Rapport – Projet TransConnect

Par Killian Begu et Clara Fadda

Le projet TransConnect vise à implémenter en C# une simulation de la gestion complète d'une société de transport de marchandises. L'objectif principal est de fournir une application fonctionnelle permettant d'administrer les salariés, les clients, les commandes de livraison, ainsi que d'optimiser les trajets entre différentes villes françaises.
Ce projet est la mise en application concrète les notions vues en TD : POO, interfaces, délégation, collection, arbre n-aire et algorithmes des plus courts chemin. En plus d'avoir consolider les notions acquises en TD, ce projet a également été l'occasion de développer de nouvelles fonctionnalités avancées dont nous parlerons dans ce rapport.

I. Architecture de l'application

Pour développer notre application, nous avons fait le choix d'utiliser une solution WPF. L'application s'articule autour de deux projets distincts, le projet CORE et le projet que nous avons nommé TransConnect_WPF ce qui permet de respecter le principe de séparation des responsabilités.

Le projet CORE contient la logique métier, les entités, les algorithmes, les services, et l'accès aux données. Il est organisé comme suit :

- Algorithms : contient les trois algorithmes de recherche du plus court chemin.
- Entities : regroupe toutes les classes représentant les entités manipulées (clients, chauffeurs, véhicules, commandes, graphes, etc.).
- Interfaces : permet de définir des interfaces pour définit des interfaces pour la persistance et les services, favorisant l'injection de dépendances et la testabilité.
- Persistence : contient les éléments liés à l'accès aux données et leur initialisation (seeders, contexte de base de données).
- Repositories : implémente les classes de type Repository pour gérer l'accès aux données de manière centralisée (Generic Repository, GrapheExcelRepository, etc).
- Services : regroupe les méthodes et services utilisés pour manipuler les entités et les données (CommandeService, OrganigrammeService, etc.).

Le projet TransConnect_WPF correspond à la couche de l'interface utilisateur (UI) de l'application. Il est structuré de la manière suivante :

- Data : contient les fichiers de données (Excel, images, configuration JSON) servant à l'initialisation ou à l'affichage des informations (distances entre villes, coordonnées géographiques, etc.).
- Views : décrit l'ensemble des interfaces utilisateur avec des pages de navigation et des fenêtres fonctionnelles pour la gestion des entités.
- Fichiers principaux :
    - App.xaml : définit les ressources et l'application WPF.
    - Program.cs : est le point d'entrée principal de l'application

II. Utilisation d'une base de données

Afin de facilité la gestion des données, nous avons fait le choix de relier notre projet à une base de données locale. Nous avons utilisé un fichier Appsetting.json qui permet de choisir les identifiants pour la connexion à la base de données. Nous l'avons configurée à l'aide de DbContext via Entity Framework Core ce qui nous a permis de gérer facilement les relations d'héritage entre les entités. Les fichiers TransConnectDbContexte, DbContextFactory et *Entité*Seeder permettent respectivement de définir le contexte de la base de données, de créer des instances et d'insérer des données initiales. Les accès aux entités sont gérés grâce à des repositories conforment au pattern Repository.

III. Fonctionnalités supplémentaires apportées

En plus des fonctionnalités attendues (création de clients, commandes, affichage de l'organigramme, calcul du trajet optimal), nous avons implémenté les méthodes, extensions et fonctionnalités suivantes :

Gestion complète de la flotte de véhicules (ajout et modification)

Nous avons trouvé judicieux de créer un module de gestion de la flotte de véhicules. Comme pour les commandes, les clients ou même les salariés, ce module permet d'une part d'ajouter un véhicule et d'autre part de modifier les caractéristiques d'un véhicule déjà existant. Ce module permet également de vérifier la disponibilité du véhicule en plus de celle du chauffeur pour éviter une double affectation.

Visualisation du graphe sur une carte

Afin de rendre notre graphe le plus visuel possible, nous avons fait en sorte qu'il s'affiche sur une carte de la France. Cela permet d'avoir une vision plus claire sur les différentes liaisons entres les villes françaises.

Ajout des villes et liens

En plus de visualiser le graphe directement sur la carte de la France, notre interface permet également d'ajouter une nouvelle ville (nouveau nœud). Notre interface permet également de créer ou de supprimer des liens. Pour créer une nouvelle ville il suffit de la nommer et d'indiquer ses coordonnées géographiques. Les nouvelles villes et les nouveaux liens sont automatiquement enregistrés dans le fichier xlsx de base.

IV. Gestion des trajets et algorithmes de plus court chemin

Visualisation du graphe

Afin de visualiser le graphe pour lequel les nœuds du graphe correspondaient aux villes et les arrêtes aux liaisons possibles entre ces villes avec un poids qui correspondait à la distance les séparant, nous avons utilisé Windows.shapes.

Présentation des algorithmes implémentés

Comme il l'était indiqué dans la consigne du projet, nous avons implémenter trois algorithmes permettant de trouver le plus court chemin d'un graphe.

- Dijkstra : calcule des plus courts chemins à partir d'une source vers tous les autres sommets en construisant progressivement un sous-graphe dans lequel sont classés les différents sommets par ordre croissant de leur distance minimale au sommet de départ.
- Bellman-Ford : calcule des plus courts chemins depuis un sommet source donné dans un graphe orienté pondéré. Il autorise la présence de certains arcs de poids négatif. L'algorithme utilise le principe de la programmation dynamique.
- Floyd-Warshall : calcule les distances des plus courts chemins entre toutes les paires de sommets dans un graphe orienté et pondéré en examinant systématiquement toutes les combinaisons intermédiaires possibles.

Comparaison des algorithmes

- Complexité algorithmique
  - Dijkstra : nous avons utilisé une liste de priorité. Notre algorithme a donc une complexité globale de O((V+E)logV)
  - Bellman-Ford : la complexité de cet algorithme est en O(|V||A|) où |V| est le nombre de sommets et |A| le nombre d'arcs.
  - Floyd-Warshall : la complexité de cet algorithme est en O(|V|^3) ou|V| est le nombre de sommets du graphe.

- Temps d'exécution (sur un même graphe)
  - Dijkstra : cet algorithme à l'avantage de ne pas parcourir tout le graphe si c'est pour calculer le plus court chemin entre deux sommets donnés ce qui lui permet d'être plus rapide.
  - Bellman-Ford : cet algorithme parcourt toutes les arêtes à chaque itération donc plus le nombre d'arêtes est grand, plus il est lent.
  - Floyd-Warshall : Si le nombre de sommet est grand, l'exécution de cet algorithme peut être très longue et couteuse en mémoire car il manipule une grosse matrice en mémoire.

- Pertinence en contexte réel
    - Dijkstra : très optimal dans le cas de la recherche du plus court chemin entre deux points donnés.
    - Bellman-Ford : il pourrait-être utile si nous ajoutions des coûts négatifs à notre graphe (si une route souvent très bouchée ou en travaux par exemple).
    - Floyd-Warshall : cette version est parfaite pour pré-calculer tous les plus courts chemins entre toutes les villes, surtout s'il n'y a pas beaucoup de sommets. Cependant dès lors qu'il y aura un changement au niveau des distances entre les villes (ce qui peut arriver à causes des intempéries, des travaux, etc.) alors il faudra de nouveau refaire tourner l'algorithme.

Justification du choix préféré dans le projet

Dans le cadre du projet TransConnect, nous avons choisi d'utiliser l'algorithme de Dijkstra. Notre application vise avant tout à déterminer le trajet optimal entre deux villes spécifiques au moment de la création d'une commande. Dans ce contexte, Dijkstra se révèle particulièrement efficace : il est rapide, optimisé pour les poids positifs (comme les distances en kilomètres), et suffisamment léger en termes de mémoire et de complexité.

Conclusion

Notre projet TransConnect a abouti à une application fonctionnelle et bien structurée, répondant à l'ensemble des exigences du cahier des charges. L'ajout de fonctionnalités complémentaires comme la gestion des véhicules (ajout et modification), l'affichage du graphe sur carte ou encore la possibilité de demander la facture de la commande, a renforcé la pertinence de l'application. L'architecture en modules bien séparés, l'intégration d'une base de données, et l'interface WPF ont permis d'assurer une bonne maintenabilité et évolutivité du code.
Nous avons tout de même pensé à des pistes d'amélioration pour une version encore plus complète dans le futur. Nous avons pensé notamment à l'intégration d'API de cartographie en temps réel (comme Google Maps ou OpenStreetMap) pour afficher plus précisément les trajets et conditions de circulation.