DOSSIER ARCHITECTURE

Projet programmation système

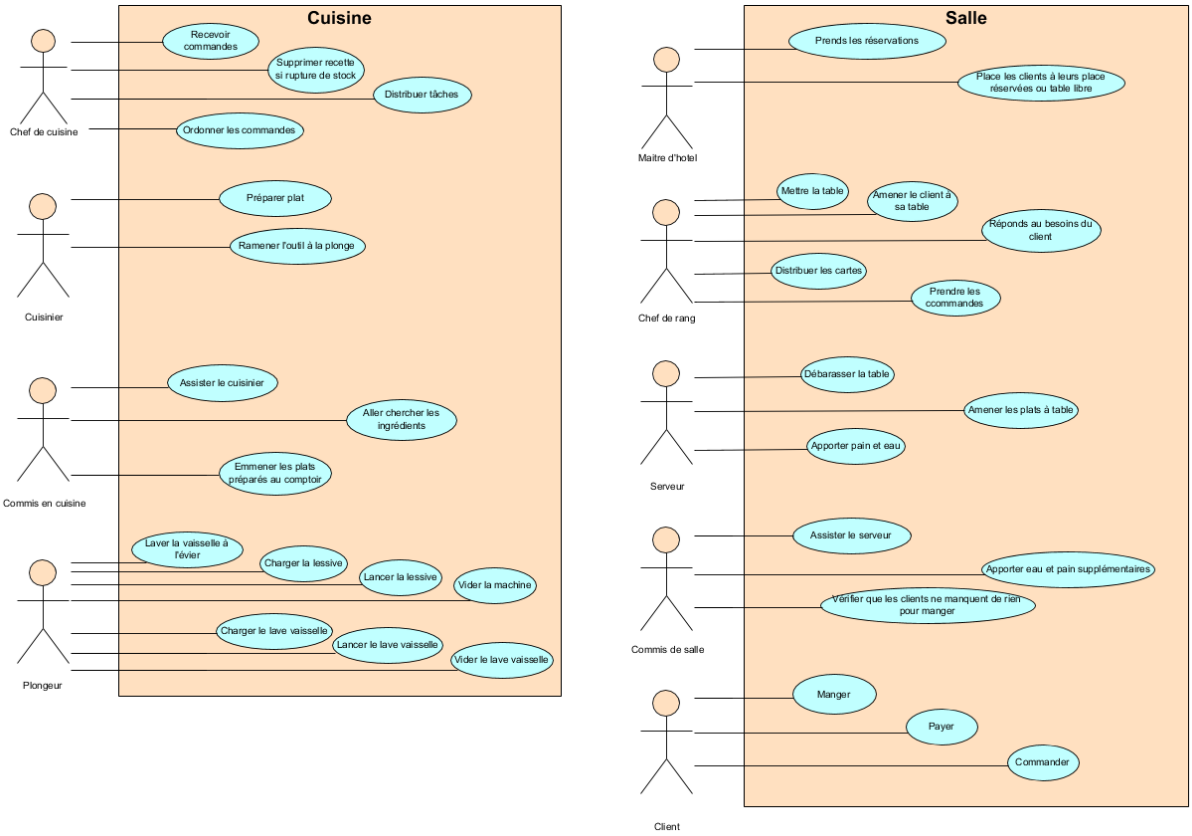
05 DECEMBRE 2018

A3 EXIA CESI
Benjamin CUNY, Thomas LEVAN, Clément FOUBERT, Laura SCHLAUDER

Table des matières

I. Livrables modélisation UML

1. Diagramme de cas d'utilisation



Le diagramme de cas d'utilisation nous permet d'organiser et simplifier les besoins et fonctionnalités d'un système. Il nous permet de visualiser les besoins de l'utilisateur depuis sa perspective et s'assurer que le système correspond bien à ses attentes.

Ici chaque acteur correspond à l'un des postes de travail du restaurant. Ils sont séparés en deux espaces distincts : la cuisine et la salle. Chaque acteur à ses actions (cas d'utilisations) représentées dans les cases ovales bleues.

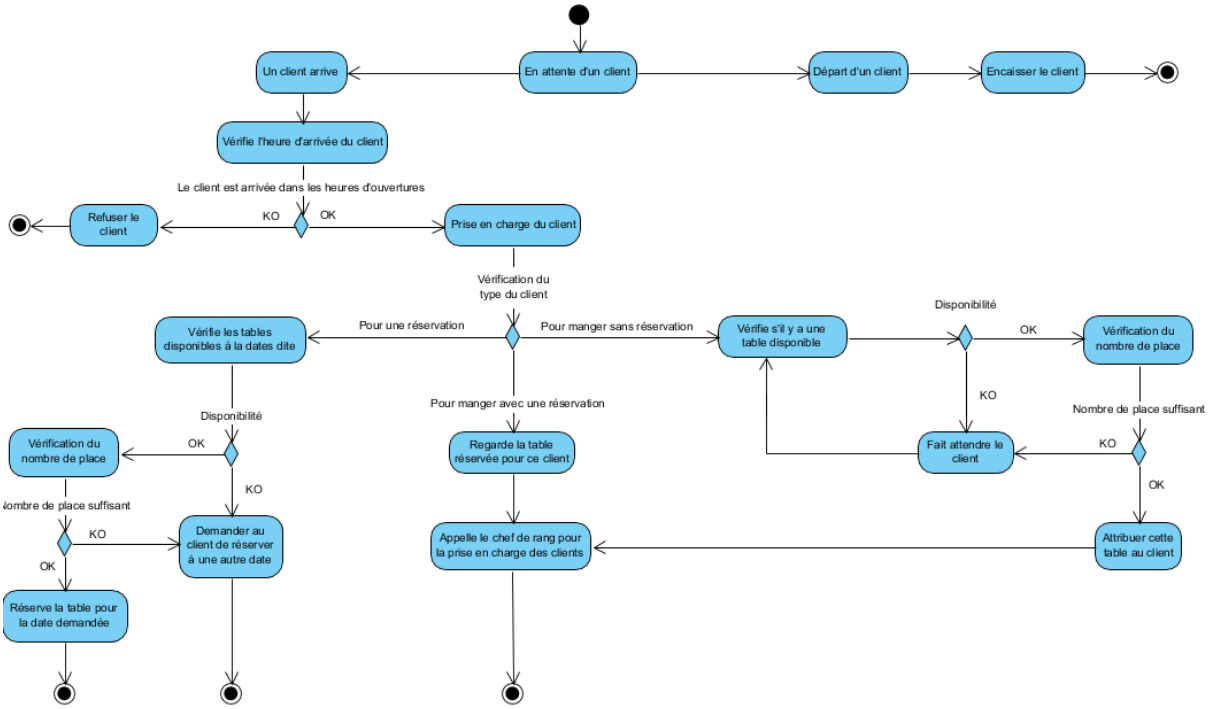
2. Diagrammes d'activité

Un diagramme d'activité nous permet de modéliser des comportements simultanés et le déclenchement d'événements en fonction des états d'un système. Il se décomposent en une suite d'actions.

Nous avons réalisé un diagramme d'activité par poste de travail présents dans notre restaurant. Chaque poste a des actions qui lui sont propres et qui seront représentés dans ces différents diagrammes.

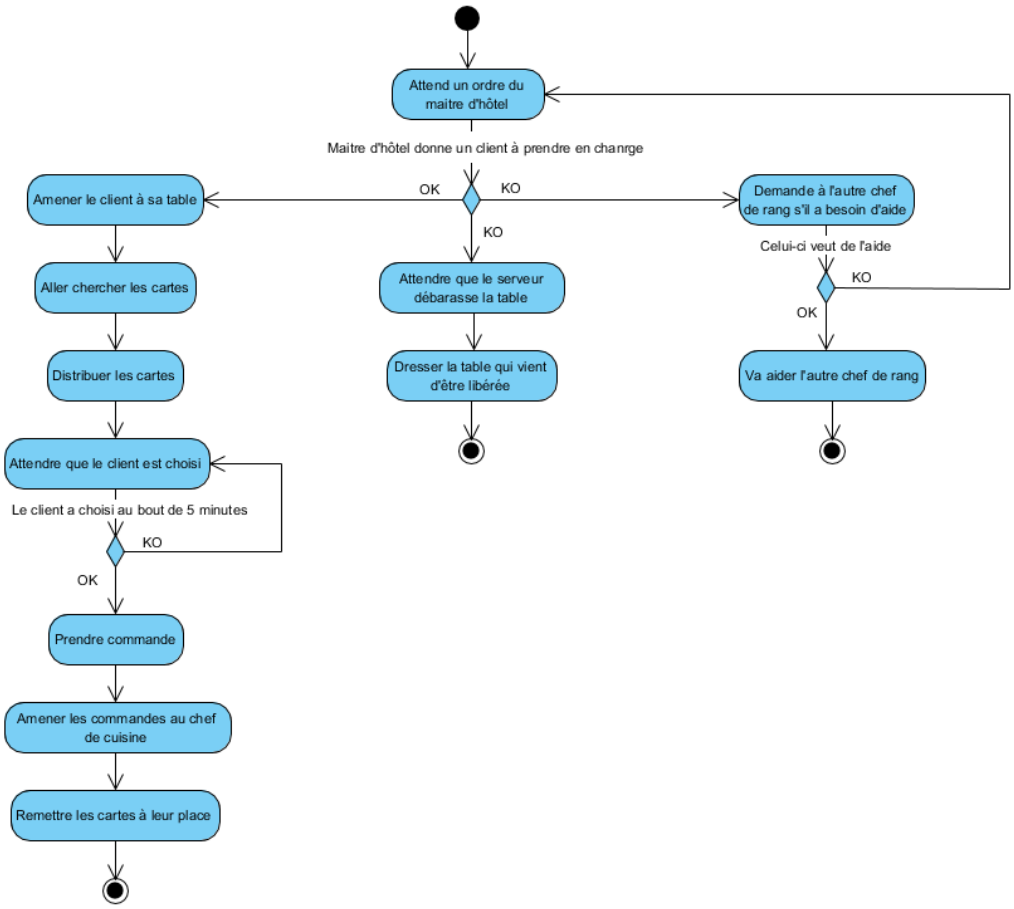
A. Maitre d'hôtel

Le maître d'hôtel est responsable de l'accueil du client. Il s'occupe de gérer les arrivées des clients qui viennent pour manger, qu'ils aient une réservation ou non, ainsi que ceux qui viennent simplement faire une réservation pour une date prochaine. Il doit également s'occuper de l'encaissement des clients.



B. Chef de rang

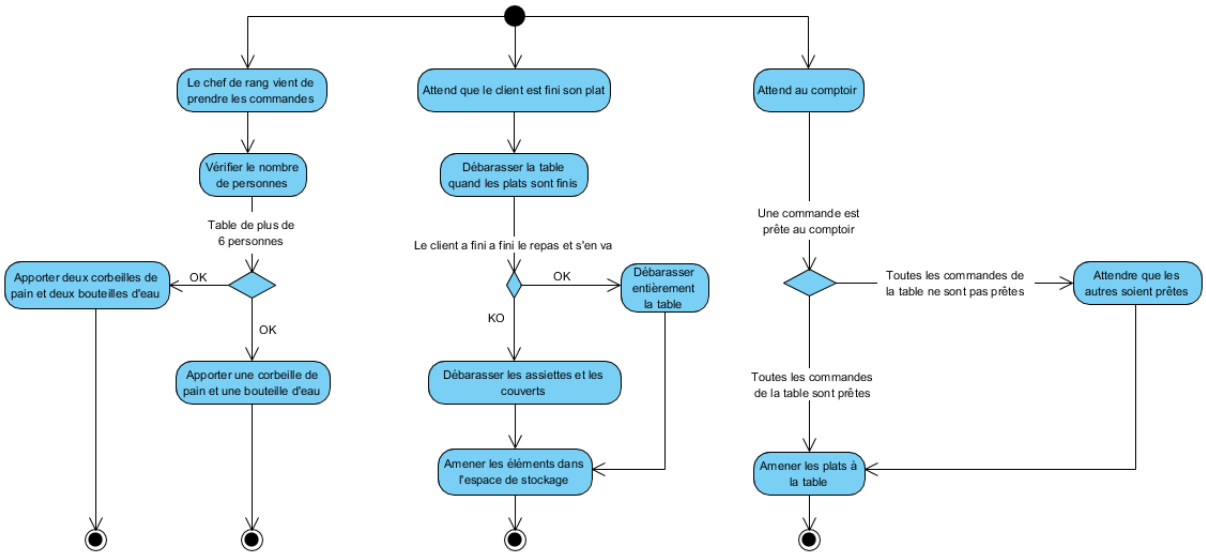
Le chef de rang est sous la responsabilité du maître d'hôtel. Il est responsable d'amener les clients à leur table assignée, de prendre les commandes et de transmettre celles-ci au chef de cuisine. Lorsqu'un serveur a fini de débarrasser une table, il doit dresser la table à nouveau. S'il n'a rien à faire, il peut également aller aider l'autre chef de rang.



C. Serveur

Le serveur doit apporter le bon nombre de corbeilles ou de bouteilles après que le chef de rang ait fini de prendre les commandes d'une des tables.

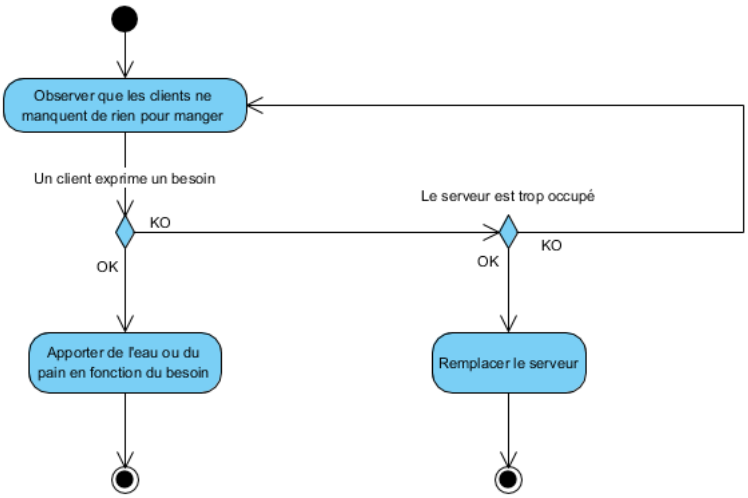
C'est lui qui amène également les plats prêts des clients en faisant attention d'amener les plats d'une même table en même temps. Il s'occupera une fois que les clients ont fini leur plat de débarrasser les assiettes et couverts. Si les clients ont fini leur repas et partent, le serveur débarrassera la table entière.



D. Commis de table

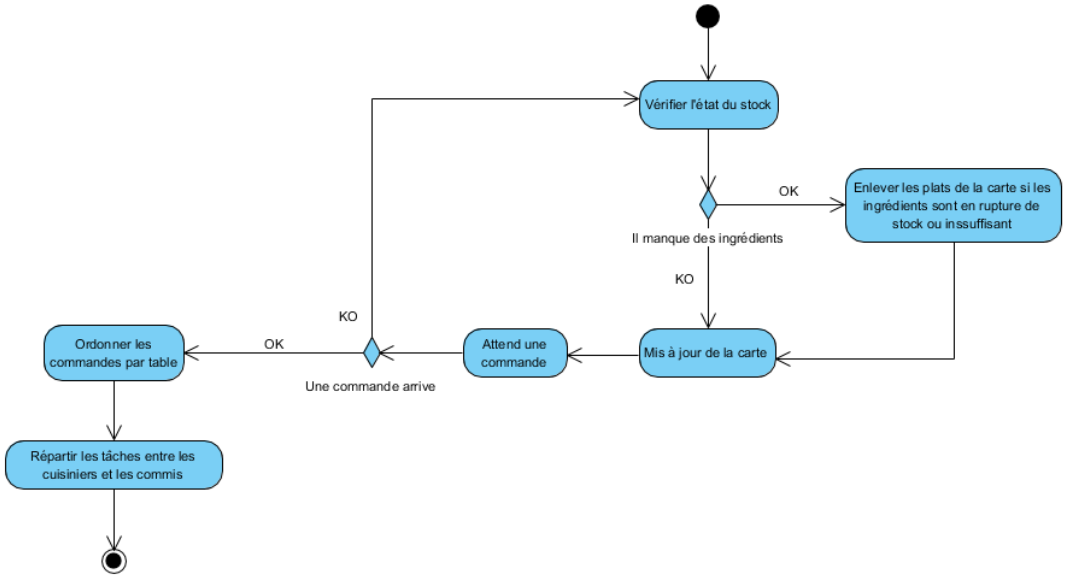
Le commis de salle doit faire attention que les clients ne manquent de rien pour manger. Il est chargé de leur amener du pain ou de l'eau au cours du repas s'ils en n'ont plus.

Il peut également remplacer le serveur si celui-ci est trop occupé.



E. Chef de cuisine

Le chef de cuisine est responsable de vérifier régulièrement l'état du stock afin de voir si les ingrédients pour une recette sont suffisants ou s'il y a une rupture de stock d'un certain ingrédient. Il devra par conséquent mettre à jour le menu et enlever les plats qui ont besoin des ingrédients manquants. Il est aussi chargé de réceptionner les commandes données par le chef de rang et de les ordonner par table. Il va ensuite répartir les tâches entre les cuisiniers et les commis.

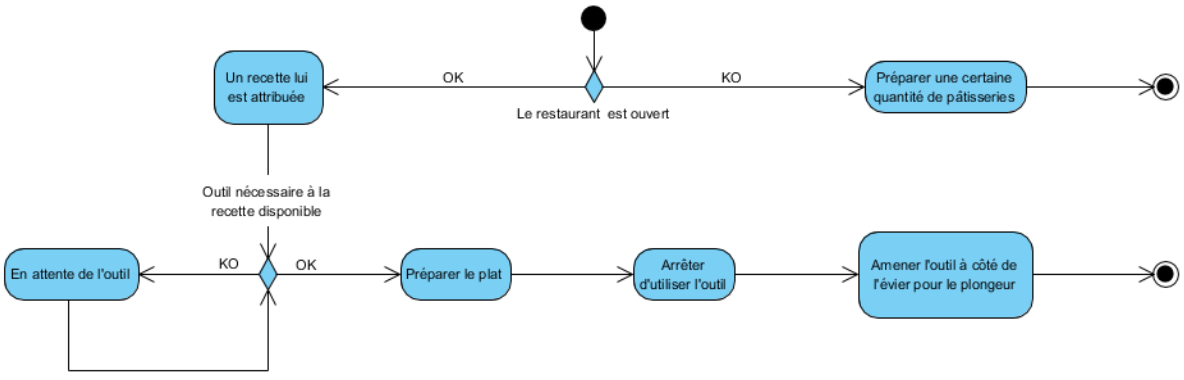


F. Cuisinier

Le cuisinier se charge de préparer les plats en cuisine.

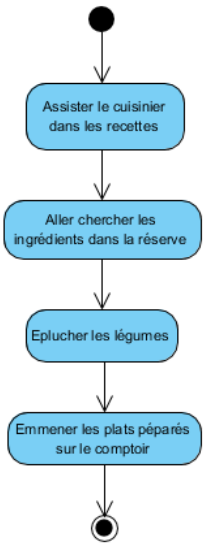
Lorsque le restaurant n'est pas ouvert, il prépare une certaine quantité de pâtisseries qui seront à la carte.

Quand le restaurant ouvre, sa fonction principale est de cuisiner une recette attribuée par le chef de cuisine. Il vérifiera si l'outil dont il à besoin est disponible. Si celui-ci est utilisé ou sale, le cuisinier attendra que l'outil soit disponible. Dans le cas contraire, il peut préparer le plat directement avec l'outil nécessaire. Il amènera ensuite l'outil sale à la pile de l'évier pour que le plongeur le lave.



G. Commis de cuisine

Le commis de cuisine sert en majorité à assister les cuisiniers dans leur travail. Il va chercher les aliments dans la réserve et épluche les légumes pour faciliter le travail des cuisiniers. Il peut également prendre leur place en cas de charge de travail importante. Enfin, il apporte les plats prêts sur le comptoir pour qu'ils soient servis.



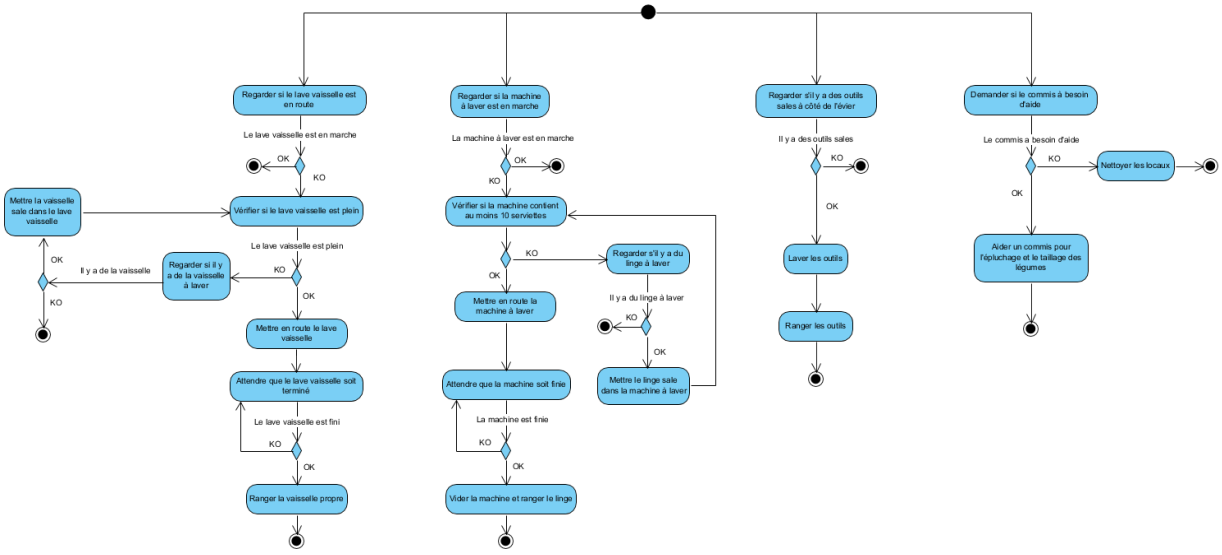
H. Plongeur

Le plongeur occupe trois rôles principaux : gérer le lave-vaisselle, la machine à laver et le lavage des outils de cuisine à l'évier.

Il vérifie si le lave-vaisselle est disponible avant de le charger et mets en route celui-ci toutes les 10 minutes. Il s'occupera également de le vider et de ranger la vaisselle.

Il procède de la même façon pour la machine à laver. Il vérifie qu'elle est disponible avant de la remplir. Il la mettra en route une fois qu'il y a au moins 10 items à l'intérieur. Une fois que la machine sera terminée, il la videra et rangera le linge.

Les outils de cuisine placés à côté de l'évier sont lavés par celui-ci puis rangés. Lorsque le plongeur n'est pas occupé, il peut proposer son aide au commis de cuisine pour l'épluchage et le taillage des légumes.



3. Diagrammes de classes

A. La cuisine

Dans la cuisine, on peut retrouver la classe principale kitchen qui regroupe les différents éléments principaux de la cuisine comme le comptoir, le frigo, le four ou les deux machines de lavage.

En dessous on peut apercevoir la classe du personnel de la cuisine avec les différents personnels et leurs actions propres comme par exemple le chef qui peut PrepareMenu().

Sur la gauche sont visible les appareils de la cuisine, regroupés en MajorAppliance et ensuite divisés entre WashingAppliance et BakingAppliance correspondants respectivement aux appareils de nettoyage et aux appareils de cuisine. Chacun de ses équipements possède sa fonction.

Sur la droite on peut voir la factory qui va nous servir à la création de nos petits objets comme les couverts ou le service de table. Ceux-ci possèdent une propriété pour connaître leur état de propreté et leur temps de lavage.

Enfin on peut apercevoir en bas la façade pour le stock de produits de notre cuisine qui nous permettra de consulter le stock de produits dans la bdd et le comptoir sur lequel on peut stocker différents éléments

B. La salle

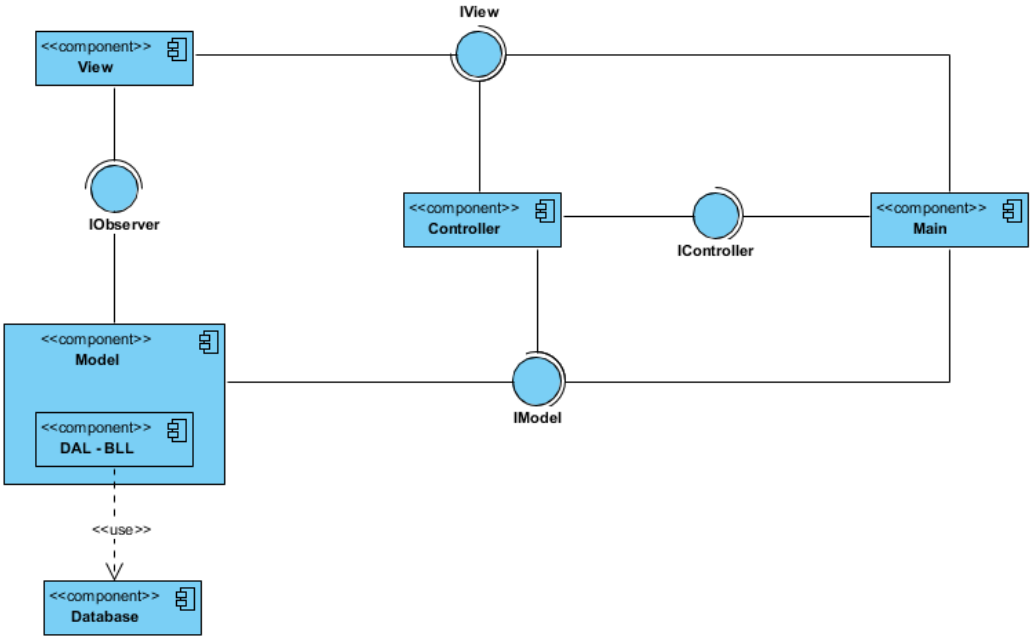
La salle quant à elle est plus simple. On y retrouve la class staff pour le personnel comme dans la cuisine, chacun ayant ses différentes fonctions. On peut voir sur la partie droite la gestion des tables et différentes places pour les clients et à gauche les plats, menus et assiettes venant de la cuisine.

C. Le DAL+BLL

Dans cette partie, on va retrouver les éléments qui permettent la communication et le CRUD à notre base de données. Cette partie sur le Data Access Layer + Business Logical Layer permet, au travers le découpage entre plusieurs classes, d'importer les données, les modifier en objet, et créer des méthodes pour les utiliser. On va donc retrouver 4 classes pour chacune de nos tables et nous avons 3 tables. Mais tout d'abord, grâce à la classe *DatabaseContext*, et à la méthode **OnConfiguring()**, on va se connecter à notre base de données. En prenant exemple sur la table *Product*, on va donc suivre le chemin suivant :

- Dans un premier temps, la classe *ProductDAO* récupère les champs de notre table ainsi que les clés étrangères.
- Ensuite, on va créer nos objets dans la classe *ProductBusiness*, on va créer les objets qui correspondent.
- Dans *ProductMapper*, on va créer les méthodes qui vont nous permettre de ''Mapper'' nos variables qui correspondent, afin de les passer d'un type à un autre, dans les deux sens.
- Pour finir, on retrouve la classe *CategoryService* qui va regrouper les méthodes du CRUD.

4. Diagrammes de composants



Chacun de nos composants représente les parties de notre programme.

Tout d'abord, chaque composant de notre MVC possède son interface (*IModel, IController,* IView) lui permettant de communiquer ses informations avec le *main* mais également avec le composant **Controller**.

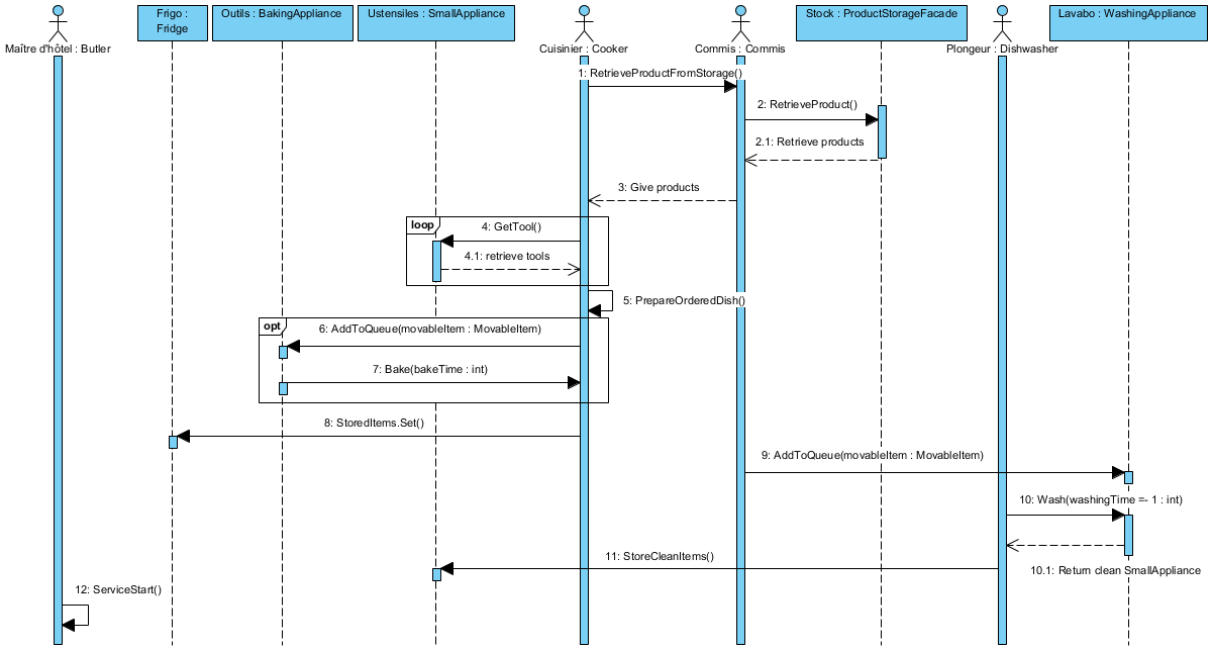
Les composants **View** et **Model** ne pouvant pas dialoguer directement, le **Model** observe la **View** au travers du Design Pattern Observer et de son interface *IObserver*.

Le composant Database représente notre base de données. Cette dernière est utilisée par le **Model**, mais c'est notre composant **Data Access Layer – Business Logic Layer** qui communique avec la base de données. Une fois que le **DAL – BLL** a récupéré les informations, le **Model** les utilise.

5. Diagrammes de séquences

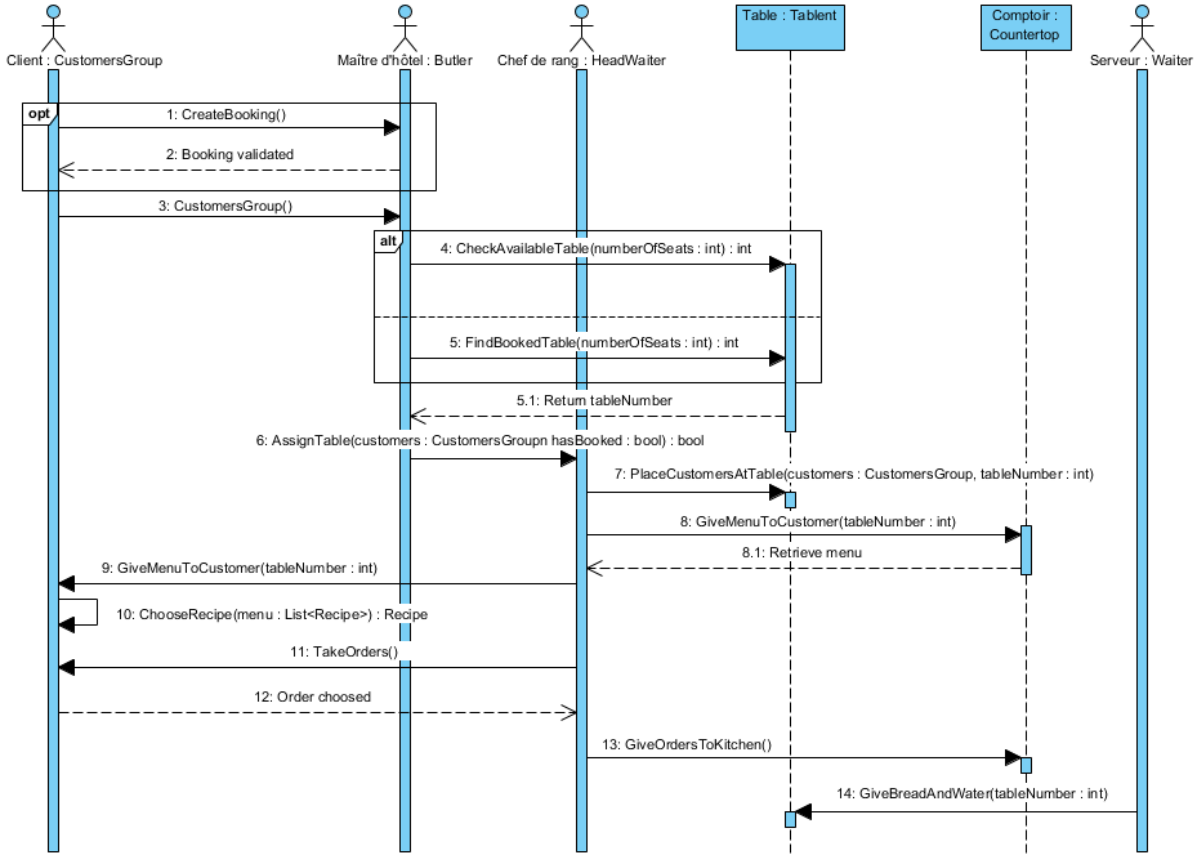
Le diagramme de cas d'utilisation nous a permis de voir le « quoi » des interactions avec les acteurs. Le diagramme de séquence va nous permettre de voir « comment » les différents éléments interagissent entre eux et avec les acteurs extérieurs au système. On peut voir sur l'image ci-dessus les différents éléments : les bonhommes représentent les différents acteurs (client et postes) tandis que les éléments carrés représentent les éléments physiques ne faisant aucune action (le stock sert juste à entreposer les aliments). Toutes nos lignes de vie sont des classes, les flèches pleines sont nos méthodes et celles en pointillées les retours. La partie en gras représente leur utilisation, pour les acteurs, ils sont actifs en permanence, même s'ils n'exécutent pas une fonction dans l'immédiat.

A. Le restaurant avant l'ouverture



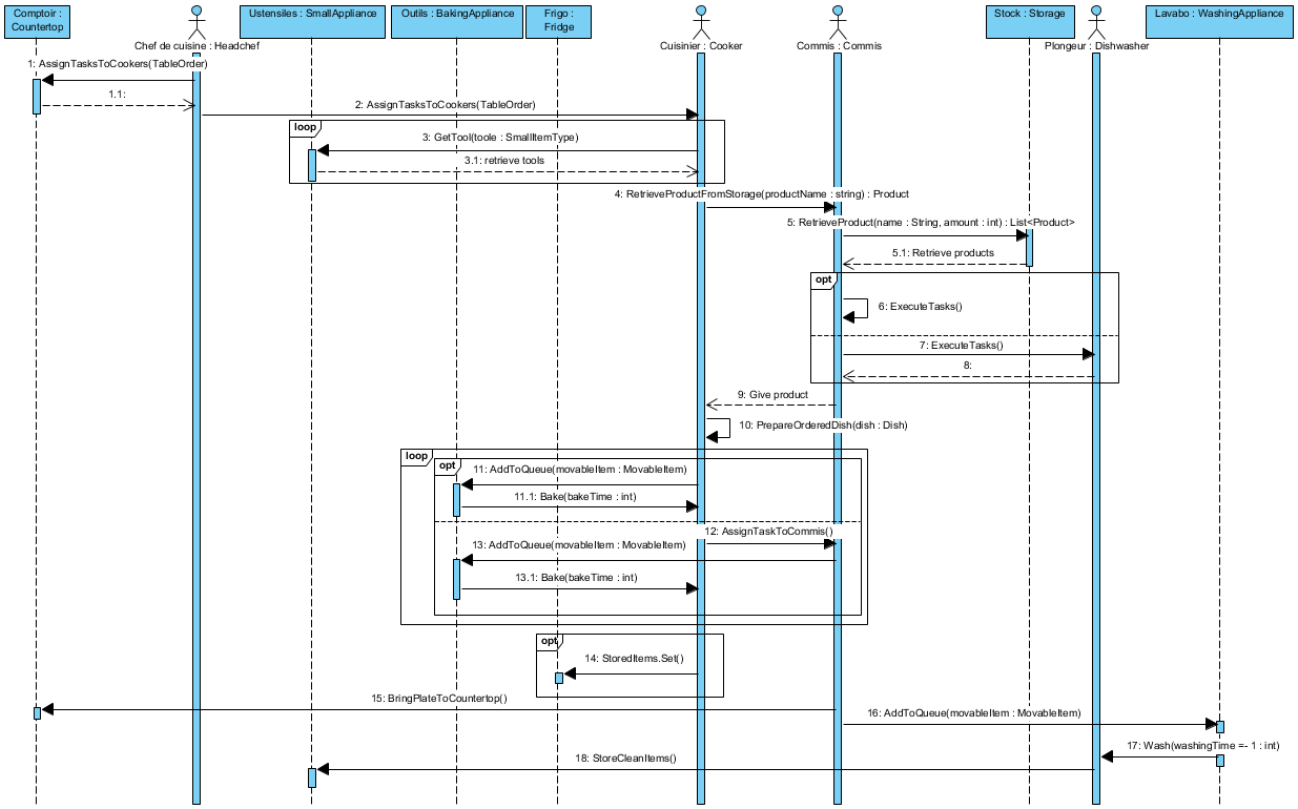
- Le cuisinier va demander au commis d'aller chercher les ingrédients qui lui faut pour ses desserts avec *RetrieveProductFromStorage().*
- Grâce à *RetrieveProduct()*, le commis récupère les produits correspondant et les rend au cuisinier.
- Avec *GetTool()*, le cuisinier récupère les bons ustensiles pour ses desserts, il ne quitte pas la **boucle** tant qu'il n'a pas les ustensiles.
- Il cuisine ensuite grâce à *PrepareOrderedDish()* et peut, en **option,** utiliser les fours en ajoutant les produits avec *AddToQueue()* et ce dernier lui renvoie le produit cuit avec *Bake()*.
- Pour finir, les pâtisseries sont stockées dans le frigo avec *StoredItems.Set()*.
- Le commis va apporter les ustensiles sales à l'évier grâce à *AddToQueue()*.
- Le plongeur lave ensuite les ustensiles grâce à l'évier et la méthode *Wash().*
- *StoreCleanItems()* va donc permettre au commis de ramener les ustensiles dans leur placard.
- Le maitre d'hôtel va démarrer le service avec *ServiceStart()* et les clients vont commencer à arriver.

B. Les commandes



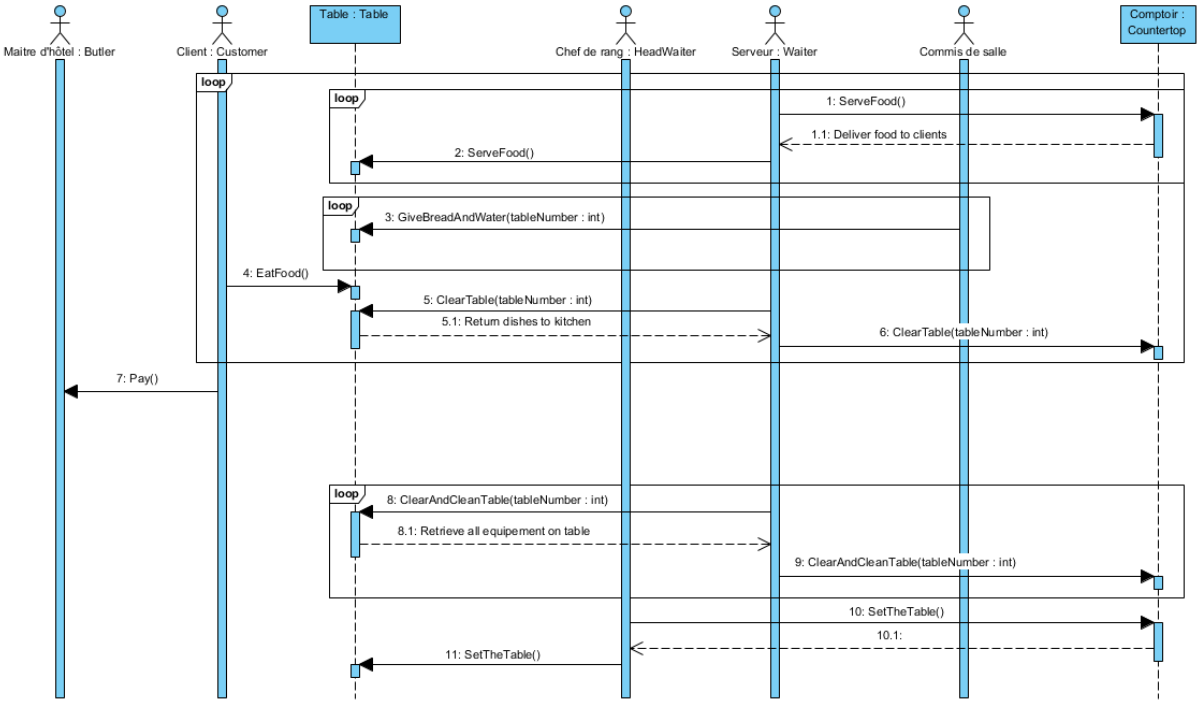
- *CreateBooking()* se trouve à l'intérieur d'une **option**, ainsi seul les clients ayant réservé passent par cette étape. Après avoir réservé, les clients sont prioritaires.
- Ensuite, le maitre d'hôtel va soit, d'abord aller chercher la table d'un client qui a réservé avec *FindBookedTable()*, soit il va aller regarder grâce à *CheckAvailableTable()* s'il y a des tables de libre. Il va donc renvoyer le numéro de la table de libre, s'il y en a une de libre grâce à une **alternative**.
- Ensuite notre maître d'hôtel va assigner une table aux clients avec *AssignTable()* en les passant au chef de rang.
- Le chef de rang va mener les clients à la table correspondante avec *PlaceCustomersAtTable()* avec en paramètre l'id des clients et le numéro de la table.
- Le chef de rang va chercher les cartes au comptoir et les amener aux clients avec *GiveMenuToCustomers()* en se rappelant le numéro de la table auquel le groupe correspond.
- Le client va choisir un menu avec *ChooseRecipe()* parmi la *List<Recipe>*. Cette liste contient en temps réel les éléments de son stock, et ainsi peut enlever les plats dont il n'y a plus assez d'ingrédient.
- Le chef de rang va demander au client son menu avec *TakeOrders()*.
- Le chef de rang ramène les cartes au comptoir et amène les commandes avec la méthode *GiveOrderToKitchen()*.
- Le serveur va amener du pain et de l'eau à la table venant de commander avec la méthode *GiveBreadAndWater()* et en paramètre la table des clients venant de commander.

C. Les cuisines



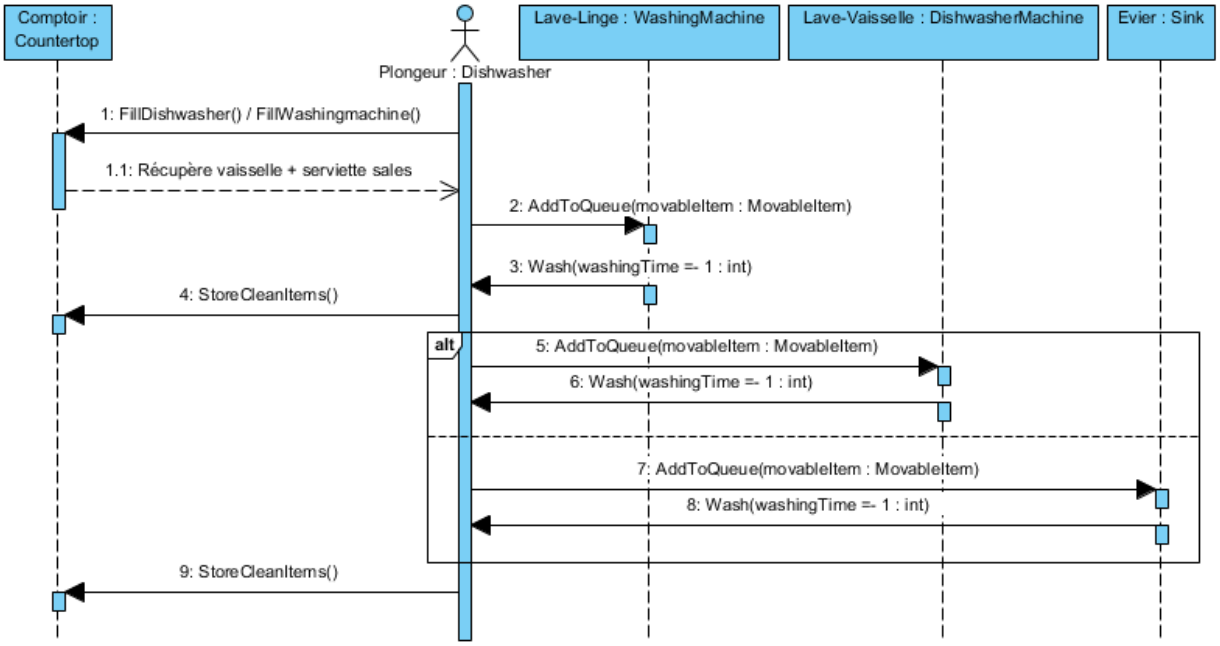
- *AssignTasksToCookers()* permet au chef de cuisine de voir les plats commandés et assigner aux cuisiniers.
- Le cuisinier va choisir ses ustensiles en fonction de son plat avec *GetTool().* Tant que les ustensiles ne sont pas disponibles, on reste dans la **boucle**.
- Il va ensuite demander les aliments qu'il souhaite au commis avec *RetrieveProductFromStorage()*.
- Ce dernier va aller chercher les ingrédients avec la méthode *RetrieveProduct()* et les donner au cuisinier.
- En **option**, le Commis va pouvoir s'occuper des légumes, ou s'il est occupé, le plongeur va pouvoir le faire à sa place. Tout se fait avec la méthode *ExecuteTasks()*.
- On va ensuite cuisiner avec *PrepareOrderedDish().*
- Le cuisinier va faire cuire ses aliments avec *AddToQueue()* et *Bake().* Ces méthodes sont dans une **boucle**, au cas où nous avons plusieurs plats à cuire. Ces plats peuvent également réaliser soit par le cuisinier, soit en **option** par le commis.
- En **option**, on peut stocker un plat au frigo avec *StoredItems.Set().*
- Le commis apporte ensuite le plat au comptoir avec *BringPlateToCountertop().*
- Comme le plat est terminé, le commis ajoute les ustensiles sales à la pile de l'évier avec *AddToQueue()*.
- Avec *Wash()*, et en paramètre le temps de lavage, ce dernier va laver les ustensiles à l'évier.
- Le commis va ensuite ranger les ustensiles avec *StoreCleanItems()*.

D. Le déroulement du repas



- Ici, nous avons 2 alternatives possible, les deux sont identiques, mais peut être réalisé soit par un serveur, soit par un commis si les deux serveurs sont occupés. Grâce à *ServeFood()*, on va récupérer les plats et les servir aux clients. Cette action est dans une **boucle** car un serveur ne peut porter que 5 plats en simultané, donc s'il y a plus de clients à table, il devra faire plusieurs trajets.
- Dès que le client manque de pain ou d'eau, le commis va en apporter de nouveau grâce à *GiveBreadAndWater(),* grâce au numéro de la table qu'il a en paramètre, et cette méthode est dans une **boucle** lui permettant de ramener du pain à chaque fois.
- Ensuite le client est notifié de son plat et le mange avec *EatFood()*
- Une fois que le client à finit, le serveur vient débarrasser son plat avec *ClearTable()*.
- Toutes ces actions précédentes sont dans une **boucle**, ainsi le client peut manger son menu entièrement avant de quitter la boucle.
- Une fois finit, il va payer auprès du maître d'hôtel grâce à *Pay()* et s'en va.
- Le serveur débarrasse ensuite la table avec *ClearAndCleanTable()* et amène la vaisselle ainsi que le textile au comptoir. Parfois, il y a plus de 5 clients, cette action est dans une **boucle**.
- C'est ensuite au chef de rang d'aller chercher au comptoir de la vaisselle ainsi que du textile propre afin de dresser la table avec la méthode *SetTheTable().*

E. Le traitement de la vaisselle et du linge



- Le plongeur va récupérer au comptoir la vaisselle ainsi que le linge sale grâce aux méthodes *FillDishwasher()* et *FillWashingmachine()*.
- Il va ensuite les ajouter à la pile du lave-linge ou du lave-vaisselle, en fonction de son type grâce à la méthode *AddToQueue()*.
- La machine correspondante va ensuite laver grâce à *Wash()*.
- *StoreCleanItems()* va pour finir permettre au plongeur de replacer la vaisselle ainsi que le textile sur le comptoir.

II. Design Patterns

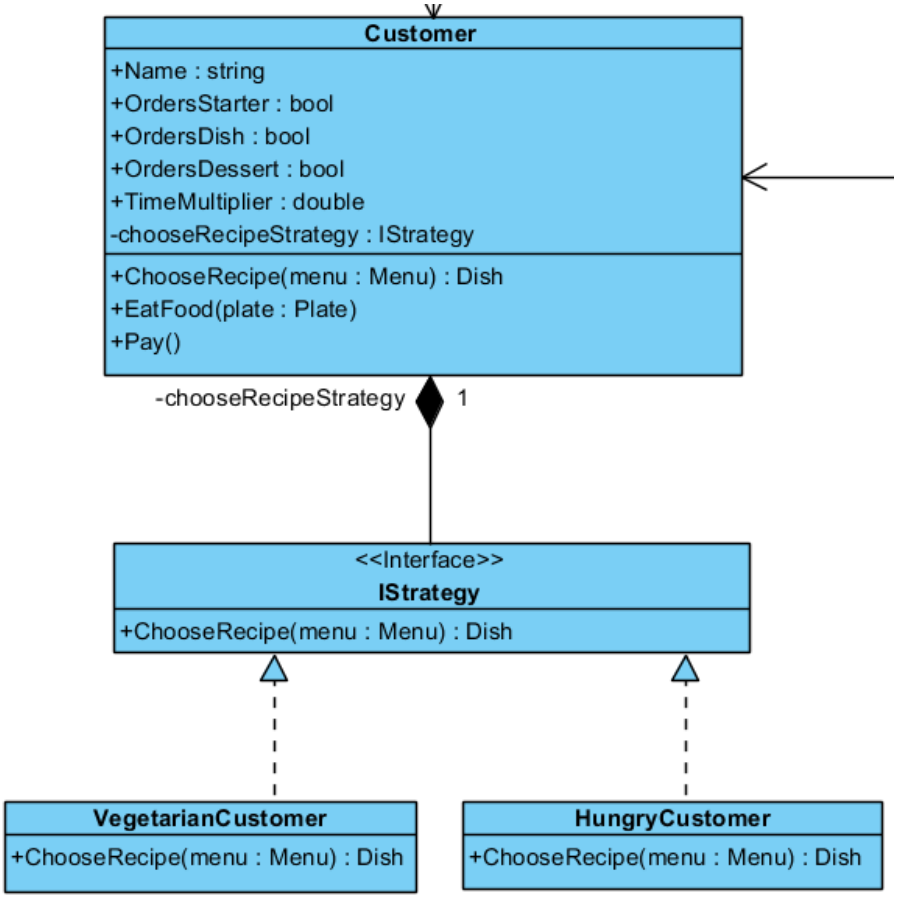
1. MVC

Le MVC, Modèle-Vue-Contrôleur, est souvent utilisé pour le développement d'interfaces utilisateurs. Ce design pattern permet de diviser l'application en trois parties qui sont interconnectées. Ce design pattern permet d'avoir une réutilisation du code efficace.

Nous utiliserons le MVC qui convient à l'application que nous allons développer qui aura une interface graphique.

2. Strategy

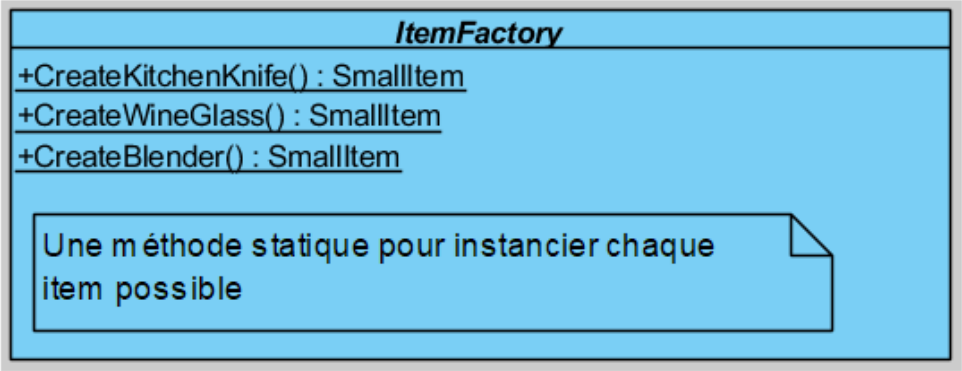
Le Design Pattern Strategy permet de définir une famille d'algorithmes qui correspondront à différentes stratégies applicables sur un objet. Ces algorithmes seront encapsulés et pourront être interchangés. Ainsi, le comportement d'une classe peut varier lors de son exécution.



Nous avons choisi d'utiliser ce design pattern lors du choix du menu des clients. En effet, les clients ne doivent pas avoir de comportements identiques. On applique donc la Strategy au choix des plats. Par exemple, le client peut choisir de prendre seulement un plat principal ou alors de prendre une entrée, un plat et un dessert.

3. Factory

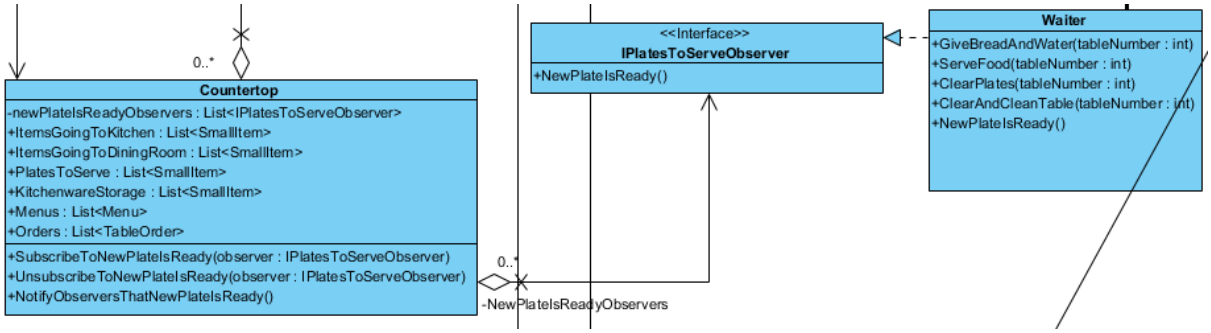
Le Design Pattern Factory que nous avons utilisé est la Static Factory. C'est la plus simple des factory. Elle consiste en une classe statique possédant autant de méthodes statiques que d'items possibles à instancier.



Dans notre restaurant, il y a un grand nombre d'objets qui composent le service de vaisselle (assiettes, couteaux, verres etc.). Au lieu de créer des assiettes de type « plate » ou des couteux de type « knife », on va utiliser une factory qui instanciera tous ces objets avec le type « SmallItem », en leur donnant les bons paramètres suivant le type d'item créé. L'accès à la création de ces objets sera donc facilité.

4. Observer

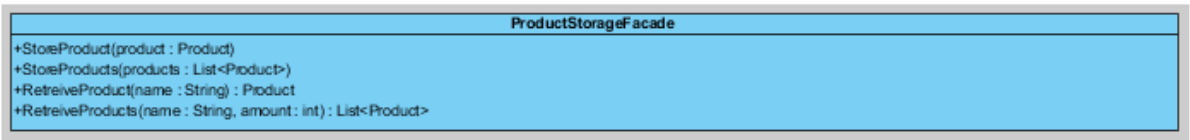
Le Design Pattern Observer permet de définir une dépendance entre un ou plusieurs objets. Lorsque l'objet qui est « observé » change d'état, tous les observateurs, ceux qui sont dépendants de celui-ci, sont notifié et mis-à-jour automatiquement.



Nous avons choisi d'utiliser l'observer pour les interactions entre le serveur et le comptoir où les plats prêts à être servi sont stockés. En effet, le serveur sera l'observateur et sera notifié lorsqu'il pourra venir au comptoir chercher le plat à amener au client qui, lui, est l'observable.

5. Facade

Le Design Pattern Facade nous permet de simplifier une partie du code avec une interface simple à utiliser pour les utilisateurs.

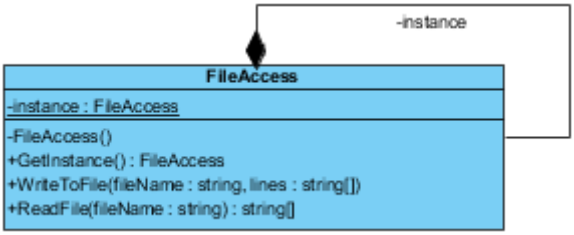


Nous utiliserons le design pattern Facade pour l'accès à la BLL (Business Logic Layer) et la DAL

(Data Access Layer) qui nous permettent d'interagir avec notre base de données. Cette couche est complexe, ce qui justifie l'utilisation d'une Facade afin de simplifier les commandes d'accès à notre base.

6. Singleton

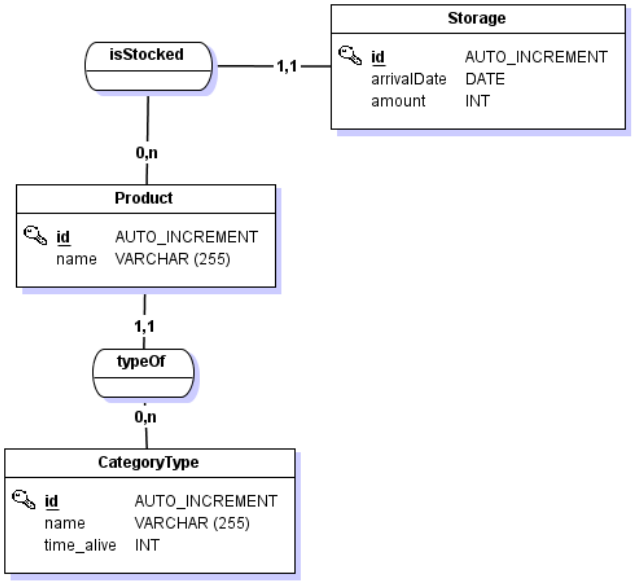
Le Design Pattern Singleton permet de contrôler qu'une classe a seulement une instance et de s'assurer qu'il n'y a qu'un seul point d'accès global à cette instance.



Nous utiliserons ce design pattern pour notre classe qui gérera l'accès aux fichiers de configurations. En effet, cet accès doit être unique afin que les informations restent dans un seul et même fichier.

III. Modélisation de la base de données

1. Modèle conceptuel des données



Notre base de données est séparée en trois tables. Nous stockerons uniquement les ingrédients, leur quantité, ainsi que leur type. Nos recettes, ainsi que nos éléments paramétrables seront implémentées dans notre programme.

La Table *CategoryType* contient donc le type de stockage, c'est-à-dire le congélateur, la réserve ou bien le frigo. On retrouve également la durée de vie correspondant à chaque type de produit.

On a ensuite la table *Product*, avec leur nom, ici sera stocké tous les produits appartenant ou ayant appartenu à notre stock avec comme clé étrangère sa catégorie.

Pour finir, on retrouve la table *Storage*, qui contient la clé étrangère d'un produit pour l'identifier ainsi que sa quantité et sa date d'arrivée. On pourra avec sa date d'arrivée et son temps de vie, dépendant de sa catégorie de produit, définir sa date de péremption et les supprimer de la table *Storage*. La quantité sera modifié en temps réel à chaque création d'une nouvelle recette, la carte des menus sera également mise à jour à chaque fois.

2. Script de création de la BDD

Aller voir le fichier *ScriptResto.sql* dans le dossier *BDD - MCD et Script SQL* des Livrables Finaux.

IV. Annexes

On peut retrouver tous ces schémas dans le fichier *ArchitectureProjet.vpp*.