**Document de justification des choix des Design Patterns**

Dans une optique d’amélioration du développement de notre application, il parait évident qu’il faille utiliser les **Design Patterns**. Les design patterns que nous avons jugés les meilleurs pour nous dans cet exercice sont ceux qui suivent :

**Design Pattern State** **:** permet à un objet de modifier son comportement lorsque son état interne change.

Il nous sera utile ici pour modéliser certains personnages comme **les commis** qui lorsqu’il n’y a pas de client sont dans un coin mais dès qu’il manque quelque chose sur la table d’un client, ils doivent venir résoudre le problème.

**Design Pattern Observer** **:** définit une dépendance un-à-plusieurs entre les objets de sorte que lorsqu'un objet change d'état, tous ses objets dépendants sont notifiés et mis à jour automatiquement.

Il nous sera utile dans notre application. Comme exemple, nous avons lorsque le client passe la commande toutes les personnes à la cuisine commencent à servir (ou ranger) les plats des clients.

**Design Pattern Command :** Il encapsule une demande en tant qu'objet, vous permettant ainsi de paramétrer les clients avec différentes demandes, demandes de file d'attente ou de journalisation, et de prendre en charge les opérations annulables.

Il nous sera utile ici pour modéliser les commandes des clients.

**Design Pattern Strategy :** définit une famille d'algorithmes, encapsule chacun d'eux et les rend interchangeables. Ce modèle permet à l'algorithme de varier indépendamment des clients qui l'utilisent.

Il nous sera utile ici pour représenter les serveurs qui doivent s’adapter aux clients

**Design Pattern Facade :** fournit une interface unifiée à un ensemble d'interfaces dans un sous-système. Ce modèle définit une interface de niveau supérieur qui facilite l'utilisation du sous-système.

Il nous sera utile ici car on va juste présenter les plats au clients mais sans toutefois montrer toutes les étapes de leur préparation.