ESILV

RAPPORT TRANSCONNECT

EDMOND Guillaume
CHANEMOUGAM Kaveen

SOMMAIRE

INTRODUCTION

Dans le cadre de notre projet de fin de semestre en C#, nous avons développé une solution de gestion d'entreprise pour aider la société TransConnect à mieux gérer son activité.

Notre objectif étant ainsi de créer un logiciel permettant de gérer les clients, les salariés et les commandes, en permettant au PDG d'effectuer diverses opérations sur tous ces aspects (allant de l'embauche ou du licenciement à la gestion de la flotte de véhicules et de chauffeurs pour faire tourner la société).

Nous nous sommes donc tout d'abord interrogés sur la structure du logiciel en effectuant plusieurs schémas et diagrammes de classes UML sur papier avant de plonger au coeur du sujet.

Au cours du développement de l'application nous avons décidé d'avancer toujours ensemble sur une ou des classes qui avaient toujours des liens entre-elles (tout en suivant la logique de nos schémas) afin d'éviter de nous éparpiller partout et de perdre le fil du projet.

PRÉSENTATION DES CLASSES

Dans cette partie du rapport, vous retrouverez la description des principales classes que nous avons créées afin de mener à terme notre projet.

02

PRÉSENTATION DES CLASSES

Durant la conception de la solution, nous avons créé 20 classes et 2 interfaces, que nous allons vous expliquer juste ci-dessous :

- Personne : Cette classe abstraite est la base de tous les types de personnes que nous avons créés, à savoir principalement les clients et les salariés. Elle contient les informations de base sur une personne comme son numéro de sécurité sociale (appelée id dans notre projet), son nom, son prénom, etc...

- Client : Un client est une personne avec en plus le montant total de ses achats et une liste de commande qui correspond à toutes les commandes qu'il a effectuées. Cette liste s'autoenrichie à chaque commande.

- Salarié : Un salarié est une personne qui dispose en plus d'une date d'entrée dans l'entreprise et du poste qu'il occupe. Cette classe sera utilisée pour tous les types de salariés sauf les chauffeurs qui disposent d'une classe à part.

- Chauffeur : Un chauffeur est un salarié mais il dispose en plus d'un numéro de chauffeur. Cela permet de savoir s'il est libre ou non un jour donné.

- Manage_Client: Cette classe permet de stocker une base de données de clients sous forme d'une liste et de gérer toutes les opérations en lien avec les clients comme l'ajout, la suppression, la modification, les différents types de tris, etc...

- Manage_Salarie: Cette classe permet de gérer tout ce qui est en lien avec les salariés comme les différents types d'embauche, le licenciement, la modification, etc... Ici, la base de données des salariés est gérée sous la forme d'un arbre n-aire qui utilise comme racine le PDG de l'entreprise. Chaque élément de l'arbre est défini comme un nœud qui est une classe à part de notre projet.

- Nœud: Grâce à cette classe, nous pouvons définir l'arbre n-aire décrit dans la classe juste au-dessus. Un nœud est définie comme un salarié qui possède un nœud successeur et un nœud frère.

- Véhicule: C'est une classe abstraite qui contient des informations comme la plaque d'immatriculation, la capacité de charge, mais également un HashSet de dates d'indisponibilités. Tous les différents types de véhicules qui hériteront de cette classe auront donc ce HashSet pour nous permettre de savoir s'ils sont libres ou non selon une date donnée.

PRÉSENTATION DES CLASSES

- Commande: Ici, on définit les éléments indispensable à une commande à savoir le numéro de la commande, le client qui l'a effectué, le chauffeur qui l'effectuera ainsi que le véhicule qu'il utilisera. D'autres éléments comme les villes de départ et d'arrivée, le prix, la distance et le statut de la commande sont également définis dans cette classe.
- Manage_Commande: Toutes les opérations en lien avec les commandes ainsi que la base de données de commande sont gérées par cette classe. C'est elle qui créera une commande adéquate en testant quel chauffeur est disponible et quel est itinéraire peut être emprunté pour effectuer le trajet le plus court. Elle calculera également le prix total de la commande en fonction de l'ancienneté du chauffeur et du nombre de kilomètres.
- Graphe: En effet, pour que la commande puisse être créée avec le chemin le plus court, il faut qu'elle soit relié à l'algorithme de Dijkstra. Pour que cet algorithme fonctionne, il lui faut un graphe constitué d'une liste d'adjacence entre les villes. C'est exactement ce que propose cette classe en initialisant un graphe (liste d'adjacence) sous la forme d'un dictionnaire de ville ayant pour valeur une liste contenant la ville à laquelle on peut accéder depuis la clé, ainsi que la distance séparant ces 2 villes. Le tout est directement initialisé depuis un fichier CSV.
- Dijkstra : En utilisant la classe graphe, nous avons créé une fonction ExecuterDijkstra qui renvoie un dictionnaire ayant pour clé une ville et pour valeur un tuple contenant en première place la distance qui le sépare avec son précurseur (2ème élément du tuple). Cette fonction fonctionne en définissant un dictionnaire de distances minimales (dictionnaire que la fonction renverra) et une file de priorité sous la forme d'un SortedSet, le tout combiné avec la liste d'adjacence définie par la classe graphe. Le but étant ainsi de récupérer la distance minimale jusqu'à la destination en partant de la source.

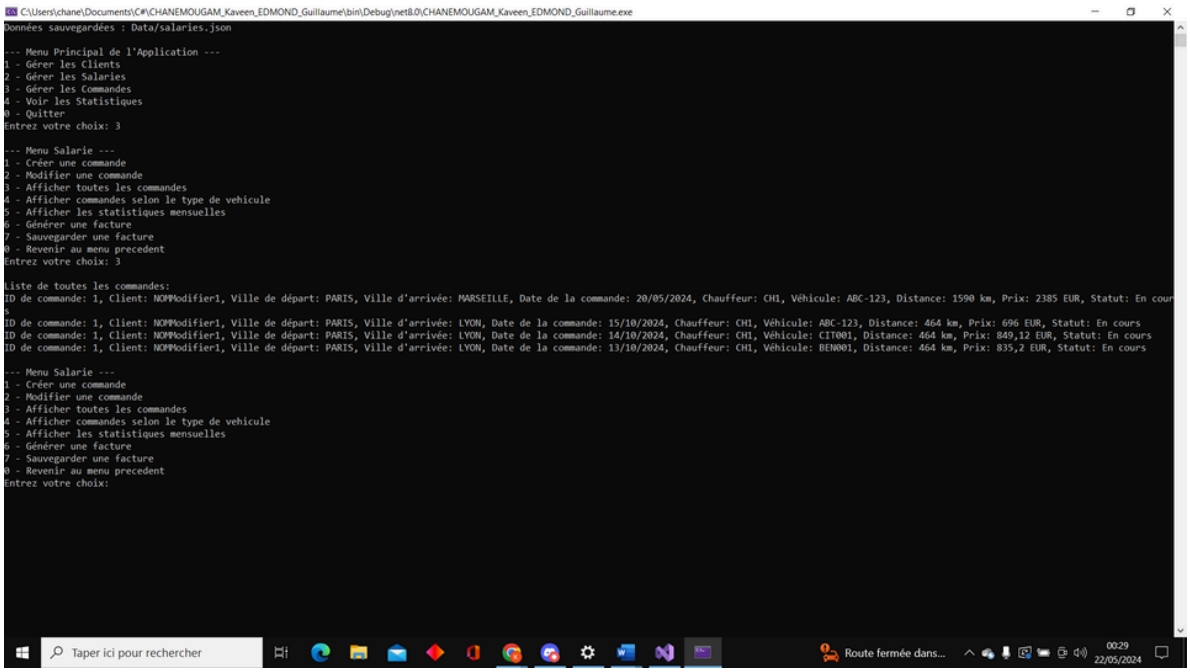
- Enregistrement: Cette classe est celle qui permet de gérer l'enregistrement de toutes les modifications apportées à notre base de données à chaque exécution du programme. Ainsi, elle gère la sauvegarde et le chargement des données grâce à des fichiers JSON.

Finalement, au cours de notre projet, nous avons ajouté plusieurs fonctionnalités en plus de celles qui étaient demandées comme la création et la sauvegarde d'une facture de la commande sur demande (un fichier .txt est généré à la demande), l'affichage des commandes effectuées selon le type de véhicule choisi, les statistiques de chacun des chauffeurs (revenus qu'il a généré, le nombre de livraisons qu'il a fait, le nombre de kilomètres qu'il a parcouru), etc...

RÉSULTATS

03

RÉSULTATS



Vous pouvez donc observer ci-dessus l'interface de notre application.

L'ensemble est découpé sous forme de plusieurs menus représentant chacun un module du cahier des charges.

Sur l'exemple donné, vous pouvez voir l'exécution du module "Commande" où nous avons affiché toutes les commandes actuellement en cours de livraison. Ces commandes ont bien évidemment été chargées depuis le fichier JSON contenant toutes les informations sur les commandes passées.

Le PDG de l'entreprise peut ainsi accéder à chacune des fonctionnalités qu'il souhaite directement depuis le terminal, tous les menus sont parfaitement fonctionnels.

Ci-dessous, voici par exemple une autre exécution de l'affichage des salariés de l'entreprise :

```
--- Menu Salarie ---
1 - Afficher l'Organigramme
2 - Embaucher un nouveau salarie
3 - Licencier un salarie
4 - Modifier un salarie
0 - Revenir au menu precedent
Entrez votre choix: 1

1 :     Directeur Général : Dupont
2 :       Directeur des Opérations : Martin
3 :         Chef d'Équipe : Leroy
4 :         Chef d'Équipe : Durand4
5 :         Chef d'Équipe : Durand5
7 :       Directeur RH : Blanc
8 :         Manager RH : Petit
14 :        comptable : Salarie
11 :      Directeur C# : EdmondChanemougam
13 :      Directeur python : edmond
```

TRANSCONNECT

MERCI POUR VOTRE ATTENTION

EDMOND Guillaume

CHANEMOUGAM Kaveen