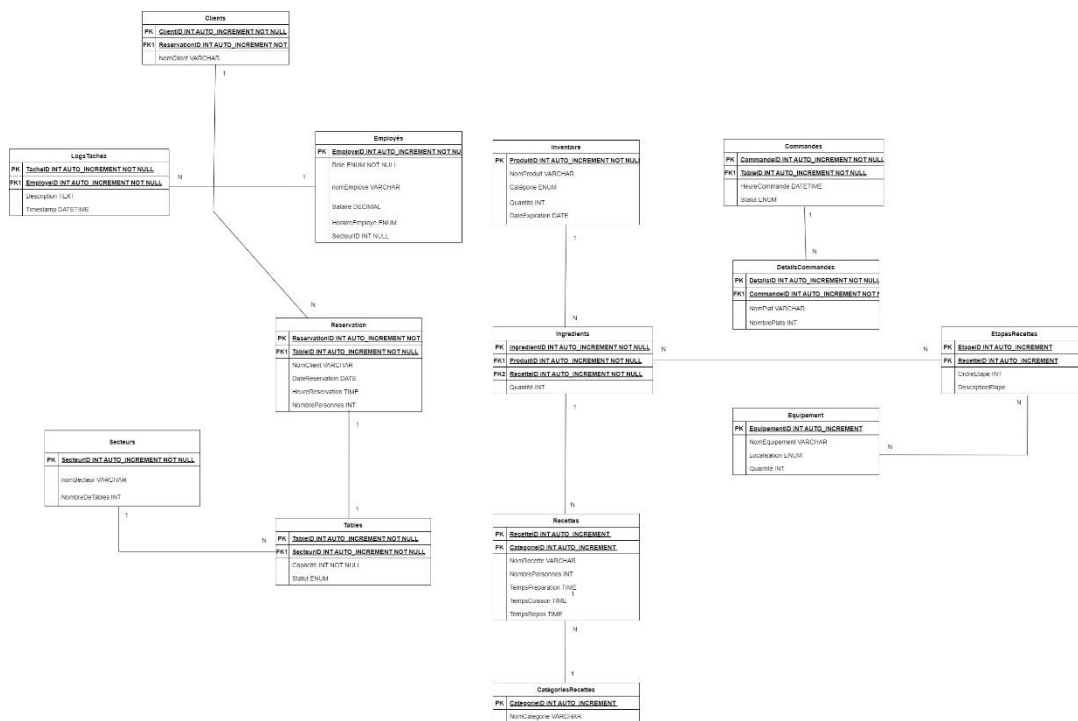


Fig 15 : Modèle conceptuel de données (MCD)

Conclusion

Ce projet a permis de développer une application de gestion d'un restaurant en utilisant l'architecture MVC (Modèle-Vue-Contrôleur), garantissant ainsi une séparation claire des responsabilités entre la gestion des données, l'interface utilisateur et la logique métier. L'utilisation de SQLite pour la gestion des bases de données a facilité la manipulation des informations tout en offrant une solution légère et efficace pour les besoins du projet.

En somme, l'application permet de gérer de manière intuitive et interactive des aspects essentiels d'un restaurant, tels que la gestion des menus, des réservations, des tables, des commandes, ainsi que la gestion des stocks. Grâce à l'architecture modulaire adoptée, le système est conçu pour évoluer facilement, ce qui ouvre la voie à de futures améliorations et fonctionnalités supplémentaires.