

- **Module Client**
 - Affichage liste de clients
 - Ajout de salarie
 - Suppression d'un salarie
 - Modification d'un client
- **Module Salariés**
 - Affichage de l'organigramme
 - Ajout de salarie
 - Suppression d'un salarie
- **Module Commandes**
 - Ajout de commandes
 - Suppression de commande
 - Modification d'une commande
- **Module Produits**
 - Affichage liste de produits
 - Ajout d'un nouveau produit
 - suppression d'un produit
- **Module Véhicules**
 - Affichage liste de Véhicules
 - Ajout d'un nouveau véhicule
 - suppression d'un produit
- **Module statistique**
 - Les fonctionnalités de ce module sont contenues dans les 4 premiers.
 - Moyenne achats clients/ Moyenne prix total commandes/Liste de commande par client/Nombre commandes par chauffeurs.

Architecture:

Le projet a été conçu via un "pseudo" MVC (Modèle-Vue-Contrôleur), qui est un motif d'architecture logicielle destiné aux interfaces graphiques lancées. "Pseudo" car je n'ai pas utilisé de réelle interface graphique par contrainte de temps, mais la console.

Le modèle Permet de construire les données à afficher, dans mon projet il y a 6 différents modèles:

- CommandeModel: Dédié à la création de l'objet commande.
- PersonneModel: Dédié à la création d'objet Salarié et d'objet Client.
- PosteModel: Dédié à la création d'objet Poste.
- DépartementModel: Dédié à la création d'objet Département.
- ProductModel: Dédié à la création d'objet Produit.
- VéhiculeModel: Dédié à la création d'objet Véhicule.

Le contrôleur est le composant qui gère les interactions entre l'utilisateur et l'application. J'ai dans mon projet créé une classe "TransconnectContrôleur", qui est un objet regroupant toutes les fonctionnalités des modules (ajout, modification, etc...).

La vue est responsable de l'affichage de l'interface utilisateur (UI) de l'application. Ma vue est gérée directement depuis le main.

Utilitaires

Certains algorithmes, type de fichier ou bien bibliothèques C Sharp ont été nécessaires pour l'implémentation de fonctionnalités, de tests unitaires, ou bien l'initialisation d'objets.

Json:

Il nous était demandé de trouver une solution pour pouvoir charger salariés/clients à l'aide de fichiers. J'ai donc opté pour la sérialisation via les fichiers Json (format de données léger qui est utilisé pour stocker et échanger des données structurées). Pour ce faire, j'ai utilisé une bibliothèque C Sharp appelée **Newtonsoft**, fournissant de nombreuses méthodes permettant la sérialisation et la désérialisation d'objets sous json. Ce fut assez simple de gérer les fichiers Json, mais j'ai pu rencontrer des problèmes de désérialisation pour les classes héritées. Tout ceci a été géré via ma classe utilitaire **JsonUtil.cs**.

CSV:

Je suis resté sur le type de fichier fourni par notre professeur, c'est-à-dire CSV (fichier au format tableau), pour la gestion des chemins entre villes. Pour ce faire, j'ai utilisé une bibliothèque nommée **CsvHelper**, qui m'a permis de récupérer les données, et de créer des objets **PathCity**. Un **PathCity** contient: une ville de départ, une ville d'arrivée et une distance (la distance entre les deux villes). La récupération de ces données a été gérée par ma classe/objet **PathCityWriter**. Dans cette classe, plusieurs attributs bénéfiques à mon algorithme de Dijkstra sont créés, tel que la matrice des distances entre villes ou bien tout simplement la liste des villes.

Dijkstra:

L'algorithme de Dijkstra est un algorithme de recherche de plus court chemin. Il est utilisé pour trouver le chemin le plus court entre deux sommets d'un graphe pondéré. Il était très important de l'implémenter pour trouver les chemins et les distances des commandes à livrer. Pour Dijkstra, j'ai tout simplement implémenté une méthode statique dans ma classe nommée **DijkstraFeaturesBI**. Cette méthode retourne la distance et instancie le chemin (string) dans un **PathCityWriter**, donné en paramètre.

N-ary tree:

Les arbres n-aires ont été bénéfiques pour la création de l'organigramme. Mon fichier utilitaire **Node.cs** contient toutes les classes nécessaires à l'implémentation de l'organigramme. Je suis partie d'une classe abstraite générique de création de nœud d'arbre nommée **Node**, puis j'ai implémenté ses classes filles **SalarieNode**. L'arbre complet est créé via la classe **SalarieTree**.

Nunit:

Pour mes tests unitaires, j'ai utilisé la bibliothèque C Sharp **NUnit3**.

LIENS

- Git: <https://github.com/shainaBk/TransconnectProject>
- Uml: <https://online.visual-paradigm.com/w/glcbxqwf/app/diagrams/#diagram:workspace=glcbxqwf&proj=0&id=1>