Système de Gestion Optimisée de Tournées de Livraison

Contexte :

Suite à l'augmentation significative du coût du carburant et du nombre de clients, une entreprise de logistique fait face à des défis majeurs dans l'optimisation de ses tournées de livraison. Actuellement, le système utilise un algorithme simple du "plus proche voisin" (Nearest Neighbor) qui génère des trajets inefficaces. L'entreprise souhaite moderniser son système en adoptant l'algorithme de Clarke & Wright pour réduire les distances parcourues. Votre mission est de développer une application web Spring Boot qui permet de gérer l'ensemble du processus de livraison et d'optimiser intelligemment les tournées.

Objectifs du projet :

- Gérer une flotte de véhicules hétérogène avec leurs contraintes spécifiques (capacité, rayon d'action)
- Planifier et optimiser automatiquement les tournées de livraison
- Comparer les performances de deux algorithmes d'optimisation ci-dessus

Relations Métier entre les Entités :

- Un Tour (Tournée) est assigné à un Vehicle spécifique pour une journée donnée

- Un Tour (Tournée) contient plusieurs Delivery ordonnées selon l'algorithme d'optimisation choisi, formant un parcours complet depuis et vers le Warehouse (entrepôt)

- Une Delivery possède un identifiant unique, une adresse de livraison représentée par des coordonnées GPS (latitude, longitude), un poids en kg, un volume en m^3, et optionnellement un créneau horaire de livraison préféré (exemple : format : 09:00-11:00 signifiant que le client préfère être livré entre 9h et 11h)

Une formule mathématique de distance (à rechercher) doit être utilisée pour calculer la distance entre une delivery et une autre grâce aux valeurs de latitude et longitude

- Une Delivery évolue à travers différents statuts (PENDING, IN_TRANSIT, DELIVERED, FAILED) selon l'avancement du Tour. Dans ce projet les valeur de statuts se modifient manuellement et pas automatiquement.

- Un Vehicle peut être de différents types (VAN, TRUCK, BIKE) avec des capacités maximales spécifiques (poids max en kg, volume max en m^3) qui limitent le nombre de Delivery possibles sur un même Tour. voir annexe1 dans la partie annexe les contraintes à respecter

- Un Warehouse est le point de départ et d'arrivée de tous les Tours, avec son adresse et ses horaires d'ouverture (06:00-22:00)

Note : Utiliser des valeurs fictives réalistes pour les simulations tout en respectant les valeurs max mentionnées dans les contraintes

L'objectif étant de délivrer toutes les marchandises chaque jour de manière optimisée. Initialement, le système utilisait l'algorithme NEAREST_NEIGHBOR (toujours choisir la livraison la plus proche) qui était rapide mais générait des trajets très longs (180km en moyenne pour 100 livraisons) et suite à l'augmentation des coûts de carburant et du nombre de clients, l'entreprise a décidé d'adopter l'algorithme CLARKE_WRIGHT. Voir annexe2 dans la partie annexe pour comprendre la démarche de calcul

Interfaces et implémentations à développer :

Libre à vous de mettre les interfaces que vous voulez mais les plus importantes sont :

TourOptimizer :

- Méthode : calculateOptimalTour(...)

TourService :

- Méthode : getOptimizedTour(...) → retourne la liste ordonnée des Delivery à suivre
- Méthode : getTotalDistance(...)

Classes d'implémentation :

- NearestNeighborOptimizer (implémente TourOptimizer)
- ClarkeWrightOptimizer (implémente TourOptimizer)

NOTE : Vous devez comprendre l'intérêt de cette démarche et sa relation avec le fichier applicationContext.xml. Cela vous aidera à bien assimiler la notion d'une application fermée à la modification et ouverte à l'extension.

Fonctionnalités à implémenter :

- Gestion CRUD des principales entités
- Visualiser la Tour optimale : liste ordonnée des Delivery à suivre pour un Tour donné
- D'autres fonctionnalités si vous voyez pertinent de les ajouter

Technologies et concepts à utiliser :

java8 ou plus

API Java à utiliser : Stream API, Java Time API, Collection API et Hashmaps, Optional …

Spring Boot :

Création de projet avec Spring Initializer/IDE

Injection de dépendance avec applicationContext.xml à créer manuellement.

Interdit d'utiliser l'Injection de Dépendance avec les annotations. Il est interdit d'utiliser : @Autowired, @Inject, @Resource, @Component, @Service, @Repository, @Bean. En cas de problème si @Repository est obligé vous pouvez la remplacer par <jpa:repositories> dans le fichier applicationContext.xml

Configuration des propriétés via application.properties (datasource, port, etc.) ET configuration des beans via applicationContext.xml (injection de dépendances)

REST API

- Endpoints CRUD pour les entités nécessaires
- Gestion des méthodes HTTP (GET, POST, PUT, DELETE) avec des annotations concernées
- Bonnes pratiques REST (nommage des URLs, codes HTTP appropriés)
- Tests API via Postman ou autre outil similaire

- Swagger pour la documentation API (dépendance sur pom.xml)

Couches applicatives : En plus des couches habituelles il faut mettre la couche **DTO**

Spring Data

- JpaRepository et ses méthodes de base
- Exemples de méthodes attendues : findByXxx , findByXxxAndYyy , Méthodes avec @Query et @Param

H2 comme SGBD ;

- **interdit de travailler avec un autre SGBD que H2**
- **Travaillez avec la dépendance H2 via pom.xml et pas l'application à installer**

Tests unitaires avec Junit

LOGGER (SLF4j, java.util.logging natif … **interdit Log4j**)

Classes d'exceptions

Validations

Design patterns (Repository, DTO, Mapper)

Outils de travail :

- Gérer le projet avec Git en utilisant des branches
- Utiliser un IDE de votre choix
- Utiliser JIRA (SCRUM) pour la gestion du projet
- Lombok
- Spring Boot DevTools
- SonarLint

pas de vues obligatoires, tout doit être testable via Postman ou un autre API Tester.

Bonus :

- mettre quelques tests d'intégration
- Travailler avec liquibase
- Docker
- Générer une interface web avec Thymeleaf

Livrables Attendus

- Code source
- Lien jira
- collection d'API sur la plateforme de l'API Tester choisi à me montrer le jour de débriefing
- Diagrammes de classes
- Fichier README

Modalités pédagogiques :

- Projet individuel
- Durée du projet : 7 jours
- Début : 20/10/25, Deadline d'envoi des rendus : 28/10/25 avant minuit

Modalités d'évaluation :

- Présentation client et technique avec démonstration (strictement en français)
- Discussions avec des questions
- Mise en situation

Critères de performance :

- Respect des exigences fonctionnelles et règles métier: Entités, contraintes, algorithmes, validations
- Qualité de l'architecture et du code
- Maîtrise des technologies : Spring Boot, JPA, REST, H2, JUnit, Swagger, Java 8+
- Gestion de projet et livrables : Git, JIRA, documentation, tests, Postman
- Qualité de la présentation et démonstration

Annexe 1 : Contraintes par type de véhicule :

BIKE (Vélo cargo) :

- Capacité max : 50kg, 0.5m³
- Nombre de livraisons : 15 max par Tour

VAN (Camionnette) :

- Capacité max : 1000kg, 8m³
- Nombre de livraisons : 50 max par Tour

TRUCK (Camion) :

- Capacité max : 5000kg, 40m³
- Nombre de livraisons : 100 max par Tour

Annexe 2 : méthodes de calcul

Explication de chaque méthode :

1. NearestNeighborOptimizer

Méthode de calcul :

Algorithme du Plus Proche Voisin. À chaque étape, choisir la livraison la plus proche de la position actuelle. Distance = MIN(toutes distances non visitées).

- ✅ Vitesse de calcul ultra rapide (~50ms pour 100 livraisons)
- ❌ Trajets très longs (180km moyenne)

2. ClarkeWrightOptimizer

Méthode de calcul :

Algorithme des économies de Clarke & Wright. Calculer les économies de fusionner deux trajets : Économie = Distance(Warehouse,A) + Distance(Warehouse,B) - Distance(A,B). Fusionner par ordre d'économie décroissante.

- ✅ Vitesse de calcul acceptable (200ms pour 100 livraisons)
- ✅ Distance réduite

Explication de formule de calcul pour chaque méthode :

Données de l'exemple : Warehouse (Point de départ) et 4 clients à livrer : A, B, C, D

Distances : Warehouse↔A : 5km | Warehouse↔B : 15km | Warehouse↔C : 10km | Warehouse↔D : 20km | A↔B : 12km | A↔C : 8km | A↔D : 18km | B↔C : 7km | B↔D : 6km | C↔D : 11km

1. NearestNeighborOptimizer

Principe : Toujours aller au client le plus proche non visité

Calcul étape par étape :

- Étape 1 : Depuis Warehouse → A (5km) est le plus proche
- Étape 2 : Depuis A → C (8km) est le plus proche non visité
- Étape 3 : Depuis C → B (7km) est le plus proche non visité
- Étape 4 : Depuis B → D (6km) seul client restant
- Étape 5 : Depuis D → Warehouse (20km) retour à l'entrepôt

Tour finale : Warehouse → A → C → B → D → Warehouse

Distance totale : 5 + 8 + 7 + 6 + 20 = 46km

2. ClarkeWrightOptimizer

Principe : Fusionner les paires de clients qui économisent le plus de kilomètres

Formule d'économie : Économie(i,j) = Distance(Warehouse,i) + Distance(Warehouse,j) - Distance(i,j)

Étape 1 : Calculer les économies

- Économie(B,D) = 15 + 20 - 6 = 29km (meilleure)
- Économie(C,D) = 10 + 20 - 11 = 19km
- Économie(B,C) = 15 + 10 - 7 = 18km
- Économie(A,B) = 5 + 15 - 12 = 8km
- Économie(A,C) = 5 + 10 - 8 = 7km
- Économie(A,D) = 5 + 20 - 18 = 7km

Étape 2 : Construire les Tours par ordre d'économie décroissante

Tours initiaux (avant fusion) :

- Tour A : Warehouse → A → Warehouse (10km)
- Tour B : Warehouse → B → Warehouse (30km)
- Tour C : Warehouse → C → Warehouse (20km)
- Tour D : Warehouse → D → Warehouse (40km)

Distance totale = 100km

Processus de fusion :

- Fusion 1 : B-D (29km) - ✅ Fusion réussie
  - Warehouse → B → D → Warehouse
  - Distance = 15 + 6 + 20 = 41km (économie de 29km réalisée)
- Fusion 2 : C-D (19km) - ✅ Fusion réussie
  - D est en fin de Tour (W→B→D→W), C est seul dans sa Tour
  - On fusionne en insérant C après D
  - Warehouse → B → D → C → Warehouse
  - Distance = 15 + 6 + 11 + 10 = 42km

Économie réalisée : on évite le trajet D→W puis W→C (20+10=30km), remplacé par D→C (11km), soit une économie de 19km

- Fusion 3 : B-C (18km) - ❌ Impossible
  - B et C sont déjà dans la même Tour
- Fusion 4 : A-B (8km) - ✅ Fusion réussie
  - B est au début de la Tour (W→B→D→C→W), A est seul
  - On insère A entre W et B
  - Warehouse → A → B → D → C → Warehouse
  - Distance = 5 + 12 + 6 + 11 + 10 = 44km
- Fusion 5 : A-C (7km) - ❌ Impossible
  - A et C sont déjà dans la même Tour
- Fusion 6 : A-D (7km) - ❌ Impossible
  - A et D sont déjà dans la même Tour

Tour finale : Warehouse → A → B → D → C → Warehouse

Distance totale : 5 + 12 + 6 + 11 + 10 = 44km

⚠️ Note importante sur cet exemple :

Sur cet exemple avec seulement 4 clients, les deux algorithmes donnent des résultats très proches (46km vs 44km, soit une amélioration de 4% grâce à Clarke & Wright). La supériorité de ClarkeWright apparaît clairement sur des cas réels avec 50-100 livraisons, où il réduit la distance de 28% en moyenne (de 180km à 130km) grâce à une vision globale des économies possibles, contrairement aux choix locaux et myopes de NearestNeighbor.