Programmation Système

**CESI**

**Ecole d’ingénieurs**

**1ère Cycle ingénieur**

**Projet Programmation Système**

**2018-2019**

Abdelmoumene Sabrina (Chef de projet)

Hasnaoui Mounir

Akbi Adel

Tachtioui Abdellilah

Table des matières

[Contexte 3](#_Toc531784628)

[Besoins et contraintes 3](#_Toc531784629)

[Ce que nous devons faire 4](#_Toc531784630)

[Choix des 5 Designs Patterns 4](#_Toc531784631)

[Pourquoi Factory ? 4](#_Toc531784632)

[Pourquoi Observer ? 4](#_Toc531784633)

[Pourquoi Singleton ? 5](#_Toc531784634)

[Pourquoi Builder ? 5](#_Toc531784635)

[Pourquoi Strategy ? 5](#_Toc531784636)

[Pourquoi Adapter ? 6](#_Toc531784637)

[Pourquoi MVC ? 6](#_Toc531784638)

Projet Programmation Système

# Contexte

Une grande chaîne de restaurants souhaite s’équiper d’une nouvelle application informatique pour améliorer ses performances (remplissage des salles, accueil des clients, gestion des réservations et travail en cuisine).

Dans ce projet, nous avons la qualification de devoir créer une application interactive en utilisant le Microsoft .NET (Visual Studio) pour pouvoir réorganiser l’activité au sein d’un restaurant c’est-à-dire réorganiser afin de pouvoir minimiser la perte de temps, et ainsi minimiser la perte de client.

## Besoins et contraintes

* Désorganisation entre les deux systèmes principaux qui sont la salle et la cuisine (manque de communication et de méthodologie des postes au sein de la salle et de la cuisine)
* Attente trop longue de la part des clients, ce qui créer une créer un goulot d’étranglement et les employés sont dépassés par la situation.
* Manque de matériel propre (Mauvaise gestion de la partie de stockage ainsi que de la partie de la plonge en cuisine)
* Client insatisfait (Au fur et à mesure de l’attente, le client s’énerve et ne peut contenir sa colère et part ce qui conduit à une baisse de la réputation du restaurant.

## Ce que nous devons faire

Le directeur du restaurant a fait appel à nous, donc nous allons devoir nous organiser en tant que tel afin de répondre au mieux à ses besoins.

Nous devons d’abord dans un premier temps détecter les problèmes liés à la perte de temps, et la désorganisation totale dans le restaurant. Ensuite nous devrons utiliser la base de données pour avoir accès à tous les types d’objets que l’on a en notre possession. Ensuite commencer à penser à une solution utile qui sera exprimée au finale en gain, pourcentage, argent et temps.

En vue de correctement faire notre travail, nous devons suivre une approche logique et ordonnée connue des développeurs, nous devons d’abord modéliser la solution, de ce fait nous devons passer par plusieurs diagrammes de modélisations.

Dans un premier temps nous faisons le diagramme de cas d’utilisation qui est le plus simple et qui va nous permettre de savoir à QUOI sert le système tout en organisant les interactions possibles avec les acteurs.

Ensuite il sera de près suivi par le diagramme d’activité qui va nous simplifier nos visions des différentes conditions et des différentes fonctions.

Quand on finit ces deux-là nous pourrons faire le diagramme de classe qui lui nous montre la structure et les liens entre les objets dont le système est composé, ainsi on spécifiera QUI sera à l’œuvre dans le système pour réaliser les fonctionnalités décrites par les diagrammes de cas d’utilisation.

Enfin on arrivera au diagramme de séquences permettant de décrire COMMENT les éléments du système interagissent entre eux et avec les acteurs.

## Choix des 5 Designs Patterns

Afin de suivre une méthodologie appropriée et correcte pour notre façon de procéder à l’exécution de ce qu’il nous a été demandé d’effectuer, nous devons à tout prix choisir 5 Design Patterns afin d’appliquer cela donc voici nos choix :

* Abstract Factory
* Observer
* Singleton
* Builder
* Strategy
* Adapter
* MVC

### Pourquoi Factory ?

Il faut savoir que c’est un patron de création

Pour le premier qui est Factory nous l’avons choisi car il permettra à notre projet de suivre de plus près les principes S.O.L.I.D et en particulier les principes de ségrégation d’interface et d’inversion de dépendance.

Il nous permet également de créer un objet sans pour autant savoir quel type d’objet est créé.

Il permet aussi une conception plus découplée et donc plus testable, il nous permet d’introduire facilement un conteneur Inversion of Control, plus de flexibilité lorsqu’on voudra changer d’application (c’est-à-dire créer de nouvelles implémentations sans changer le code dépendant).

En résumé, le pattern Abstract Factory va nous permettre d’instancier des familles de produits dépendant les uns des autres sans qu’il soit nécessaire de préciser leur type concret.

### Pourquoi Observer ?

Ici il faut savoir que l’objet crée est appelé sujet, tiendra à jour une liste de ses dépendants qui sont appelés observateurs et les informera de tout changement d’état généralement en appelant l’une de leurs méthodes.

Dans notre cas à nous nous avons pas mal d’objets voir des tas, donc nous devons avoir un recensement de l’état de tous ces objets, et qu’ils soient à jours au fur et à mesure et c’est là que Observer représente un atout majeur car si l’état d’un objet change, les objets dépendants vont être mis à jour automatiquement, et aussi un objet pourra notifier un nombre illimité d’objets.

### Pourquoi Singleton ?

Pour notre application, on aura parfois l’obligation de créer une classe qui sera utilisé plusieurs fois au sein de cette même application, mais qui ne pourra être instanciée qu’une seule fois.

On peut effectivement l’instancier au début dans une variable globale mais ça ne reste pas une bonne solution car on enfreindra les principes d’encapsulations et rien ne nous garantit qu’on n’instancie pas la classe une deuxième fois.

Donc pour cela on utilisera le Design Patterns Singleton qui lui va nous permettre de rajouter une sorte de contrôle sur le nombre d’instances que peut retourner la classe, et cela en empêchant les développeurs d’utiliser les constructeur de la classe pour l’instancier après cela on va déclarer une méthode statique qui va retourner l’objet correspondant au type de la classe cela sera comme un pseudo constructeur donc le fait qu’elle soit statique elle va nous permettre de l’appeler sans posséder d’instance de la classe.

### Pourquoi Builder ?

Généralement quand on a besoin de construire un objet propre et fait étape par étape on utilise ce Design Pattern en question.

Dans notre application il y a des objets que l’on devra créer qui seront très complexe donc on devra séparer la construction de l’objet de sa représentation comme sa le même processus de construction pourra créer différentes représentations (en général ce n’est pas en un seul appel de fonction que l’on préparera un repas entiers avec l’entrée le plat et le dessert). Builder peut également utiliser Factory et Builder dans leurs implémentations c’est-à-dire (Builder se concentre sur la construction d’un objet complexe, étape par étape ensuite Factory va mettre l’accent sur une famille d’objets de produits et enfin le Builder renvoie le produit en tant qu’étape finale.

Quand je parle d’objets je parle de constructions de repas c’est-à-dire entrée, plat et dessert, donc dans ce cas précis le processus de construction reste le même car au final le chef de cuisine va recevoir la recette en question et ensuite donnera les plats à préparer aux cuisiniers, le cuisinier va ensuite s’exécuter et ensuite dresser les assiettes et mettre les plats dedans.

### Pourquoi Strategy ?

IL nous aide à la conception.

Ce Pattern va nous permettre à nous et notre application de sélectionner de manière dynamique une méthode appropriée au moment de l’exécution, en fonction du contexte de son hôte.

Donc en résumer il nous permet de modifier un algorithme de façon transparente, car nous avons des éléments à trier selon des méthodes différentes et donc lui va nous expliquer le mécanisme rendant le choix de l’algorithme transparent pour l’objet qu’on va utiliser.

### Pourquoi Adapter ?

C’est un Pattern de structure, il permet d’avoir les providers de ADO.NET comme SqlConnection et OleDbConnection, et chacun d’entre eux est un adaptateur pour la base de données cible. Nous allons l’utiliser car en .NET c’est l’un des meilleure moyen de mettre en connexion notre base de données sous SQL Server et notre application .NET.

### Pourquoi MVC ?

Le modèle vue contrôleur est déjà intégré aux fonctionnalités de .NET et de ASP.NET telles que les pages maîtres et l’authentification basée sur l’appartenance. L’infrastructure MVC est définie dans l’assembly System.Web.Mvc.

Dans ce cas si notre infrastructure est définie comme suit :

* Modèles. Les objets de modèle sont les parties de l'application qui implémentent la logique du domaine de données de l'application. Souvent, ils récupèrent l'état du modèle et le stockent dans une base de données. Par exemple, un objet Product peut récupérer des informations dans une base de données, les exploiter, puis réécrire les informations mises à jour dans une table Products d'une base de données SQL Server.

Dans les petites applications, le modèle est souvent une séparation conceptuelle plutôt qu'une séparation physique. Par exemple, si l'application sert uniquement à lire un groupe de données et à l'envoyer à la vue, elle ne comporte pas de couche de modèle physique, ni de classe associée.Dans ce cas, le groupe de données joue le rôle d'un objet de modèle.

* Vues. Les vues sont les composants qui affichent l'interface utilisateur (IU) de l'application. En général, cette interface utilisateur est créée à partir des données du modèle. Il peut s'agir par exemple d'une vue d'édition d'une table Products affichant des zones de texte, des listes déroulantes et des cases à cocher en fonction de l'état actuel d'un objet Product.
* Contrôleurs. Les contrôleurs sont les composants qui gèrent les interventions de l'utilisateur, exploitent le modèle et finalement sélectionnent une vue permettant de restituer l'interface utilisateur. Dans une application MVC, la vue sert uniquement à afficher les informations ; le contrôleur gère les entrées et interactions de l'utilisateur, et y répond. Par exemple, il gère les valeurs de chaîne de requête et les passe au modèle, qui peut à son tour les utiliser pour interroger la base de données.

Donc en vue de cette explication on en déduit que le MVC nous permet de gérer plus facilement la complexité en décomposant l’application, et aussi il offre une meilleure prise en charge du développement axé sur des tests et nous permet de contrôler pleinement notre application.