05 décembre 2018

Nicola bazillio julien zolli adrien dossang paul brouet DELAHAYE MATHIEU

Cesi

Mont Saint Aignan

Dossier architecture

projet programmation système 2018-2019

Table des matières :

[Diagrammes 2](#_Toc531860172)

[Diagrammes de cas d’utilisation : 2](#_Toc531860173)

[Diagrammes d’activité : 6](#_Toc531860174)

[Diagramme de séquence : 10](#_Toc531860175)

[Diagramme de composants : 11](#_Toc531860176)

[Diagramme de classe : 11](#_Toc531860177)

[MCD : 11](#_Toc531860178)

[Explications des Design Pattern 12](#_Toc531860179)

[MVC : 12](#_Toc531860180)

[Factory : 13](#_Toc531860181)

[Bridge : 13](#_Toc531860182)

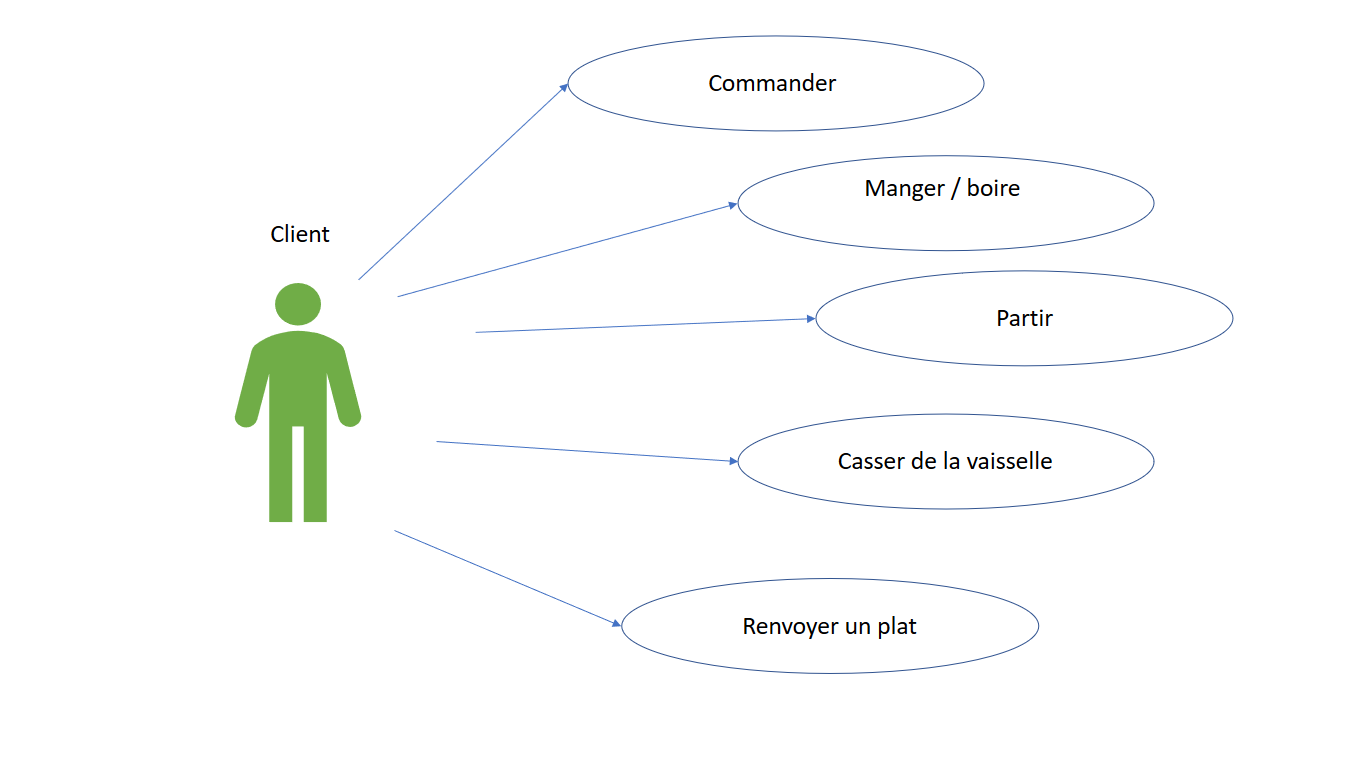
[Singleton : 13](#_Toc531860183)

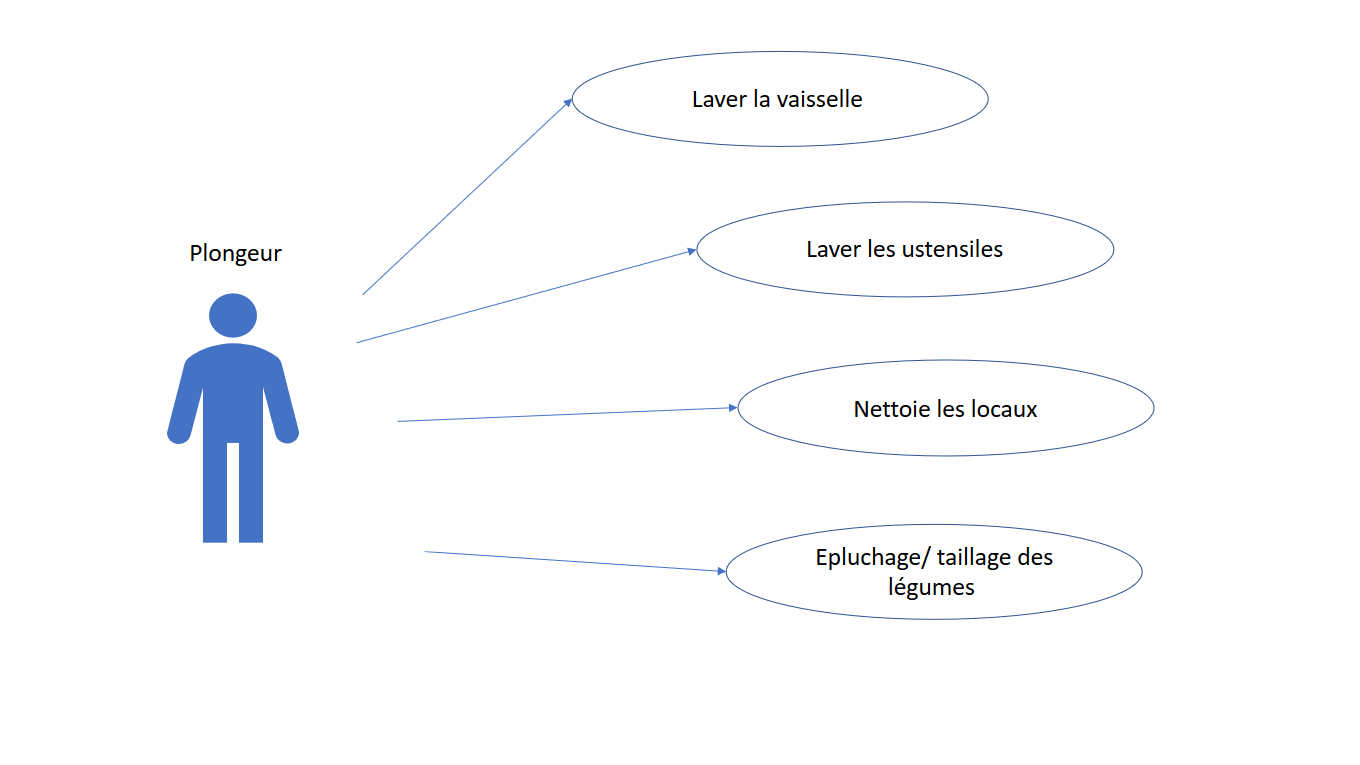
[Observer : 14](#_Toc531860184)

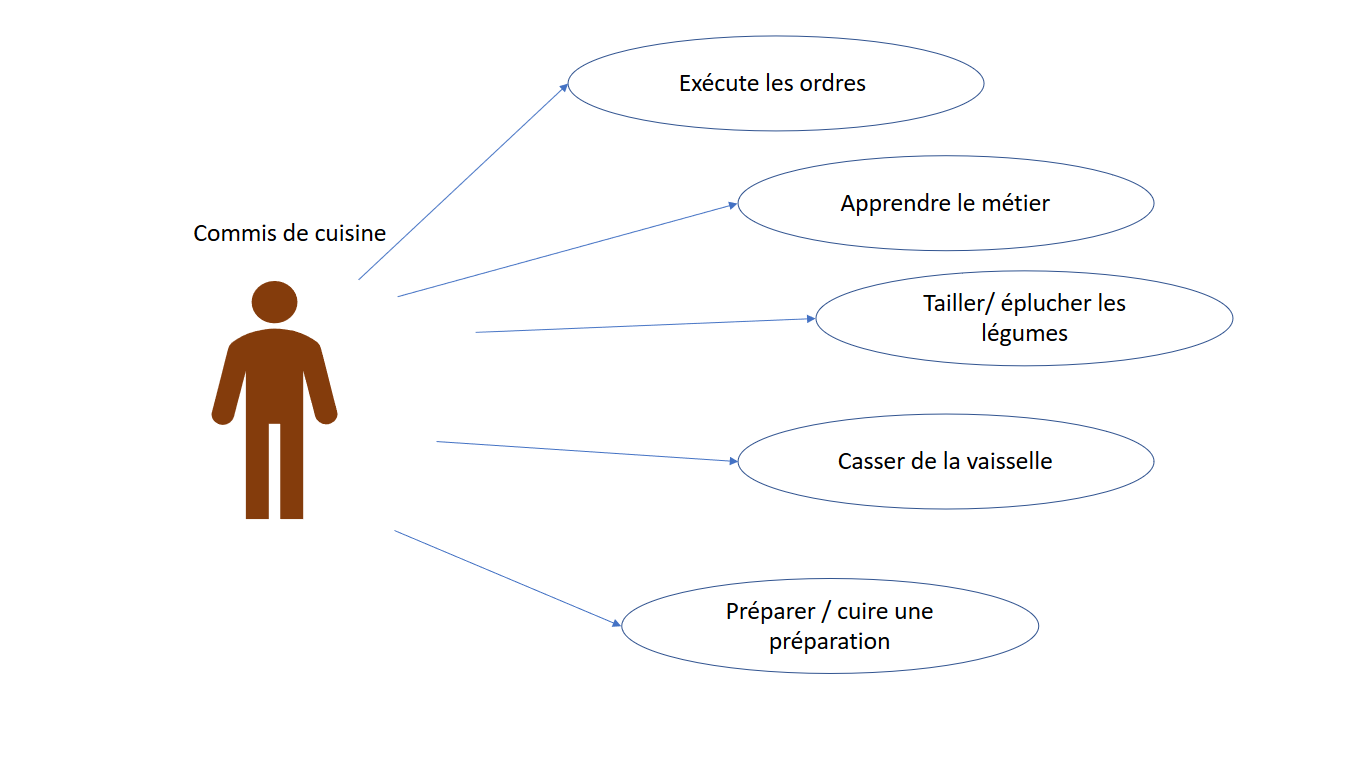
[Iterator : 14](#_Toc531860185)

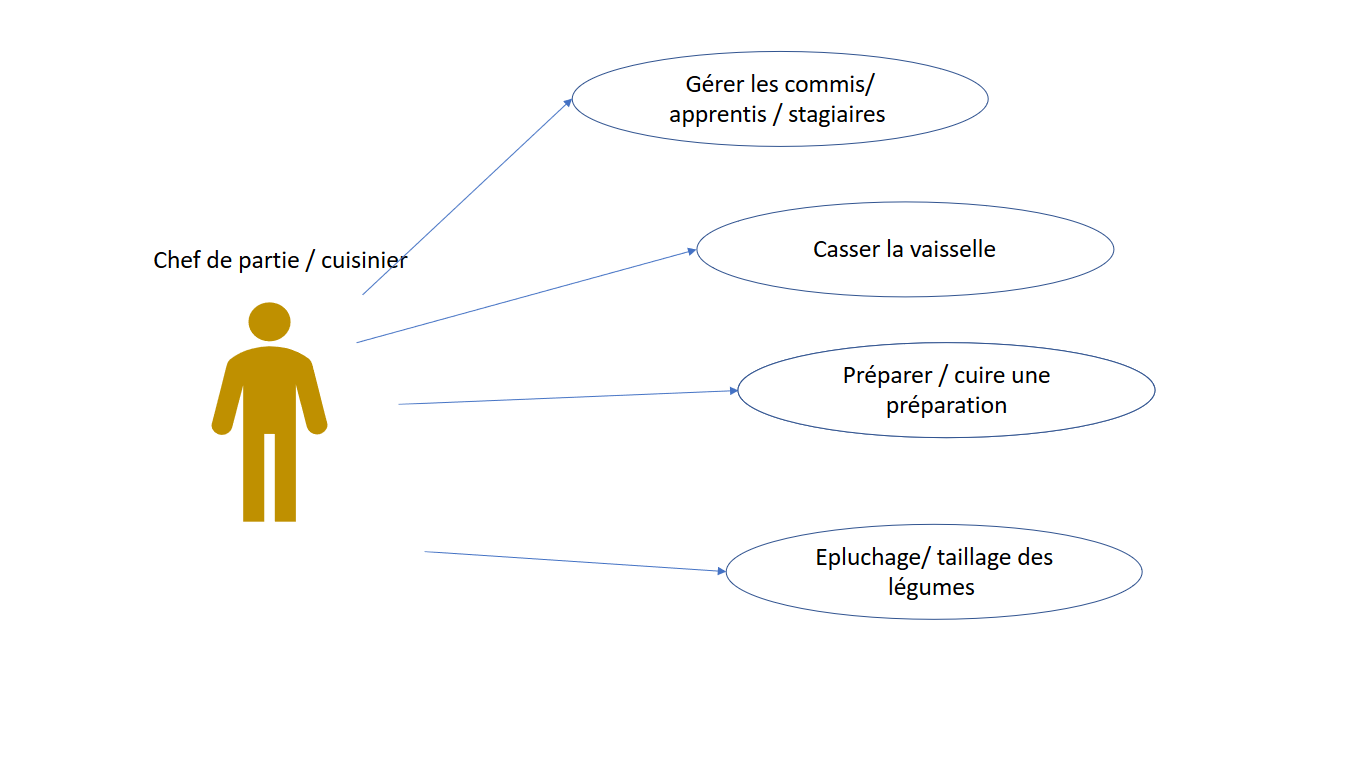
# Diagrammes

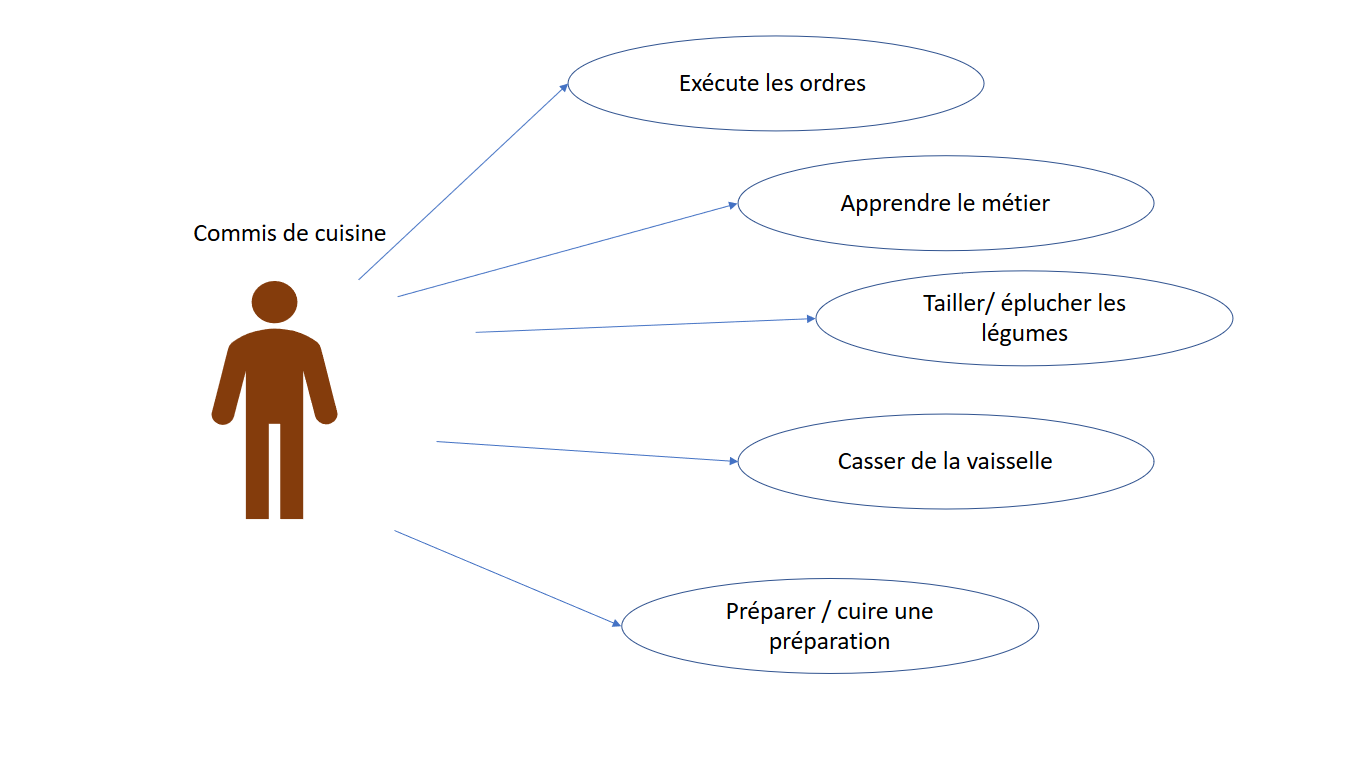
## Diagrammes de cas d’utilisation :

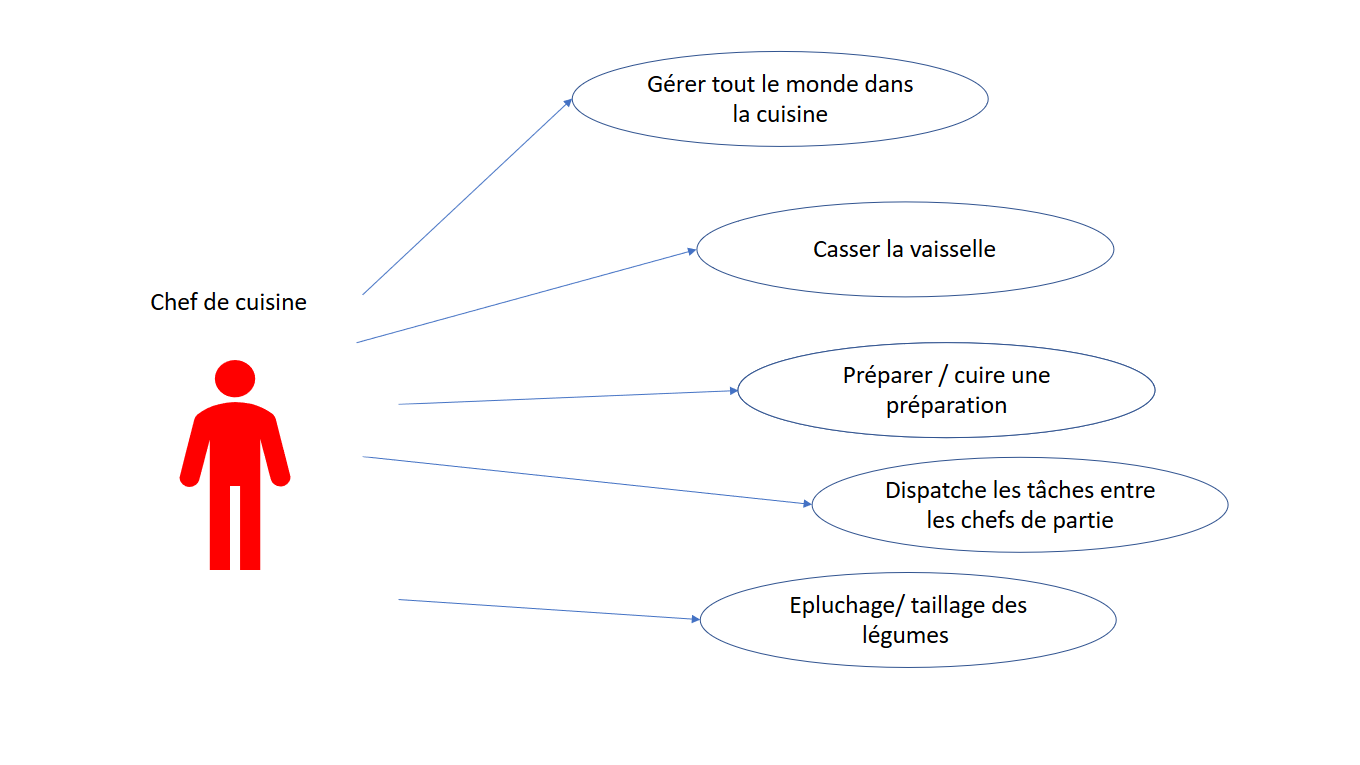


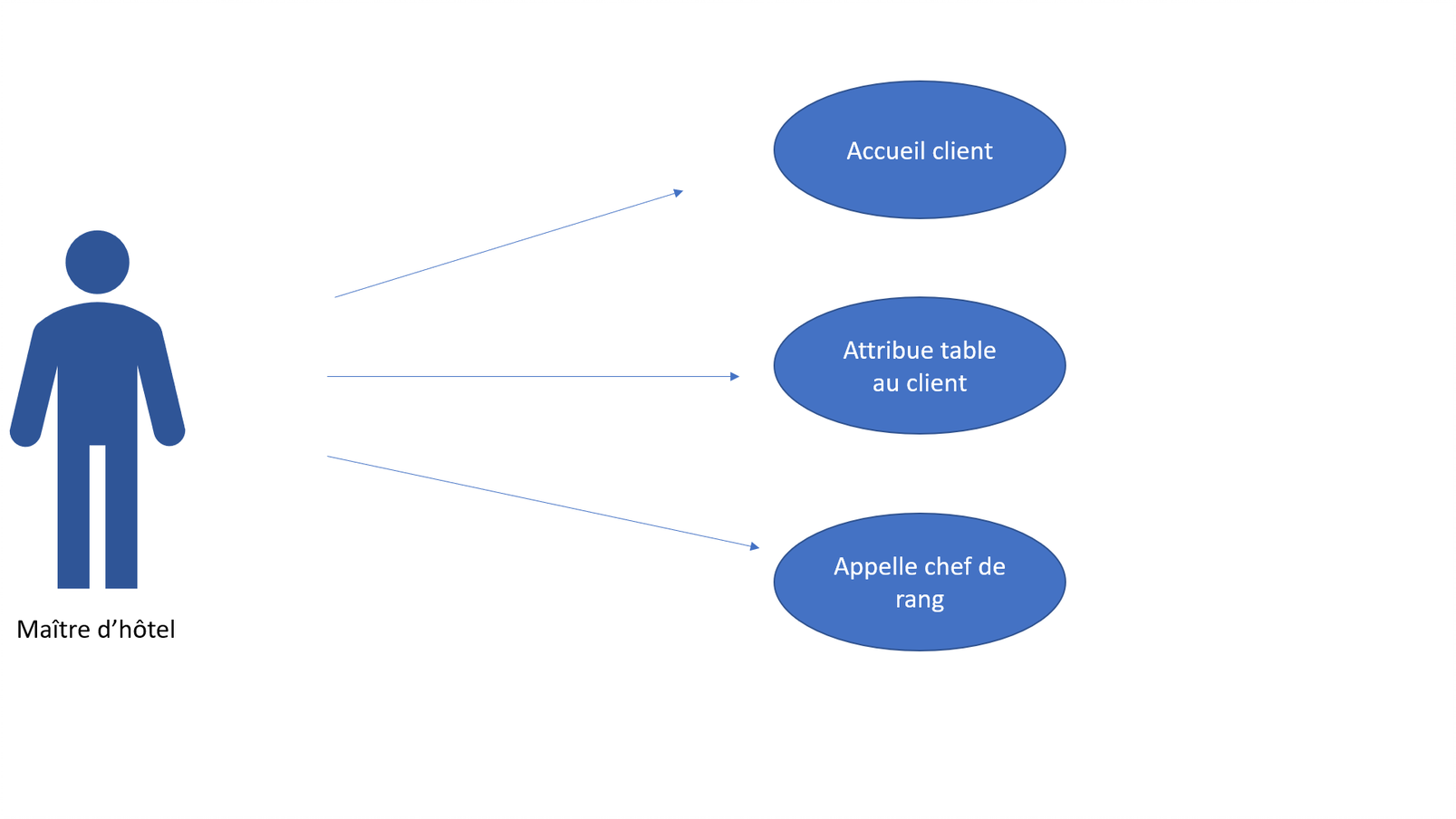


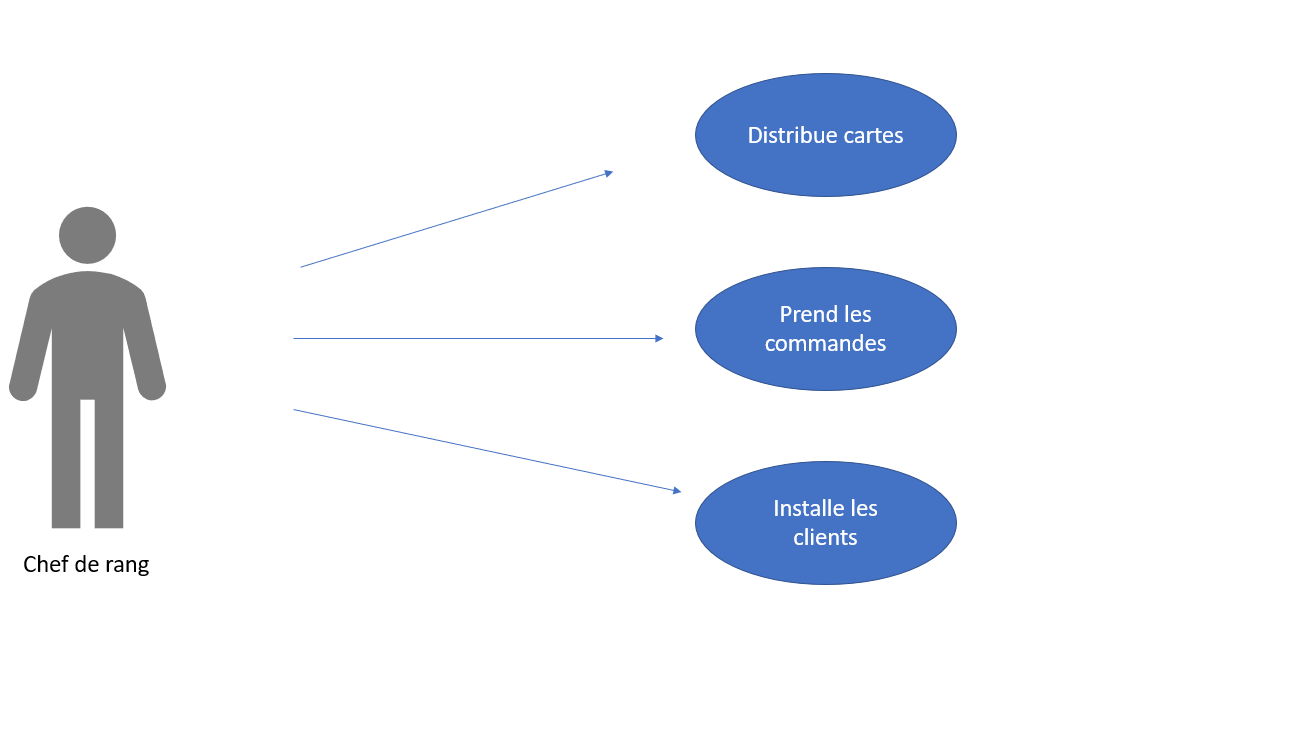


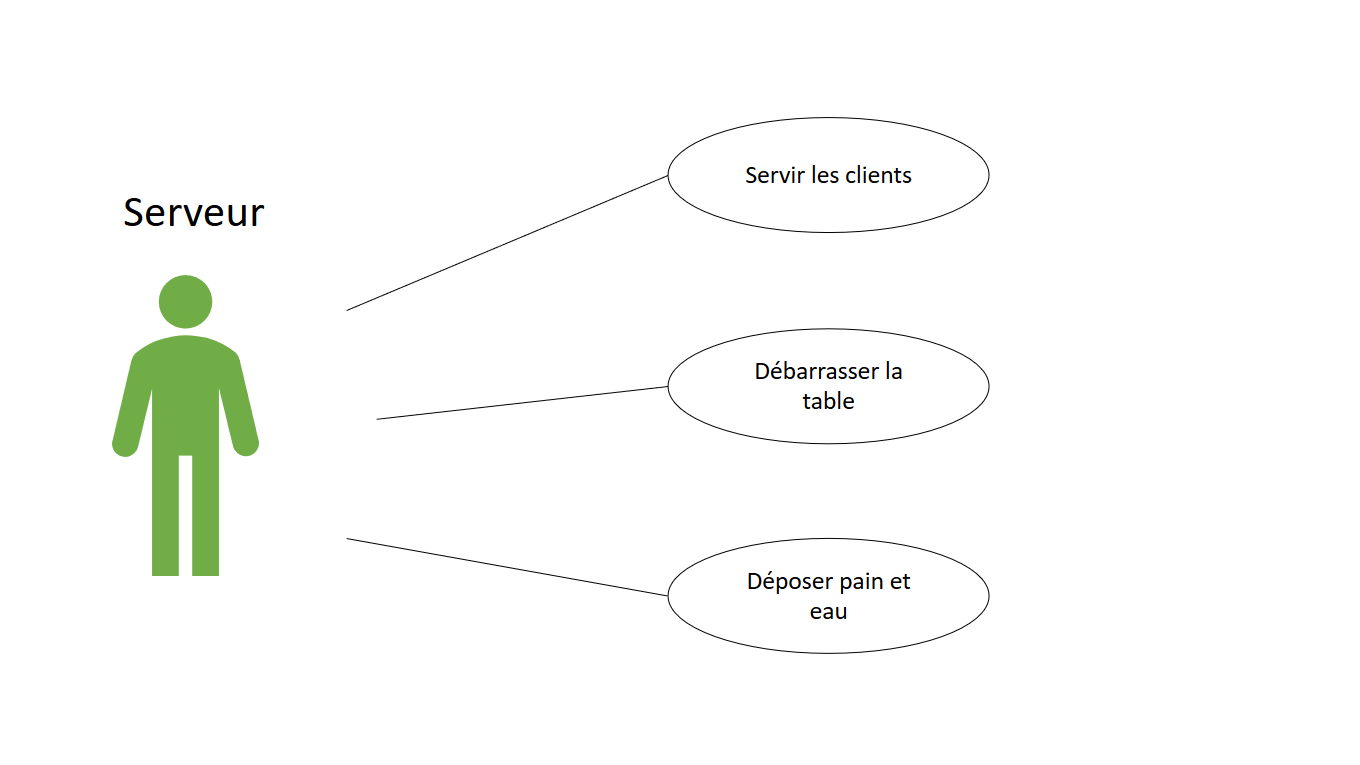


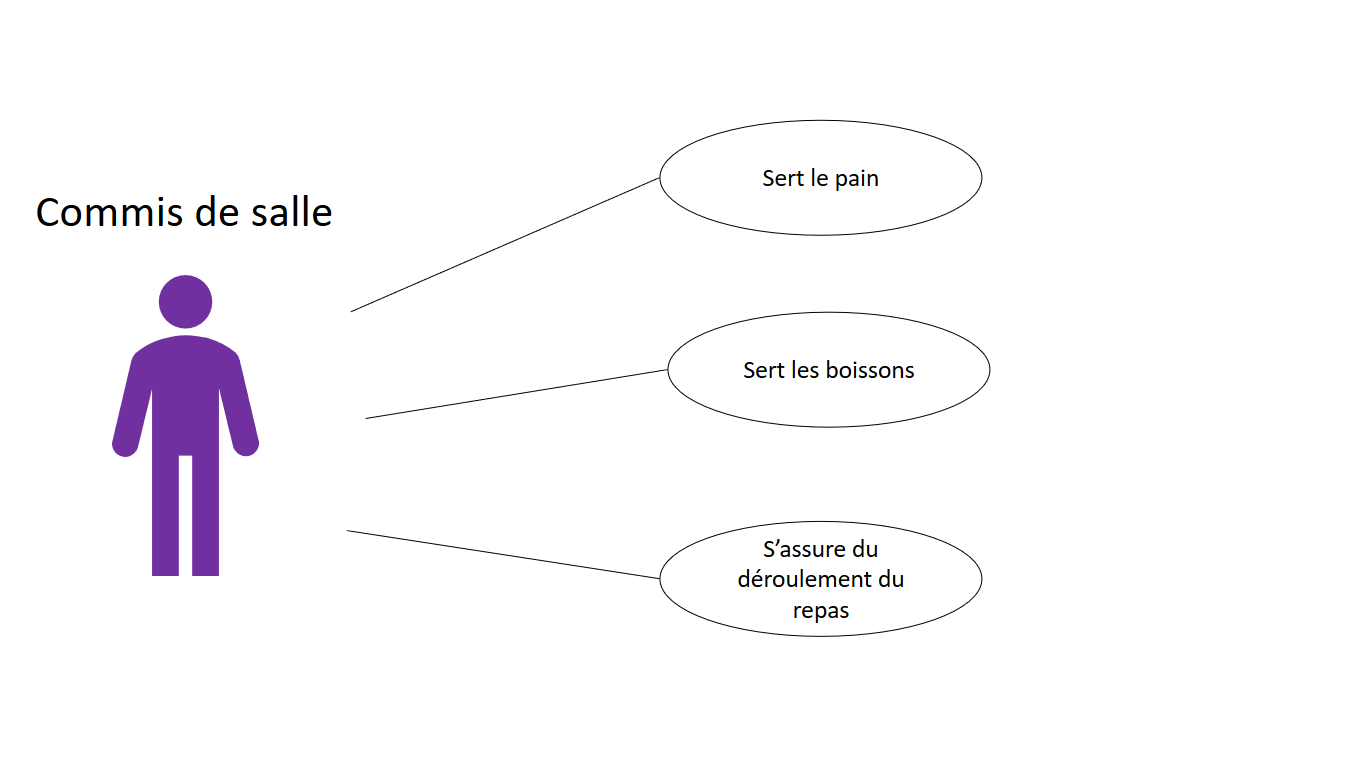




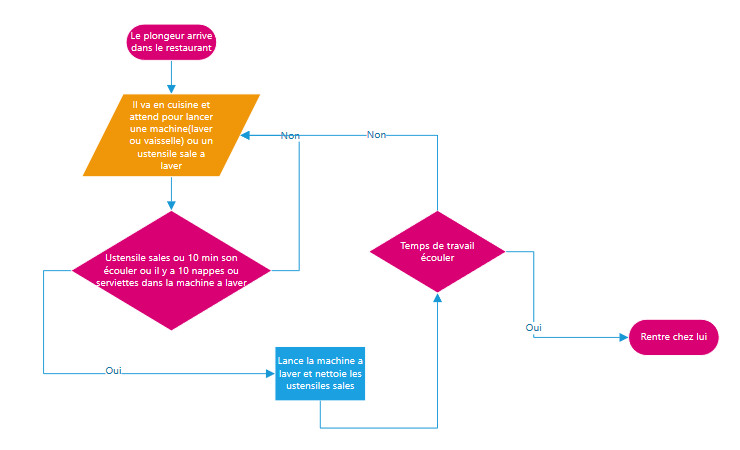




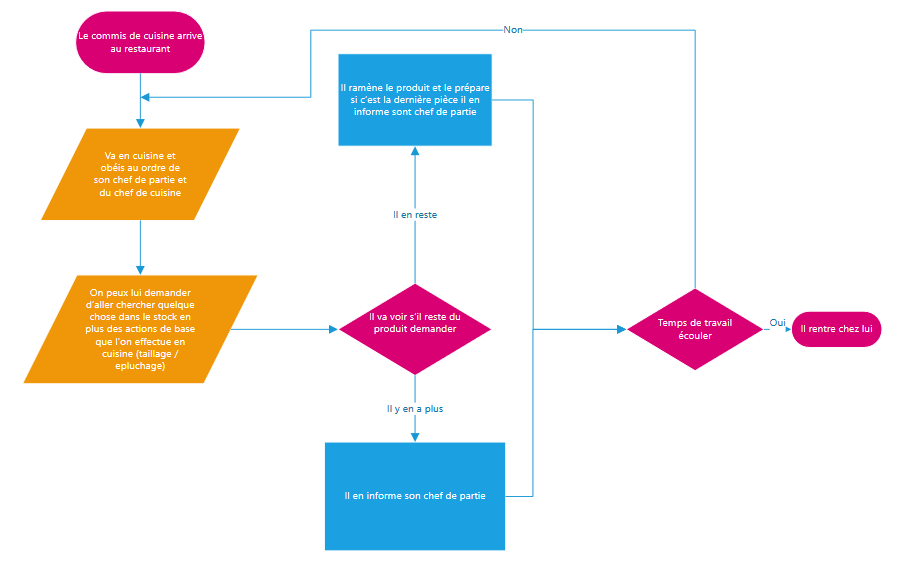




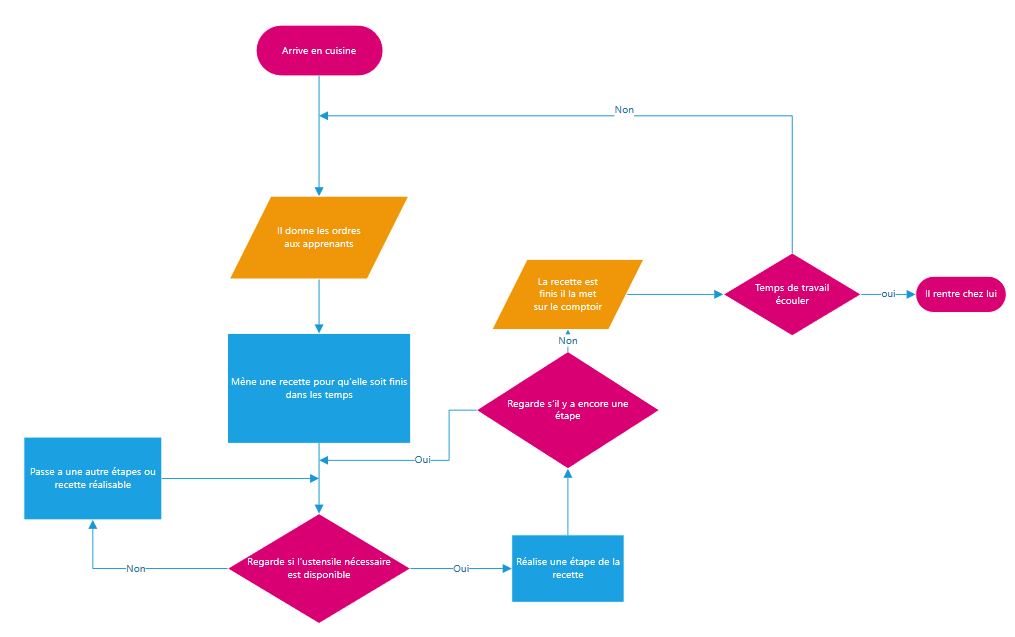
## Diagrammes d’activité :



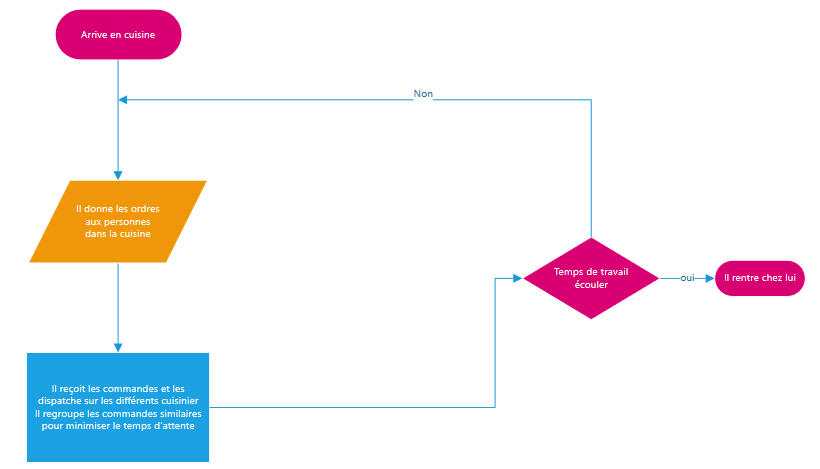
**Plongeur**



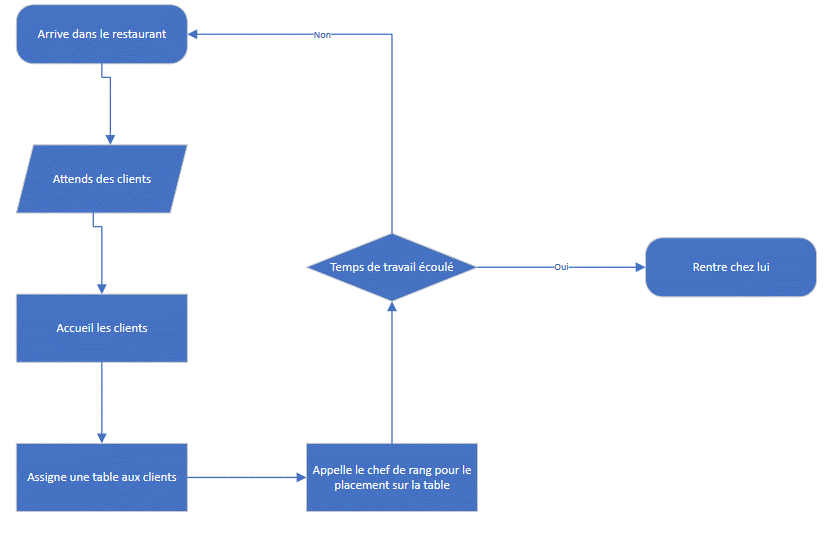
**Commis de cuisine**



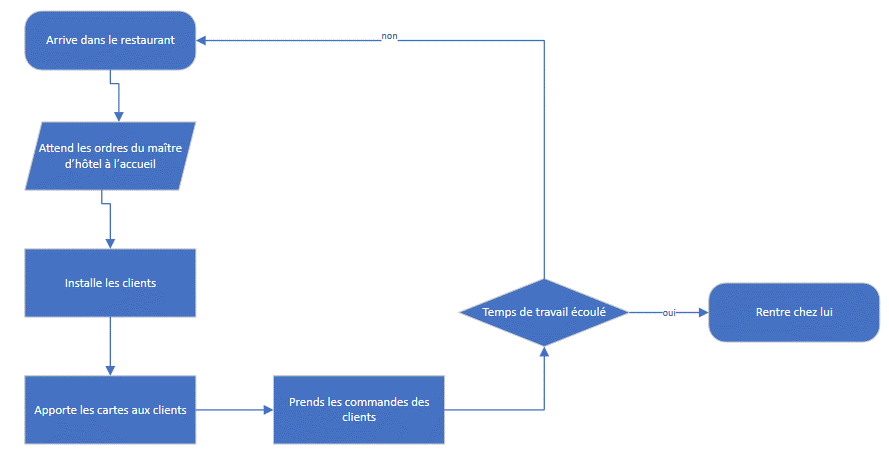
**Chef de partie**



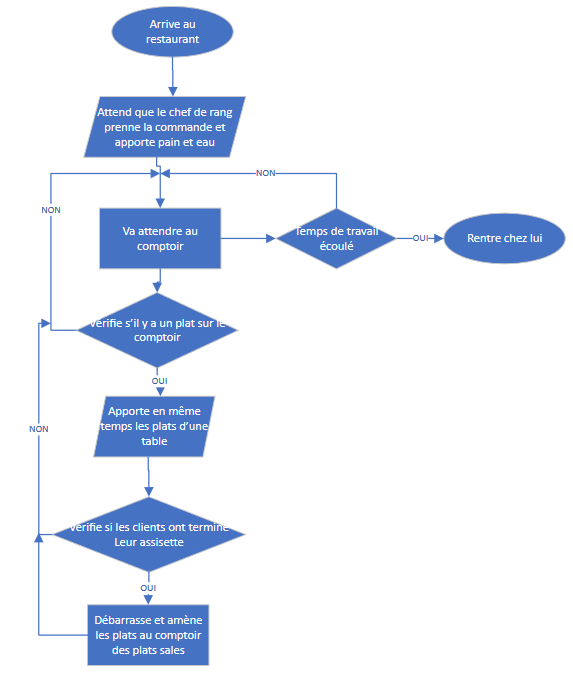
**Chef de cuisine**



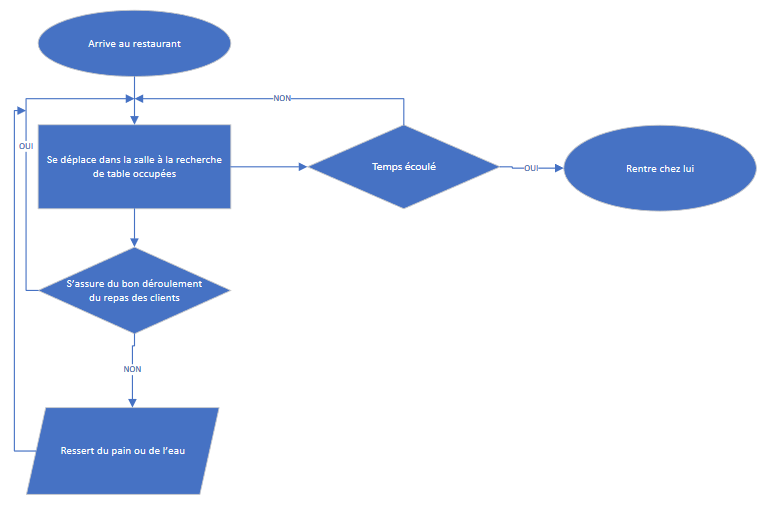
**Maître d’hôtel**



**Chef de rang**

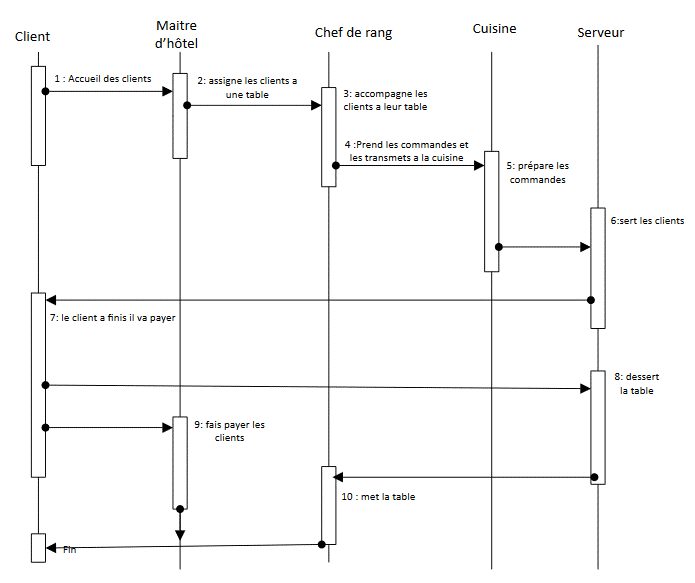


**Serveur**

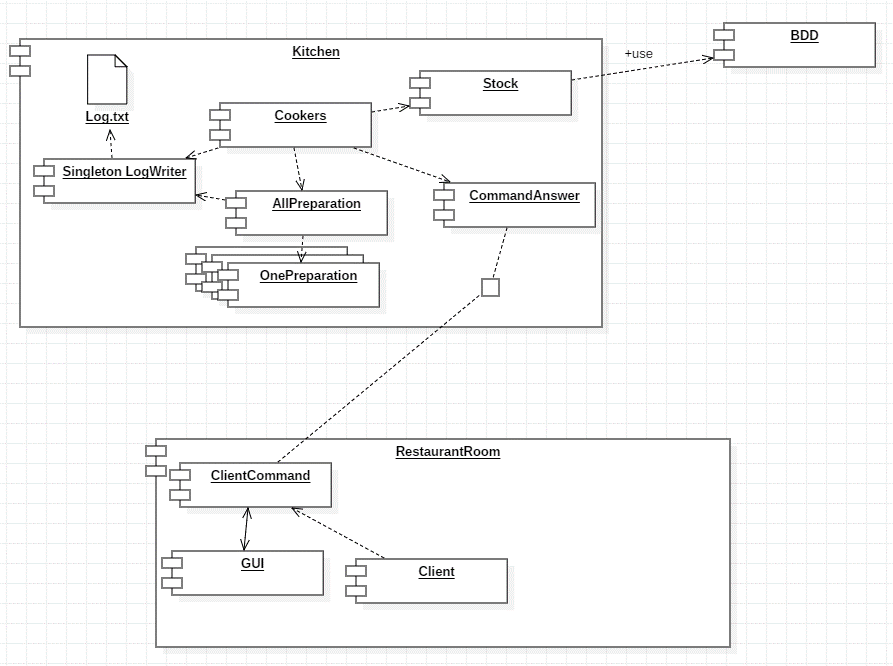


**Commis de salle**

## Diagramme de séquence :

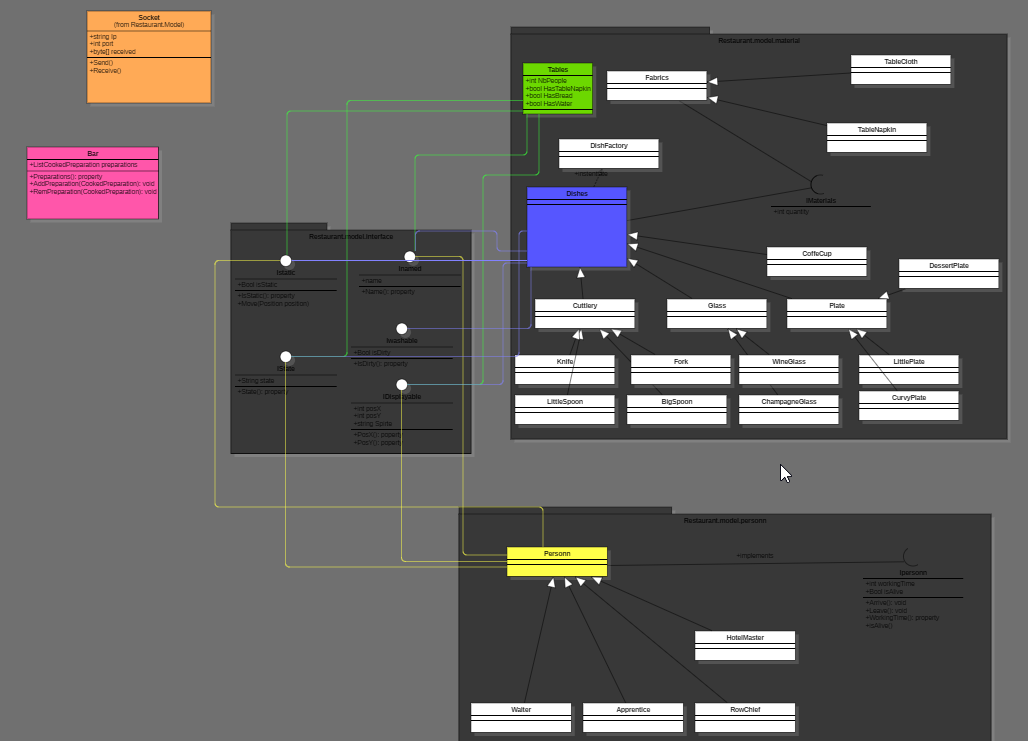


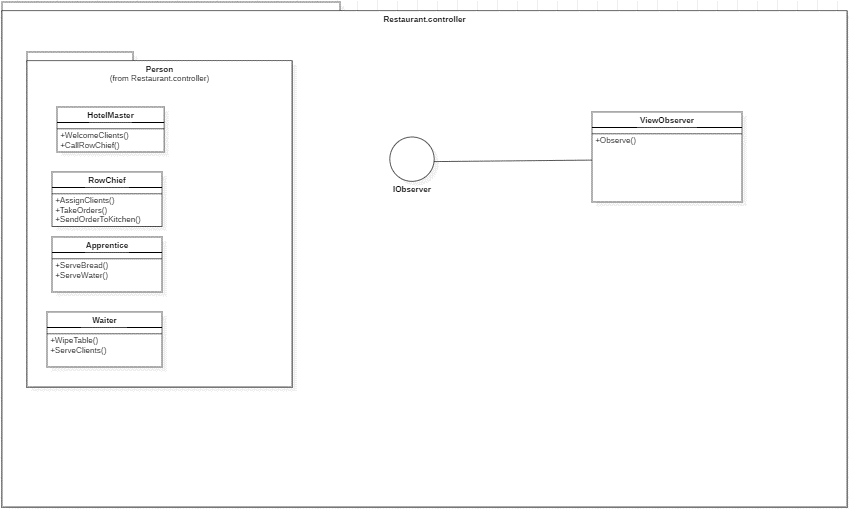
## Diagramme de composants :

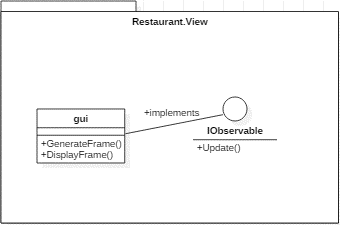


## Diagramme de classe :

### Salle de restaurant :







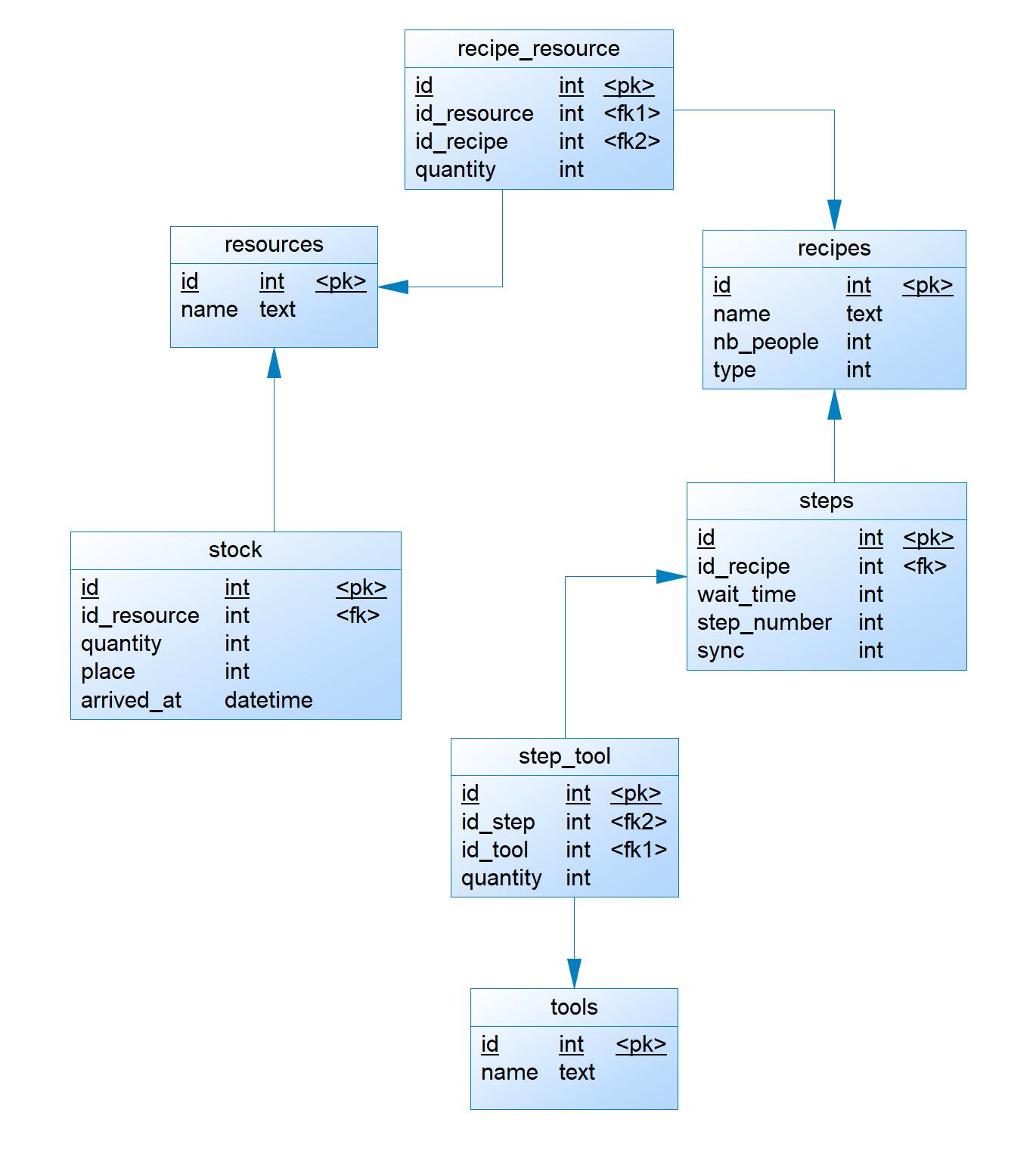
### Cuisine :

### https://cdn.discordapp.com/attachments/519069126195412994/520191205598822400/unknown.png

### https://cdn.discordapp.com/attachments/519069126195412994/520191373249609728/unknown.png

### https://cdn.discordapp.com/attachments/519069126195412994/520191926918447104/unknown.png

## MCD :



# Explications des Design Pattern

## MVC :

Modèle-Vue-Contrôleur ou bien MVC est une architecture logicielle adaptée aux interfaces graphiques. Il est composé de trois types de modules ayant trois responsabilités différentes :

* Le modèle détermine les données que l’application doit contenir. Si l’état de ces données change, alors le modèle va avertir la vue et l’affichage peut changer au besoin. Il peut aussi avertir le contrôleur si une logique différente est nécessaire afin de contrôler la modification de la vue.
* La vue va déterminer comment les données vont être affichées. Elle va recevoir les données du modèle pour les afficher.
* La contrôleur va contenir l’ensemble de la logique de mise à jour du modèle et de la vue en fonction des actions de l’utilisateurs sur l’application.

Dans notre cas de design Pattern MVC est utilisé pour le carde structurel de notre application, il permet de séparer les différents composants. Il permet d’avoir une structure bien définie dès le départ ainsi que de prévoir l’évolutivité du programme.

## Factory :

C’est un patron de conception qui définit une interface pour la création d’un objet en déléguant à ses sous-classes le choix des classes à instancier. Le pattern factory permet l'instanciation d'objets non définis dans une classe concrète à partir d'une méthode d'instanciation issue d'une classe abstraite.

Dans notre cas la hiérarchie de classes parallèles nous permet d’instancier des classes de l’autre. L’un des principaux avantages de Factory est la possibilité d’utiliser des noms descriptifs précis.

## Bridge :

Le bridge est un design pattern permettant de découpler l’interface d’une classe et son implémentation.

Dans notre cas, comme nous utilisons des interfaces, le design pattern bridge nous permet de découpler l’abstraction d’un concept de son implémentation et permet à l’abstraction et l’implémentation de varier indépendamment.

## Singleton :

L’objectif de ce patron de conception est de restreindre l’instanciation d’une classe à un seul objet. Il est utilisé dans le cas où nous avons besoin d’un objet pour coordonner des opérations systèmes. Il peut être utilisé lorsque nous avons des objets similaires. Ce qui est le cas avec notre projet.

Cependant nous devons l’utiliser avec précaution : en effet si deux threads sont exécutés en même temps, il faut vérifier qu’un seul est créé.

Pour remédier à ce problème nous pouvons utiliser l’exclusion mutuelle pour indiquer que l’objet est en cours d’instanciation.

## Observer :

Le design pattern observer est utilisé pour envoyer un signal à des modules qui jouent le rôle d’observateur. Dans le cas d’une notification les observateurs effectuent l’action voulue en fonction des informations fournies.

Dans notre cas d’utilisation, certains évènements devront être exécutés simultanément. Il permet de limiter le couplage entre les modules aux seuls phénomènes à observer.

## Iterator :

Le design pattern Iterator est un des plus commun dans les modèle de type comportementaux. L’idée derrière Iterator est de limiter la vision d’une collection par un utilisateur. L’interface Iterator permet à l’utilisateur de ne pas être concerné par la manière de gérer les objets dans un tableau, une liste etc. Dans notre cas, Iterator est utilisé avec des objets List<>.