Rapport de Design Pattern

I. Introduction :

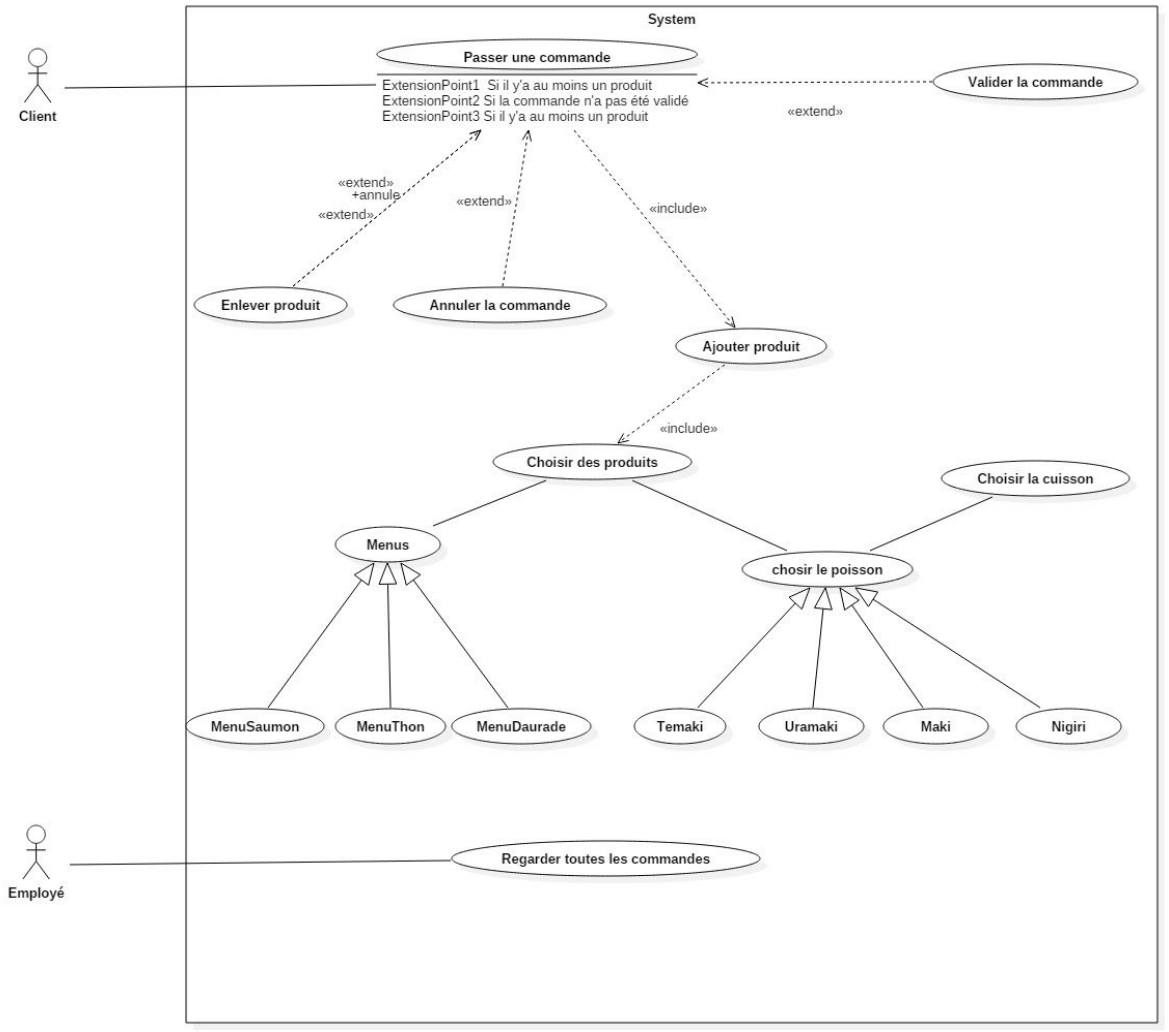
Le but était de réaliser pour notre client un logiciel qui permettrait de gérer les commandes pour son restaurant asiatique. Il fallait répondre aux besoins de notre client qui étaient bien précis.

Les fonctionnalités souhaitées par notre client sont les suivantes :

- L'utilisateur peut effectuer une commande
- Lorsqu'il effectue sa commande il a plusieurs choix :
    - Ajouter un produit
    - Enlever le dernier produit
    - Valider la commande
    - Annuler la commande
- Chaque produit possède un prix, un temps de préparation et une déclinaison possible
    - Nigiri
    - Maki
    - Temaki
    - Uramaki
- L'utilisateur peut commander des menus s'il le souhaite
- Le restaurant souhaite voir toutes les commandes

Voici un court extrait de notre discussion avec le client pour que vous puissiez comprendre pourquoi on a utilisé certain design pattern tel que le singleton:

Nous avons demandé à notre client :
"Comptez vous étendre votre nombre de restaurant car si ce n'est pas le cas nous pouvons faciliter notre développement en utilisant ce que l'on appelle un singleton mais cela implique de ne plus pouvoir étendre cette application à d'autre restaurant, signé les dev"
et il nous a répondu :
"Salut les Cocos, nous n'avons pas prévu d'avoir d'autre restaurant, vous pouvez donc utiliser ce que vous appelez un singleton mais dépêchez vous nous souhaitons breveté cette application le plus tôt possible, signé le client"



Voici notre diagramme de cas d'utilisation pour mieux illustrer notre architecture :

II. Implémentation des patterns :

- ➢ Comme nous l'avons dit en introduction nous avons utilisé un singleton qui permet de créer un objet tout en s'assurant qu'un seul objet est bien créé pour le restaurant (l'objet est le restaurant).
  Avec ce pattern nous n'avons qu'un seul restaurant.
- ➢ Nous avons un pattern factory (SushiCreator) pour créer une "fabrique de sushi" qui sera utile pour toute nos déclinaison avec nos différents produits (saumon, thon et daurade).

- ➢ Nous avons un pattern state (IPoisson) pour savoir si l'état du poisson est cru ou cuit. En fonction de l'état du poisson le comportement des classes changent. par exemple le prix et le temps de préparation sera modifié.
- ➢ Nous avons utilisé le pattern composite (IMenu) pour la création des 3 menus. Pour la composition des menus c'est pratique car nous avons plusieurs objets menus différents qui dépendent de notre interface IMenu.
- ➢ Le pattern iterator (Restaurant) permet de récupérer toutes les commandes validés du restaurant.