République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Tunis El Manar
École Nationale d'Ingénieurs de Tunis

Département Technologies de l'Information et de la Communication

Rapport De Stage Ingénieur

Application intelligente d'affectation et d'optimisation des livraisons

PROGRESS Engineering

Présenté par :

Belsem JEMAI **2ATEL1**

Encadré par :

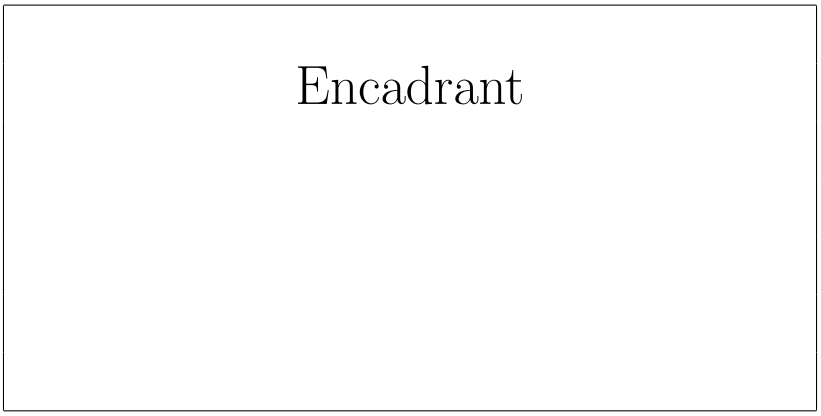
M. Mabrouk ZHILI

Durée de stage :

De 01/07/2026 à 30/08/2026

Année Universitaire : 2025/2026

Signature



Remerciements

Au terme de ce travail, je tiens à exprimer ma sincère gratitude à toutes les personnes qui ont contribué, de près ou de loin, à la réussite de ce stage.

Je remercie tout d'abord la société **Progress Engineering** de m'avoir accueilli en son sein et de m'avoir offert l'opportunité de réaliser ce projet dans un cadre professionnel stimulant.

J'adresse mes remerciements les plus particuliers à mon encadrant, **M. Mabrouk ZHILI**, Directeur Technique, pour la confiance qu'il m'a accordée, pour sa disponibilité tout au long du stage et pour ses conseils avisés qui ont guidé mes choix techniques et enrichi ce travail.

Je tiens également à remercier l'ensemble de l'équipe du service informatique pour son accueil chaleureux et pour l'esprit de collaboration qui a rendu cette expérience aussi agréable qu'instructive.

Mes remerciements s'adressent enfin au corps enseignant de l'**École Nationale d'Ingénieurs de Tunis (ENIT)** pour la formation de qualité dispensée, ainsi qu'aux membres du jury qui me font l'honneur d'évaluer ce travail.

Résumé

Ce projet, réalisé dans le cadre d'un stage d'ingénieur au sein de la société Progress Engineering, porte sur la conception et la réalisation d'une application web d'optimisation de tournées de livraison dans un contexte multi-dépôts. Le problème traité relève du problème de tournées de véhicules multi-dépôts avec contrainte de capacité (MDVRP), appliqué à un cas d'étude comportant cinq dépôts, une centaine de destinations réparties sur le territoire tunisien et une flotte d'une dizaine de véhicules soumis à des contraintes opérationnelles. La solution développée s'appuie sur un moteur d'optimisation fondé sur la bibliothèque OR-Tools, exploitant une décomposition du problème par dépôt source, ainsi que sur une couche d'analyse de données assurant un zonage géographique des destinations. L'ensemble est exposé au travers d'une application web complète, comprenant une interface de saisie, un tableau de bord d'indicateurs et une cartographie interactive. Sur le cas d'étude, le système produit une solution servant l'intégralité des livraisons, tout en quantifiant le coût des principales contraintes métier.

Mots-clés : optimisation de tournées, MDVRP, OR-Tools, recherche opérationnelle, application web, cartographie, apprentissage automatique.

Abstract

This project, carried out during an engineering internship at Progress Engineering, focuses on the design and implementation of a web application for delivery route optimization in a multi-depot context. The problem addressed is the Multi-Depot Vehicle Routing Problem with capacity constraints (MDVRP), applied to a case study involving five depots, around one hundred destinations across Tunisia, and a fleet of about ten vehicles subject to operational constraints. The developed solution relies on an optimization engine based on the OR-Tools library, using a decomposition of the problem by source depot, together with a data analysis layer providing a geographic zoning of the destinations. The whole is delivered through a complete web application, including a data-entry interface, a performance dashboard and an interactive map. On the case study, the system produces a solution serving all deliveries, while quantifying the cost of the main business constraints.

Keywords : route optimization, MDVRP, OR-Tools, operations research, web application, mapping, machine learning.

Table des matières

Remerciements **2**

Résumé **3**

Abstract **4**

Liste des figures **9**

Liste des tableaux **10**

Liste des acronymes **12**

Introduction Générale **12**

1 Contexte général et étude préalable **14**

1.1 Présentation de l'organisme d'accueil 14

1.1.1 Progress Engineering et son activité 14

1.1.2 Le service informatique 15

1.1.3 Encadrement et positionnement du projet 16

1.2 Contexte et problématique 16

1.2.1 La distribution logistique et l'enjeu de l'optimisation des tournées . 16

1.2.2 Le contexte multi-dépôts 17

1.2.3 Problématique et objectif du projet 17

1.3 Le problème de tournées de véhicules (VRP) 18

1.3.1 Du VRP au problème avec contrainte de capacité (CVRP) 18

1.3.2 Le problème multi-dépôts (MDVRP) 19

1.3.3 Complexité et enjeux algorithmiques 19

1.4 Approches de résolution 20

1.4.1 Méthodes exactes 20

1.4.2 Heuristiques et métaheuristiques 21

1.4.3 Positionnement et choix d'OR-Tools 21

1.5 Objectifs et méthodologie du stage 22

1.5.1 Objectifs et périmètre du projet 22

1.5.2 Méthodologie de travail 23

1.5.3 Traçabilité des choix de conception 24

1.6 Conclusion 24

2 Analyse et conception **25**

2.1 Spécification des besoins 25

2.1.1 Besoins fonctionnels 25

2.1.2 Besoins non fonctionnels 26

2.1.3 Acteurs et cas d'utilisation 27

2.2 Formulation mathématique du problème 27

2.2.1 Notations et données du problème 28

2.2.2 Fonction objectif 28

2.2.3 Contraintes du modèle 29

2.3 Architecture générale de la solution 30

2.3.1 Vue d'ensemble 30
2.3.2 Choix de la pile technologique 31
2.3.3 Flux de données et interactions entre couches 32
2.4 Conception du modèle de données 33
2.4.1 Schéma relationnel 33
2.4.2 Principe " une donnée à un seul endroit " 34
2.4.3 Séparation entre schéma de données et logique du solveur 34
2.5 Conception des composants métier 34
2.5.1 Conception du solveur 34
2.5.2 Conception de la couche d'analyse de données 35
2.5.3 Conception de la cartographie 36
2.6 Conclusion 36

3 Réalisation du cœur d'optimisation 38

3.1 Environnement de développement 38
3.1.1 Poste de travail et gestion de l'environnement 38
3.1.2 Gestion de version 39
3.1.3 Organisation du code et rythme d'intégration 39
3.2 Préparation des données géographiques 39
3.2.1 Géocodage des destinations 39
3.2.2 Matrices de distances : géodésique et routière 40
3.2.3 Asymétrie du réseau, plancher de distance et mise en cache 40
3.3 Implémentation du solveur 41
3.3.1 Décomposition en sous-problèmes par source 41
3.3.2 Modélisation des contraintes dans OR-Tools 41

3.3.3 Retour au dépôt d'origine et budget de temps 42
3.3.4 Gestion des lots non servis 43
3.4 Fractionnement des lots de gros volume 43
3.4.1 Principe et calcul du découpage 43
3.4.2 Livraison " tout ou rien " 44
3.4.3 Validation du comportement 44
3.5 Couche d'analyse de données 44
3.5.1 Zonage géographique par k-means 44
3.5.2 Validation par l'indice de silhouette et recoupement 45
3.5.3 Estimation de la durée des tournées 46
3.6 Conclusion 47

4 Réalisation applicative, résultats et validation 48

4.1 Réalisation du backend 48
4.1.1 Architecture de l'API 48
4.1.2 Endpoints d'optimisation et de consultation 49
4.1.3 Endpoints cartographie, indicateurs et évaluation 49
4.2 Réalisation du frontend 50
4.2.1 Architecture multi-écrans 50
4.2.2 Saisie des lots et lancement de l'optimisation 50
4.2.3 Consultation des tournées et tableau de bord 51
4.3 Cartographie interactive 52
4.3.1 Du rendu serveur au rendu client 53
4.3.2 Code couleur des destinations 53
4.3.3 Affichage des violations et évaluation manuelle 54

4.4 Résultats et indicateurs 55
4.4.1 Indicateurs des tournées 55
4.4.2 Coût des contraintes 56
4.4.3 Structure du zonage 56
4.5 Validation et limites 57
4.5.1 Validation fonctionnelle 57
4.5.2 Limites 57
4.5.3 Perspectives 58
4.6 Conclusion 58

Conclusion générale **59**

Bibliographie **60**

Liste des figures

2.1 Diagramme de cas d'utilisation de l'application 27

2.2 Architecture en couches de la solution 31

2.3 Diagramme de séquence du déroulement d'une optimisation 32

2.4 Modèle de données de l'application 33

2.5 Décomposition du problème global en sous-problèmes par dépôt source . . 35

3.1 Zonage géographique des destinations en sept zones 45

3.2 Indice de silhouette moyen selon le nombre de zones 46

4.1 Écran de saisie des lots d'une vague 51

4.2 Lancement d'une optimisation et indicateur de progression 51

4.3 Consultation détaillée des tournées d'un run 52

4.4 Tableau de bord des indicateurs d'un run 52

4.5 Carte interactive des tournées d'un run 53

4.6 Affichage d'une destination non livrée et de son motif 54

4.7 Évaluation d'une affectation manuelle avec signalement d'une violation caisson 54

Liste des tableaux

4.1 Principaux points d'accès de l'API 49

4.2 Indicateurs globaux de la solution sur le cas d'étude 55

4.3 Effet des principales contraintes sur la distance 56

4.4 Répartition des destinations par zone géographique 57

Liste des acronymes

API *Application Programming Interface* (interface de programmation d'application)

CVRP *Capacitated Vehicle Routing Problem* (problème de tournées avec contrainte de capacité)

GeoJSON *Geographic JSON* (format d'échange de données géographiques)

GIZ *Deutsche Gesellschaft für Internationale Zusammenarbeit* (Agence allemande de coopération internationale)

GLS *Guided Local Search* (recherche locale guidée)

HTTP *HyperText Transfer Protocol*

JSON *JavaScript Object Notation*

KPI *Key Performance Indicator* (indicateur clé de performance)

MDVRP *Multi-Depot Vehicle Routing Problem* (problème de tournées multi-dépôts)

ML *Machine Learning* (apprentissage automatique)

ORS *OpenRouteService*

OSM *OpenStreetMap*

REST *Representational State Transfer*

SHA *Secure Hash Algorithm*

SSII Société de Services et d'Ingénierie Informatique

TSP *Traveling Salesman Problem* (problème du voyageur de commerce)

VRP *Vehicle Routing Problem* (problème de tournées de véhicules)

Introduction Générale

Durant ma formation en télécommunications à l'École Nationale d'Ingénieurs de Tunis, j'ai eu l'opportunité d'effectuer un stage ingénieur au sein de Progress Engineering, société d'ingénierie implantée au Parc Technologique El Ghazala. Cette expérience m'a offert une immersion concrète dans le monde du travail et m'a permis d'approfondir mes connaissances en développement d'applications et en optimisation.

Durant cette période, j'ai participé à la conception et à la réalisation d'une application web complète dédiée à l'optimisation des tournées de livraison multi-dépôts. J'ai ainsi abordé aussi bien la modélisation du problème et sa résolution par la recherche opérationnelle que la mise au point d'une interface interactive de visualisation cartographique. Ce travail m'a permis de relier la théorie acquise en classe, qu'il s'agisse d'algorithmique, de traitement de données ou de programmation, à la réalité d'un projet concret soumis à de véritables contraintes métier.

Ce stage a constitué pour moi une excellente occasion d'approfondir mes compétences, de découvrir le fonctionnement d'une société d'ingénierie et, surtout, de mesurer l'importance d'un travail rigoureux, structuré et guidé par des décisions bien argumentées.

Dans les sections suivantes, je détaillerai les missions auxquelles j'ai contribué, les connaissances que j'ai acquises et les enseignements que je retire de cette première expérience dans l'univers professionnel.

Chapitre 1

Contexte général et étude préalable

Ce chapitre pose le cadre général du projet. Il présente d'abord l'entreprise d'accueil, **Progress Engineering**, ainsi que le service informatique au sein duquel s'est déroulé le stage. Il expose ensuite la problématique de la livraison multi-dépôts à l'origine du projet, avant d'introduire le cadre théorique du problème de tournées de véhicules (VRP) et les principales approches de résolution. Il se termine par la présentation des objectifs, du périmètre et de la méthodologie de travail adoptée.

1.1 Présentation de l'organisme d'accueil

Cette première section présente l'entreprise **Progress Engineering** et son domaine d'activité, le service informatique qui l'anime, puis le cadre d'encadrement et le positionnement du projet réalisé.

1.1.1 Progress Engineering et son activité

Progress Engineering est une société de services et d'ingénierie informatique (SSII) tunisienne, créée en 2000 et basée au Parc Technologique El Ghazala à Ariana [1]. Spécialisée dans l'étude, le conseil et le développement informatique, elle conçoit, développe et intègre des solutions logicielles clés en main adaptées à des besoins spécifiques. Orientée client, l'entreprise intervient aussi bien auprès d'organismes publics, y compris des ministères tunisiens, que d'acteurs privés.

Le Parc Technologique El Ghazala, où se situe l'entreprise, constitue le premier technopôle de Tunisie et regroupe de nombreux acteurs des technologies de l'information et de la communication [2], ce qui offre un environnement propice à l'innovation logicielle.
Depuis sa création, Progress Engineering a mené plusieurs projets d'envergure nationale. Elle a notamment développé et maintient, depuis 2004, la plateforme nationale d'inscription universitaire en ligne en Tunisie [3]. Parmi ses références plus récentes figurent des plateformes développées pour Tunisair Handling, ainsi que les solutions livrées pour la Cour des Comptes en partenariat avec l'Agence allemande de coopération internationale (GIZ) [4].

1.1.2 Le service informatique

Le service informatique constitue le cœur opérationnel de Progress Engineering, puisque l'activité de l'entreprise repose essentiellement sur l'ingénierie et le développement logiciel. C'est au sein de ce service que s'est déroulé mon stage, ce qui m'a permis de participer aux différentes étapes de réalisation d'un projet logiciel.

Objectifs du service :

Le service informatique a pour objectif principal de **concevoir**, **développer** et **maintenir des solutions logicielles** répondant aux besoins spécifiques des clients de l'entreprise. Ses objectifs s'articulent autour de plusieurs axes :

- Assurer la réalisation de projets de développement logiciel de qualité, dans le respect des délais et des budgets convenus avec les clients.
- Garantir la fiabilité, la sécurité et la maintenabilité des applications livrées, tout au long de leur cycle de vie.
- Accompagner les clients dans l'analyse de leurs besoins et leur proposer des solutions adaptées et évolutives.
- Rester à la pointe des technologies en intégrant continuellement de nouveaux outils et méthodes (intelligence artificielle, technologies web modernes, optimisation des processus), afin d'offrir des solutions innovantes et compétitives.

Rôle du service :

Le rôle du service couvre l'ensemble du cycle de vie des projets, depuis l'étude du besoin jusqu'à la mise en production et la maintenance. Concrètement, il prend en charge l'analyse et la spécification des besoins exprimés par les clients, la conception des architectures techniques et fonctionnelles, le développement des applications, ainsi que les phases de test, de déploiement et de maintenance. Il assure également un rôle de conseil et d'expertise technique, en orientant les choix technologiques et en garantissant la cohérence des solutions développées.

1.1.3 Encadrement et positionnement du projet

Mon stage s'est déroulé au sein du service informatique, sous l'encadrement de **M. Mabrouk ZHILI**, Directeur Technique, qui a assuré le suivi technique du travail et la validation des choix de conception tout au long du projet.
Le projet confié s'inscrit dans la démarche d'innovation du service : il porte sur la conception d'une application web d'optimisation des tournées de livraison, appliquée à un contexte multi-dépôts. Il combine un moteur d'optimisation, une couche d'analyse de données et une interface de visualisation cartographique, ce qui en fait un projet transversal mobilisant à la fois des compétences algorithmiques et des compétences de développement full-stack. La problématique métier précise ainsi que le cadre théorique sous-jacent font l'objet des sections suivantes.

1.2 Contexte et problématique

Après avoir présenté l'organisme d'accueil, cette section expose le contexte métier à l'origine du projet : les enjeux de la distribution logistique, la particularité d'une organisation multi-dépôts, puis la problématique d'optimisation qui a motivé le travail réalisé.

1.2.1 La distribution logistique et l'enjeu de l'optimisation des tournées

La distribution physique de marchandises constitue un poste de coût majeur pour toute activité logistique. L'acheminement des produits depuis les points de stockage jusqu'aux

clients finaux repose sur l'organisation de tournées de véhicules, dont dépendent directement les délais de livraison, la consommation de carburant et le taux d'utilisation de la flotte. Une planification manuelle de ces tournées atteint rapidement ses limites : dès que le nombre de destinations augmente, le nombre d'itinéraires possibles devient trop grand pour être exploré à la main, et les solutions retenues empiriquement sont rarement optimales.

L'optimisation des tournées vise précisément à réduire la distance totale parcourue et à mieux répartir la charge entre les véhicules, tout en respectant un ensemble de contraintes opérationnelles (capacité des véhicules, nature des marchandises, disponibilité de la flotte). Le gain attendu est double : une réduction des coûts d'exploitation et une amélioration de la qualité de service.

1.2.2 Le contexte multi-dépôts

Le projet s'inscrit dans un contexte de distribution à l'échelle nationale, réparti sur l'ensemble du territoire tunisien. La particularité de ce contexte tient à la présence de plusieurs points de départ : les marchandises ne partent pas d'un entrepôt unique, mais de **cinq dépôts** répartis géographiquement. Chaque livraison doit être rattachée à un dépôt d'origine, et chaque véhicule effectue une tournée au départ de son dépôt avant d'y revenir.

Concrètement, le problème traité comporte cinq dépôts, une centaine de destinations à desservir à travers le pays, regroupées en environ cent vingt lots de livraison, et une flotte d'une dizaine de véhicules. Cette configuration multi-dépôts complexifie sensiblement la planification : il ne s'agit plus seulement de déterminer l'ordre des visites, mais aussi de décider quel dépôt et quel véhicule prennent en charge chaque livraison, ce qui multiplie le nombre de décisions à optimiser.

1.2.3 Problématique et objectif du projet

La planification manuelle d'un tel volume de livraisons, sur un territoire aussi étendu et avec plusieurs dépôts, n'est ni fiable ni reproductible. Elle repose sur l'expérience des planificateurs, ne garantit aucune optimalité, et rend difficile l'évaluation objective d'une organisation par rapport à une autre.

La problématique du projet peut ainsi se formuler de la manière suivante : *comment concevoir un outil capable de calculer automatiquement des tournées de livraison de bonne qualité, dans un contexte multi-dépôts soumis à des contraintes opérationnelles, tout en restant utilisable par un non-spécialiste ?*

Répondre à cette question suppose de relever plusieurs défis : modéliser correctement le problème et ses contraintes, disposer de données de distance réalistes reflétant le réseau routier, mettre en œuvre un moteur d'optimisation capable de produire une solution en un temps raisonnable, et restituer les résultats sous une forme visuelle et exploitable. C'est l'ensemble de cette démarche que le présent rapport décrit.

1.3 Le problème de tournées de véhicules (VRP)

Le projet réalisé relève d'une classe de problèmes d'optimisation combinatoire bien identifiée dans la littérature : les problèmes de tournées de véhicules. Cette section en présente le cadre théorique, depuis la formulation de base jusqu'à la variante multi-dépôts qui correspond au cas traité, avant d'en souligner la complexité algorithmique.

1.3.1 Du VRP au problème avec contrainte de capacité (CVRP)

Le problème de tournées de véhicules (*Vehicle Routing Problem*, VRP) a été introduit pour la première fois en 1959 par Dantzig et Ramser, sous le nom de *Truck Dispatching Problem*, à propos de la livraison de carburant depuis un terminal vers un ensemble de stations-service [5]. Il constitue l'un des problèmes les plus étudiés de la recherche opérationnelle et généralise le célèbre problème du voyageur de commerce (*Traveling Salesman Problem*, TSP).

Dans sa forme la plus simple, le VRP se modélise sur un graphe $G = (V, A)$, où l'ensemble des sommets $V = \{0, 1, \ldots, n\}$ comprend un dépôt (le sommet 0) et n clients à desservir, et où l'ensemble des arcs A est muni d'une matrice de coûts c_{ij} représentant la distance (ou le temps) de déplacement entre les sommets i et j. Il s'agit alors de déterminer un ensemble de tournées, effectuées par une flotte de véhicules partant du dépôt et y revenant, de manière à visiter chaque client exactement une fois tout en minimisant le coût total, c'est-à-dire la somme des distances parcourues par l'ensemble des véhicules.

La variante la plus classique, et celle qui sert de socle à la plupart des extensions, est le problème avec contrainte de capacité (*Capacitated VRP*, CVRP). Chaque client i y est associé à une demande q_i, et chaque véhicule dispose d'une capacité maximale Q qui ne peut être dépassée sur une tournée [6]. L'ajout de cette contrainte de capacité transforme le problème en une généralisation nettement plus difficile du TSP, puisqu'il faut désormais décider non seulement de l'ordre de visite des clients, mais aussi de leur répartition entre les différentes tournées.

1.3.2 Le problème multi-dépôts (MDVRP)

Le VRP classique suppose que tous les véhicules partent d'un dépôt unique. Or de nombreux contextes logistiques réels, dont celui du présent projet, font intervenir *plusieurs* dépôts répartis géographiquement. On parle alors de problème de tournées de véhicules multi-dépôts (*Multi-Depot VRP*, MDVRP).

Dans cette variante, l'ensemble des sommets est partitionné entre un ensemble de dépôts et un ensemble de clients. Chaque véhicule est rattaché à un dépôt d'origine, effectue sa tournée depuis ce dépôt et doit y revenir en fin de parcours. Le MDVRP introduit ainsi un niveau de décision supplémentaire par rapport au CVRP : il ne suffit plus d'ordonnancer les visites et de répartir les clients entre les véhicules, il faut également **affecter chaque client au dépôt qui le prendra en charge**. Cette décision d'affectation est couplée aux décisions de routage, ce qui accroît considérablement la taille de l'espace des solutions à explorer.

Le cas traité dans ce projet correspond précisément à un MDVRP avec contrainte de capacité : cinq dépôts répartis sur le territoire tunisien, une flotte de véhicules à capacité limitée, et un ensemble de livraisons à affecter et à ordonnancer. À ces caractéristiques de base s'ajoutent des contraintes opérationnelles propres au contexte de l'entreprise, qui seront détaillées lors de la formulation mathématique du problème au chapitre suivant.

1.3.3 Complexité et enjeux algorithmiques

Le VRP, comme le TSP dont il dérive, appartient à la classe des problèmes *NP-difficiles* [7]. Concrètement, cela signifie qu'aucun algorithme connu ne permet de garantir la solution

optimale en un temps raisonnable dès que la taille de l'instance devient importante : le nombre de solutions possibles croît de façon combinatoire (factorielle) avec le nombre de clients, rendant l'énumération exhaustive impraticable au-delà de quelques dizaines de points.

Pour une instance comportant une centaine de destinations réparties entre plusieurs dépôts, comme dans le cas présent, la recherche de la solution strictement optimale est donc hors de portée en pratique. L'objectif réaliste n'est pas d'obtenir *la* meilleure solution au sens mathématique absolu, mais une solution de **très bonne qualité**, proche de l'optimum, calculée en un temps compatible avec un usage opérationnel.

Cette caractéristique fondamentale du problème conditionne directement le choix des méthodes de résolution : plutôt que des méthodes exactes, dont le temps de calcul explose avec la taille du problème, on privilégie en général des méthodes approchées capables de fournir rapidement de bonnes solutions. Ces différentes approches, ainsi que le positionnement de l'outil retenu pour ce projet, font l'objet de la section suivante.

1.4 Approches de résolution

La complexité du problème de tournées, évoquée précédemment, a donné lieu à une littérature abondante proposant des méthodes de résolution très variées. On les regroupe habituellement en deux grandes familles : les méthodes exactes, qui garantissent l'optimalité au prix d'un temps de calcul potentiellement prohibitif, et les méthodes approchées, qui privilégient la rapidité et la qualité pratique des solutions. Cette section présente ces deux familles, puis situe l'outil retenu pour le projet.

1.4.1 Méthodes exactes

Les méthodes exactes visent à déterminer la solution strictement optimale du problème. Elles reposent généralement sur une formulation en programmation linéaire en nombres entiers, résolue par des techniques telles que le *branch-and-bound*, le *branch-and-cut* ou le *branch-and-price*, qui explorent l'espace des solutions de manière arborescente tout en élaguant les branches ne pouvant conduire à une meilleure solution [6].

Ces méthodes présentent l'avantage décisif de fournir une garantie d'optimalité. Toutefois,

en raison de la nature NP-difficile du problème, leur temps de calcul croît très rapidement avec la taille de l'instance. En pratique, elles ne permettent de résoudre de façon fiable que des instances de taille limitée, de l'ordre de quelques dizaines à une ou deux centaines de clients selon les variantes et les contraintes. Pour un problème industriel comportant une centaine de destinations réparties sur plusieurs dépôts et soumis à des contraintes additionnelles, le recours exclusif à une méthode exacte n'est donc pas envisageable dans un cadre opérationnel où la solution doit être obtenue en quelques secondes ou minutes.

1.4.2 Heuristiques et métaheuristiques

Face à ces limites, les méthodes approchées ne cherchent pas à garantir l'optimum, mais à produire rapidement une solution de bonne qualité. On distingue deux niveaux.

Les **heuristiques** sont des procédures spécifiques, souvent inspirées du bon sens, qui construisent ou améliorent une solution selon des règles simples. L'une des plus anciennes et des plus connues est l'algorithme des gains de Clarke et Wright, proposé en 1964, qui construit les tournées en fusionnant progressivement les trajets offrant les plus fortes économies de distance [8]. Ces heuristiques sont très rapides, mais la qualité de leurs solutions reste variable et peut rester éloignée de l'optimum.

Les **métaheuristiques** constituent un cadre de recherche plus général, capable de guider une exploration intelligente de l'espace des solutions pour échapper aux optima locaux. Parmi les plus employées pour le VRP figurent la recherche taboue, le recuit simulé, les algorithmes génétiques ou encore la recherche locale à voisinage large [9]. Elles offrent en général le meilleur compromis entre qualité de la solution et temps de calcul, et constituent aujourd'hui l'approche de référence pour les instances de grande taille rencontrées en pratique.

1.4.3 Positionnement et choix d'OR-Tools

Le choix d'une méthode de résolution pour ce projet a été guidé par un impératif pratique : disposer d'un outil capable de traiter un MDVRP avec contraintes, à l'échelle d'une centaine de destinations, en un temps compatible avec un usage opérationnel, sans avoir à réimplémenter de zéro des algorithmes complexes.

La solution retenue est **Google OR-Tools**, une bibliothèque d'optimisation open source largement utilisée, aussi bien dans l'industrie que dans le monde académique, pour la résolution de problèmes de tournées [10]. Son module dédié au routage repose précisément sur les approches décrites ci-dessus : il construit une première solution à l'aide d'une heuristique, puis l'améliore par une métaheuristique de recherche locale, dans la limite d'un budget de temps fixé par l'utilisateur. Ce fonctionnement correspond exactement au compromis recherché entre qualité et rapidité.
OR-Tools présente en outre plusieurs atouts déterminants pour le projet : une prise en charge native des contraintes de capacité et des configurations multi-dépôts, une grande souplesse dans la modélisation de contraintes spécifiques, et une intégration aisée dans un environnement Python, cohérent avec le reste de la pile technologique. Ces raisons ont motivé son adoption comme moteur d'optimisation de l'application, dont la conception détaillée fait l'objet du chapitre suivant.

1.5 Objectifs et méthodologie du stage

Après avoir posé le cadre métier et théorique du projet, cette section précise les objectifs assignés au stage et le périmètre exact du problème traité, puis décrit la méthodologie de travail adoptée tout au long de la réalisation ainsi que la manière dont les choix effectués ont été tracés.

1.5.1 Objectifs et périmètre du projet

L'objectif général du stage est de concevoir et de réaliser une application web d'aide à la décision pour l'optimisation des tournées de livraison, dans le contexte multi-dépôts décrit précédemment. L'application doit permettre de calculer automatiquement des tournées de bonne qualité, de les visualiser sur une carte et d'en évaluer les principaux indicateurs de performance, tout en restant utilisable par un opérateur non spécialiste de l'optimisation.
Cet objectif général se décline en plusieurs objectifs spécifiques :

— modéliser le problème de tournées multi-dépôts et l'ensemble de ses contraintes opérationnelles (capacité des véhicules, compatibilité des marchandises, rattachement à un dépôt d'origine, fractionnement des livraisons de gros volume) ;

— constituer des données de distance réalistes reflétant le réseau routier réel, plutôt que de simples distances à vol d'oiseau ;
— mettre en œuvre un moteur d'optimisation capable de produire une solution en un temps compatible avec un usage opérationnel ;
— enrichir l'outil d'une couche d'analyse de données (regroupement géographique des destinations, estimation de la durée des tournées) ;
— restituer les résultats via une interface web comprenant une visualisation cartographique et un tableau de bord d'indicateurs.

Le périmètre du problème traité est le suivant : **cinq dépôts** répartis sur le territoire tunisien, une centaine de destinations à desservir, regroupées en environ **cent vingt lots** de livraison, et une flotte d'une **dizaine de véhicules** à capacité limitée. Ce périmètre, représentatif d'un cas d'usage réel de l'entreprise, est suffisamment vaste pour rendre l'optimisation nécessaire, tout en restant maîtrisable dans le cadre d'un stage.

1.5.2 Méthodologie de travail

La réalisation du projet a suivi une démarche **itérative et incrémentale**, structurée en étapes successives. Plutôt que de développer l'ensemble du système d'un seul tenant, le travail a été découpé en briques fonctionnelles indépendantes, développées et validées l'une après l'autre.

Cette démarche repose sur deux principes appliqués systématiquement. D'une part, **chaque choix de conception est validé avant sa mise en œuvre**, afin de ne pas engager de développement sur une base incertaine. D'autre part, **chaque brique de code est testée et vérifiée avant de passer à la suivante**, ce qui permet de détecter les erreurs au plus tôt et de garder à tout moment un système fonctionnel. Le suivi régulier avec l'encadrant a ponctué cette progression, chaque étape importante faisant l'objet d'une validation avant de poursuivre.

Sur le plan technique, cette organisation s'est appuyée sur un outil de gestion de version (Git), garantissant la traçabilité des évolutions du code et la possibilité de revenir à un état antérieur en cas de besoin.

1.5.3 Traçabilité des choix de conception

La conception d'un système soumis à de nombreuses contraintes suppose de trancher un grand nombre de questions techniques, dont chacune peut influencer le comportement de l'ensemble. Afin de garder la maîtrise de ces décisions et de pouvoir les justifier a posteriori, une démarche de **traçabilité** a été adoptée tout au long du projet.
Chaque choix d'architecture ou de modélisation significatif a été consigné, avec sa justification, sous la forme d'une décision identifiée. De la même manière, les points nécessitant un arbitrage de l'entreprise (précisions sur les données, interprétation de certaines règles métier) ont été recensés comme questions ouvertes, soumises à l'encadrant. Cette discipline présente un double intérêt : elle assure la cohérence des décisions dans le temps, en évitant de revenir involontairement sur un choix déjà tranché, et elle facilite la rédaction du présent rapport, en fournissant un historique clair des raisons ayant guidé la conception.

1.6 Conclusion

Ce premier chapitre a posé le cadre général du projet. Il a d'abord présenté l'organisme d'accueil, la société Progress Engineering, et le service informatique au sein duquel s'est déroulé le stage. Il a ensuite exposé la problématique métier à l'origine du projet : l'optimisation de tournées de livraison dans un contexte multi-dépôts couvrant le territoire tunisien. Le cadre théorique du problème a été introduit, du problème de tournées de véhicules classique jusqu'à sa variante multi-dépôts avec contrainte de capacité, en soulignant sa complexité NP-difficile qui justifie le recours à des méthodes de résolution approchées, et en particulier à la bibliothèque OR-Tools retenue pour le projet. Enfin, les objectifs, le périmètre et la méthodologie de travail ont été précisés.
Le cadre étant désormais établi, le chapitre suivant s'attache à l'**analyse et à la conception** de la solution : il détaille la spécification des besoins, la formulation mathématique du problème et de ses contraintes, puis l'architecture générale du système et les choix de conception de ses différents composants.

Chapitre 2

Analyse et conception

Le cadre général du projet ayant été posé, ce chapitre est consacré à l'analyse et à la conception de la solution. Il précise d'abord les besoins fonctionnels et non fonctionnels ainsi que les acteurs du système. Il formalise ensuite le problème d'optimisation sous forme mathématique, en explicitant la fonction objectif et l'ensemble des contraintes. Il présente enfin l'architecture générale retenue, la conception du modèle de données et celle des principaux composants métier : le solveur, la couche d'analyse de données et la cartographie.

2.1 Spécification des besoins

Cette section identifie ce que le système doit accomplir et dans quelles conditions. Elle distingue les besoins fonctionnels, qui décrivent les services attendus, des besoins non fonctionnels, qui portent sur la qualité de ces services, avant de préciser les acteurs et les principaux cas d'utilisation.

2.1.1 Besoins fonctionnels

Les besoins fonctionnels décrivent les fonctionnalités que l'application doit offrir à ses utilisateurs. Ils ont été dégagés à partir de la problématique métier et des échanges avec l'encadrant. Le système doit permettre de :

— saisir et gérer les livraisons à effectuer, en les regroupant en lots caractérisés par un volume, un dépôt d'origine et une destination ;
— constituer une *vague* de livraisons, c'est-à-dire un ensemble de lots à planifier ensemble lors d'une même optimisation ;
— lancer le calcul d'optimisation des tournées à partir d'une vague donnée, en tenant compte de l'ensemble des contraintes opérationnelles ;
— consulter le résultat d'une optimisation sous forme détaillée : tournées par véhicule, ordre des visites, lots affectés et lots éventuellement non servis ;
— visualiser les tournées sur une carte interactive, avec un code couleur distinguant les destinations servies, abandonnées ou hors périmètre ;
— consulter un tableau de bord d'indicateurs de performance synthétisant les résultats d'une optimisation (distances, taux de service, etc.) ;
— évaluer manuellement une affectation proposée par l'utilisateur, afin d'en vérifier la faisabilité et d'en estimer le coût.

2.1.2 Besoins non fonctionnels

Les besoins non fonctionnels portent sur les qualités attendues du système, indépendamment des fonctionnalités elles-mêmes. Les principaux retenus pour ce projet sont :

— **Performance** : le calcul d'une optimisation, pour une instance d'une centaine de destinations, doit aboutir en un temps compatible avec un usage opérationnel, de l'ordre de la minute.
— **Fiabilité** : les résultats produits doivent être reproductibles et cohérents ; une même vague optimisée dans les mêmes conditions doit conduire à des résultats stables.
— **Utilisabilité** : l'application doit rester accessible à un opérateur non spécialiste de l'optimisation, au moyen d'une interface claire et d'une restitution visuelle des résultats.
— **Maintenabilité** : le code doit être organisé de manière modulaire et documenté, afin de faciliter son évolution ultérieure.
— **Évolutivité** : l'architecture doit permettre le passage d'une base de données locale à une base de production, ainsi que l'ajout de nouvelles contraintes ou fonctionnalités sans refonte majeure.

2.1.3 Acteurs et cas d'utilisation

Le système s'adresse principalement à un acteur unique, l'**opérateur logistique**, chargé de planifier les livraisons. Cet acteur interagit avec l'application pour saisir les lots, constituer les vagues, lancer les optimisations et consulter les résultats sous leurs différentes formes (tableaux, carte, indicateurs).

Les principaux cas d'utilisation se résument ainsi : *saisir les lots et créer une vague*, *lancer une optimisation*, *consulter les tournées d'un résultat*, *visualiser les tournées sur la carte*, *consulter le tableau de bord* et *évaluer une affectation manuelle*. Ces cas d'utilisation correspondent directement aux besoins fonctionnels précédemment énumérés et structureront la conception de l'interface et des services applicatifs. Le diagramme de cas d'utilisation correspondant est présenté à la figure 2.1.

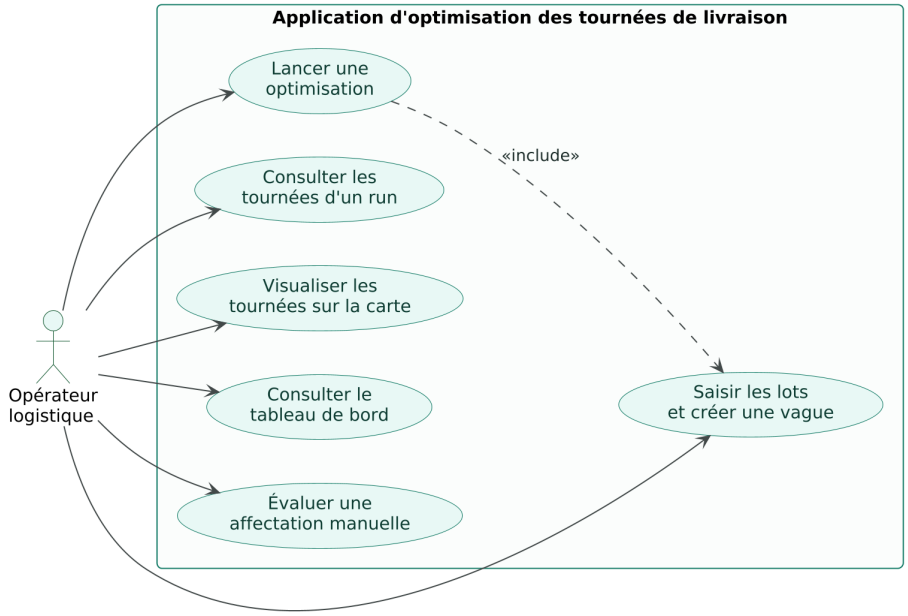


FIGURE 2.1 – Diagramme de cas d'utilisation de l'application

2.2 Formulation mathématique du problème

Avant d'aborder la conception logicielle, il est nécessaire de formaliser le problème d'optimisation sous-jacent. Cette section précise les notations et les données, puis exprime la fonction objectif et l'ensemble des contraintes du modèle. La formulation retenue reprend

le cadre classique du problème de tournées avec contrainte de capacité [6], enrichi des contraintes spécifiques au contexte de l'entreprise.

2.2.1 Notations et données du problème

Le problème se modélise sur un graphe $G = (V, A)$, où $V = \mathcal{D} \cup \mathcal{C}$ regroupe l'ensemble des dépôts $\mathcal{D}$ et l'ensemble des destinations $\mathcal{C}$, et où A est l'ensemble des arcs reliant ces sommets. On note :

- $\mathcal{L}$ l'ensemble des lots à livrer et $\mathcal{K}$ l'ensemble des véhicules disponibles ;
- pour chaque lot $l \in \mathcal{L}$: son volume v_l, son dépôt d'origine $\sigma(l) \in \mathcal{D}$, sa destination $\delta(l) \in \mathcal{C}$, et un indicateur $\rho_l \in \{0, 1\}$ valant 1 si le lot requiert un véhicule à caisson spécial ;
- pour chaque véhicule $k \in \mathcal{K}$: sa capacité Q_k, son dépôt d'attache $o(k) \in \mathcal{D}$, et un indicateur $\gamma_k \in \{0, 1\}$ valant 1 s'il est équipé d'un caisson spécial ;
- c_{ij} la distance routière de déplacement entre les sommets i et j ;
- P une pénalité forfaitaire associée à l'abandon d'un lot.

Les variables de décision sont les suivantes :

- $x_{ijk} \in \{0, 1\}$ vaut 1 si le véhicule k emprunte directement l'arc (i, j) ;
- $y_{lk} \in \{0, 1\}$ vaut 1 si le lot l est affecté au véhicule k ;
- $z_l \in \{0, 1\}$ vaut 1 si le lot l n'est pas servi (abandonné).

2.2.2 Fonction objectif

L'objectif du problème est de minimiser le coût total, défini comme la somme des distances parcourues par l'ensemble des véhicules, augmentée d'une pénalité pour chaque lot non servi :

$$\min \quad \sum_{k \in \mathcal{K}} \sum_{(i,j) \in A} c_{ij}\, x_{ijk} \;+\; P \sum_{l \in \mathcal{L}} z_l. \tag{2.1}$$

Le premier terme traduit l'objectif principal de réduction du kilométrage total. Le second terme, associé à une pénalité P de valeur élevée, décourage fortement l'abandon de lots : le solveur ne renoncera à servir un lot que lorsque aucune affectation faisable n'existe, faisant de l'abandon une solution de dernier recours plutôt qu'un moyen commode de réduire la distance.

2.2.3 Contraintes du modèle

Le modèle est soumis à un ensemble de contraintes traduisant les règles métier du problème.

2.2.3.0.1 Affectation des lots. Chaque lot doit être soit affecté à exactement un véhicule, soit déclaré non servi :

$$\sum_{k \in \mathcal{K}} y_{lk} + z_l = 1, \qquad \forall l \in \mathcal{L}. \tag{2.2}$$

2.2.3.0.2 Capacité des véhicules. Le volume total des lots affectés à un véhicule ne peut dépasser sa capacité :

$$\sum_{l \in \mathcal{L}} v_l \, y_{lk} \leq Q_k, \qquad \forall k \in \mathcal{K}. \tag{2.3}$$

2.2.3.0.3 Cohérence de la source. Un lot ne peut être pris en charge que par un véhicule dont le dépôt d'attache coïncide avec le dépôt d'origine du lot :

$$y_{lk} = 0, \qquad \forall l \in \mathcal{L},\ \forall k \in \mathcal{K} \text{ tel que } o(k) \neq \sigma(l). \tag{2.4}$$

2.2.3.0.4 Compatibilité caisson. Un lot nécessitant un caisson spécial ne peut être affecté qu'à un véhicule équipé :

$$\rho_l \, y_{lk} \leq \gamma_k, \qquad \forall l \in \mathcal{L},\ \forall k \in \mathcal{K}. \tag{2.5}$$

2.2.3.0.5 Continuité des tournées. Chaque véhicule utilisé quitte son dépôt d'attache, dessert les destinations de ses lots et revient à ce même dépôt. La conservation du flux impose qu'un véhicule entrant dans un sommet en ressorte :

$$\sum_{j:(i,j) \in A} x_{ijk} = \sum_{j:(j,i) \in A} x_{jik}, \qquad \forall i \in V,\ \forall k \in \mathcal{K}. \tag{2.6}$$

Le lien entre affectation et routage est assuré en imposant que la destination d'un lot affecté à un véhicule soit effectivement visitée par ce véhicule. Enfin, des contraintes d'élimination des sous-tours garantissent que chaque tournée forme un unique circuit connecté au dépôt, sans boucle isolée.

Le retour au dépôt d'*origine* constitue un choix de modélisation important : un véhicule ne termine pas sa tournée au dépôt le plus proche, mais revient à son point de départ, ce qui reflète fidèlement la réalité opérationnelle et évite de sous-estimer les distances.

2.2.3.0.6 Traitement des lots de gros volume. Un lot dont le volume excède la capacité d'un véhicule ne peut être servi tel quel. Un tel lot est alors **fractionné**, en phase de prétraitement, en $\lceil v_l / c_{\max} \rceil$ parts de volume égal, où $c_{\max}$ désigne la capacité maximale disponible. Ces parts sont traitées comme des lots distincts, mais liées entre elles par une règle de *tout ou rien* : soit toutes les parts sont servies, soit aucune ne l'est, afin de ne jamais livrer une fraction incomplète d'une même commande.

2.3 Architecture générale de la solution

Cette section décrit l'organisation d'ensemble du système, les technologies retenues pour chaque couche, et la manière dont ces couches communiquent entre elles.

2.3.1 Vue d'ensemble

L'application adopte une architecture en couches, séparant nettement la présentation, la logique métier et la persistance des données. Quatre composants principaux structurent le système :

- un **backend** exposant une interface de programmation (API) qui centralise la logique applicative et l'accès aux données ;
- un **moteur d'optimisation** (solveur), sollicité par le backend pour calculer les tournées ;
- une **couche d'analyse de données** assurant le regroupement géographique des destinations et l'estimation de durées ;
- un **frontend** web fournissant l'interface utilisateur, y compris la visualisation cartographique.

Cette séparation présente l'avantage de rendre chaque composant modifiable indépendamment des autres : le solveur peut évoluer sans impact sur l'interface, et la base de

données peut être remplacée sans toucher à la logique métier. L'organisation générale de ces composants est illustrée à la figure 2.2.

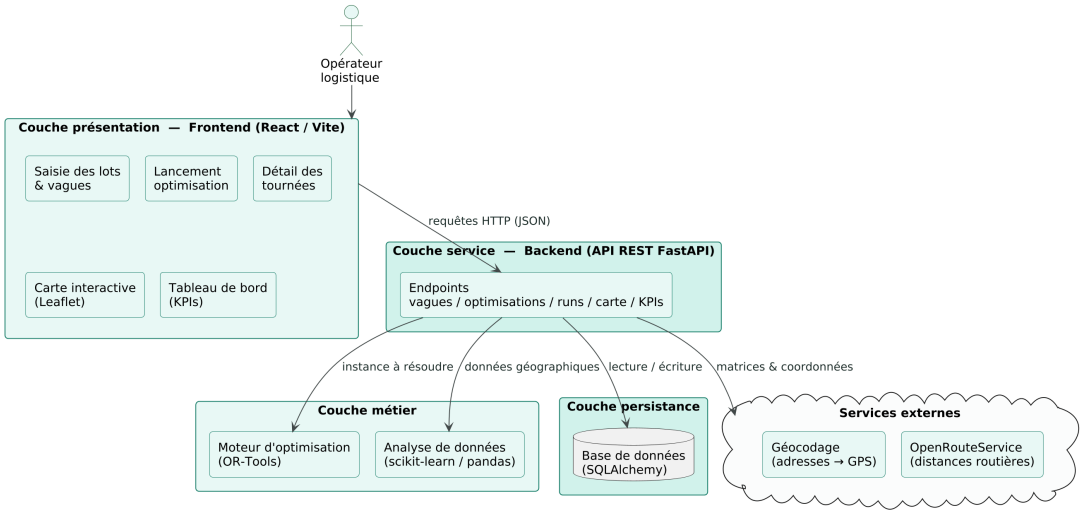


FIGURE 2.2 – Architecture en couches de la solution

2.3.2 Choix de la pile technologique

Les technologies ont été choisies pour leur adéquation au problème, leur maturité et leur cohérence d'ensemble autour du langage Python côté serveur.

Le backend repose sur le framework **FastAPI** [11], retenu pour ses performances, sa simplicité et sa génération automatique de documentation d'API. L'accès aux données est assuré par le pont objet-relationnel **SQLAlchemy** [12], associé à un outil de migration de schéma, ce qui permet de faire évoluer la base de manière contrôlée. La base de données est une base légère de type fichier en phase de développement, conçue pour être remplacée par une base relationnelle de production sans modification du code métier.

Le moteur d'optimisation s'appuie sur la bibliothèque **Google OR-Tools** [10], dont le choix a été justifié au chapitre précédent. La couche d'analyse de données utilise **scikit-learn** [13] pour les algorithmes d'apprentissage automatique et la bibliothèque `pandas` pour la manipulation des données.

Côté client, le frontend est développé avec la bibliothèque **React** [14], associée à un outil de construction moderne, et la visualisation cartographique s'appuie sur la bibliothèque **Leaflet** [15]. Enfin, les distances routières réalistes entre les points sont obtenues auprès du service **OpenRouteService** [16], et le géocodage des adresses s'appuie sur un service

de conversion adresse–coordonnées.

2.3.3 Flux de données et interactions entre couches

Le déroulement d'une optimisation illustre l'interaction entre les couches. L'opérateur saisit les lots et constitue une vague depuis le frontend, qui transmet ces informations au backend via l'API. Lors du lancement d'une optimisation, le backend rassemble les données nécessaires (lots de la vague, véhicules, matrice de distances), les transmet au solveur, puis récupère la solution calculée. Cette solution est ensuite persistée en base, sous forme de tournées et d'affectations, et rendue disponible via l'API. Le frontend interroge alors le backend pour afficher les résultats, que ce soit sous forme de tableaux, de carte ou d'indicateurs. La couche d'analyse de données, quant à elle, exploite les données géographiques pour produire le zonage et les estimations, consommés par les mêmes canaux. La figure 2.3 détaille les échanges entre couches lors d'une optimisation.

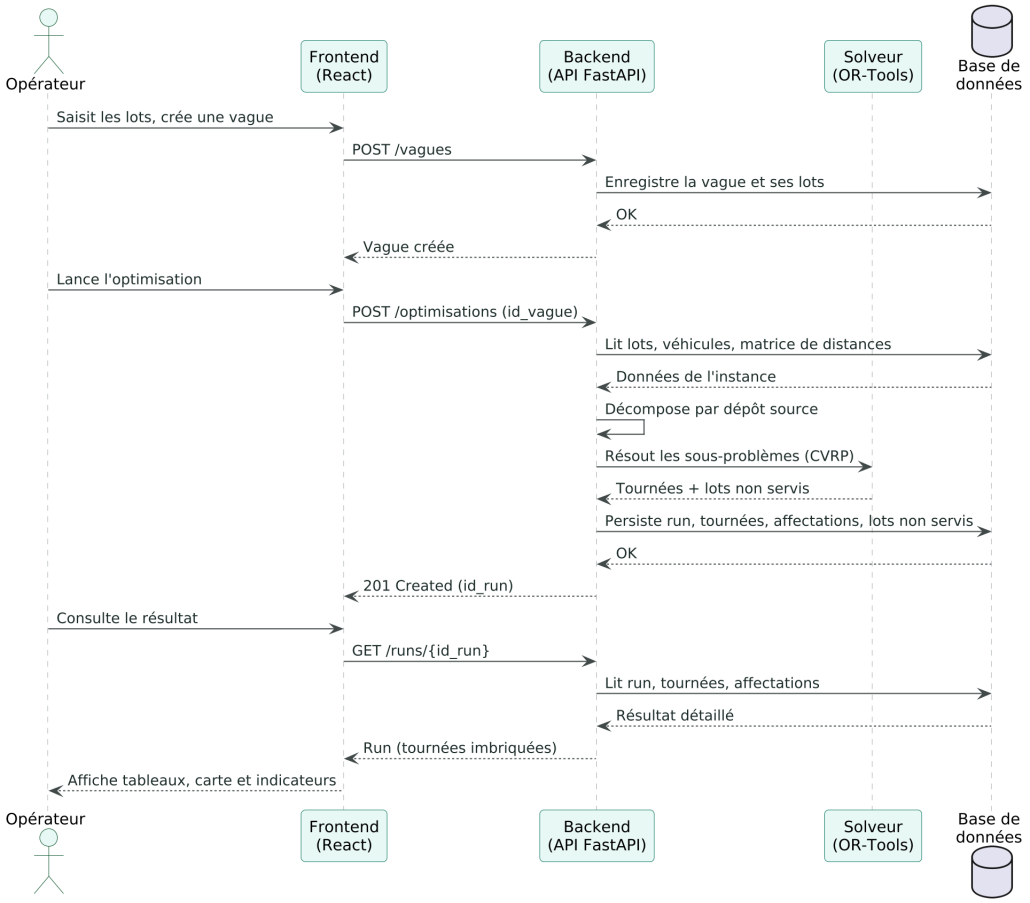


FIGURE 2.3 – Diagramme de séquence du déroulement d'une optimisation

2.4 Conception du modèle de données

Le modèle de données constitue le socle du système. Sa conception a été guidée par un souci de cohérence et de non-redondance, afin d'éviter les incohérences difficiles à diagnostiquer.

2.4.1 Schéma relationnel

Le schéma s'organise autour de quelques entités principales. Les **stations** (dépôts et destinations) portent les informations géographiques de référence. Les **lots** décrivent les livraisons à effectuer, avec leur volume, leur dépôt d'origine et leur destination. Une **vague** regroupe les lots à optimiser ensemble. Le résultat d'une optimisation est matérialisé par un **run**, auquel sont rattachées des **tournées** (une par véhicule utilisé), elles-mêmes composées d'**affectations** ordonnées reliant chaque étape à sa destination. Enfin, une entité dédiée recense les **lots non servis**, en précisant pour chacun la raison de l'abandon. La structure complète du modèle de données et les relations entre ces entités sont représentées à la figure 2.4.

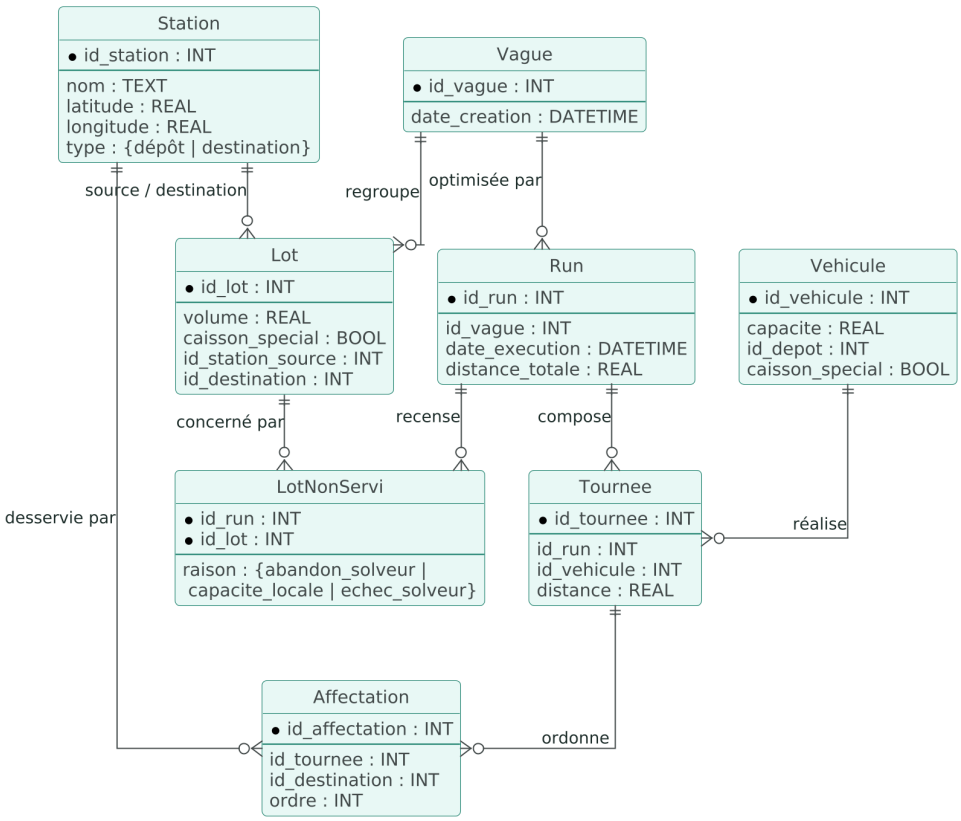


FIGURE 2.4 – Modèle de données de l'application

2.4.2 Principe ” une donnée à un seul endroit ”

Un principe directeur a présidé à la conception du schéma : chaque information ne doit être stockée qu'à un seul endroit. Par exemple, l'identifiant d'un run n'est porté que par la tournée, et non répété sur chaque affectation ; l'appartenance d'une affectation à un run se déduit par lecture de la tournée correspondante. Cette règle évite la duplication d'information et, surtout, le risque que deux copies d'une même donnée divergent au fil des évolutions. De la même manière, le statut d'une destination sur la carte (servie, abandonnée, hors périmètre) est déterminé à partir d'une source unique de vérité, la table des lots non servis, plutôt que reconstruit indépendamment à plusieurs endroits, ce qui garantit la cohérence de l'affichage.

2.4.3 Séparation entre schéma de données et logique du solveur

Une distinction nette a été maintenue entre les données telles qu'elles sont stockées et les données telles qu'elles sont fournies au solveur. Le solveur travaille sur des structures qui lui sont propres, construites à la volée à partir de la base : matrice de distances éventuellement ajustée, index internes des nœuds, regroupement des lots par dépôt d'origine. Ces transformations ne sont pas persistées et restent confinées à la couche d'optimisation. Cette séparation permet de modifier la manière dont le problème est présenté au solveur, par exemple en ajustant la matrice de distances, sans altérer les données de référence stockées en base.

2.5 Conception des composants métier

Cette section présente les choix de conception propres aux trois composants métier du système : le solveur, la couche d'analyse de données et la cartographie.

2.5.1 Conception du solveur

La conception du solveur repose sur une idée structurante : la contrainte de cohérence de la source rend les livraisons de dépôts différents totalement indépendantes les unes des

autres. Un lot issu d'un dépôt donné ne pouvant être servi que par un véhicule de ce même dépôt, le problème global se **décompose naturellement en autant de sous-problèmes indépendants qu'il y a de dépôts sources**. Chaque sous-problème est un problème de tournées avec contrainte de capacité, de taille réduite, résolu séparément. Cette décomposition simplifie la modélisation et accélère la résolution, sans perte d'optimalité puisque les sous-problèmes ne partagent aucune ressource. Le principe de cette décomposition est schématisé à la figure 2.5.

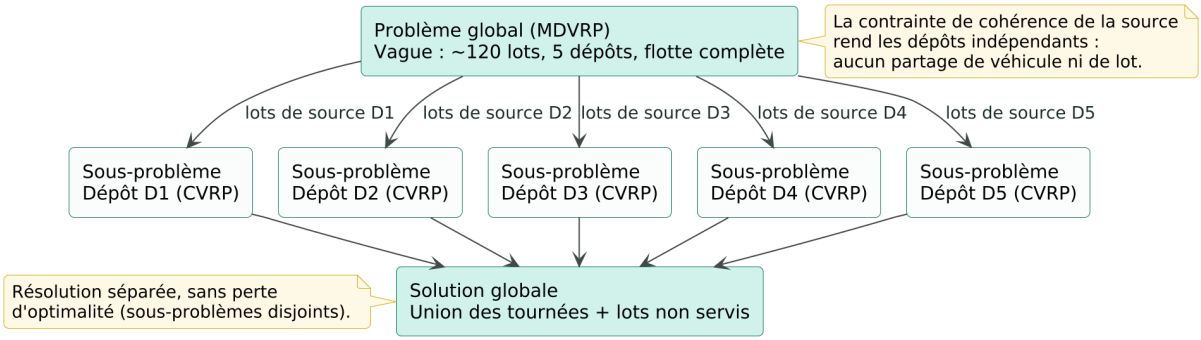


FIGURE 2.5 – Décomposition du problème global en sous-problèmes par dépôt source

Un second choix de conception concerne le traitement des contraintes lors de l'*évaluation* d'une affectation proposée manuellement par l'utilisateur. Dans ce mode, les contraintes sont vérifiées de manière **non bloquante** : plutôt que de rejeter une affectation infaisable, le système la calcule tout de même et signale les violations constatées, laissant à l'opérateur le soin d'apprécier la situation. Enfin, lorsqu'un lot ne peut être servi, la raison de l'abandon est **typée et conservée** (capacité insuffisante, échec du solveur, etc.), afin de fournir un diagnostic exploitable plutôt qu'un simple constat d'échec.

2.5.2 Conception de la couche d'analyse de données

La couche d'analyse de données a été conçue comme un apport complémentaire au moteur d'optimisation, à visée d'aide à la décision. Elle comporte deux volets.

Le premier est un **zonage géographique** des destinations, obtenu par un algorithme de regroupement non supervisé. L'objectif est de faire émerger des zones cohérentes du point de vue géographique, indépendamment de la partition induite par les dépôts, afin d'offrir une grille de lecture supplémentaire du territoire desservi. La conception prévoit une validation de la qualité du regroupement par un indicateur quantitatif et par recoupement

avec une seconde méthode de regroupement.
Le second volet est un modèle d'**estimation de la durée** d'une tournée à partir de ses caractéristiques (distance, nombre d'arrêts, volume, part de lots à caisson). Ce modèle est conçu comme illustratif : en l'absence de données réelles de durée en quantité suffisante, il ne saurait être utilisé en production et doit être présenté comme une preuve de concept, dont les résultats devront être revalidés sur des données de terrain.

2.5.3 Conception de la cartographie

La cartographie constitue le principal moyen de restitution visuelle des résultats. Sa conception vise à donner à l'opérateur une lecture immédiate de la solution. Les tournées sont tracées sur un fond de carte interactif, chaque véhicule étant représenté par un itinéraire distinct.
Un **code couleur** des destinations a été conçu pour synthétiser leur statut vis-à-vis d'une optimisation : une couleur pour les destinations effectivement servies dans le run, une autre pour celles dont le lot figurait dans la vague mais n'a pas été servi, et une troisième pour celles situées hors du périmètre de la vague. Une règle de priorité tranche les cas ambigus lorsqu'une destination pourrait relever de plusieurs statuts. Ce code couleur, alimenté par la source unique de vérité évoquée plus haut, garantit une représentation cohérente entre les différentes vues de l'application.

2.6 Conclusion

Ce chapitre a traduit la problématique du projet en une solution conçue. Il a d'abord dégagé les besoins fonctionnels et non fonctionnels ainsi que les acteurs du système. Il a ensuite formalisé le problème d'optimisation, en explicitant la fonction objectif, la minimisation de la distance totale assortie d'une pénalité d'abandon, et l'ensemble des contraintes : capacité, cohérence de la source, compatibilité caisson, continuité des tournées avec retour au dépôt d'origine, et fractionnement des lots de gros volume. Il a enfin présenté l'architecture en couches retenue, la pile technologique associée, un modèle de données guidé par le principe de non-redondance, et les choix de conception des composants métier, au premier rang desquels la décomposition du problème par dépôt source.

La solution étant désormais entièrement conçue, le chapitre suivant en aborde la **réalisation**, en se concentrant sur le cœur du système : la préparation des données géographiques, l'implémentation du solveur et de ses contraintes, et la mise en œuvre de la couche d'analyse de données.

Chapitre 3

Réalisation du cœur d'optimisation

Après avoir défini l'architecture et la conception de la solution, ce chapitre en présente la réalisation, centrée sur le cœur du système. Il décrit d'abord l'environnement de développement mis en place. Il détaille ensuite la préparation des données géographiques, étape déterminante pour la qualité des résultats, puis l'implémentation du solveur et de ses contraintes. Il expose enfin le mécanisme de fractionnement des lots de gros volume et la couche d'analyse de données qui complète le moteur d'optimisation.

3.1 Environnement de développement

Cette section décrit le cadre matériel et logiciel dans lequel s'est déroulé le développement, ainsi que les pratiques adoptées pour organiser le travail et en assurer la traçabilité.

3.1.1 Poste de travail et gestion de l'environnement

Le développement a été mené sur un poste sous Windows, à l'aide de l'interpréteur de commandes PowerShell. Le langage principal est Python, exécuté dans un environnement virtuel dédié au projet, ce qui isole les dépendances et garantit la reproductibilité de l'installation. L'ensemble des bibliothèques (FastAPI, SQLAlchemy, OR-Tools, scikit-learn, pandas, geopy) y est installé à une version fixée.
L'exécution s'organise autour de plusieurs terminaux fonctionnant en parallèle : l'un pour le serveur de développement du frontend, un autre pour le serveur applicatif backend,

et un troisième pour les tâches ponctuelles (migrations de base, scripts d'analyse). Un point mérite d'être relevé : le serveur backend est lancé *sans* rechargement automatique pendant les mesures de performance du solveur. Le rechargement à chaud, pratique en développement, fausse en effet la mesure du temps de résolution, car il partage le temps processeur ; sa désactivation garantit des mesures de durée fiables.

3.1.2 Gestion de version

Le code est versionné avec Git et hébergé sur un dépôt distant privé. Chaque évolution significative fait l'objet d'un *commit* accompagné d'un message décrivant la modification apportée. Cette pratique assure la traçabilité complète de l'historique du projet et la possibilité de revenir à tout état antérieur en cas de régression. Le travail s'est déroulé sur une branche principale unique, adaptée à un projet individuel.

3.1.3 Organisation du code et rythme d'intégration

Conformément à la démarche itérative décrite au chapitre précédent, le développement a progressé **une brique fonctionnelle à la fois**. Chaque brique n'est intégrée qu'une fois testée et validée, et un commit ne clôt une étape qu'après vérification complète de son bon fonctionnement. Cette discipline, associée à la consignation systématique des choix d'architecture, a permis de conserver à tout moment un système cohérent et fonctionnel, et de documenter au fil de l'eau les décisions qui structurent le reste de ce chapitre.

3.2 Préparation des données géographiques

La qualité d'une optimisation de tournées dépend directement de la fidélité des données de distance qui l'alimentent. Cette section décrit la constitution de ces données, depuis le géocodage des points jusqu'à la construction et la mise en cache des matrices de distances.

3.2.1 Géocodage des destinations

La première étape consiste à associer à chaque point (dépôt ou destination) des coordonnées géographiques exploitables. Ce géocodage s'appuie sur le service Nominatim, fondé

sur les données ouvertes d'OpenStreetMap [17], interrogé au moyen de la bibliothèque geopy. Les coordonnées obtenues sont consolidées dans un fichier de référence, qui fait autorité pour l'ensemble du système : la base de données est peuplée à partir de ce fichier, et jamais l'inverse. Ce principe de source unique évite toute divergence entre les coordonnées utilisées par le solveur et celles affichées sur la carte.

3.2.2 Matrices de distances : géodésique et routière

Une distinction essentielle a été faite entre deux notions de distance. La distance *géodésique* (à vol d'oiseau) est simple à calculer mais irréaliste pour un problème de livraison routière, car elle ignore le réseau des routes. La distance *routière*, au contraire, reflète les trajets réellement empruntables.

Les distances routières sont obtenues auprès du service OpenRouteService [16], qui calcule les itinéraires sur le réseau réel. L'ensemble des distances entre tous les points est précalculé et stocké sous forme d'une matrice de dimension 105×105, conservée dans un fichier binaire pour un accès rapide. Une matrice géodésique de secours est maintenue en parallèle, utilisée comme repli en cas d'indisponibilité du service externe, ce qui garantit que le système reste fonctionnel même hors connexion.

3.2.3 Asymétrie du réseau, plancher de distance et mise en cache

Le recours aux distances routières fait apparaître un phénomène absent des distances à vol d'oiseau : l'**asymétrie**. En raison des sens uniques et de la topologie du réseau, la distance d'un point i vers un point j diffère en général de celle de j vers i. Cette asymétrie est bien réelle dans les données du projet, où l'écart maximal observé entre les deux sens atteint plusieurs dizaines de kilomètres pour un même couple de villes. Le modèle en tient compte en traitant la matrice comme non symétrique.

Un second ajustement concerne les très courtes distances. Un **plancher** est appliqué aux distances inférieures à un seuil de trois kilomètres, afin d'éviter que des points très proches ne soient traités comme quasi confondus par le solveur. Ce plancher est appliqué *uniquement en entrée du solveur*, au travers d'une fonction de transformation dédiée, et n'altère pas la matrice de référence stockée : la séparation entre données de référence et données

du solveur, prévue à la conception, est ainsi respectée.
Enfin, le calcul des matrices étant coûteux, un mécanisme de **mise en cache** évite de le refaire inutilement. Une empreinte numérique (condensé SHA-256) est calculée sur les données d'entrée ; tant que cette empreinte est inchangée, la matrice en cache est réutilisée, et elle n'est recalculée que si les données sous-jacentes évoluent.

3.3 Implémentation du solveur

Le solveur constitue le composant central du système. Cette section décrit sa structure, la traduction des contraintes du modèle dans la bibliothèque OR-Tools, ainsi que les choix relatifs au retour au dépôt et à la gestion des lots non servis.

3.3.1 Décomposition en sous-problèmes par source

Conformément à la conception retenue, le problème global est résolu par décomposition en sous-problèmes indépendants, un par dépôt source. Concrètement, les lots sont regroupés selon leur dépôt d'origine, et un problème de tournées avec contrainte de capacité est construit et résolu séparément pour chaque groupe. Les solutions partielles sont ensuite réunies pour former la solution globale. Cette organisation réduit la taille de chaque instance traitée et simplifie la modélisation, sans perte d'optimalité puisque les sous-problèmes ne partagent aucune ressource.

3.3.2 Modélisation des contraintes dans OR-Tools

Chaque sous-problème est modélisé à l'aide du module de routage d'OR-Tools. La contrainte de capacité est exprimée au moyen d'une *dimension* de capacité, qui cumule le volume des lots le long de chaque tournée et en interdit le dépassement.
La contrainte de compatibilité caisson, qui restreint les véhicules autorisés à servir certains lots, a nécessité un traitement particulier. La méthode native prévue à cet effet s'est révélée inutilisable en raison d'une anomalie dans la couche d'interfaçage entre la bibliothèque

(écrite en C++) et Python. Le contournement adopté consiste à fixer directement l'ensemble des véhicules autorisés pour chaque nœud concerné, en incluant la valeur spéciale -1 qui autorise la non-desserte :

```
# Restreint les vehicules autorises pour un noeud donne
# (-1 autorise l'abandon du noeud)
routing.VehicleVar(index).SetValues(vehicules_autorises + [-1])
```

Cette contrainte de compatibilité a un coût : réserver certains lots à une sous-flotte équipée réduit la marge d'optimisation et allonge sensiblement la distance totale par rapport à un scénario sans contrainte de caisson. Ce surcoût structurel, mesuré lors des essais, est analysé au chapitre suivant.

3.3.3 Retour au dépôt d'origine et budget de temps

Un choix de modélisation important concerne la fin des tournées. Plutôt que de laisser chaque véhicule terminer au dépôt le plus proche, le solveur impose le **retour au dépôt d'origine**, fidèle à la réalité opérationnelle. Ce choix augmente mécaniquement la distance totale par rapport à une modélisation à dépôt d'arrivée libre, mais corrige une sous-estimation qui aurait faussé l'évaluation des tournées.
La stratégie de résolution combine une heuristique de construction rapide et une métaheuristique d'amélioration. La solution initiale est construite par insertion au moindre coût (`PARALLEL_CHEAPEST_INSERTION`), puis améliorée par une recherche locale guidée (`GUIDED_LOCAL_SEARCH`). Un budget de temps est imparti à cette phase d'amélioration :

```
params.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.
       PARALLEL_CHEAPEST_INSERTION)
params.local_search_metaheuristic = (
    routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH
       )
params.time_limit.FromSeconds(45)
```

La durée de ce budget a été calibrée expérimentalement : au-delà d'une quarantaine de secondes, la qualité de la solution n'évolue pratiquement plus, l'amélioration ayant atteint

un plateau. Cette valeur offre donc un bon compromis entre qualité et temps de réponse.

3.3.4 Gestion des lots non servis

Le modèle autorise, en dernier recours, l'abandon d'un lot lorsqu'aucune affectation faisable n'existe. Cet abandon est fortement pénalisé par une constante de valeur élevée dans la fonction objectif, si bien que le solveur ne l'utilise que lorsqu'il n'a pas d'autre choix. Surtout, lorsqu'un lot n'est pas servi, la **raison** de cet abandon est déterminée et conservée en base, dans une table dédiée. Trois motifs sont distingués : l'abandon décidé par le solveur faute de solution globale (`abandon_solveur`), l'incapacité locale liée à un dépassement de capacité au niveau d'un dépôt (`capacite_locale`), et l'échec technique de résolution d'un sous-problème (`echec_solveur`). Ce typage transforme un simple constat d'échec en un diagnostic exploitable, qui pourra être restitué à l'opérateur.

3.4 Fractionnement des lots de gros volume

Certaines livraisons présentent un volume supérieur à la capacité d'un seul véhicule et ne peuvent donc être traitées comme un lot indivisible. Cette section décrit le mécanisme conçu pour les prendre en charge.

3.4.1 Principe et calcul du découpage

Lorsqu'un lot dépasse la capacité maximale disponible, il est **fractionné** en plusieurs parts lors d'une phase de prétraitement, en amont du solveur. Le nombre de parts est déterminé par la formule :

$$k = \left\lceil \frac{v_l}{c_{\max}} \right\rceil, \tag{3.1}$$

où v_l est le volume du lot et $c_{\max}$ la capacité maximale d'un véhicule. Le volume est réparti en k parts égales, chacune traitée par la suite comme un lot autonome, susceptible d'être affecté à un véhicule distinct.

3.4.2 Livraison " tout ou rien "

Fractionner un lot introduit un risque : servir seulement une partie des parts reviendrait à livrer une commande incomplète, ce qui n'a pas de sens sur le plan métier. Pour l'éviter, les parts issues d'un même lot sont liées par une contrainte de **tout ou rien** : soit toutes les parts sont servies, soit aucune ne l'est. Cette contrainte est réalisée en couplant les variables d'activation des parts dans le modèle, de sorte que le solveur ne puisse pas en retenir un sous-ensemble strict.

3.4.3 Validation du comportement

Le comportement du mécanisme a été vérifié sur des cas de test comportant des lots volumineux. Le fractionnement produit bien le nombre de parts attendu, et la contrainte de tout ou rien est respectée dans tous les cas. Il convient de noter que le fractionnement a un coût de distance intrinsèque : les parts d'un même lot étant livrées par des véhicules différents, chacun effectue son propre aller-retour, ce qui augmente la distance totale par rapport à une livraison hypothétique en un seul voyage. Ce surcoût est une conséquence inévitable de la contrainte physique de capacité, et non un défaut de la solution.

3.5 Couche d'analyse de données

En complément du moteur d'optimisation, une couche d'analyse de données apporte des éclairages supplémentaires à visée d'aide à la décision. Elle comprend un volet de zonage géographique et un volet d'estimation de durée.

3.5.1 Zonage géographique par k-means

Le zonage vise à regrouper les destinations en zones géographiquement cohérentes. Il s'appuie sur l'algorithme des k-moyennes (*k-means*) [18], appliqué aux coordonnées des destinations. Une précaution méthodologique a été prise concernant la projection : aux latitudes de la Tunisie (environ 35,7N), un degré de longitude correspond à une distance bien plus courte qu'un degré de latitude. Utiliser les coordonnées brutes introduirait une

distorsion est-ouest importante. Une projection équirectangulaire corrige cet effet en pondérant la longitude par le cosinus de la latitude, de sorte que les distances du plan reflètent les distances réelles au sol.

Le nombre de zones a été fixé à sept, valeur pour laquelle la qualité du regroupement, mesurée par l'indice de silhouette, est la plus élevée. Les zones ainsi obtenues couvrent le territoire du nord au sud de manière cohérente, indépendamment de la partition induite par les dépôts. Le résultat de ce zonage est présenté à la figure 3.1.

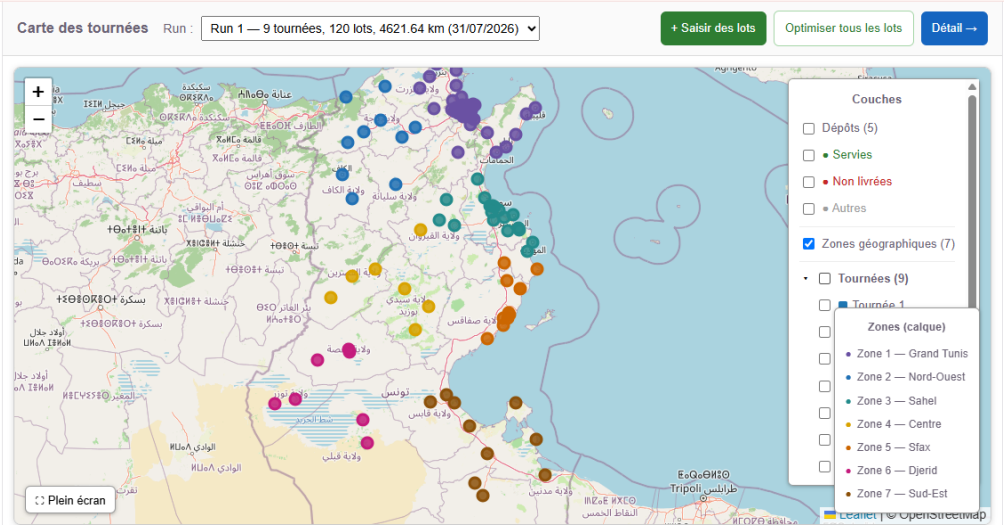


FIGURE 3.1 – Zonage géographique des destinations en sept zones

3.5.2 Validation par l'indice de silhouette et recoupement

La qualité du zonage a été évaluée quantitativement au moyen de l'indice de silhouette [19], qui mesure, pour chaque point, à quel point il est proche des points de sa propre zone comparativement aux zones voisines. L'indice a été calculé pour un nombre de zones variant de deux à neuf ; les valeurs obtenues sont reportées à la figure 3.2.

Ces valeurs se situent toutes dans une bande étroite (entre 0,47 et 0,54), ce qui indique une structure géographique réelle mais sans découpage unique s'imposant nettement. La valeur maximale (0,537) est atteinte pour $k = 2$, mais elle correspond à un découpage trivial du territoire en deux blocs nord et sud, sans intérêt opérationnel. Parmi les découpages suffisamment fins pour être exploitables ($k \geq 4$), c'est $k = 7$ qui obtient le meilleur indice (0,532), tout en produisant des zones géographiquement lisibles ; c'est cette valeur qui a été retenue.

Un fait mérite d'être souligné : le découpage en cinq zones, qui coïnciderait avec le nombre de dépôts, correspond à un *creux local* de l'indice (0,514), inférieur à celui de ses voisins immédiats $k = 4$ (0,523) et $k = 6$ (0,522). Autrement dit, les regroupements géographiques naturels ne s'alignent pas sur la partition induite par les dépôts, ce qui justifie de traiter le zonage comme une grille de lecture analytique distincte. Ce constat a été consolidé par recoupement avec une seconde méthode de regroupement hiérarchique, la méthode de Ward [20], dont les zones obtenues concordent largement avec celles des k-moyennes.

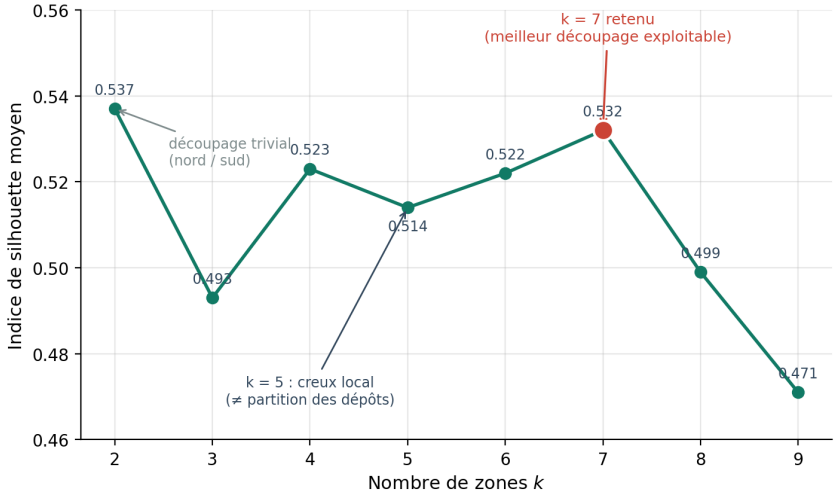


FIGURE 3.2 – Indice de silhouette moyen selon le nombre de zones

3.5.3 Estimation de la durée des tournées

Le second volet est un modèle d'estimation de la durée d'une tournée à partir de quatre caractéristiques : la distance parcourue, le nombre d'arrêts, le volume total transporté et la part de lots à caisson spécial. Un modèle de régression linéaire a été entraîné sur ces variables.

Il est essentiel de préciser la nature de ce modèle. En l'absence de données réelles de durée en quantité suffisante, l'entraînement a été réalisé sur un jeu de données **synthétique**. Les performances obtenues, bien qu'excellentes en apparence (coefficient de détermination R^2 voisin de 0,99, supérieur à celui d'un modèle de forêt aléatoire testé en comparaison), ne sauraient donc être interprétées comme une validation opérationnelle. Ce modèle constitue une **preuve de concept** : il démontre la faisabilité de l'approche et la pertinence des variables retenues, mais devra être ré-entraîné et revalidé sur des données de terrain avant

tout usage en production. Cette limite est assumée et rappelée à chaque présentation des résultats correspondants.

3.6 Conclusion

Ce chapitre a présenté la réalisation du cœur du système. Il a décrit l'environnement de développement et les pratiques de traçabilité, puis la préparation des données géographiques : géocodage, construction des matrices de distances routières, prise en compte de l'asymétrie du réseau et mise en cache. Il a ensuite détaillé l'implémentation du solveur, décomposition par dépôt, traduction des contraintes dans OR-Tools, retour au dépôt d'origine et gestion typée des lots non servis, ainsi que le mécanisme de fractionnement des lots volumineux. Il s'est achevé sur la couche d'analyse de données, avec un zonage géographique validé par l'indice de silhouette et un modèle d'estimation de durée présenté comme preuve de concept.

Le moteur d'optimisation et ses composants analytiques étant désormais opérationnels, le chapitre suivant aborde la **réalisation applicative** : l'exposition de ces traitements au travers de l'API, l'interface utilisateur et la cartographie, puis l'analyse des résultats obtenus et la validation de la solution.

Chapitre 4

Réalisation applicative, résultats et validation

Le moteur d'optimisation étant opérationnel, ce dernier chapitre en présente l'exposition et l'exploitation. Il décrit d'abord la réalisation du backend et de son interface de programmation, puis celle du frontend et de la cartographie interactive. Il analyse ensuite les résultats obtenus sur le cas d'étude, indicateurs des tournées et coût des différentes contraintes, avant de conclure sur la validation de la solution, ses limites et les perspectives d'évolution.

4.1 Réalisation du backend

Le backend constitue le point d'entrée unique de la logique applicative. Cette section décrit l'organisation de son interface de programmation et les principaux services qu'elle expose.

4.1.1 Architecture de l'API

Le backend est bâti sur le framework FastAPI, qui expose les fonctionnalités du système sous la forme d'une interface de programmation web de type REST. Chaque fonctionnalité correspond à un point d'accès (*endpoint*) identifié par une méthode HTTP et une route. Les données échangées transitent au format JSON, et la documentation interactive de

l'API est générée automatiquement, ce qui facilite les tests et l'intégration avec le frontend. L'accès à la base de données est encapsulé par la couche SQLAlchemy, et les traitements métier (solveur, analyse de données) sont invoqués depuis les endpoints correspondants. Le tableau 4.1 récapitule les principaux points d'accès de l'API.

TABLE 4.1 – Principaux points d'accès de l'API

Méthode	**Route**	**Rôle**
POST	`/vagues`	Créer une vague de lots à planifier
POST	`/optimisations`	Lancer une optimisation (synchrone)
GET	`/runs`	Lister les optimisations effectuées
GET	`/runs/{id}`	Détail d'un run (tournées et affectations)
GET	`/runs/{id}/kpis`	Indicateurs de performance d'un run
GET	`/zones`	Zonage géographique des destinations
POST	`/evaluations`	Évaluer une affectation manuelle

4.1.2 Endpoints d'optimisation et de consultation

Le point d'accès central est `POST /optimisations` : il déclenche la résolution du problème sur l'état courant de la base et renvoie, en cas de succès, un code `201` accompagné de l'identifiant du run créé. Le traitement est **synchrone** : la réponse n'est renvoyée qu'une fois l'optimisation terminée, ce qui est acceptable compte tenu du budget de temps borné du solveur.

La consultation des résultats s'effectue via `GET /runs/{id}`, qui renvoie une représentation **agrégée** du run : les tournées y sont imbriquées, chacune contenant ses affectations ordonnées, avec les noms lisibles des destinations plutôt que leurs seuls identifiants. Cette agrégation à la lecture évite de dupliquer l'information en base : l'identifiant du run n'est porté que par la tournée, et le rattachement des affectations en est déduit. L'endpoint `GET /runs` fournit quant à lui la liste synthétique des optimisations passées.

4.1.3 Endpoints cartographie, indicateurs et évaluation

Trois points d'accès complètent l'API. `GET /zones` expose le zonage géographique calculé à la volée par la couche d'analyse. `GET /runs/{id}/kpis` renvoie les indicateurs de per-

formance d'un run (distances, taux de service, remplissage), consommés par le tableau de bord.

Enfin, `POST /evaluations` offre une fonctionnalité d'aide à la décision : l'opérateur soumet une affectation de son choix, que le système réordonne pour en minimiser la distance et dont il vérifie les contraintes de manière **non bloquante**. Les violations éventuelles sont signalées sans être rédhibitoires, et **rien n'est persisté** : la réponse (code `200`) est purement consultative. Ce point d'accès permet de comparer une intuition humaine à la solution automatique, sans altérer les données.

4.2 Réalisation du frontend

Le frontend fournit l'interface par laquelle l'opérateur interagit avec le système. Cette section en présente l'organisation et les principaux écrans.

4.2.1 Architecture multi-écrans

L'interface est développée avec React et l'outil de construction Vite. Elle est organisée en plusieurs écrans, la navigation entre eux étant gérée par un routeur côté client. Chaque écran correspond à une étape du flux de travail : saisie des lots, lancement de l'optimisation, consultation des tournées, visualisation cartographique et tableau de bord. L'adresse du backend n'est pas codée en dur mais fournie par une variable d'environnement, ce qui permet de changer d'environnement (développement, production) sans modifier le code.

4.2.2 Saisie des lots et lancement de l'optimisation

Le premier écran permet à l'opérateur de saisir les lots à livrer et de les regrouper en une vague. Chaque lot est caractérisé par un volume, un type de caisson, un dépôt source et une destination, sélectionnés dans des listes déroulantes alimentées par la base. Deux modes sont alors proposés : l'optimisation automatique par le solveur ou la composition manuelle des tournées. Cet écran est présenté à la figure 4.1.

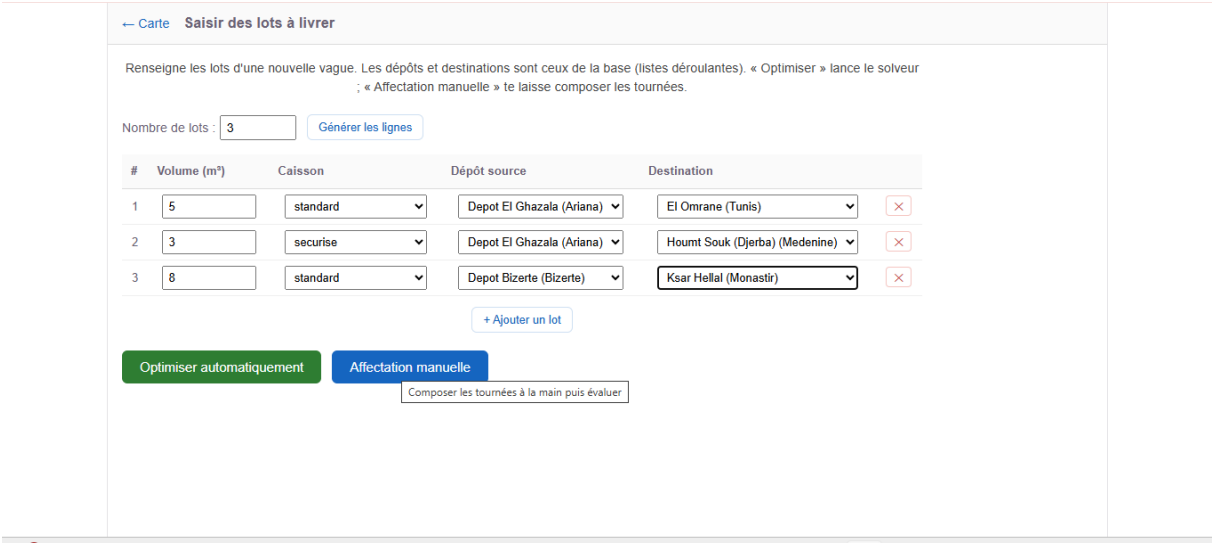


FIGURE 4.1 – Écran de saisie des lots d'une vague

Le lancement de l'optimisation s'effectue depuis un écran dédié. Pendant le calcul, un indicateur d'attente et un chronomètre informent l'utilisateur que le traitement est en cours, jusqu'à réception du résultat, comme l'illustre la figure 4.2.

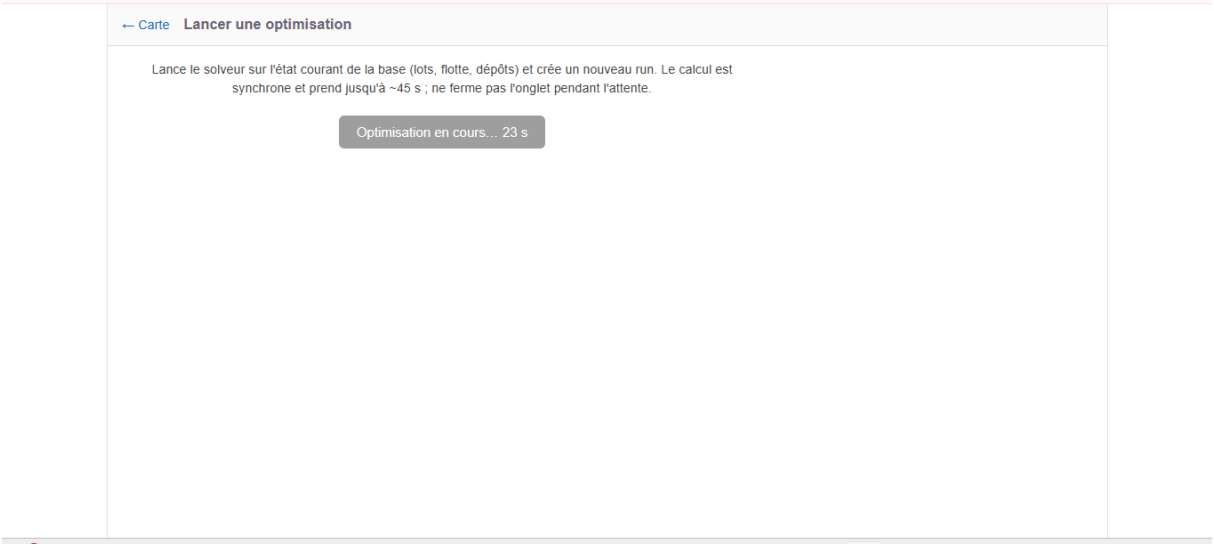


FIGURE 4.2 – Lancement d'une optimisation et indicateur de progression

4.2.3 Consultation des tournées et tableau de bord

Une fois l'optimisation terminée, l'opérateur consulte le détail des tournées sous forme tabulaire : pour chaque véhicule, le dépôt d'attache, l'ordre des visites, les lots affectés et les volumes correspondants (figure 4.3).

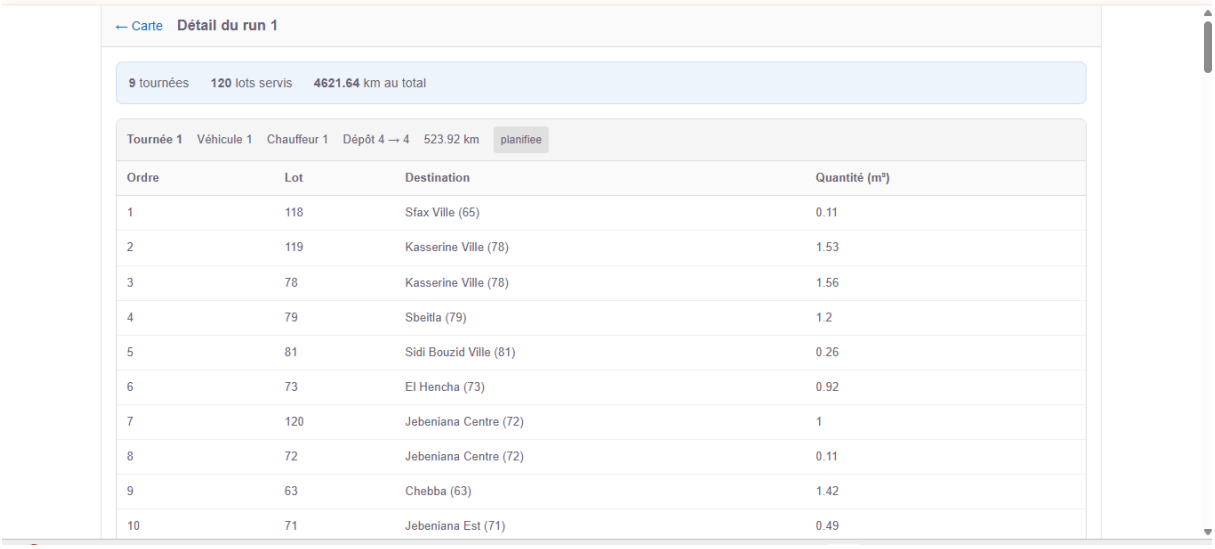


FIGURE 4.3 – Consultation détaillée des tournées d'un run

Un tableau de bord dédié synthétise par ailleurs les indicateurs clés du run (distance totale, taux de remplissage, lots servis, ainsi que l'équilibrage des charges et des distances entre tournées) offrant une vue d'ensemble immédiate de la qualité de la solution (figure 4.4).

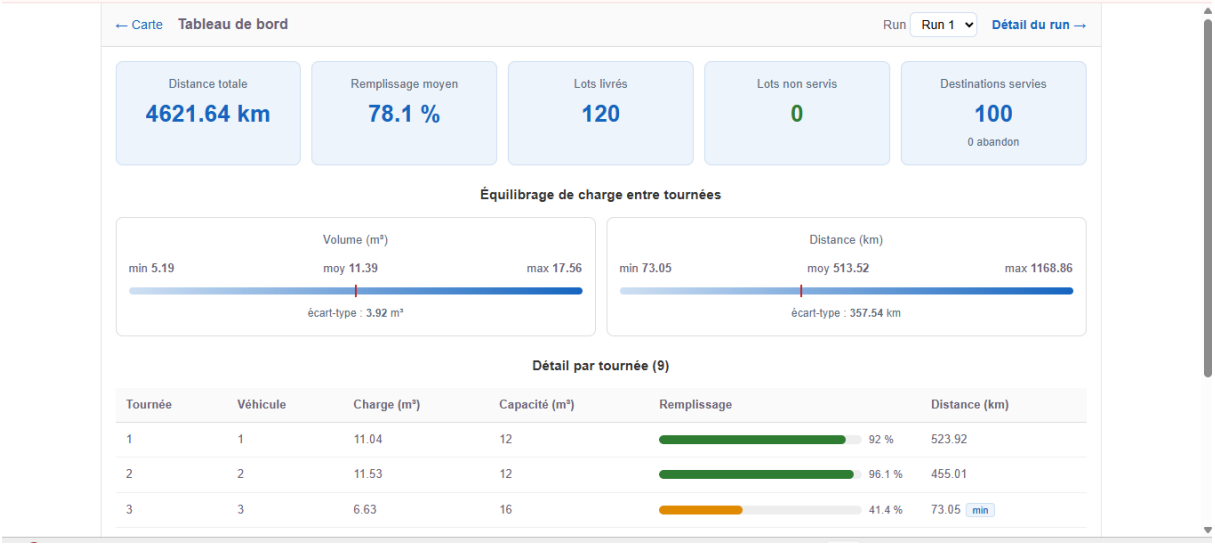


FIGURE 4.4 – Tableau de bord des indicateurs d'un run

4.3 Cartographie interactive

La cartographie est le principal moyen de restitution visuelle des résultats. Cette section décrit sa mise en œuvre et les conventions adoptées pour rendre la lecture immédiate.

4.3.1 Du rendu serveur au rendu client

Deux approches de cartographie ont été explorées successivement. La première reposait sur un rendu *côté serveur* : la carte était générée en HTML par la bibliothèque Folium [21], puis servie à l'interface. La seconde, retenue, adopte un rendu *côté client* : le backend expose les données géographiques au format GeoJSON, et l'affichage est réalisé dans le navigateur par la bibliothèque React-Leaflet [22]. Cette seconde approche rend la carte réellement interactive et allège le serveur, tout en conservant un point d'accès cartographique distinct des endpoints de consultation. Un panneau de contrôle des couches, développé spécifiquement, permet d'afficher ou de masquer les dépôts, les tournées et les zones géographiques. La carte des tournées est illustrée à la figure 4.5.

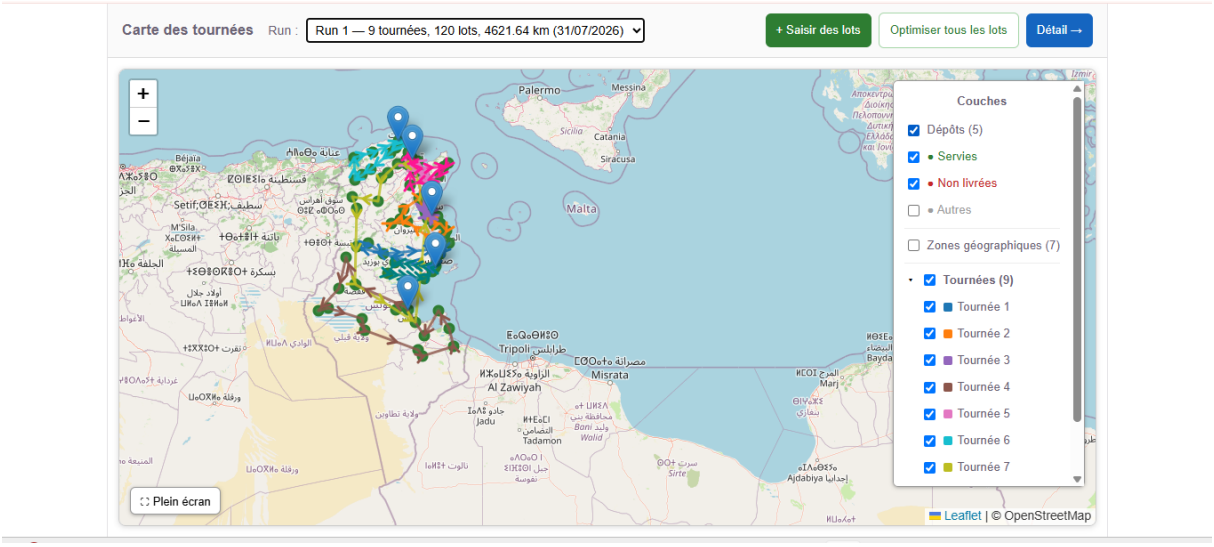


FIGURE 4.5 – Carte interactive des tournées d'un run

4.3.2 Code couleur des destinations

Pour synthétiser l'état de chaque destination vis-à-vis d'une optimisation, un code couleur a été mis en place. Une destination effectivement desservie dans le run apparaît en **vert** ; une destination dont le lot figurait dans la vague mais n'a pas été servi apparaît en **rouge** ; une destination hors du périmètre de la vague apparaît en **gris**. Une règle de priorité tranche les cas où une destination pourrait relever de plusieurs statuts. Ce code couleur est alimenté par une source unique de vérité, la table des lots non servis, plutôt que reconstruit indépendamment, ce qui garantit sa cohérence entre les différentes vues. Au

survol d'une destination non servie, une infobulle indique le motif de l'abandon, comme le montre la figure 4.6.

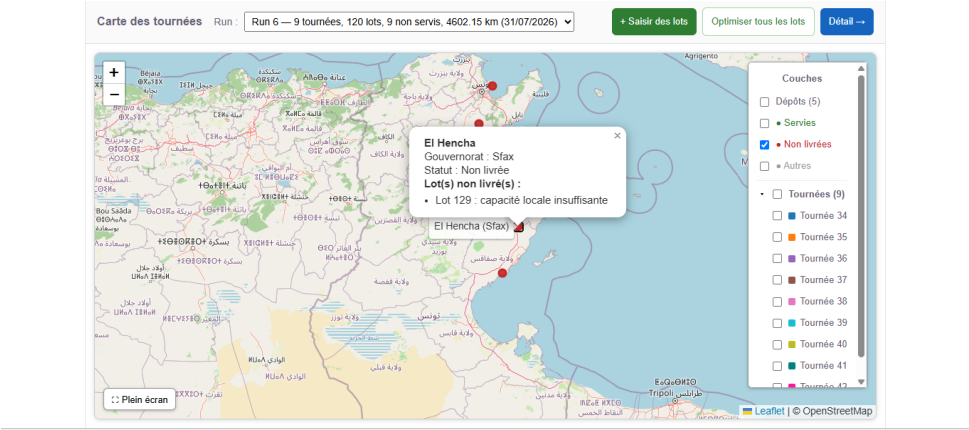


FIGURE 4.6 – Affichage d'une destination non livrée et de son motif

4.3.3 Affichage des violations et évaluation manuelle

La fonctionnalité d'évaluation manuelle laisse l'opérateur composer ses tournées en affectant chaque lot à un véhicule. Le système réordonne alors chaque tournée pour en minimiser la distance et vérifie les contraintes de façon non bloquante : les violations sont signalées visuellement sans empêcher le calcul. La figure 4.7 illustre ce cas, où une affectation viole la contrainte de compatibilité caisson : la violation est signalée et le segment concerné mis en évidence, confrontant ainsi une proposition humaine aux règles du problème.

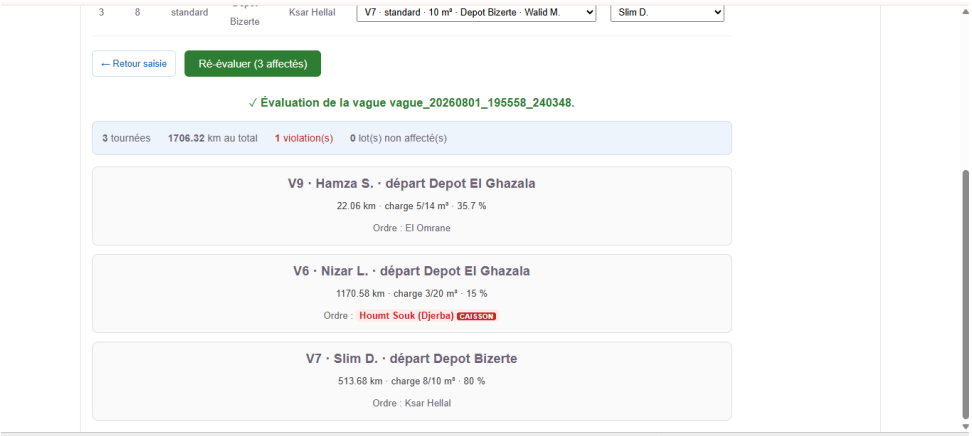


FIGURE 4.7 – Évaluation d'une affectation manuelle avec signalement d'une violation caisson

4.4 Résultats et indicateurs

Cette section présente les résultats obtenus sur le cas d'étude complet, puis analyse le coût des principales contraintes et la structure du zonage.

4.4.1 Indicateurs des tournées

L'optimisation de la vague complète, regroupant l'ensemble des lots, produit une solution servant la totalité des livraisons. Les indicateurs globaux, extraits du tableau de bord, sont reportés au tableau 4.2.

TABLE 4.2 – Indicateurs globaux de la solution sur le cas d'étude

Indicateur	**Valeur**
Nombre de tournées	9
Lots planifiés	120
Lots servis	120 (100 %)
Destinations servies	100
Distance totale parcourue	4 621,64 km
Taux de remplissage moyen	78,1 %
Charge par tournée (min / moy / max)	5,19 / 11,39 / 17,56 m^3
Distance par tournée (min / moy / max)	73,05 / 513,52 / 1 168,86 km

Le taux de service de 100 % indique qu'aucun lot n'a dû être abandonné : toutes les livraisons de la vague ont pu être affectées à un véhicule dans le respect des contraintes. Le taux de remplissage moyen de 78,1 % traduit une bonne utilisation de la capacité de la flotte. On observe néanmoins une dispersion notable des distances entre tournées (de 73 à près de 1 169 km, pour un écart-type d'environ 358 km) : certaines tournées, desservant des zones périphériques et éloignées, sont structurellement bien plus longues que celles concentrées autour du Grand Tunis. Cet écart reflète la géographie du problème plutôt qu'un défaut d'optimisation.

4.4.2 Coût des contraintes

Au-delà de la solution elle-même, il est instructif de mesurer le coût, en distance, de certains choix de modélisation. Ces mesures sont résumées au tableau 4.3.

TABLE 4.3 – Effet des principales contraintes sur la distance

Contrainte / choix de modélisation	**Effet sur la distance**
Retour au dépôt d'origine (vs dépôt le plus proche)	+ 208 km
Compatibilité caisson spécial	+ 38,1 %

Le retour imposé au dépôt d'origine ajoute environ 208 km à la distance totale par rapport à une modélisation autorisant l'arrivée au dépôt le plus proche ; ce surcoût est le prix d'une modélisation fidèle à la réalité opérationnelle, qui évite de sous-estimer les distances. La contrainte de compatibilité caisson, qui réserve certains lots à une sous-flotte équipée, se révèle nettement plus coûteuse : elle allonge la distance d'environ 38 %, en réduisant la liberté d'affectation du solveur. Ce résultat quantifie un arbitrage métier concret entre respect d'une contrainte physique et efficacité des tournées.
Le partitionnement par dépôt source, quant à lui, ne figure pas dans ce tableau car il ne constitue pas une contrainte optionnelle mais une *règle métier structurelle* : un lot ne peut, par nature, être servi que depuis son dépôt d'origine. C'est cette règle qui fonde la décomposition du problème en sous-problèmes indépendants. Quantifier son coût supposerait de résoudre un problème hypothétique à flotte mutualisée, hors du périmètre opérationnel réel, et n'a donc pas été mesuré.

4.4.3 Structure du zonage

Le zonage géographique décrit au chapitre précédent répartit les cent destinations en sept zones, dont les effectifs sont donnés au tableau 4.4. Cette répartition, ordonnée du nord au sud, met en évidence la concentration des destinations dans le Grand Tunis et le Sahel, et confirme que la structure géographique du territoire desservi ne se réduit pas à la partition induite par les cinq dépôts.

TABLE 4.4 – Répartition des destinations par zone géographique

Zone	Nom	Destinations
1	Grand Tunis	37
2	Nord-Ouest	10
3	Sahel	18
4	Centre	7
5	Sfax	11
6	Djérid	7
7	Sud-Est	10
Total		**100**

4.5 Validation et limites

Cette section fait le bilan de la validation de la solution, en explicite honnêtement les limites et esquisse les évolutions envisageables.

4.5.1 Validation fonctionnelle

La solution a été validée fonctionnellement tout au long du développement, chaque brique étant testée avant intégration. Sur le cas d'étude complet, le système produit une solution servant l'intégralité des lots, dans le respect des contraintes de capacité, de source et de compatibilité caisson. Les comportements particuliers, fractionnement des lots volumineux, gestion typée des abandons, évaluation non bloquante d'affectations manuelles, ont été vérifiés sur des cas dédiés. La reproductibilité est assurée : à données et paramètres identiques, une optimisation conduit à des résultats stables.

4.5.2 Limites

Plusieurs limites doivent être clairement énoncées. La plus importante concerne le modèle d'estimation de durée : entraîné sur des données **synthétiques**, il ne constitue qu'une preuve de concept et ne saurait être utilisé en production en l'état, malgré un coefficient de détermination élevé en apparence. Sa validation nécessitera des données réelles de durée de tournée, non disponibles à ce stade.

D'autres limites tiennent au périmètre du projet : les coordonnées de certains points reposent sur un géocodage automatique perfectible, et quelques paramètres métier (terminologie exacte des volumes, nombre définitif de véhicules) restaient à arbitrer avec l'entreprise au moment de la rédaction. Enfin, la base de données utilisée en développement devra être migrée vers une base de production avant tout déploiement.

4.5.3 Perspectives

Plusieurs pistes d'évolution se dégagent. À court terme, l'enrichissement des réponses de l'API avec les noms de destinations pour les lots non servis améliorerait la lisibilité, et la migration vers une base de données de production préparerait le déploiement. À moyen terme, la collecte de données réelles de durée permettrait de transformer le modèle d'estimation, aujourd'hui illustratif, en un véritable outil de prédiction opérationnel. Enfin, le cadre de modélisation retenu se prête naturellement à l'ajout de nouvelles contraintes (fenêtres horaires de livraison, contraintes de tournée multiples), ouvrant la voie à des extensions futures du système.

4.6 Conclusion

Ce chapitre a présenté la réalisation applicative de la solution et l'exploitation de ses résultats. Il a décrit le backend et son interface de programmation, le frontend multi-écrans et la cartographie interactive, avec ses conventions de restitution. Il a ensuite analysé les résultats obtenus sur le cas d'étude : une solution servant la totalité des cent vingt lots à l'aide de neuf tournées pour une distance de 4 621,64 km, ainsi que le coût mesuré des principales contraintes, au premier rang desquelles la compatibilité caisson. Il s'est achevé sur une validation honnête de la solution, la reconnaissance explicite de ses limites, en particulier le caractère synthétique du modèle d'estimation de durée, et les perspectives d'évolution du système.

Conclusion générale

Ce rapport a présenté, dans un premier temps, le cadre général du projet : l'entreprise d'accueil, Progress Engineering, société de services et d'ingénierie informatique, ainsi que la problématique métier à l'origine du travail, à savoir l'optimisation de tournées de livraison dans un contexte multi-dépôts couvrant le territoire tunisien. Cette mise en contexte a permis de situer les enjeux logistiques et de poser le cadre théorique du problème de tournées de véhicules.

Dans un second temps, une étude du problème et de sa résolution a été menée : du problème de tournées classique jusqu'à sa variante multi-dépôts avec contrainte de capacité, en passant par l'analyse de sa complexité et des approches de résolution existantes. Cette étude a justifié les choix de conception retenus, architecture en couches, formulation mathématique des contraintes, et sélection de la bibliothèque OR-Tools comme moteur d'optimisation.

Enfin, la dernière partie a été consacrée à la réalisation du système. Celle-ci a couvert un large éventail de composants : la préparation des données géographiques et des matrices de distances routières, l'implémentation du solveur et de ses contraintes (capacité, source, compatibilité caisson, fractionnement des lots), une couche d'analyse de données pour le zonage géographique, ainsi qu'une application web complète, backend, interface multi-écrans et cartographie interactive, pour la saisie, le lancement et la restitution des résultats. Sur le cas d'étude complet, le système produit une solution servant l'intégralité des cent vingt lots à l'aide de neuf tournées, tout en quantifiant le coût des principales contraintes métier.

Ce stage a constitué une expérience particulièrement enrichissante, tant sur le plan humain que technique. Il m'a permis d'appliquer mes connaissances académiques à un cas concret, de mettre en œuvre une chaîne complète allant de la modélisation d'un problème

d'optimisation jusqu'à son exploitation dans une application web, et de renforcer mes compétences en recherche opérationnelle comme en développement full-stack. Il m'a également confronté aux exigences d'un projet réel : rigueur dans les choix de conception, validation progressive et honnêteté dans l'analyse des résultats et de leurs limites.

Plusieurs perspectives prolongent naturellement ce travail. La collecte de données réelles de durée permettrait de transformer le modèle d'estimation, aujourd'hui illustratif, en un véritable outil de prédiction. La migration vers une base de données de production préparerait un déploiement effectif. Enfin, le cadre de modélisation retenu se prête à l'ajout de nouvelles contraintes, telles que des fenêtres horaires de livraison, ouvrant la voie à des extensions futures du système.

Bibliographie

[1] Progress Engineering, *Accueil et présentation*. `https://www.progress.com.tn/`

[2] Pôle El Ghazala des Technologies de la Communication, *Parc Technologique El Ghazala*. `https://www.elgazala.tn/`

[3] Progress Engineering, *Nos réalisations : Plateforme nationale d'inscription universitaire*. `https://www.progress.com.tn/nos-realisations/`

[4] Progress Engineering, *Actualités : Projets Tunisair Handling, Cour des Comptes et GIZ*. `https://www.progress.com.tn/actualites/`

[5] G. B. Dantzig et J. H. Ramser, « The Truck Dispatching Problem », *Management Science*, vol. 6, n° 1, pp. 80–91, 1959.

[6] P. Toth et D. Vigo (dir.), *Vehicle Routing : Problems, Methods, and Applications*, 2e éd., MOS-SIAM Series on Optimization, SIAM, Philadelphie, 2014.

[7] J. K. Lenstra et A. H. G. Rinnooy Kan, « Complexity of Vehicle Routing and Scheduling Problems », *Networks*, vol. 11, n° 2, pp. 221–227, 1981.

[8] G. Clarke et J. W. Wright, « Scheduling of Vehicles from a Central Depot to a Number of Delivery Points », *Operations Research*, vol. 12, n° 4, pp. 568–581, 1964.

[9] G. Laporte, « The Vehicle Routing Problem : An Overview of Exact and Approximate Algorithms », *European Journal of Operational Research*, vol. 59, n° 3, pp. 345–358, 1992.

[10] Google, *OR-Tools : Google Optimization Tools*. `https://developers.google.com/optimization`

[11] S. Ramírez, *FastAPI : FastAPI framework, high performance, easy to learn*. `https://fastapi.tiangolo.com/`

[12] M. Bayer, *SQLAlchemy : The Python SQL Toolkit and Object Relational Mapper*. `https://www.sqlalchemy.org/`

[13] F. Pedregosa *et al.*, « Scikit-learn : Machine Learning in Python », *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] Meta Open Source, *React : The library for web and native user interfaces*. `https://react.dev/`

[15] V. Agafonkin, *Leaflet : an open-source JavaScript library for interactive maps*. `https://leafletjs.com/`

[16] Heidelberg Institute for Geoinformation Technology (HeiGIT), *OpenRouteService*. `https://openrouteservice.org/`

[17] OpenStreetMap Foundation, *OpenStreetMap*. `https://www.openstreetmap.org/`, consulté le 4 septembre 2026.

[18] J. B. MacQueen, « Some Methods for Classification and Analysis of Multivariate Observations », dans *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, University of California Press, pp. 281–297, 1967.

[19] P. J. Rousseeuw, « Silhouettes : A Graphical Aid to the Interpretation and Validation of Cluster Analysis », *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[20] J. H. Ward, « Hierarchical Grouping to Optimize an Objective Function », *Journal of the American Statistical Association*, vol. 58, n° 301, pp. 236–244, 1963.

[21] Folium contributors, *Folium : Python data, Leaflet.js maps*. `https://python-visualization.github.io/folium/`

[22] P. Le Callennec et contributeurs, *React-Leaflet : React components for Leaflet maps*. `https://react-leaflet.js.org/`