Dossier Architecture

**Groupe 1 :** Florentin CHÉRAUD, Victor GOURBILIÈRE, Tanguy MOUNIER, Vincent PASQUET, Amaury VINCENT

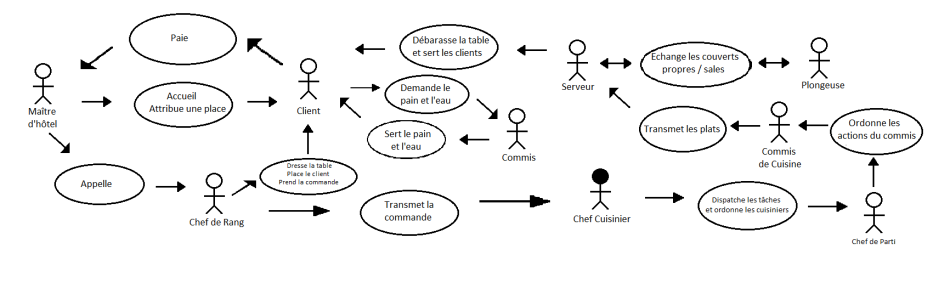
Projet programmation système

# Dossier GIT

<https://github.com/Amaknes/SystemProgTeam1>

L’ensemble des images présenté ci-dessous sont consultable dans le répertoire GIT.

# Diagramme cas d’utilisation



# Diagramme d’activités

Voici les diagrammes d’activités pour les différents acteurs dans le projet :

Serveur

Chef de Rang

Maître d’Hôtel

Commis de Salle

Client

Chef Cuisinier

Chef Patissier

Plongeur

Commis de Cuisine

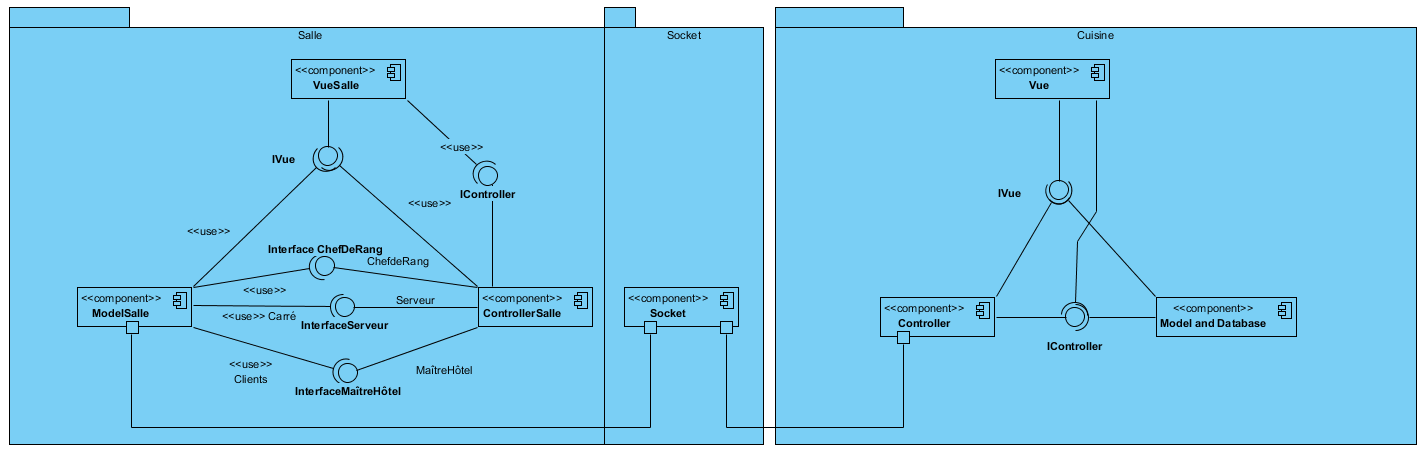
Cuisinier

# Diagramme de classes

Le diagramme de classe est mis dans le répertoire.

# Diagramme de composants

Le diagramme de composants pour le projet est le suivant :



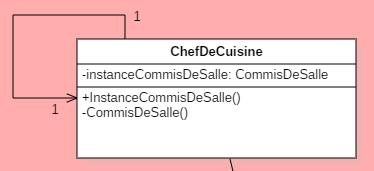
# Design Pattern

## MVC

Le design Pattern MVC est un design pattern visant à couper en 3 un logiciel, avec une partie vue qui sert d’interface, d’une partie controller qui sert à stocker la logique et d’une partie Model qui sert pour accéder aux données, le pattern est visible sur notre diagramme de Classe, avec les différents packages.

## Singleton

Le singleton est un design pattern dont l'objectif est de restreindre l'instanciation d'une classe à un seul objet. Il est utilisé lorsqu'on a besoin exactement d'un objet dans le système. Le constructeur est privé pour n'être accédé que dans la commande GetInstance, lorsqu'on accède à celle-ci, soit la fonction renvoie l'instance existante de l'Objet mis en variable statique, soit-il en créé une et la place dans la variable. De ce fait un seul objet est instancié et peut être appelé partout. Ici nous l’utiliserons pour le chef de cuisine, unique et qui a parfois besoin d’être récupéré à n’importe quel moment en cuisine par les autres cuisiniers.

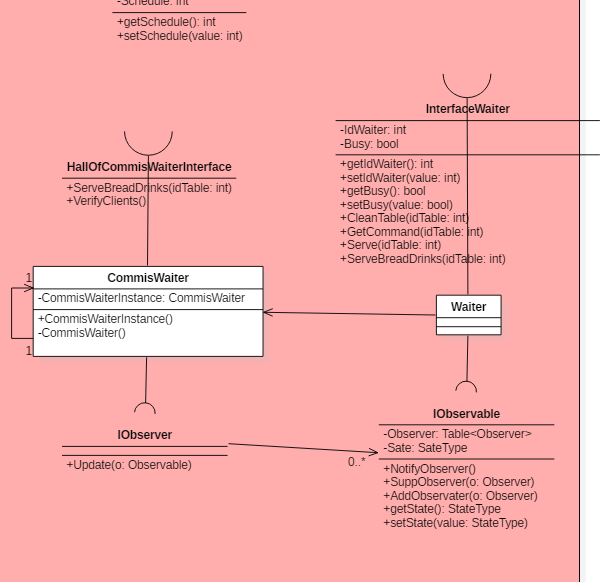


## Observer

Le cahier des charges du projet demande au commis de réagir lorsqu’un client souhaite lui parler ainsi que lorsqu’un serveur est trop occupé pour exécuter certaines tâches.

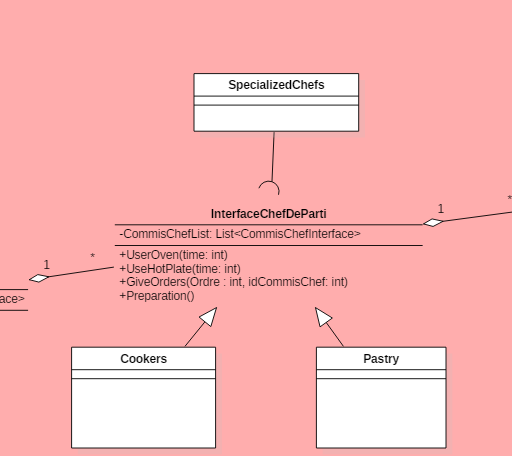
Nous avons donc décider d’appliquer le Design Pattern Observer aux couples Commis de Salle / Serveur et Commis de Salle / Client.

Le Commis de Salle implémente alors l’interface Observer tandis que les Serveur et Clients implémentent l’interface Observable.



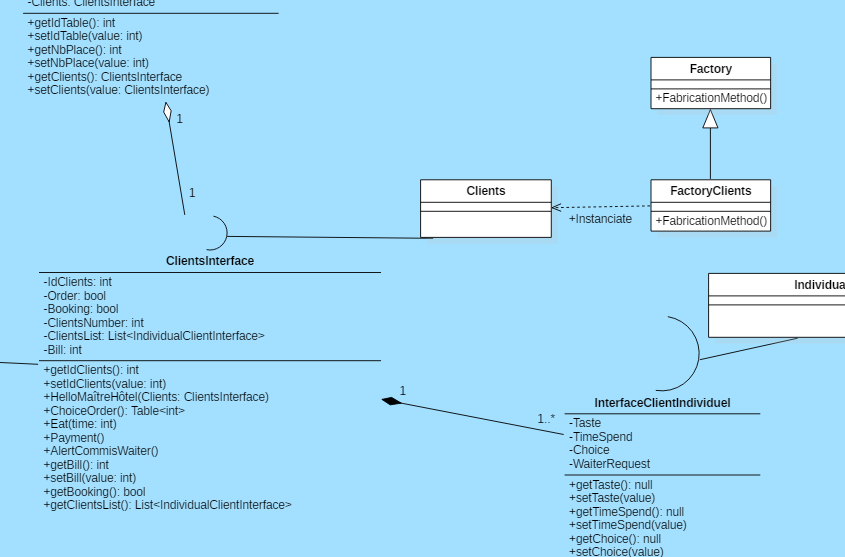
## Strategy

Le Design Pattern Strategy est un design pattern permettant de choisir une stratégie en fonction d’une situation donnée. Nous l’utilisons pour déterminer quel rôle aura les chefs de partie.



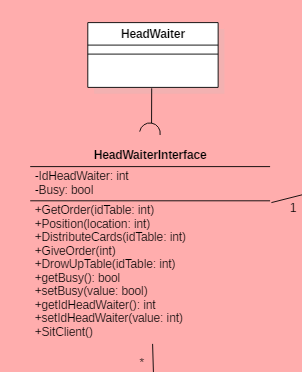
## FACTORY

Le Design pattern Factory est un design Pattern permettant d’avoir une factory de différents objets, dans le projet nous l’utilisons pour créer plusieurs clients aux comportement différents et aléatoires

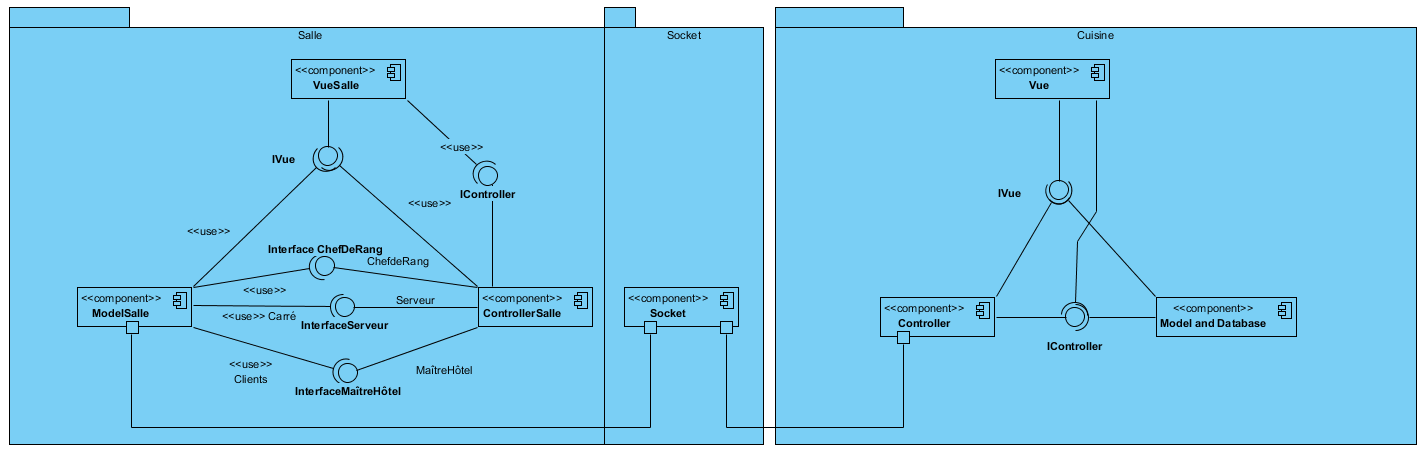


## BRIDGE

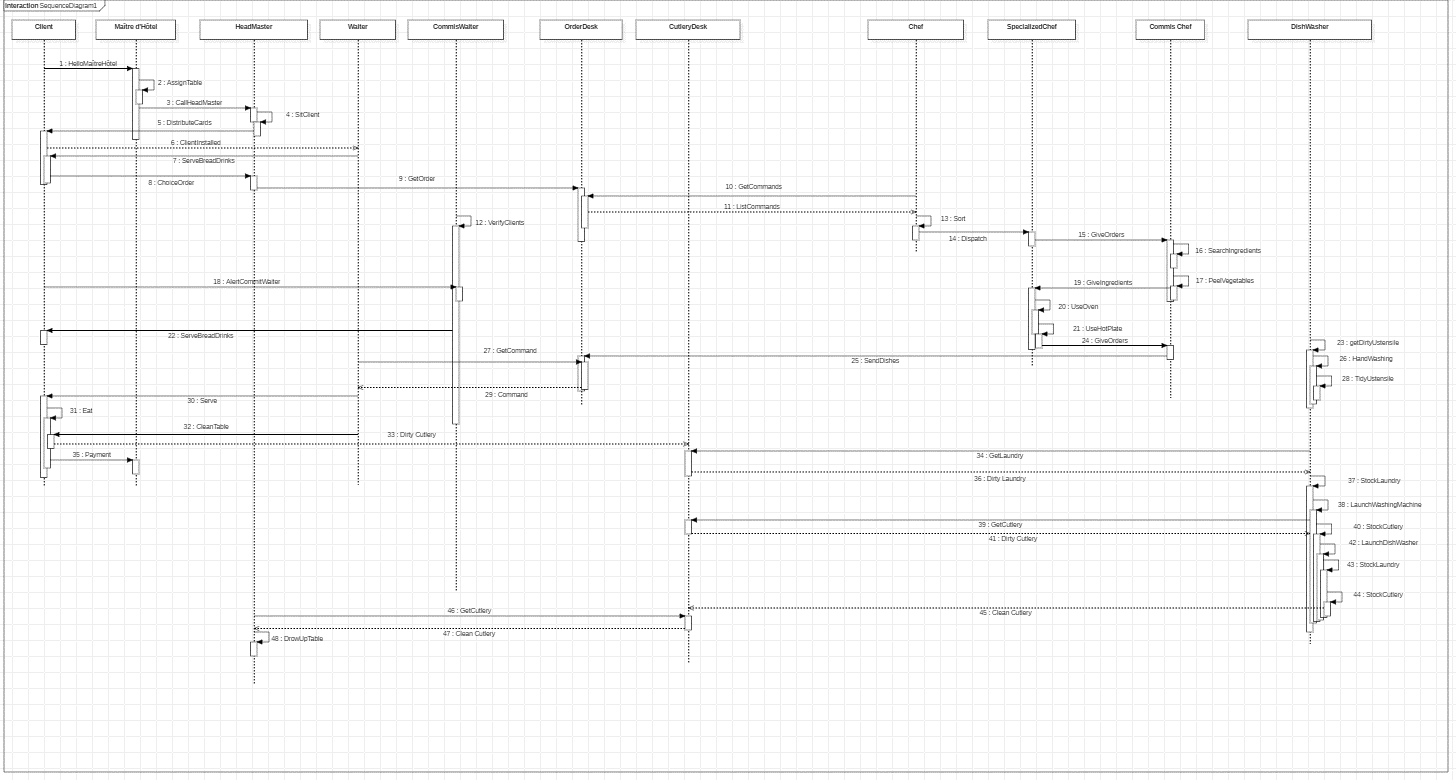
Le Design Pattern Bridge est un design pattern permettant de découpler une classe de son implémentation. Elle permet à la classe d’être donc modulable sans changer son code de base, en changeant l’implémentation pour s’adapter à la situation. Ici nous l’avons utilisé sur la plupart de nos classes, dans le but de pouvoir changer la classe en cas de problème tout en gardant les méthodes de l’interface.



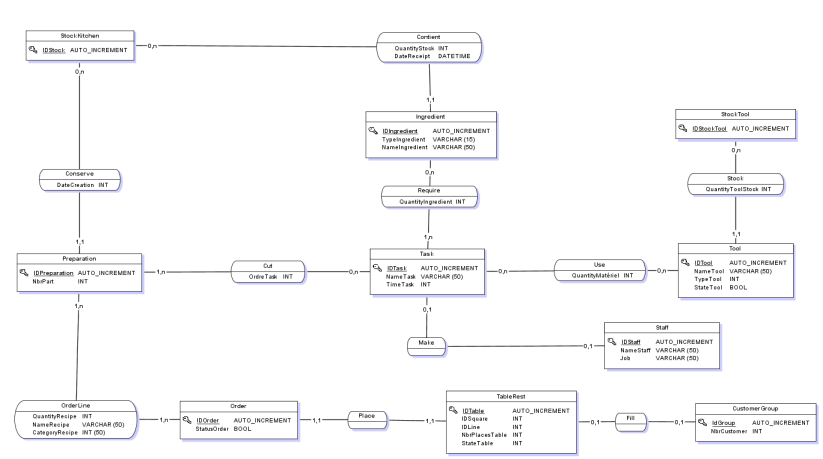
# Diagramme de composants



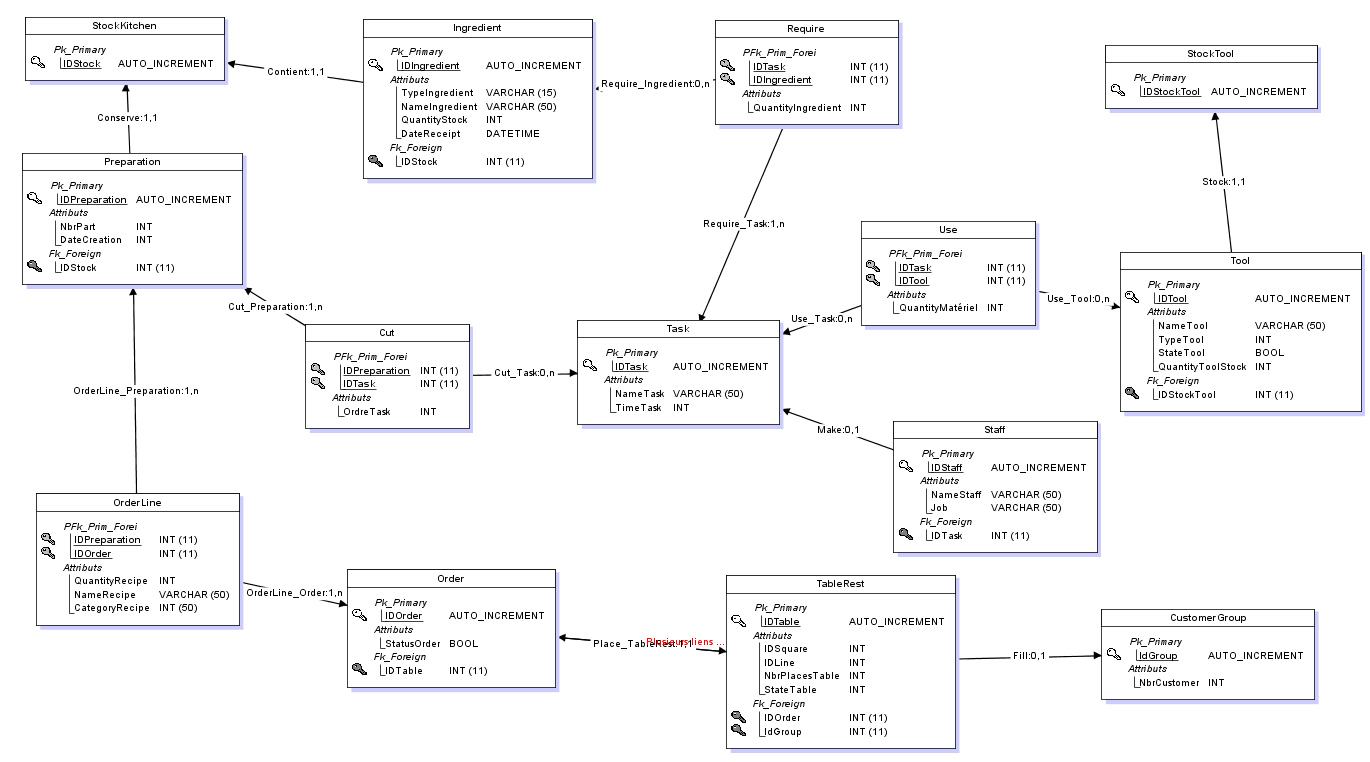
# Diagramme de séquence



# MCD



# MLD



# Script SQL

Le script SQL peut être généré grâce au fichier du MCD présent dans le répertoire.