МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

БЕЗДУШНИЙ ВЛАДИСЛАВ ОЛЕГОВИЧ

Допускається до захисту:
завідувач кафедри комп'ютерних наук та інформаційних технологій, к.т.н., доцент
_____ Т.В.Нескородєва

«___» _____20__р.

ІНФОРМАЦІЙНА СИСТЕМА МАРШРУТИЗАЦІЇ ВАНТАЖІВОК. РЕАЛІЗАЦІЯ БІЗНЕС-ЛОГІКИ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Штовба С.Д., професор кафедри
_____,
(назва кафедри)
професор

Оцінка:____/ ____/ _____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2021

АНОТАЦІЯ

Бездушний В.О. Інформаційна система маршрутизації вантажівок. Реалізація бізнес-логіки. Спеціальність 122 «Комп’ютерні науки», освітня програма «Сучасні інформаційні технології та програмування», Донецький національний університет імені Василя Стуса, Вінниця, 2021.

У кваліфікаційній (бакалаврській) роботі досліджена проблема маршрутизації вантажівок. Показані методи та підходи її вирішення. На основі цих підходів був встановлений оптимальний алгоритм для рішення і розроблена система, що буде вирішувати поставлену задачу.

47 с., 3 табл., 19 рис., 5 дод., 41 дрежело.

Ключові слова: VRP, VRPTW, Spring, Java, Jsprit, інформаційна система, маршрутизація

Abstract

Bezdushnyi V.O. Information system for solving the vehicle routing problem: Implementation of business logic. Specialization 122 "Computer science", educational program "Modern information technologies and Programming", Vasyl’ Stus Donetsk National University, Vinnytsia, 2021.

In the qualification (bachelor's) work the vehicle routing problem has been investigated. Methods and approaches of hanging are shown. Based on that approaches, the optimal algorithm for the solution was established and a system that will solve the problem was developed.

47 p., 3 tab., 19 figures., 5 app., 41 ref.

Keywords: VRP, VRPTW, Spring, Java, Jsprit, information system, routing

ЗМІСТ

ВСТУП

Проблема маршрутизації транспортних вантажівок (VRP – Vehicle Routing Problem) – це комбінаторна задача оптимізації та цілочисельного програмування, яка задається питанням: “Який оптимальний набір маршрутів для проїзду вантажівок для доставки товару певному набору клієнтів?”.

Знаходження оптимального рішення VRP є задача NP складності, тому розмір проблем, які можна вирішити оптимально за допомогою математичного програмування або комбінаторної оптимізації, може бути обмеженим. Тому комерційні вирішувачі, як правило, використовують евристику через розмір реальних VRP, які їм потрібно вирішити.

Задача та її вирішення широко розповсюджені у промисловості, тому що використовування оптимізації маршруту може зберегти компаніям багато грошей, так як доставка товарів це одна з найважливіших задач яка також є доволі великою частиною від ціни продукту, зазвичай приблизно 10% [1], та і транспортний сектор приносить приблизно 10% від ВВП Європейського Союзу.

Актуальність проблеми зумовлена тим, що майже всі існуючі вирішувачі можуть вирішувати лише список «стандартних» задач, без їх ускладнення та поєднання, комбінування з іншими типами, тому ця частина є недостатньо вивченої та реалізованою.

Метою цієї роботи є розробка інформаційної системи, що буде будувати шлях вантажівок по точкам-замовникам, щоб мінімізувати час та витрати на перевезення товару, а також кількість вантажівок які потрібні для перевезень.

Для досягнення мети ставляться такі завдання:

1) аналіз сучасного стану методів та інформаційних технологій вирішення задачі маршрутизації вантажівок;

2) генерація варіантів створення нової інформаційної системи маршрутизації вантажівок та їх багатокритеріальний аналіз;

3) розробка алгоритмів та програмна реалізація бізнес-логіки інформаційної системи;

4) тестування інформаційної системи та розробка інструктивних матеріалів.

Об’єкт дослідження: задача маршрутизації вантажівок з часовими вікнами (Vehicle Routing Problem with Time Windows).

Предмет дослідження: інформаційна система маршрутизації вантажівок з часовими вікнами.

Практична значення отриманих результатів полягає в тому, що розроблена інформаційна система дозволяє підвищити якість сервісів доставки вантажів та скоротити витрати ресурсів за рахунок раціонального розподілу замовлень за транспортними засобами та прокладення для них оптимальних маршрутів.

Апробація результатів дослідження проводилась на двох семінарах кафедри інформаційних технологій ДонНУ імені Василя Стуса в 2021 р.

РОЗДІЛ 1
ОГЛЯД СТАНУ ПИТАННЯ ТА ФОРМАЛІЗАЦІЯ ЗАДАЧ РОЗРОБКИ

1.1. Автоматизація маршрутизації вантажівок як об’єкт дослідження

Задача маршрутизації вантажівок узагальнює добре відому проблему комівояжера (TSP – Travelling Salesman Problem). Вперше ця проблема з’явився в роботі Д. Данцига та Д. Рамзера в 1959 р., в якій був написаний перший алгоритмічний підхід і застосовувався до поставок бензину. Часто контекстом є доставка товарів, розташованих у центральному депо, споживачам, які зробили замовлення на такі товари. Завданням VRP є мінімізація загальної вартості маршруту. У 1964 році Кларк Г. та Райт В. вдосконалили підхід Данцига та Рамзера, використовуючи ефективний жадібний підхід, який називається алгоритмом заощаджень.

Ця задача є задачею оптимізації, стосується послуг компанії доставки. Як речі доставляються з одного або декількох складів, який має певний набір транспортних засобів та експлуатується набором водіїв, які можуть рухатися по даній дорожній мережі до якогось набору клієнтів. Задача вимагає визначити набір маршрутів, (один маршрут для кожного транспортного засобу, який повинен стартувати і закінчуватись в депо), таким чином, щоб усі вимоги клієнтів та експлуатаційні обмеження були задоволені, а загальні транспортні витрати мінімізовані.

Дорожню мережу можна описати за допомогою графа, де дуги – це дороги, а вершини – перехрестя між ними. Кожна дуга має пов'язану з нею вартість, яка, як правило, є її тривалістю[2]. На рисунку 1.1 зображений приклад маршруту як графа, де квадрат посередині це депо.

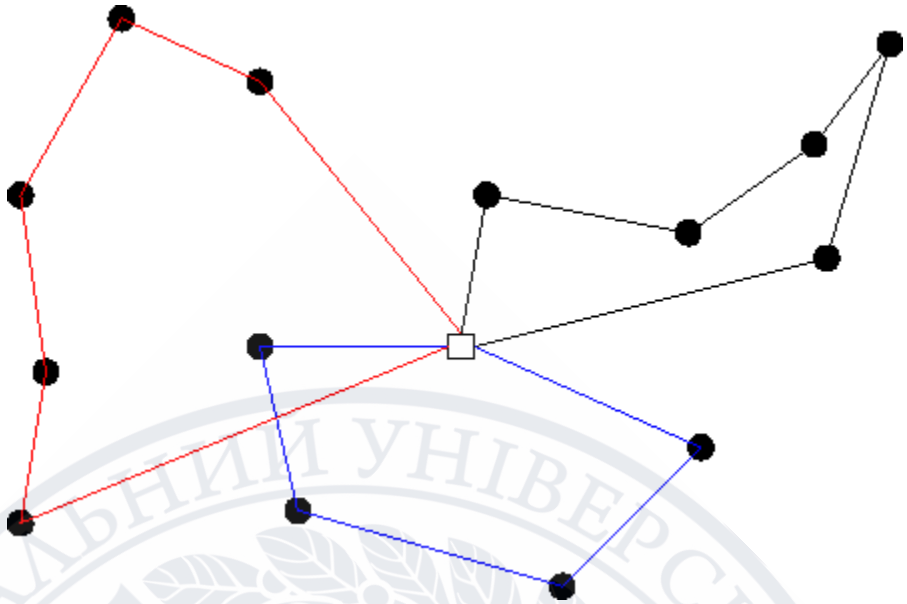


Рисунок 1.1 – Представлення маршруту як графа

Щоб знати загальну вартість кожного маршруту, слід знати вартість проїзду та час у дорозі між кожним клієнтом та депо. Для цього оригінальний графік перетворюється на такий, де вершинами є клієнти та депо, а дуги – дороги між ними. Вартість на кожній дузі – це найнижча вартість між двома точками на вихідній дорожній мережі.

Іноді неможливо задовольнити всі запити клієнта, і в таких випадках вирішувачі можуть зменшити вимоги деяких клієнтів або залишити деяких клієнтів необслуженими. Для вирішення цих ситуацій може бути введена змінна для пріоритету для кожного клієнта або відповідні штрафи за часткову або повну відсутність послуги для кожного клієнта.

- Цільова функція VRP може бути дуже різною залежно від конкретного застосування результату, але є список найпоширеніших цілей цієї задачі: мінімізування глобальних транспортних витрат, виходячи із загальної пройденої відстані, а також постійних витрат, пов’язаних із використаними транспортними засобами;
- мінімізування кількісті транспортних засобів, необхідних для обслуговування всіх клієнтів;
- мінімізування покарання за низьку якість обслуговування;

- максимізація прибутку [2].

Існує декілька варіантів цієї задачі, а саме:

- Проблема маршрутизації вантажівок з прибутком (VRPP – Vehicle Routing Problem with Profits): проблема максимізації, коли не обов'язково відвідувати всіх клієнтів. У цій проблемі представляється числова величина прибутку, що представляє суму вигоди, отриманої під час візиту до кожного клієнту, і дозволяється залишати певних (менш привабливих) клієнтів невідвіданими. Отже, замість того, щоб мінімізувати загальну відстань подорожі по всім клієнтам, ця задача максимізує зібраний прибуток шляхом визначення клієнтів для відвідування та маршрути до цих клієнтів, задовольняючи обмеження на споживання ресурсів на кожному маршруті. Транспортні засоби повинні запускатись і закінчуватися в депо. Цей варіант задачі розглядався у роботах [8, 29, 30, 31].
- Проблема маршрутизації вантажівок із самовивозом та доставкою (VRPPD – Vehicle Routing Problem with Pickup and Delivery): певну кількість товарів потрібно перемістити з певних місць прийому в інші місця доставки. У випадку цієї проблеми, клієнти поділяються на два класи: набір клієнтів для доставки товару та набір клієнтів у яких товар набирати. Мета – знайти оптимальні маршрути для вантажівок щоб приїхати в місця прийому та вивезення. Цей варіант задачі розглядався у роботах [11, 32, 33, 34, 35, 40].
- Проблема маршрутизації вантажівок з часовими вікнами (VRPTW – Vehicle Routing Problem with Time Windows): Місця доставки мають часові вікна, протягом яких повинні бути здійснені доставки. Часові обмеження гарантують наявність вантажівки у клієнта протягом певного періоду часу. Транспортний засіб може прибути раніше часу коли вікно «відкривається», але обслуговування клієнта неможливе,

поки не відкриється це часове вікно. Вантажівці заборонено прибувати після того як вікно «закривається». Деякі моделі допускають раннє або пізнє обслуговування, але з певною формою додаткових витрат або штрафу, проте більшість досліджень проводились без цієї моделі. Щодо моделі «м'яких» часових вікон – клієнтів можна обслуговувати навіть якщо вантажівка не встигла у призначений час, проте тоді на неї будуть накладені деякі штрафи, але ця задача з загальною цільовою функцією не може бути ефективно вирішена. Також ця задача розширюється до задачі с декількома часовими вікнами VRPMTW (Vehicle Routing Problem with Multiple Time Windows). Цей варіант задачі дозволяє клієнтам мати декілька часових вікон у які може приїхати вантажівка та обслужити їх. Ця задача розглядалась у роботах. Цей варіант задачі розглядався у роботах [16, 17, 18, 19, 39].

- Проблема маршрутизації вантажівок з ємністю: CVRP (Capacitated Vehicle Routing Problem) або CVRPTW (Capacitated Vehicle Routing Problem with Time Windows): задача в якій вантажівки з обмеженою вантажопідйомністю повинні брати або доставляти товари до клієнтів. Товари мають такі параметри як кількість, вага або об'єм, а транспортні засоби мають параметр вантажопідйомності. Проблема полягає в тому, щоб забрати або доставити товари за найменшу ціну, але при цьому ніколи не перевищувати пропускну вантажопідйомність вантажівок. CVRPTW є розширенням цієї задачі, і до всіх перелічених параметрів додаються ще й часові вікна. Цей варіант задачі розглядався у роботах [20, 38].
- Проблема маршрутизації вантажівок при кількох поїздках (VRPMT – Vehicle Routing Problem with Multiple Trips): транспортні засоби можуть пройти більше одного маршруту. Ця проблема має дуже практичне значення. Наприклад, при доставці додому швидкопсувних

товарів, таких як їжа, маршрути є короткочасними і повинні поєднуватися так, щоб сформувати повний робочий день. Цей варіант задачі розглядався у роботах [9, 21, 22, 23, 24].

- Відкрита проблема маршрутизації вантажівок (OVRP – Open Vehicle Routing Problem): транспортні засоби не зобов'язані повертатися до депо. Сама по собі задача зустрічається дуже рідко, але має популярне розширення. В реальності, транспортні проблеми з цим типом проблеми вже зі старту має деякі нові обмеження. Відкрита проблема маршрутизації вантажівок з часовим вікном (OVRPTW) додає обмеження у вигляді часового вікна, щоб задовольнити клієнта. Замість проблеми з одним депо, пропонується проблема з декількома депо в маршруті і задача розширюється до (MDOVRP). Цей варіант задачі розглядався у роботах [10, 25, 26, 27, 28].
- Проблема маршрутизації вантажівок, що мають декілька депо (MDVRP – Multi-Depot Vehicle Routing Problem): існує кілька депо, з яких транспортні засоби можуть починати і закінчувати. У задачі може бути більше одного депо. Кожне депо може мати свій парк вантажівок або вантажівки можуть базуватися у різних депо. Також існує два варіанти задачі. Для першого варіанту зазвичай передбачається, що транспортні засоби повинні повертатися до того ж депо. У другому варіанту можуть існувати така умова, при якій кількість транспортних засобів, які прибувають на депо дорівнює кількості транспортних засобів, які виїжджають із депо. Цей варіант задачі розглядався у роботах [3, 36, 37].

Цільова функція у різних варіантах поставленої задачі може відрізнятися, нижче наведений список цільових функцій:

- мінімізація кількості машин,
- мінімізація довжини маршруту,
- мінімізація часу подорожі,

- максимізація кількості клієнтів, до яких можна заїхати з заданою кількістю вантажівок,
- мінімізація часу простою вантажівок,
- мінімізація мінливості під час подорожі вантажівки,
- ефективне навантаження вантажівок.

Також можливі і комбінації цих функцій. Різновиди наведених задач, та яка з якої слідує можна побачити на рисунку 1.2:

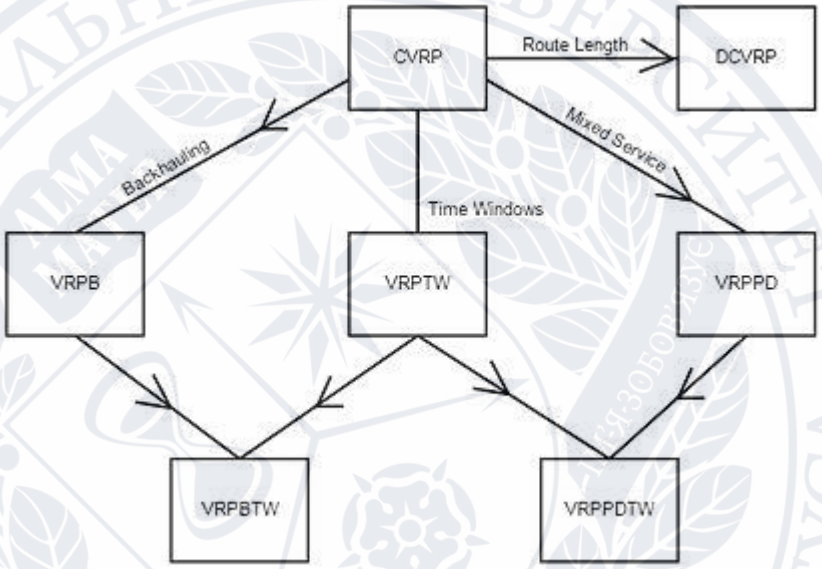


Рисунок 1.2 – Відношення розповсюджених VRP [41]

1.2 Формалізована постановка задачі автоматизації

Для цієї роботи була обрана Capacitated Vehicle Routing Problem with Time Windows **(**CVRPTW**)** як одна з найбільш поширених та «зрозумілих» з точки зору реального життя. Це зумовлено тим, що якщо, наприклад, потрібно розвести товари в мережу магазинів, то зрозуміло що вантажівка має свої ліміти вантажопідйомності та деякі магазини можуть бути відкритими чи доступними для завезення товару в різний час, тому вирішення такої проблеми має прикладний характер.

Формалізація задачі. Задача описується набором вантажівок, які позначимо як V, набором клієнтів C, та якийсь граф G на якому буде відображатись шлях. Граф буде складатися з |C|+2 вершин, де клієнти будуть

позначатися як 1, 2, 3, …, n, а депо нехай буде представлено як вершини 0 та n+1. Набір вершин 0, 1, 2,…, n позначимо як N. Набір дуг, який позначимо як A, представляє зв'язки депо з замовниками та замовники з замовниками. Жодна дуга не закінчується вершиною 0 та жодна дуга не починається з вершини n+1. З кожною дугою (i,j) треба закріпити ціну або дистанцію C_{ij}.

Кожна вантажівка має вантажопідйомність q, і кожен клієнт і потребує d_i товару. Це означає, що q, і та d_i є невід'ємними цілими числами.

Для кожної дуги (i,j), де i≠j; i≠n+1; j≠0, та для кожної вантажівки k, X_{ijk} буде визначений як

$$x_{ij} = \begin{cases} 1, & \text{якщо вантажівка } k \text{ не використовує дугу } (i,j) \\ 0, & \text{в іншому випадку} \end{cases}$$

Задача створити набір маршрутів з мінімальною вартістю, по одному для кожної вантажівки, так щоб кожен клієнт обслуговувався рівно один раз і так щоб маршрут брав початок у вершині 0 і закінчувався на вершині n+1. Тому можна математично записати цю задачу так:

$$min \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \tag{1.1}$$

де:

$$\sum_{k \in V} \sum_{i \in N} x_{ijk} = 1, \quad \forall i \in C \tag{1.2}$$

$$\sum_{i \in C} d_i \sum_{j \in N} x_{ijk} \leq q, \quad \forall k \in V \tag{1.3}$$

$$\sum_{j \in N} x_{ojk} = 1, \quad \forall k \in V \tag{1.4}$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0, \ \ \forall h \in C, \forall k \in V \tag{1.5}$$

$$\sum_{i \in N} x_{in+ik} = 1, \ \ \forall k \in V \tag{1.6}$$

$$x_{ijk} \in \{0,1\}, \forall i, j \in N, \forall k \in V \tag{1.7}$$

Цільова функція (1.1) мінімізує загальну пройдену відстань. Обмеження (1.2) говорить те, що кожен клієнт буде відвіданий рівно один раз, а (1.3) говорить про те що, ні одна із вантажівок не навантажена більше ніж дозволяє вантажопідйомність. Наступних три рівняння (1.4, 1.5, 1.6) забезпечують те, що кожна вантажівка покидає депо 0, а після прибуття до клієнту вантажівка його покидає та відправляється до депо n+1. Обмеження (1.7) є інтегральним обмеженням.

1.3 Аналіз методів вирішення задачі

Для заданої проблеми маршрутизації вантажівок з часовими вікнами існує декілька алгоритмів які вирішують її, і ці алгоритми можна поділити на 3 групи:

- алгоритми побудови;
- алгоритми вдосконалення;
- метаевристичні алгоритми.

Евристичні алгоритми, що будують набір маршрутів, відомі як алгоритми побудови і використовуються для побудови початкового рішення проблеми. Алгоритми, які намагаються знайти вдосконалене рішення за допомогою початкового рішення називаються алгоритмами вдосконалення. Метаевристичні підходи є переважно поєднаннями цих двох підходів та базуються на різних речах [15].

Алгоритми побудови використовуються для створення можливих маршрутів. Ці алгоритми можна класифікувати як послідовні та паралельні алгоритми. В послідовних алгоритмах будується спочатку один маршрут, а інші

тоді, коли це необхідно, хоча в паралельному алгоритмі будуються одночасно багато маршрутів.

Перший алгоритм послідовної побудови був запропонований Е. Бейкером і Д. Шаффером (1986 р.) Алгоритм починається з всіх можливих одиничних маршрутів до клієнта на кожній ітерації, а два маршрути з максимальною економією комбінуються [4].

Орієнтована на час евристика найближчого сусіда – це еврестичний алгоритм послідовної побудови запропонований М. Соломоном (1987 р.). Кожен маршрут ініціалізується, вибираючи найближчого клієнта до депо. Близькість може бути як географічною так і тимчасовою. Вставка необслуженого клієнта здійснюється шляхом вибору клієнта, який є найближчим до останнього клієнту у маршруті, доданого на останній ітерації. Коли немає жодної можливості для вставки будь-якої клієнта, ініціалізується новий маршрут, і вставка триває, поки всі не призначені клієнти не будуть додані [5].

Також існують алгоритми вдосконалення, як вже писалося раніше. Перший з них це «пошук сусіда». Майже кожен алгоритм вдосконалення маршруту базується на концепції сусідства. Концепція сусідства використовується як мінімум 40 років для поєднання проблеми оптимізації. Одне з найперших використань – це Г. Крос (1958 р.), який використовував ідея вдосконалити рішення задачі комівояжера [6].

Більшість алгоритмів вдосконалення для VRPTW використовують алгоритм зміни сусідства, щоб отримати краще рішення. Два класичних алгоритми, які були спочатку запропонованими для задачі комівояжера є евристичні алгоритми k-Opt та Or-Opt.

Евристика k-Opt замінює набір зв’язків у маршруті іншим набором k зв’язків. На складність евристики найбільше впливає розмір k. Для більших значень k, евристика, як правило, дає кращі результати, але обчислювальний час також збільшується. Обмеження у вигляді часових вікон може спричинити те,

що задача може бути не вирішеною, оскільки часові вікна змінюють порядок клієнтів у маршруті.

Евристика Or-Opt яка спочатку була запропонована І. Орем (1976 р.) для задачі комівояжера, вилучає ланцюжок із щонайбільше трьох клієнтів підряд з маршруту і намагається вставити цей ланцюжок у всі можливі місця на маршрутах. Ця евристика є дещо модифікованою, що дозволяє їй вставляти ланцюг в той самий маршрут і до інших маршрутів [14].

Одним із найбільш популярних метаевристичних алгоритмів це Tabu Search (TS), цей алгоритм базується на пошуку сусідів з локальним оптимом уникнення, але детермінованим способом, який намагається моделювати процеси пам'яті людини. TS має методи, які було розроблено для перетину меж доцільності або локальні оптимуми які зазвичай розглядаються як бар'єри, і систематично нав'язувати і звільняти обмеження, що дозволяють досліджувати заборонені, в іншому випадку, регіони [7]. Отже, ідея, яка лежить у ТС, це систематично порушувати можливість здійснення умови. На кожній ітерації досліджуються сусіди поточного рішення і найкращим рішенням обирається як нове поточне рішення. Однак, на відміну від класичної техніки локального пошуку, процедура не зупиняється на першому локальному оптимумі для якого більше неможливе ніяке покращення. Найкраще рішення береться як поточне рішення, навіть якщо це рішення гірше ніж поточне.

Часто ще використовується алгоритм імітації відпалу (Simulated annealing, SA). Це техніка стохастичної релаксації, яка має походження в статистичній механіці [12]. Його методологія подібна до обробка відпалу твердих речовин. Щоб уникнути метастабільних станів, що утворюються під час гартування, метали часто дуже повільно охолоджуються, що дозволяє їм перейти в стабільний стан, структурно зміцніти. Цей процес називають відпалом. На кожному етапі модельованого процесу відпалу виникає у системи новий стан досягнутий з поточного стану шляхом надання випадкової точці випадково вибрану частинку. Якщо енергія нового стану нижча за енергію поточного стану,

приймається нове рішення. Іншими словами, це алгоритм працює шляхом пошуку набору всіх можливих рішень, зменшення шансів застрягти в поганому локальному оптимумі дозволяючи рухатися до гірших рішень під контролем рандомізованої схеми [13].

Існує ще один, більш новий, метаевристичний алгоритм для вирішення цієї задачі, який був створений Девідом Пісінгером та Стефаном Ропком (2007 р.) на основі іншого метаевристичного алгоритму, що був створений Г. Шримпфом (2000 р.), який сформулював принцип руйнування і відтворення. Це, знову ж таки, алгоритм пошук по сусідам, який поєднує в собі елементи модельованого відпалу та прийняття порогових алгоритмів. По суті, це працює наступним чином: починаючи з якогось початкового рішення, алгоритм розкладає частини рішення, що ведуть до (i) набору клієнтів, які більше не обслуговуються вантажівками, та до (ii) часткового рішення, що містить усіх інших клієнтів. Таким чином, цей крок називається кроком руйнування. На основі часткового рішення (ii) всі клієнти з (i) знову реінтегруються, що, таким чином, називається кроком відтворення, що поступається новому рішенню. Якщо нове рішення відповідає певним критеріям якості, воно приймається як нове найкраще рішення, після чого починається нова ітерація руйнування і відтворення. Цей алгоритм є одним з кращих завдяки тому, що він найкраще підходить для складних задач к великою кількістю обмежень, це є універсальною евристикою яка може вирішувати ряд класичних проблем маршрутизації вантажівок, та основні стратегії пошуку можна легко змінювати на малі або великі, залежно від того наскільки проблема важка та комплексна.

У таблиці 1.1 наведений огляд найбільш популярних робот авторів, які працювали над рішенням проблеми, та цільовими функціями, що були передбачені у їх вирішенні.

Таблиця 1.1 – Цільові функції для VRPTW або аналогічної TSP

Автор	Цільова функція
Е. Бейкер і Д. Шаффер (1986 р.)	Мінімальна ціна, мінімальна кількість вантажівок
М. Соломон (1987 р.)	Мінімальна кількість вантажівок, мінімальна дистанція маршруту, мінімальний час простою
Крос Г. (1958 р.)	Мінімальна дистанція маршруту
І. Ор (1976 р.)	Мінімальна дистанція маршруту
Фред В. Гловер, М. Лагуна (1997 р.)	Мінімальна кількість вантажівок, мінімальна дистанція маршруту, мінімальний час простою
Д. Пісінгер, С. Ропк (2007 р.)	Мінімальна кількість вантажівок, мінімальна дистанція маршруту, мінімальний час простою, мінімальний час створення розкладу

1.4 Специфікація вимог до інформаційної системи

На основі даних, що були отримані вище потрібно сформулювати специфікацію системи. Загалом, тут все доволі просто. Система має мати:

- зовнішній інтерфейс, за допомогою якого вона зможе взаємодіяти з користувачем. Цим інтерфейсом може виступати як і відкрите для використання будь ким API (Application programming interfaces), так зване Public API(s), так і закрите для звичайного користувача API яке буди взаємодіяти з веб-інтерфейсом, так званим Front-End’ом.
- Система, що перевіряє вхідні дані на коректність;
- Система, що має обробляти помилки;
- Система, що має вирішувати поставлену задачу за осмислений людиною час, це означає, що рішення не може займати занадто багато

часу. Одна із головних задач системи це вирішувати поставлену задачу якомога швидше та точніше.

Всі ці підсистеми активно взаємодіють одна з іншою, шляхом внутрішніх API, для ефективної роботи всієї системи в цілому. Сервер потребує для коректної роботи деякі дані, а саме список машин та клієнтів, та їх дані, такі як координати, для вантажівок вантажопідйомність, а для клієнтів показник скільки вантажу потрібно, також часові вікна кожного із клієнтів.

1.5 Специфікація проектних завдань

Для досягнення мети роботи потрібно виконати такі завдання:

1. Створити інтерфейси. Вони важливі, щоб програма не була «сам в собі», а щоб можна було динамічно передавати їй дані для рішення. Інтерфейс взагалі, це межа між об’єктами системи та всім іншим, і через цю межу ці об’єкти можуть якось взаємодіяти, у даному випадку взаємодії людини з сервером. Інтерфейс буде реалізований як REST (Representational State Transfer) API. Це є доволі простим та популярним рішенням для back-end розробки.
2. Створити моделі вихідних даних та моделі вхідних даних. Модель вихідних даних не обов’язкова, але бажана для створення документації. Вона просто буде мати такий же вигляд як і справжні дані, що повертаються з серверу. Моделі вхідних даних є обов’язковими. Вони потрібні і для документації, щоб користувач знав як саме потрібно дані серверу відправляти, і для коректної роботи програми. Оскільки сервер пишеться на об’єктно орієнтованій мові, то потрібно створити клас який міг би зберігати в собі дані для їх подальшої обробки. Для вантажівок будуть зберігатися такі дані: ціну простою, вантажопідйомність (в кг.), координати, довгота і широта, та ідентифікатор вантажівки. Для клієнтів зберігається дещо схожий набір інформації, а саме: кількість вантажу, що потребується (в кг.), відрізок часу в який потрібно встигнути заїхати вантажівці, який

відображається найбільш раннім та найбільш пізнім часом коли можна до клієнту заїхати, також координати клієнта, довгота і широта, і також ідентифікатор клієнту.

3. Створити допоміжні класи та методи для валідації даних, для конфігування документації та для обробки помилок. Хоча валідація даних це є обов'язковим функціоналом кожного серверу, з очевидних причин, то документація і власний обробник помилок є «хорошим тоном» у розробці, щоб спростити життя іншим розробникам, тестувальникам та і простим користувачам, у разі якщо розробляється публічне API.
4. Створити підсистему, що буде вирішувати саму VRPTW. Причому, в цій роботі розв'язується не класична VRPTW, а з можливістю додавання декількох депо. Як було сказано вище, для вантажівок зберігаються координати, завдяки цьому можна додати вантажівки до задачі які знаходяться у різних місцях. Це ускладнює задачу, але в той самий час дає їй більшу гнучкість, що дозволяє вирішувати більший спектр проблем. Задача все ще може буди розв'язана як класична, просто додавши декілька вантажівок с однаковими координатами, але при потребі, вона масштабується з Vehicle routing problem with time windows до Multi-depot vehicle routing problem with time windows. Після того, як задача буде вирішена, результат роботи сервера повернеться до front-end частини для графічного відображення для користувача.

Висновки до розділу 1

У розділу досліджена задача маршрутизації вантажівок, отримані знання щодо трьох основних груп алгоритмів та популярних робіт, що вирішуються задачу. Найкращим варіантом був обраний алгоритм руйнування та відтворення, і саме він буде використовуватись у подальшій розробці системи, специфікація якої теж була виявлена.

РОЗДІЛ 2
ВАРІАНТНИЙ АНАЛІЗ ПРОЕКТНИХ РІШЕНЬ

2.1 Варіанти проектних рішень

Для того, щоб вирішити поставлену задачу, існує не так вже й багато підходів. Ці підходи – це написання вирішення вручну, з нуля, або використовувати якийсь фреймворк. Для Java існує два основних, великих фрейморвка які надають розробнику інструменти для розробки та конфігурування рішення задачі, і їх назви це Jsprit та OptaPlanner.

Jsprit це написаний на Java фреймворк з відкритим кодом для вирішення задач комівояжера та проблем маршрутизації вантажівок. Фреймворк цей доволі «легкий» з точки зори кількості займаємої на жорсткому диску пам’яті, дуже гнучкий та простий для використання. Jsprit надає інструменти для вирішування та конфігурування ряду задач, а саме: проблему маршрутизації вантажівок із самовивозом та доставкою, проблема маршрутизації вантажівок з часовими вікнами, проблему маршрутизації вантажівок з ємністю, відкриту проблему маршрутизації вантажівок, проблему маршрутизації вантажівок, що мають декілька депо, проблему с початковим маршрутом, проблему з навичками водіїв та багато інших. Jsprit надає доволі багато прикладів задач, варіантів рішення, прикладів вхідних даних, щоб код працював правильно, тобто має доволі непогано «базу знань» для розробника. Також має свій форум на якому є багато цікавих запитань від інших розробників, на які автори фреймворки намагаються давати якомога повну та зрозумілу відповідь з точки зору теорії та часто підкріпляють ці слова фрагментами коду які мають вирішувати задане запитання, що дає змогу оперативно вирішувати якісь незрозумілі питання, це означає, що фрейморку має гарну службу технічної підтримки.

OptaPlanner це також написаний Java фреймворк який дозволяє більш зручно вирішувати оптимізаційні задачі, серед яких є проблема маршрутизації вантажівок. Цей фреймворк, рівно як і Jsprit, не займає багато місця на

жорсткому диску та гнучкий. OptaPlanner базується на штучному інтелекті, та використовує ряд оптимізаційних алгоритмів, в тому числі і Tabu Search, про який вже писалося раніше. Може давати інструменти для вирішування майже такого же списку проблем, що і Jsprit, хоча і не має можливості давати водіям навички, або занадто сильно змішувати проблеми різних типів, так як цей фреймворк дає інструменти для більшого спектру різнопланових оптимізаційних задач, і тому не зациклюється лише на одному типі, як це робить Jsprit. OptaPlanner також має свою «базу знань» хочу вона і дещо менша для конкретно проблеми маршрутизації вантажівок, тому що фреймворк дає більше можливостей стосовно інших оптимізаційних задач, як вже писалося раніше. Також присутній форум де кожен бажаючий може залишити питання стосовно реалізації будь чого зв'язаного з фреймворком, але на цьому форумі важче знайти інформації стосовно однієї якоїсь теми, знову ж таки, через універсальність фреймворку.

Написання вирішення вручну. Для цього варіанта існує ряд проблем та переваг. Написання чогось з нуля, без використання сторонніх бібліотек неминуче приведе до того, що розробка дуже сильно затягнеться як і в часі, так і у вартості. Для написання своєї бібліотеки потрібно бути висококваліфікованим розробником, щоб ця бібліотека просто нормально працювала. При такому підході на стадії впровадження буде багато проблем також. Але з іншої сторони це дозволяє створити максимально продуктивне рішення та дозволяю розвивати бібліотеку у потрібному напрямку, але це актуально тільки при умові, що розробник дуже гарно орієнтується в суті проблеми і знає мову на високому рівні, знову ж таки приходимо до проблеми кваліфікації. Це все приводить до того, що таке є рішення доволі сумнівним, так немає ніякої документації або форуму, до яких можна було би звернутись у разі непередбачуваної помилки з технічної точки зору, також, знову ж таки, потрібно дуже добре розуміти математичну частину алгоритмів, щоб правильно написати рішення. Та навіть якщо це все правильно написати, ця бібліотека буде потребувати тестування. На

відмінну від великих фреймворків і бібліотек, які тестуються тисячами розробників вже великий період часу, які їх використовують, власна бібліотека, причому у рамках цієї конкретної роботи, не зможе бути так же гарно протестована, адаптована і гарно масштабуватись.

2.2 Критерії оцінки проектних рішень

Потрібно сформувати список критеріїв за якими можна були би цих три варіанти порівняти, та знайти найкращий з них. Для позначення результатів будуть використовуватись бали 1, 2 та 3, де 1 – поганий результат, мало; 2 – «середина», не ідеально, але і не погано; 3 – гарний результат, всього достатньо.

Складність розробки. Як складно використовувати це рішення, чи наскільки кваліфікований спеціаліст потрібний. Ці питання описуються один з найважливіших критеріїв про розробці будь чого. Цей критерій буде відображати рівень складності, неоднозначності та і цілком незрозумілості інструментів, підходів та методів які будуть використані при розробці рішення задачі. Також критерій буде відображати рівень знань які потрібно мати розробнику, щоб можна було розібратись з кодом, з алгоритмом та почати працювати. Також до цього критерію відносяться наявність документації, та її якість, тому що це завжди був, є і буде основним способом отримати технічну інформацію щодо якогось класу, метода, чи аргументу. Так як цей критерій показує «негативну» сторону, тому при підрахунку найкращого рішення він буде братися за принципом «чим менше – тим краще».

Трудомісткість розробки. Цей критерій відображає як багато прийдеться працювати. Він показує як багато функціоналу треба писати самому, як багато треба переписувати, або дописувати, скільки часу та сил потрібно потратити щоб написати та адаптувати систему під конкретні цілі. До цього критерію також відносяться документація, знову ж таки, так як це дуже цінне джерело інформації, а також наявність відкритого початкового коду, щоб розробник сам

міг передивись код якогось конкретного класу, методу і при потребі переписати його, адаптувати під конкретну реалізацію чи якісь інші умови. Адаптування чужих бібліотек, їх зміна та переписування це доволі поширена практика, тому що багато що пишеться с прицілом на універсальність, а тому може бути не самим оптимальним варіантом при конкретній реалізації, і через це цей критерій теж є важливим. Так як цей критерій показує «негативну» сторону, тому при підрахунку найкращого рішення він буде братися за принципом «чим менше – тим краще».

Підтримка системи. Цей критерій відображає гнучкість системи та її масштабованість. Гнучкість у виборі або додаванні нових алгоритмів, за необхідністю, гнучкість у виборі цільової функції цих алгоритмів, гнучкість у виборі обмежень для задачі, її типу, її поглиблення та розширення, причому не ламаючи те, що вже є. Додавання функціоналу для інтеграцій з іншими сервісами, синхронізація з ними, вчасне обновлення документації, це все є підтримкою системи, і можливості її підтримки сильно залежать від рівня попередніх критеріїв, але будь яку систему все одно потрібно підтримувати, щоб вона не замкнулась та не була забутою та викинутою через її непотрібність, це все теж дужа важливо. Так як цей критерій показує «негативну» сторону, тому при підрахунку найкращого рішення він буде братися за принципом «чим менше – тим краще».

Підтримка рішень. Цей критерій показує наскільки добре рішення підтримується розробником. У випадку з фреймворком це показує як часто виходять оновлення, наскільки вони корисні, чи приносять новий функціонал, нові інструменти, чи покращуються ті які вже існують, чи закриваються якийсь старий функціонал і чи намагаються створити для нього якусь кращу реалізацію. У випадку з власним рішенням це значить як часто розробник бібліотеки приносить щось нове, вивчає нові підходи та алгоритми, щоб їх додати, оптимізує вже існуючі та взагалі розвиває те, що написав.

2.3 Багатокритеріальна оцінка проектних рішень

Маючи ці критерії потрібно оцінити рішення, і вибрати найкраще з них. Для цього треба створити таблицю та записати у нею відповідні бали. Бали які виставлені для «Рішення з нуля» є суб'єктивними для автора роботи, і можуть змінюватись залежно від того, який розробник і з якими знаннями буде розробляти таке рішення.

Таблиця 2.1 – Варіанти проектних рішень

	Складність розробки	Трудомісткість розробки	Підтримка системи	Підтримка рішень
Рішення з нуля	3	3	2	2
Jsprit	1	1	1	2
OptaPlanner	2	2	2	3

Для того, щоб знайти найкращий варіант потрібно використати метод лінійної згортки (метод зваженої суми критеріїв). Для цього потрібно нормалізувати значення у таблиці за допомогою наступної формули:

$$p_{ij} = \frac{x_{\max j} - x_{ij}}{x_{\max j} - x_{\min j}} \quad (2.1)$$

$$p_{ij} = \frac{x_{ij} - x_{min\,j}}{x_{\max j} - x_{\min j}} \quad (2.2)$$

У формулі (2.1) вираховується нормалізоване значення за принципом «чим менше – тим краще», а у формулі (2.2) навпаки «чим більше – тим краще». У таблиці 2.2 наведені нормалізовані значення, які були отримані при використанні формул

Таблиця 2.2 – Нормалізована таблиця

	Складність розробки	Трудомісткість розробки	Підтримка системи	Підтримка рішень
Рішення з нуля	0	0	0	0.5
Jsprit	1	1	1	0.5
OptaPlanner	0.5	0.5	0	1

Далі, надавши всім критеріям оцінку 0.25, так як всі вони є однаково важливими, кожне значення відповідного проектного рішення було помножене на оцінку та всі результати множень були підсумовування, таким чином були отримані наступні оцінки: Рішення з нуля – 0.125, Jsprit – 0.875, OptaPlanner–0.5.

Базуючись на результатах отримаємо те, що найкращим варіантом для розробки системи буде фреймворк Jsprit. На його основі, за допомогою тих інструментів, що він надає і буде вирішуватись поставлена задача, адже ті інструменти що він надає з незначною перевагою переважують ті інструменти, що надає OptaPlanner, і доволі сильно переважує варіант розробки з нуля, так як цей варіант має потребує дуже добрих знань як і мови програмування, так і математичної частини, про що вже писалось раніше.

Але потрібно зауважити, що варіант розробки з нуля може бути релевантним для якихось специфічних задач, для яких немає прямого рішення у популярних бібліотеках та фреймворках, або для тих задач які методами фреймворків вирішуються занадто довго, особливо у випадку коли дужа багато клієнтів.

Для технічної реалізації рішення задачі була обрана мова програмування Java. Java є простою з точки зору синтаксиса, та доволі простою для вивчення. Розробники цієї мови приділили багато часу тому, щоб зробити її простою, і завдяки цьому код на ній доволі просто писати, компілювати, відлагоджувати та

аналізувати. Java є об’єктно-орієнтованою мовою програмування, що дозволяє створювати модульні програми, і перевикористовувати код . Також добре написанні програми з використанням ООП самі себе документують, адже такий код і таке представлення даних більше зрозуміле людині, що дозволяє менше часу використовувати для аналізу и читання коду і швидше почати писати свій код. Ще однією дуже вагомою перевагою стало те, що Java не залежить від платформи, а це дозволяє без жодних проблем переносити програму з однієї системи на іншу, також так як ця мова на залежить від платформи як на рівні початково коду, так і на рівні двійкового представлення, програми на ній можна запускати на будь яких системах, що особливо важливо для серверів які працюють в інтернеті. Простота Java, незалежність від платформи та вбудовані функції захисту роблять цю мову однієї з найкращих для створення середніх та великих рішень які будуть працювати в інтернеті.

Також для цієї мови існує велика кількість фреймворків для написання рішень для будь яких задач. І рішення поставленої задачі буде будуватися на стеку фреймворка Spring. Spring сильно спрощує розробку веб-додатків на базі стеку JavaEE (Java Enterprise Edition), що є набором специфікацій, що розширюють звичайну Java для створення веб-сервісів. Spring має величезну спільноту користувачів, що означає, що можна безкоштовно знайти навчальні матеріали та курси по ньому. Більше того, фреймворк допомагає автоматично налаштовувати всі компоненти додатку, сприяє створенню тестів та взагалі тестування додатків надаючи за замовчуванням інструменти для юніт тестів та інтеграційних тестів, допомагає уникати написання анотацій, складних XML конфігурувань, має вбудовані HTTP сервери (Jetty, Tomcat) тестування веб-додатків, надає доступ до дуже зручних інструментів таких як Spring Data, Spring Security, Spring JDBC. Також фреймворк надає безліч плагінів для роботи з вбудованими базами даних, та базами, що знаходяться в пам’яті машини, дозволяю легко підключатися до всіх популярних баз даних, таких як Oracle,

PostgreSQL, MySQL, MongoDB та інші. Як вже зрозуміло, надає всі потрібні будь якому розробнику інструменти, та навіть більше.

Більше того, ця мова є досить популярною, про це говорить статистика від GitHub за 2020 рік. Незважаючи на те що ця мова трішки загубила свою популярність за останній рік, вона все ще знаходиться у числі перших трьох мов. Це надає змогу отримувати для цієї мові актуальні знання, приклади коду та інше.

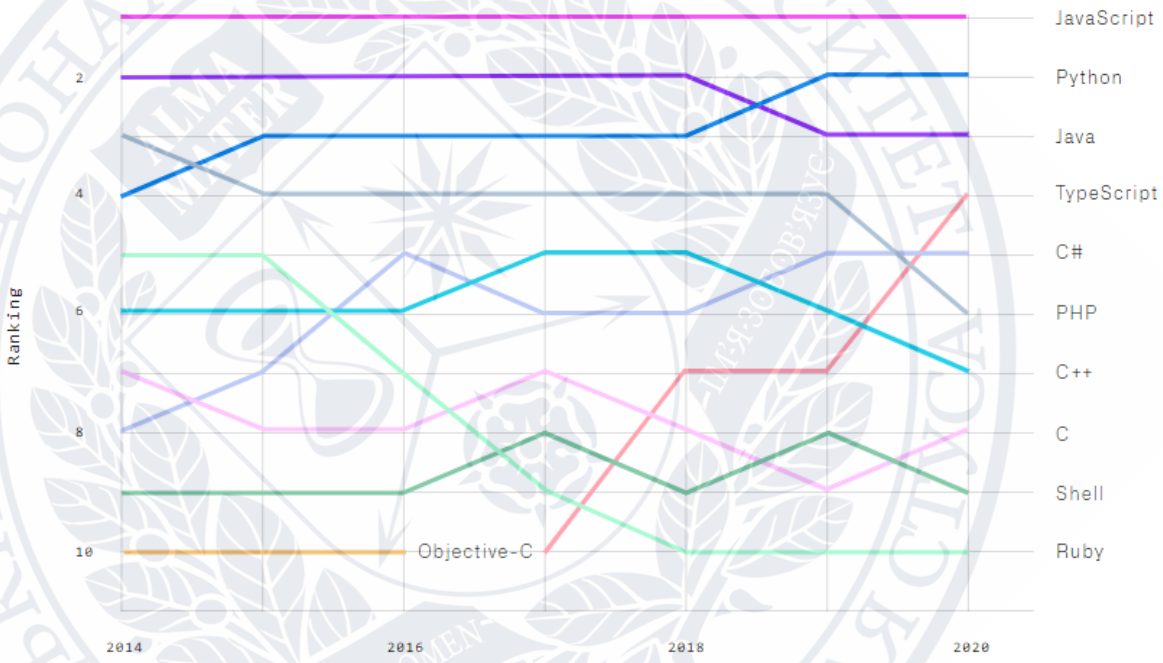


Рисунок 3.2 – Найпопулярніші мови програмування на GitHub

Все це надає змогу швидко розпочати розробку, мінімізувати час який потрібний для написання розповсюджених задач та спростити повсякденні проблеми.

Висновки до розділу 2

У цьому розділу була отримана інформація щодо рішень якими можна скористатися при розробці свого сервіса для рішення проблеми маршрутизації

вантажівок. Також були розглянуті критерії за якими був виявлений оптимальний підхід у цій роботі для вирішення поставленої задачі та була обрана мова програмування на якій розроблятись система.

РОЗДІЛ 3

ПРОГРАМНА РЕАЛІЗАЦІЯ БІЗНЕС-ЛОГІКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Архітектура проекту

Архітектурним рішенням для проекту виступає MVC (Model-View-Controller) патерн. Його ціль це розділяти додаток на три основних логічних компоненти: на модель, на представлення та на контролер. Кожен з цих компонентів створений для обробки конкретних аспектів розробки додатку. MVC – це один з найпоширеніших стандартів розробки веб-додатків для створення масштабованих, розширюваних проектів. Завдяки такій архітектурі додаток розділяється на декілька рівнів, і кожен рівень виконує тільки свою частину і не знає, що відбувається на інших рівнях і тому ніяк від них не залежить. Через це код будь якого з рівнів можна змінювати не ризикуючи отримати конфлікт між цими рівнями.

Компонент Model відповідає з відображення усієї логіки проекту, з якою працює користувач. Цей компонент може дані, що передається між компонентами View та Controller, або з будь-якими іншими даними бізнес-логіки додатку. Модель ніяк не залежить від View, бо не знає як візуалізувати дані, та від Controller, бо не має ніяк «контакту» з користувачем, лише просто надає доступ до даних і методів для управління ними.

Компонент View це та частина програми, яка представляє дані користувачеві. Цей компонент створюється з даних, що були зібрані з компонентів Model та Controller. Цей компонент повторно вимагає дані від моделі, щоб відобразити відповідь сервера користувачеві, якщо ті були якось модифіковані. У цьому проекті компонент View розділений на дві частини, адже для сервера цим компонентом є вихідний JSON (JavaScript Object Notation). Потрібно зауважити, що JSON є дуже поширеним стандартом для обміну даними між браузером та сервером, та між сервером та сервером, і незважаючи на те, що він оснований на мові програмування JavaScript, він є незалежним від цієї мови і

майже всі сучасні мови мають інструменти для роботи з ним. На відміну від XML, JSON є більше лаконічним та добре читається людьми, і саме тому такий спосіб представлення даних був обраний для серверної частини додатку. Цей JSON передається на front-end частину за допомогою HTTP і вже на основі даних отриманих від сервера, front-end частина «будує» зручне для користувача представлення с елементами інтерфейсу.

Компонент Controller це частина додатку, яка виступає інтерфейсом між компонентами View та Model, для обробки бізнес логіки проекту та вхідних запитів, маніпулює даними за допомогою компонента Model, взаємодіє з компонентом View для надання кінцевого результату. Цей компонент надсилає моделі команду оновити її стан, наприклад коли якась сутність була модифікована. Також Controller надсилає команду своєму представленню для зміни того, щоб відображається користувачеві.

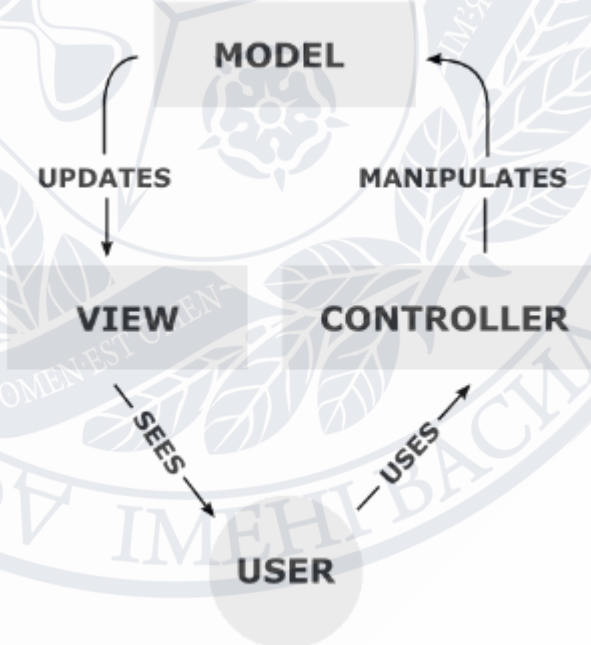


Рисунок 3.1 – Архітектура Model-View-Controller

Цей патерн не має чіткої реалізації, тому від може бути реалізований інакше, залежно від розробника. Немає якогось стандарту де має бути розміщена бізнес-логіка проекту, вона може знаходитися як і в контролері, так і в моделі.

Так як проект написаний за MVC патерном, то різні його частини відповідаються за різнці функції. Модель вхідних даних VrpModel складається з двох інших моделей, таких як ServiceModel та DepotModel.

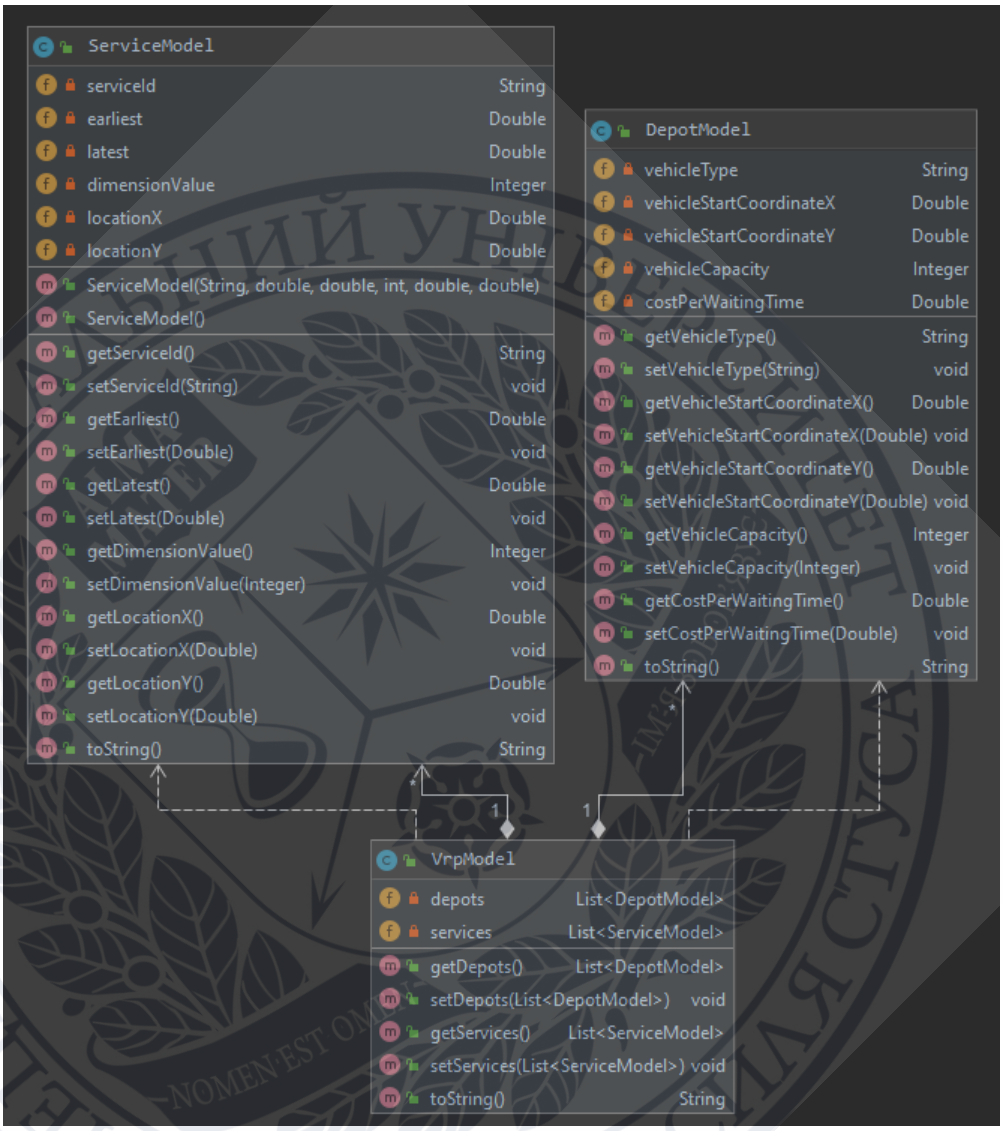


Рисунок 3.3 - Модель VrpModel

ServiceModel та DepotModel зберігаються дані про клієнтів та депо відповідно, і ці моделі зберігаються у моделі VrpModel.

Проте існує також і вихідна модель, яка є дельшо більша, в ній зберігається набагато більше даних, і вона створюється, модицікується під час виконання рішення поставленої задачі.

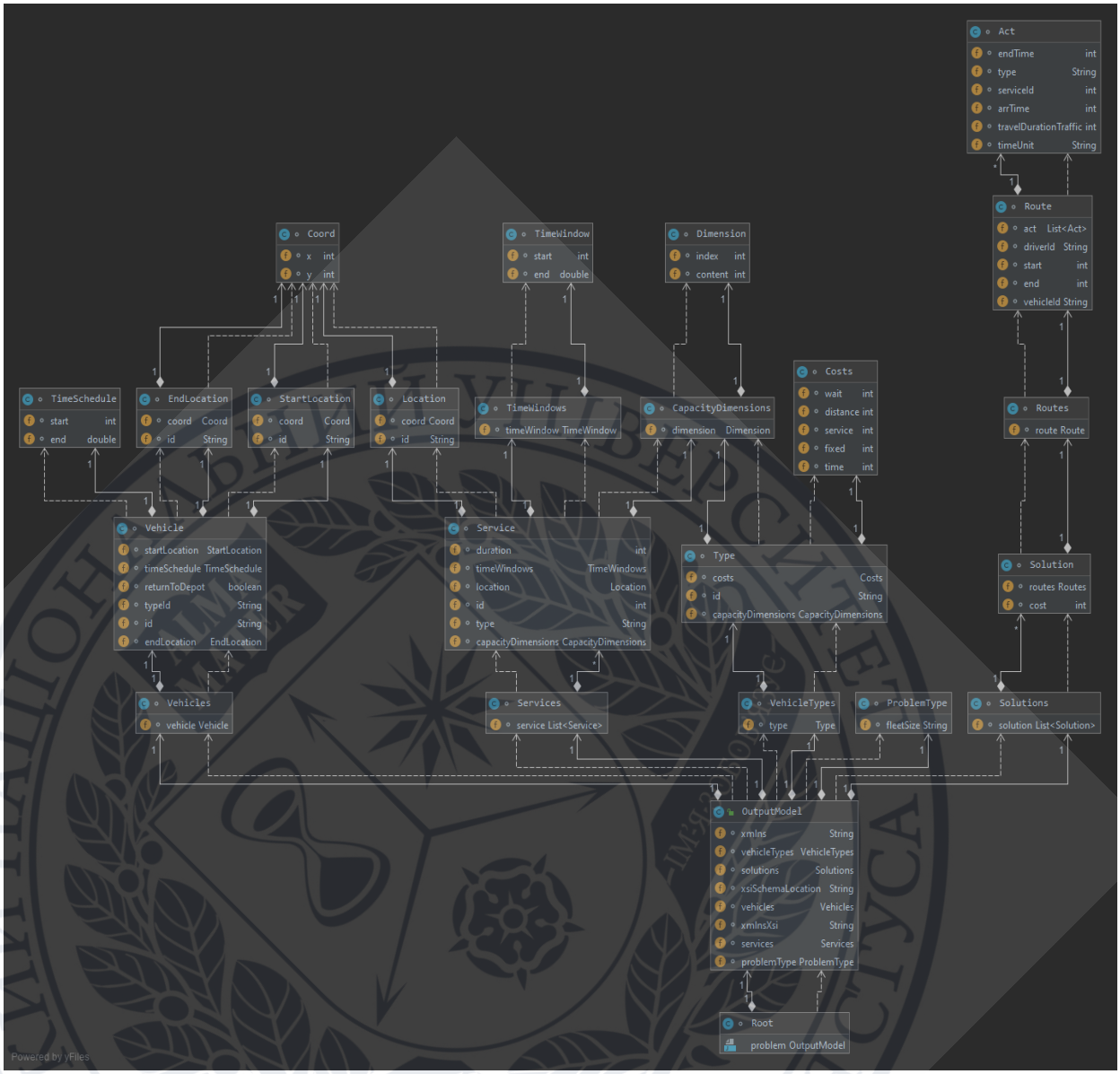


Рисунок 3.4 – Модель OutputModel

OutptuModel зберігає всю інформацію, яка відноситься до рішення задачі. Тут зберігається вся інформація про машини, координати, сервіси (клієнтів), розклад, локацію, ціну подорожі, а також масив шляхів для кожної вантажівки.

Так виглядає Model частина проекту. Оскільки в моделях дужа зберігається дужа багато параметрів, для кожного параметру є свої методи, то на рисунках були відображені не повністю всі методи і весь функціонал моделей, оскільки тоді рисунки стають занадто великими та нечитабельними.

View частиною проекту виступає front-end частина, але щоб ця частина працювала потрібно їй передати інформацію. Як вже писалося раніше, цю інформацію представляє JSON файл. Так як цей JSON є теж доволі великим, то на рисунку буде представлений лише його основний фрагмент.

Рисунок 3.5 – JSON View

У цій частині відповіді серверу показаний один елемент масиву шляху для якоїсь вантажівки. Потім у front-end частині генерується вивід зручної для користувача інформації, саме цей JSON є однією з основних частин front-end частини. Також цей JSON зберігає передає багато мета-даних пов’язаних з вантажівками ті клієнтами, з вартістю та матрицею вартості.

Останньою частиною проекту є Controller. Ця частина приймає дані, тобто моделі, передає обробляє їх та визначає вихідну модель. В цьому проекті вся бізнес логіка виконується в контролерах та допоміжних йому класах.

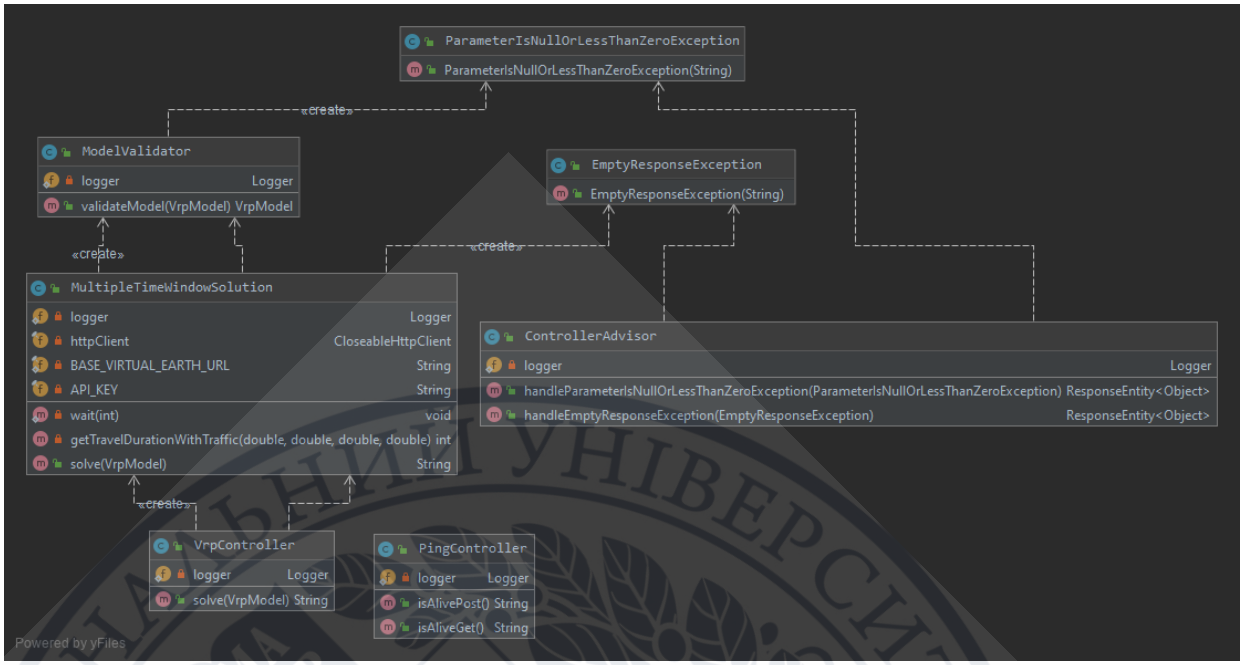


Рисунок 3.6 – Controller

Як видно з рисунку, існує два контролера. Один з потрібен лише для того, щоб перевірити чи сервер працює, та змусити вийти його с режиму сну, якщо він в цьому режимі. Інший контролер, VrpController, якраз і приймає дані, валідує їх, та передає їх на вирішення. Це є основною частиною для взаємодії сервером з «зовнішнім» світом.

У цьому проекті буде розроблене рішення для заданої задачі, на основі фреймворку Jsprit, що стоїть на стеку Spring. Всі використані рішення вже тим або іншим чином «готовими» але завдання буде їх адаптувати під «реальні» значення, адаптувати їх для того щоб всі ці фреймворки працювали разом і не створювали один іншому конфліктів, також задачу буде добитися достатньої швидкодії рішення, щоб забезпечити користувачеві максимальний комфорт. Також серед задач буде весь цей сервер «увімкнути», щоб його можна були використовувати через інтернет, і не потрібно були його локально розвертати, щоб протестувати або з ним працювати.

3.2 Модуль «Моделі»

Моделі є основним джерелом даних для цього проекту, і тому вони мають бути повними. Як було видно з попередніх рисунків, вхідна модель VrpModel складається з двох інших. Всередині цієї моделі маємо 2 списки:

- private List<DepotModel> depots
- private List<ServiceModel> services

У списку depots зберігаються дані про депо в якому знаходяться вантажівки і яке повинно буди початковою та кінцевою точкою подорожі для кожної з вантажівок. Серед даних депо є:

- private String vehicleType, що зберігає в собі дані про конкретну вантажівку, а саме її тип, або ідентифікатор. Наразі у цього параметру немає ніякої практичної користі, ніякого використання в середині алгоритму вирішення, цей параметр потрібний щоб розділяти вантажівки, їх ідентифікатори у результаті роботи алгоритму, у його вихідній моделі, так як для кожної з вантажівок буди побудований її власний шлях, який потім потрібно відобразити на front-end частині проекту.
- private Double vehicleStartCoordinateX, цей параметр зберігає X початкову координату вантажівки, а саме її довготу. Цей параметр береться з Google Maps API за назвою якої небудь локації, за допомогою скриптів на front-end частині.
- private Double vehicleStartCoordinateY, аналогічно з попереднім параметром, але зберігає цей параметр широта, а не довготу початкового місця вантажівки.
- private Integer vehicleCapacity, це є параметром, що зберігає вантажопідйомність вантажівки. Він є дуже важливим так як велика доля майбутнього шляху вантажівки залежить від того, скільки вантажу вона може з собою везти. Все ж таки VRPTW є розширенням задачі CVRP.

- private Double costPerWaitingTime, цей параметр зберігає значення, яке потрібно для вираховування ціни простою. Насправді, цей параметр не є важливою частиною задачі, так як цільова функція це мінімізація маршруту та часу подорожі вантажівок. Проте цей параметр все одно використовується під час рішення, щоб вираховувати кошти які були втрачені під час якихось несподіваних ситуацій, як , наприклад, затор на дорозі.

Всі ці параметри є обов'язково у тілі запиту до сервера, якщо хоча б одного з них не буде, то сервер поверне помилку з вказівкою на параметр, якого немає, але про обробник помилок буде в іншому розділі.

Також потрібно уточнити модель клієнтів, яка зберігає в собі всі потрібна дані які відносяться до клієнтів. Це дані це:

- private String serviceId, цей параметр зберігає дані про ідентифікатор клієнту. Це потрібно щоб можна було при передачі даних зрозуміти які дані до якого клієнту відносяться. Також після побудови маршруту сервер повертає відсортований згідно маршруту кожної з вантажівок, і щоб уникнути неоднозначності та додаткових проблем з написанням коду, ідентифікатори і потрібні.
- private Double earliest, це параметр є частиною того часового вікна, яке є ядром задачі VRPTW, і цей параметр показує найбільш ранній час коли вантажівка може приїхати до заданого клієнта.
- private Double latest, аналогічно попередньому параметру цей параметр теж є частиною часового вікна, проте цей параметр показує найпізніший час коли до заданого клієнту може заїхати вантажівка.
- private Integer dimensionValue, цей параметр відображає скільки конкретному замовнику потрібно товару. Знову ж таки, цей параметр існує тому, що поставлена задача є розширенням задачі з об'ємом перевезення.

- private Double locationX, цей параметр, аналогічно X координаті вантажівки є X координатою клієнта, його довготою. Рівно як і інші координати отримується за допомогою Google Maps API.
- private Double locationY, це є Y координата клієнту, його шириною.

Також, як і в попередній моделі, всі параметри є обов'язковими, без, хоча б, одного модель буде некоректною та сервер поверне помилку.

І останньою моделлю у системі є вихідна модель, яка відображає дані та рішення задачі. Серед глобальних даних цієї моделі є:

- public VehicleTypes vehicleTypes, об'єкт який в собі містить дані про кожну існуючу вантажівку, їх типи. Серед даних це ідентифікатори типів машин, щоб потім можна було вантажівкам закріпити тип за ним, також для цих типів існує їх вантажопідйомність, тип вантажопідйомності (об'єм або вага), також для кожного типу існують його «розцінки» такі як ціна за одиницю часу простою, ціна за одиницю дистанції, ціна за одиницю часу проведення обслуговування, і це все може використовуватись як для просто підрахунку ціни подорожі, так і для зміни задачі під іншу цільову функцію.
- public Vehicles vehicles, це об'єкт що містить всю інформацію про кожну вантажівку, а саме X та Y початкову координату кожної з них, ідентифікатор цієї координати, «розклад» для вантажівки, час коли вона виїжджає та коли вона повертається, параметр який визначає чи вантажівка має повернутися назад у депо, чи це не потрібно (але по умові задачі це потрібно, тому параметр завжди має значення true), для кожної вантажівки прикріплений її персональний ідентифікатор та ідентифікатор її типу, а також координати кінцевої локації вантажівки (так як по умові задачі вантажівка завжди повертається у

депо, то ці координати завжди такі ж, як і початкові координати) та ідентифікатор цих координат.

- public Services services, це об’єкт який містить інформацію про кожного конкретного клієнта. Містяться такі дані як час скільки вантажівка розвантажувалась у клієнта, часове вікно цього клієнта яке, знову ж таки, складається з двох частин, координати клієнта та ідентифікатор цих координат, ідентифікатор самого клієнта, його тип, та кількість товару яка йому потрібна та індекс вантажу (об’єм або вага).
- public Solutions solutions, це об’єкт-розв’язок задачі. У цій частині зберігаються короткі дані, такі як координати початкові, ті ідентифікатор вантажівки, про кожну вантажівку, яка задіяна у розв’язку, також разом з даними про цю вантажівку йде список клієнтів, до яких вона закріплена. Цей список вже відсортований у правильному порядку, у тому в якому вантажівка має поїхати, і цей список також містить дані про клієнтів, такі як ідентифікатор клієнта, його координати та час скільки займе поїздка до нього.
- public ProblenType problemType, це останній об’єкт у моделі який містить в собі лише тип вирішуваної задачі. Фреймворк дозволяє вирішувати задачу двох типів «кінцеву» та «нескінченну». Для кінцевої задачі береться як факт те, щоб вантажівки один раз проїхали по деяким клієнтам, та повернувшись в депо вона в ньому залишаються доки не буде поставлена нова задача. Для нескінченної береться як факт те що, вантажівки «циркулюють» по тим самим клієнтам час від часу. Тобто ця задача постійно повторюється з одними і тими же даними. Для реалізації системи був обраний «кінцевий» підхід, так як він реалізується дещо простіше, і не має ніяких суттєвих відмінностей.

Лістинг коду моделей знаходиться у Додатку А.

3.3 Модуль «Обробник помилок»

Так як власний обробник помилок є хорошим тоном взагалі при розробці він також був реалізований. Більше того він дає більшу гнучкість розробникам при створенні власних помилок, їх кодів та опису. Також такий обробник помилок дає більш розгорнуті та зрозумілі повідомлення користувачеві в разі якої небудь помилки. Так як у цьому проекті не так вже й багато способів створити помилку, то різних помилок не так вже й багато, тому модуль, що відповідає за їх обробку теж не дуже великий.

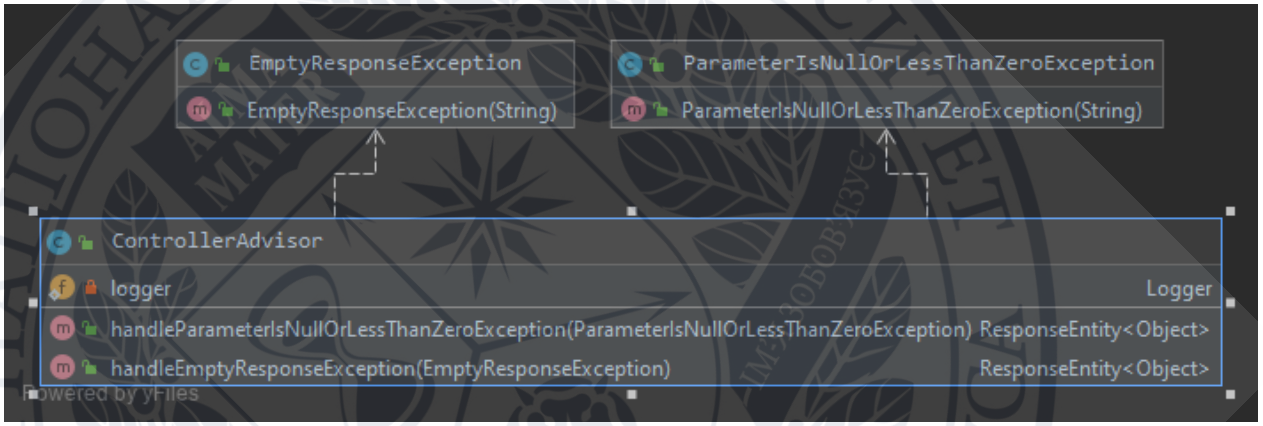


Рисунок 3.7 – Діаграма взаємодії помилок та їх «радника»

Загалом передбачено лише два типи помилок, це помилка 400 Bad Request та 422 Unprocessable Entity. Помилка 422 виникає в тому разі, якщо якогось параметру не вистачає, або він некоректно був введений. Цю помилку може створити лише валідатор який перевіряє всі вхідні запити на сервер.

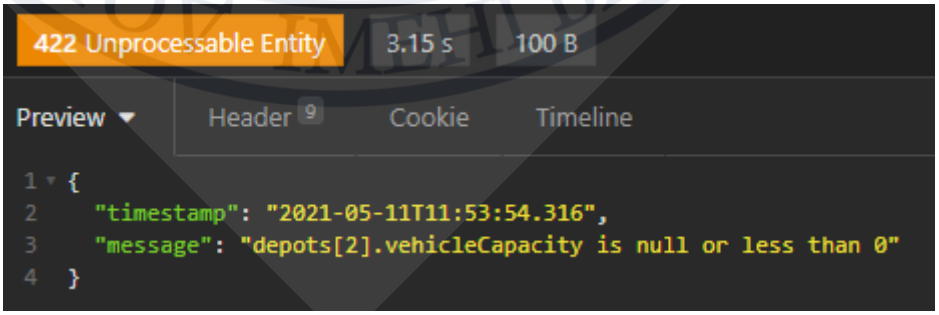


Рисунок 3.8 – Приклад помилки 422 Unprocessable Entity

Якщо вхідні дані пройшли валідацію, то вони передаються далі, і підчас виконання обчислень виникають з ним проблеми, або ж якщо були отримані дані які з точки зору своєї наявності та відповідності мінімальним перевіркам є нормальними, але ж з ними неможливо вирішити задачу, то сервер поверне помилку 400 Bad Request. Наприклад у разі коли всі клієнти будуть потребувати більше товару, ніж будь яка з вантажівок може перевозити.

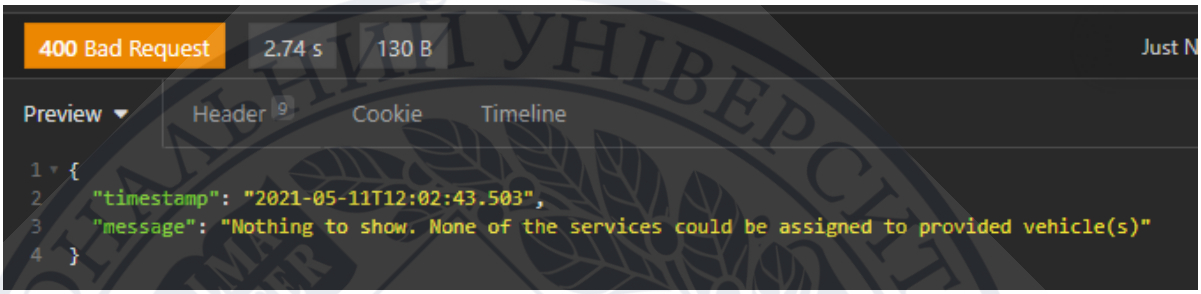


Рисунок 3.9 – Приклад помилки 400 Bad Request

Всі помилки, що відбуваються на сервері логуються, та можуть бути переглянуті якщо потрібно. У разі якщо виникає якась незначна помилка, наприклад якщо сервер не може привести один тип даних до іншого, він спочатку спробує усунути цю помилку самостійно, і тільки в разі якщо у нього це не виходить, то повертається помилка користувачеві. Зазвичай у такому випадку повертається помилка 500 Internal Server Error, але все одно у ній є повідомлення з вказівкою що саме пішло не так і де потрібно шукати помилку. Сервер ніколи не поверне користувачеві помилку з вказівкою на номер рядку коду де вона відбулась, або який метод її визвав. Для користувача ніколи не буде видно stack trace додатку.

Цей обробник будувався щоб забезпечити потенційного користувача та розробника front-end частини максимально зручним та дружнім інтерфейсом взаємодії.

Лістинг коду обробника помилок знаходиться у Додатку Б.

3.4 Модуль «Валідатор моделі»

Валідатор існує для того, щоб перевіряти тіло вхідного JSON’у на наявність усіх обов’язкових параметрів та їх коректність.

Перевірка даних є важливою частиною будь-якого проекту, незалежно від того, чи завдання полягає у зборі даних, аналізу даних чи підготовки даних до якогось представлення. Якщо дані не є коректними з самого початку, то результати також точно не будуть коректними. Ось чому необхідно перевірити та перевірити дані перед їх використанням.

Більше того, дані, що приходять зі сторони front-end частини обов’язково потрібно перевірити, так як ці дані можуть бути змінені багатьма варіантами, щоб обійти будь які валідації на цій стороні.

Тому кожен раз коли на сервер приходять будь які дані, сервер намагається знайти серед них лише тік, які потрібні, причому повністю ігнорує будь які дані які не відносяться до вхідної моделі, а ті які відносяться перевіряє

Для прикладу потрібно розглянути коректну модель.

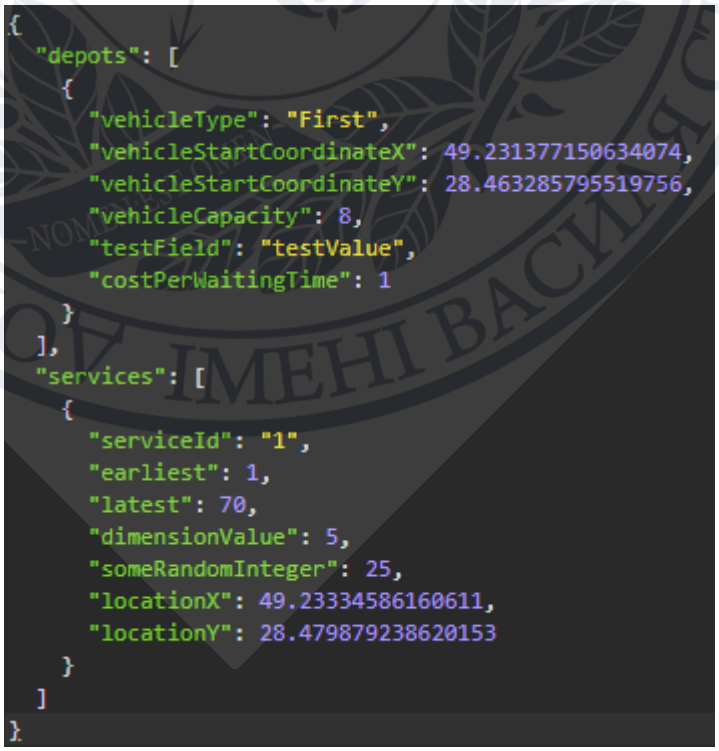


Рисунок 3.10 – Коректно побудована модель

У цій моделі є всі обов’язкові поля, ніяке з них не виходить за «дозволені» межі, тобто не є меншим за нуль (за виключенням координат), але у ній є надлишкові дані у вигляді testField: testValue та someRandomInteger: 25, але це не є проблемою адже сервер ці дані просто проігнорує і буде працювати лише з тими, які були представлені в моделях раніше.

Тепер розглянемо не правильно побудовану модель.

Рисунок 3.11 – Некоректно побудована модель

У такій моделі, по-перше, від’ємна вантажопідйомність вантажівки, що не пройде валідацію та повернеться помилка, по-друге, параметр latest менший за earlist, що спричинить логічну помилку адже таке часове вікно не може існувати, по-третє не вистачає Y координати для клієнта, що зробить неможливим побудову його локації та прокладення до нього шляху, і це теж спричинить повернення користувачеві помилки. І знову ж таки, якщо б тут були якісь додаткові параметри, то сервер просто їх проігнорував.

Лістинг коду валідатора моделі знаходиться у Додатку В.

3.5 Модуль «Конфігурування документації»

Також однією з найважливіших частин будь якого проекту, нехалежно від його розміру, є його документування, опис його роботи, опис даних.

Для документування проекту був використаний тип документації Swagger 2. Swagger – це фреймворк для специфікації RESTful API. Крім того, що цей фреймворк дозволяє не тільки інтерактивно переглядати інформацію щодо контролеру, так ще й надає змогу прямо з документації відправляти запити до серверу через Swagger UI.

Ця документації пишеться на основі коду. Все що потрібно це додати залежність у проект через фреймворк автоматичної збірки проектів Maven, у контролерах то моделях просто разставити потрібні анотації, та сконфігурувати створення самої документації.

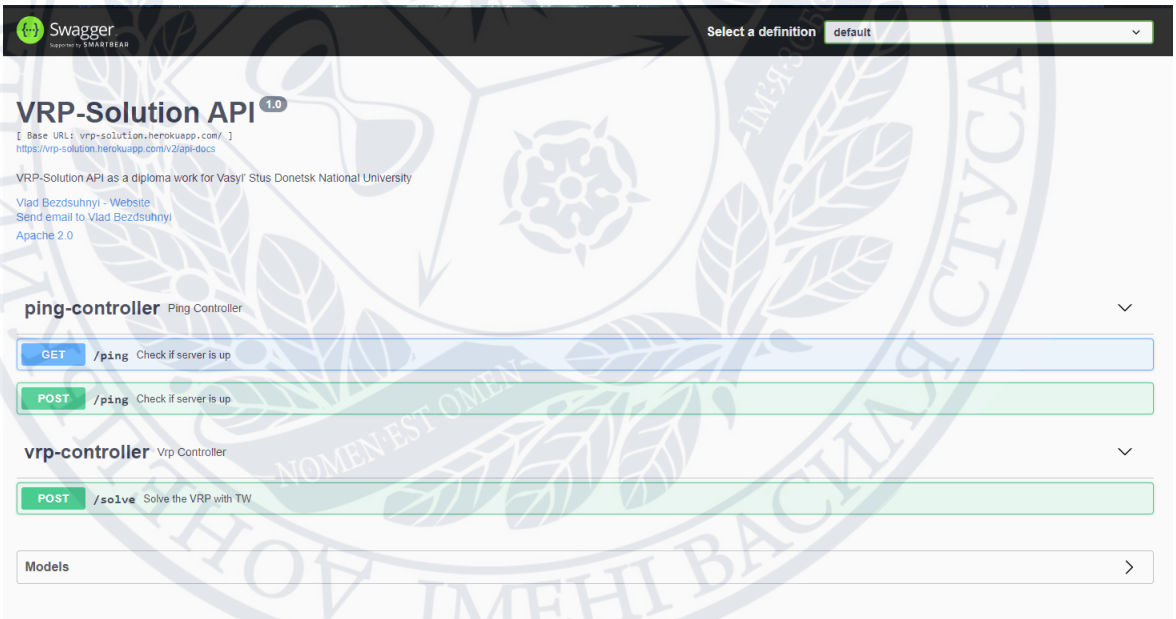


Рисунок 3.12 – Приклад Swagger-UI

Як можна побачити з рисунку, Swagger документує всі потрібні розробнику контролери, документує кожен ендпоінт який присутній у цьому контролері, метод який потрібний для роботи ендпоінта. Для кожного ендпоінта є список параметрів, як обов’язкових, так і опціональних, є список всіх можливих кодів

відповідей серверу та опис повідомлення яке повертається при кожному конкретному коді.

Останнім, але важливим, функціоналом є можливість прямо з документації відправити запит на сервер та отримати на нього відповідь.

Нижче наведений приклад такого запиту.

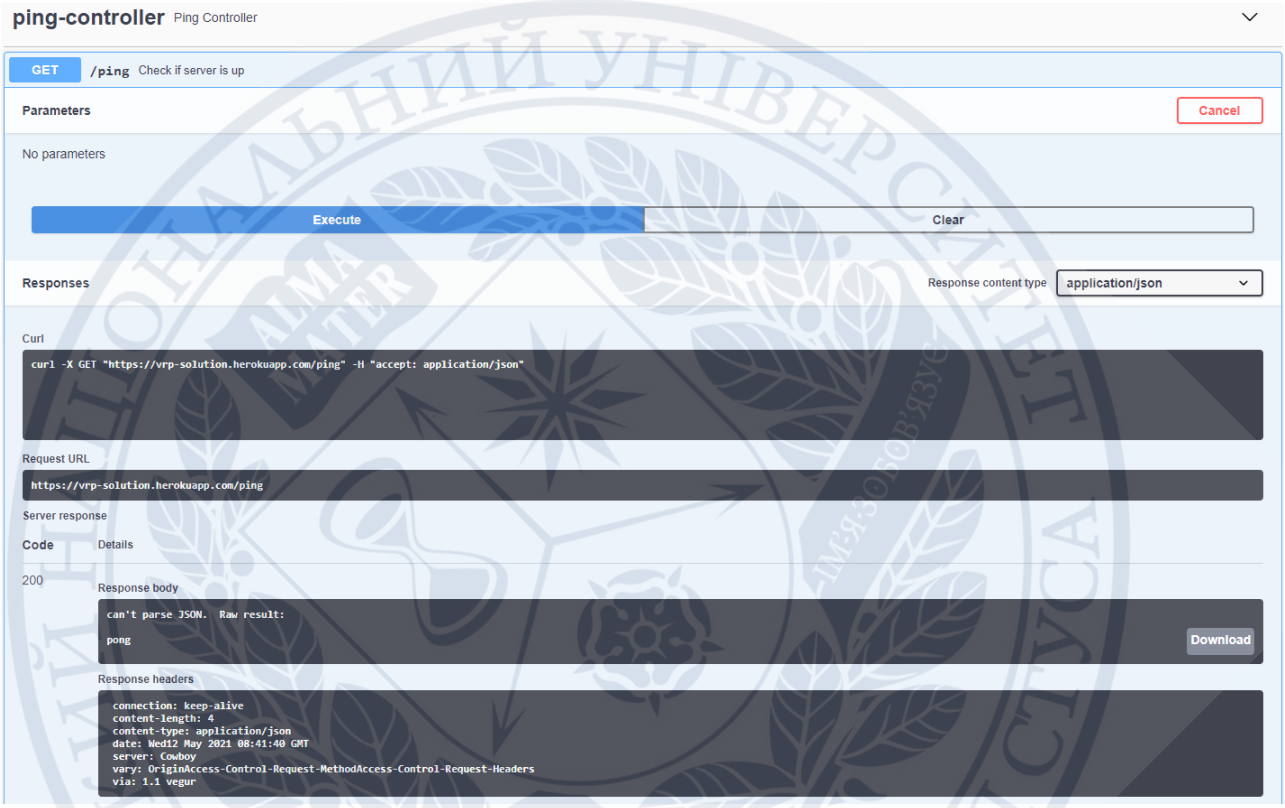


Рисунок 3.13 – Приклад запиту через Swagger UI

За допомгою кнопки «Execute» відправляється запит до серверу, і ничже можна бачити відповідь від нього. У цьому конкретному випадку на запит до ендпоінту «/ping» користувач отримає відповідь «pong».

Лістинг коду конфігуратора документації знаходиться у Додатку Г.

3.6 Модуль «Контролери та бізнес-логіка»

У проекті існує всього два контролери. Перший з них це ping-controller який містить у собі всього один ендпоінт з двома HTTP методами, POST та GET.

HTTP POST – це метод при якому дані приймаються сервером у тілі запиту.

HTTP GET – це метод при якому, зазвичай, просто отримують дані якогось ресурсу. Проте через нього також можна передавати параметри за допомогою

адресного рядка. Наприклад «https://www.example.com/test?someParam=value». Як видно після спеціального символу «?» був переданий параметр «someParam» зі значенням «value».

Контролер може приймати запити цих двох типів, і якщо сервер не відключений, тобто працює, то користувачеві повернеться у обох випадках текст «pong». Цей текст сигналізує про те що, сервер можна використовувати, він працює, не спить. Також ці ендпоінти можна використовувати для того, щоб сервер «пінганути» під час того, поки завантажується сторінка front-end частини, щоб сервер одразу був готовий до роботи, тому що якщо сервер довго не використовується, він переходить до режиму сну и щоб з нього вийти треба деякий час, зазвичай приблизно 15-20 секунд.

Другим, та основним контролером є vrp-controller. У нього є всього один ендпоінт «/solve» з методом POST. Саме цей ендпоінт приймає JSON з front-end частини та повертає у відповіді вирішену задачу. Саме цей ендпоінтом є «вікном» зв’язку сервера, бізнес-логіки, з front-end частиною. Цей ендпоінт приймає дані у заданому форматі, передає їх на валідацію, і в разі якщо все добре, то передає бізнес логікі на обробку.

```
{
  "depots": [
    {
      "costPerWaitingTime": 0,
      "vehicleCapacity": 0,
      "vehicleStartCoordinateX": 0,
      "vehicleStartCoordinateY": 0,
      "vehicleType": "string"
    }
  ],
  "services": [
    {
      "dimensionValue": 0,
      "earliest": 0,
      "latest": 0,
      "locationX": 0,
      "locationY": 0,
      "serviceId": "string"
    }
  ]
}
```


Рисунок 3.14 – Формат вхідного JSON’у

Після того як всі дані будуть передані до бізнес-логіки починається будуватись задача.

Спочатку обирається тип вантажу (вага або об’єм). Задача завжди працює з вагою, щоб уникнути неоднозначностей та конфліктів коли вантажівка везе якусь вагу якогось товару, але клієнт потребує об’єм якогось іншого товару. Далі створюється пустий список клієнтів, а також записується список клієнтів з вхідної моделі. Далі кожен з клієнтів у вхідній моделі перетворюються на список клієнтів який потрібний для фреймворку. Для кожного клієнта з вхідної моделі створюється, за допомогою патерну Builder, об’єкт Service який має такі аргументи: ідентифікатор, часове вікно, індекс, що відповідає за тип вантажу та кількість цього вантажу та будується локація цього клієнту на основі вхідних координат. Кожен такий об’єкт записується в відповідний список.

Далі за допомогою все того ж патерну Builder та патерну Singleton, що гарантує існування тільки одного об’єкту цього типу у системі, створюється, поки що, пустий об’єкт задачі.

Потім для кожного депо з вхідної моделі будується відповідні об’єкти VehicleTypeImpl та VehicleImpl. VehicleTypeImpl будується залежно від того, скільки депо існує, від того яка вантажопідйомність у вантажівок у цьому депо та яка вартість для вантажівок у цьому депо, у цьому рішенні це вартість простою, і також кожен тип має свій ідентифікатор. Якщо буде декілька однакових депо по цим параметрам, то у системі не буди для кожного депо свій тип, вони всі отримають один і той же, збудований на основі першого. Далі вже будуються самі вантажівки VehicleImpl, кількість яких завжди дорівнює кількості у вхідній моделі. Вантажівка будується отримуючи такі параметри як тип депо, стартові координати, які беруться з вхідної моделі і потім всі вантажівки також записуються у відповідний список.

Після чого до пустого об’єкту задачі додається усі вантажівки а також клієнти, далі для задачі обирається тип, а саме «кінцевий» і задача будується.

Наступним кроком є створення алгоритму, який іде «прямо з коробки» разом з фреймворком. Далі створюється колекція рішень на основі ітерацій

руйнування та відтворення за Девідом Пісінгером та Стефаном Ропком, про цей алгоритм писалося у розділі 1, та з цієї колекції обирається найкраще рішення яке потім передається назад з методів фреймворку у код програми.

Після того як найкраще рішення було обране та повернути назад потрібно його зберегти як JSON, так як фреймворк повертає або відформатований текст, або XML файл, потрібно дописати невеликий метод який просто трансформує це до JSON'у. Після того як ці задачі були виконані це рішення повторно перевіряється за допомогою Microsoft Virtual Earth сервісу. Virtual Earth дуже схожий на Google Maps але набагато кращий від нього, з точки зору розробки та тестування, так як не має таких суворих обмежень.

Для кожного об'єкту з масиву маршрутів вантажівок за допомогою цього сервіса повторно перераховується час подорожі від попередньої локації до поточної.

Базова адреса сервісу це «http://dev.virtualearth.net/REST/v1/Routes». Серед параметрів обов'язковим є API ключ (key), який можна безкоштовно отримати на персональній сторінці Bing Maps. Далі приймається параметр «waypoint.n» який передає координати локації під номером n. Приклади використання: waypoint.1=47.610,-122.107 (координати); wp.1=Seattle,WA (орієнтир); waypoint.1=1%20Microsoft%20Way%20Redmond%20WA%20 (адреса). Номер локації може починатись як з нуля, так і з одиниці, але всі наступні локації мають мати номер на один більший ніж попередній, іншими словами дати локації з номер один, а потім під номер три є некоректним та приведе до помилки, так як номер два пропущений. Наступним параметром передається параметр «avoid» який призначений для того, щоб уникати якісь частини дороги. Він приймає значення «minimizeTolls», що означає, що шлях буде будуватися з мінімізацією платних доріг на ньому, проте цей параметр може набувати і інших значень, таких як:

- highways – повністю уникати шосе;

- tolls – повністю уникати платні дороги;
- ferry – повністю уникати переправи, minimizeHighways – мінімізація шосе у дорозі;
- borderCrossing – уникати перетин кордону країн.

Наступним параметром є «optimize», який визначає за яким параметром потрібно оптимізувати подорож, і приймає значення «timeWithTraffic» який визначає, що шлях треба оптимізувати шлях під мінімальний час з урахуванням поточного трафіку на дорогах, але як і інші параметри цей може набувати і інших значень:

- distance – мінімізувати шлях подорожі без урахування трафіку;
- time – мінімізувати час подорожі без урахування трафіку;
- timeAvoidClosure – мінімізувати час подорожі з уникненням перекритих доріг, без урахування трафіку.

Наступним параметром є «distanceUnit», який може мати лише два значення, а саме:

- Mile - миля;
- Kilometer - кілометр.

Звісно ж, він приймає значення Kilometer.

Останнім параметром є «travelMode» який має значення «Transit», хоча може мати і інші, такі як:

- Driving - водіння;
- Walking - пішки.

Запит до Virtual Earth з такими параметрами буде мати приблизно такий вигляд:

http://dev.virtualearth.net/REST/v1/Routes?waypoint.0=49.23445616040541,28.417748999611494&waypoint.1=49.22833284241811,28.39853100001332&key=***&o

ptimize=timeWithTraffic&avoid=minimizeTolls&distanceUnit=Kilometer&travelMode=Transit (API ключ був замінений на *** у цілях безпеки).

Останнім кроком є розширення відповіді серверу, так як відповідь за замовчуванням не містить у собі всі важливі поля, і тому до неї потрібно ще додати координати, отриманий час подорожі. Після того як це готово, остаточно формується вихідна модель та повертається як відповідь серверу.

Лістинг коду контролерів та бізнес логіки знаходиться у Додатку Д.

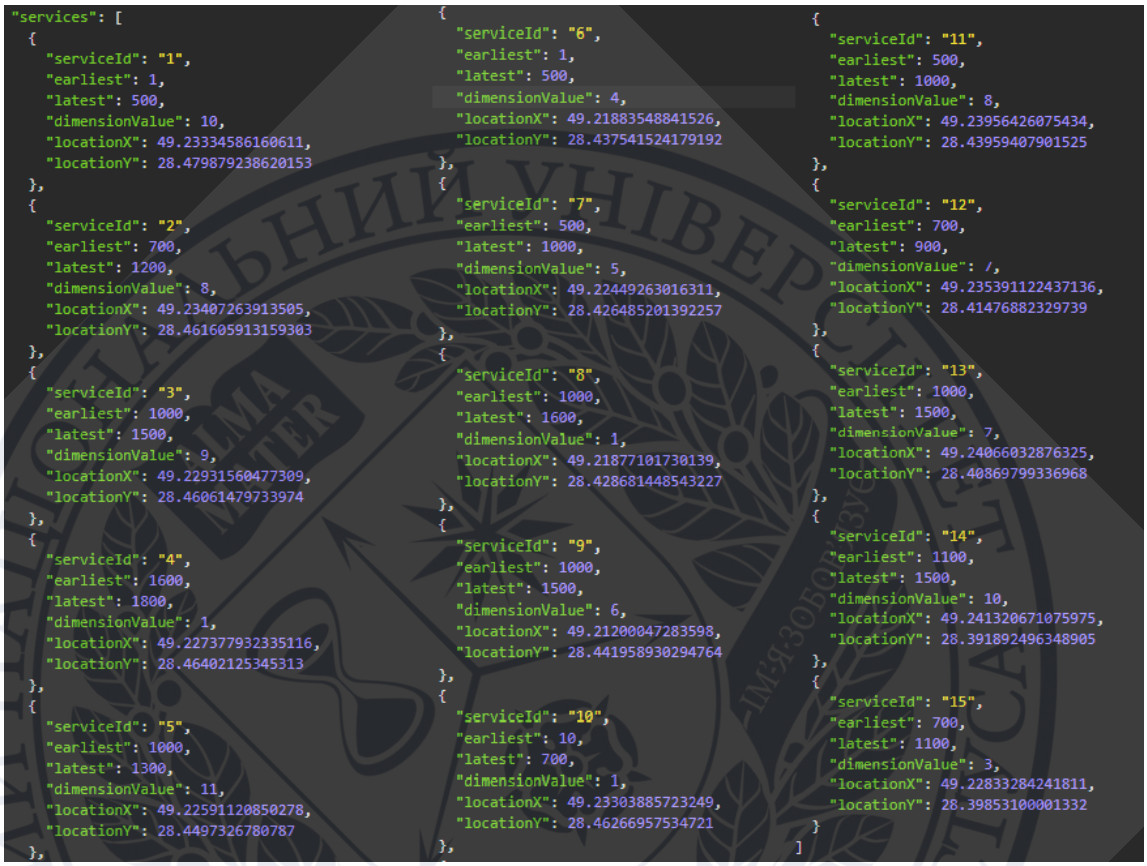
3.7 Тестування бізнес-логіки системи

Для тестування та демонстрації того як працює проект, потрібно обрати якийсь тестовий приклад.

Буде створена задача з трьома депо: вантажівка у першому депо буде розташована на координатах 49.231377150634074, 28.463285795519756 і буде мати вантажопідйомність 30 кг. Друга буде мати таку ж вантажопідйомність а координати її будуть 49.23445616040541, 28.417748999611494. У третьої вантажівки буде мати вантажопідйомність 25 кг. а координати будуть 49.224085597032904, 28.41158467155847.

Рисунок 3.15 – Вхідна модель Depots

Також, буде створені п’ятнадцять клієнтів розташованих на різних координатами, з різними часовими вікнами, також кожен з цих клієнтів потребують певну кількість товару.



```
"services": [
  {
    "serviceId": "1",
    "earliest": 1,
    "latest": 500,
    "dimensionValue": 10,
    "locationX": 49.23334586160611,
    "locationY": 28.479879238620153
  },
  {
    "serviceId": "2",
    "earliest": 700,
    "latest": 1200,
    "dimensionValue": 8,
    "locationX": 49.23407263913505,
    "locationY": 28.461605913159303
  },
  {
    "serviceId": "3",
    "earliest": 1000,
    "latest": 1500,
    "dimensionValue": 9,
    "locationX": 49.22931560477309,
    "locationY": 28.46061479733974
  },
  {
    "serviceId": "4",
    "earliest": 1600,
    "latest": 1800,
    "dimensionValue": 1,
    "locationX": 49.227377932335116,
    "locationY": 28.46402125345313
  },
  {
    "serviceId": "5",
    "earliest": 1000,
    "latest": 1300,
    "dimensionValue": 11,
    "locationX": 49.22591120850278,
    "locationY": 28.4497326780787
  },
  {
    "serviceId": "6",
    "earliest": 1,
    "latest": 500,
    "dimensionValue": 4,
    "locationX": 49.21883548841526,
    "locationY": 28.437541524179192
  },
  {
    "serviceId": "7",
    "earliest": 500,
    "latest": 1000,
    "dimensionValue": 5,
    "locationX": 49.22449263016311,
    "locationY": 28.426485201392257
  },
  {
    "serviceId": "8",
    "earliest": 1000,
    "latest": 1600,
    "dimensionValue": 1,
    "locationX": 49.21877101730139,
    "locationY": 28.428681448543227
  },
  {
    "serviceId": "9",
    "earliest": 1000,
    "latest": 1500,
    "dimensionValue": 6,
    "locationX": 49.21200047283598,
    "locationY": 28.441958930294764
  },
  {
    "serviceId": "10",
    "earliest": 10,
    "latest": 700,
    "dimensionValue": 1,
    "locationX": 49.23303885723249,
    "locationY": 28.46266957534721
  },
  {
    "serviceId": "11",
    "earliest": 500,
    "latest": 1000,
    "dimensionValue": 8,
    "locationX": 49.23956426075434,
    "locationY": 28.43959407901525
  },
  {
    "serviceId": "12",
    "earliest": 700,
    "latest": 900,
    "dimensionValue": 7,
    "locationX": 49.235391122437136,
    "locationY": 28.41476882329739
  },
  {
    "serviceId": "13",
    "earliest": 1000,
    "latest": 1500,
    "dimensionValue": 7,
    "locationX": 49.24066032876325,
    "locationY": 28.40869799336968
  },
  {
    "serviceId": "14",
    "earliest": 1100,
    "latest": 1500,
    "dimensionValue": 10,
    "locationX": 49.241320671075975,
    "locationY": 28.391892496348905
  },
  {
    "serviceId": "15",
    "earliest": 700,
    "latest": 1100,
    "dimensionValue": 3,
    "locationX": 49.22833284241811,
    "locationY": 28.39853100001332
  }
]
```


Рисунок 3.16 – Вхідна модель Services

Після того як ці дані була введені, за допомогою REST сервісу Insomnia зробимо запит на адресу https://vrp-solution.herokuapp.com/solve . Зробивши запит, отримаємо відповідь, що буде наведена нижче.

Першій машині First потрібно їхати за таким маршрутом:

Депо --> Клієнт 1 --> Клієнт 10 --> Клієнт 11 -->Клієнт 15 --> Клієнт 2 --> Депо.

Друга машина має їхати за таким маршрутом:

Депо --> Клієнт 6 -->Клієнт 12 --> Клієнт 3 --> Клієнт 14 --> Депо

Третя машина повинна їхати за таким маршрутом:

Депо --> Клієнт 5 --> Клієнт 9 --> Клієнт 8 --> Клієнт 13 --> Депо

Клієнти з номером 4 та 7 були виключені з маршруту тому, що вантажівки не зможуть встигнути, або ж тому що вантажівкам не вистачить вантажу, щоб розгрузитися у клієнтів.

```
"act": [
  {
    "lng": 49.23334586160611,
    "endTime": 1,
    "type": "service",
    "serviceId": 1,
    "arrTime": 0.01856215407243411,
    "travelDurationTraffic": 295,
    "lat": 28.479879238620153,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.23303885723249,
    "endTime": 10,
    "type": "service",
    "serviceId": 10,
    "arrTime": 1.0175166676465608,
    "travelDurationTraffic": 297,
    "lat": 28.46266957534721,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.23956426075434,
    "endTime": 500,
    "type": "service",
    "serviceId": 11,
    "arrTime": 10.029600899853808,
    "travelDurationTraffic": 427,
    "lat": 28.43959407901525,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.22833284241811,
    "endTime": 700,
    "type": "service",
    "serviceId": 15,
    "arrTime": 500.05229449733815,
    "travelDurationTraffic": 648,
    "lat": 28.39853100001332,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.23407263913505,
    "endTime": 700.0688147098629,
    "type": "service",
    "serviceId": 2,
    "arrTime": 700.0688147098629,
    "travelDurationTraffic": 780,
    "lat": 28.461605913159303,
    "timeUnit": "seconds"
  }
],
"driverId": "noDriver",
"lng": 49.231377150634074,
"start": 0,
"end": 700.0731900807243,
"vehicleId": "First",
"lat": 28.463285795519756
},
{
"act": [
  {
    "lng": 49.21883548841526,
    "endTime": 1,
    "type": "service",
    "serviceId": 6,
    "arrTime": 0.03541319655784747,
    "travelDurationTraffic": 568,
    "lat": 28.437541524179192,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.235391122437136,
    "endTime": 700,
    "type": "service",
    "serviceId": 12,
    "arrTime": 1.0393283349036793,
    "travelDurationTraffic": 564,
    "lat": 28.41476882329739,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.22931560477309,
    "endTime": 1000,
    "type": "service",
    "serviceId": 3,
    "arrTime": 700.0519214917064,
    "travelDurationTraffic": 692,
    "lat": 28.46061479733974,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.241320671075975,
    "endTime": 1100,
    "type": "service",
    "serviceId": 14,
    "arrTime": 1000.0807273672938,
    "travelDurationTraffic": 1012,
    "lat": 28.391892496348905,
    "timeUnit": "seconds"
  }
],
"driverId": "noDriver",
"lng": 49.23445616040541,
"start": 0,
"end": 1100.032721013933,
"vehicleId": "Second",
"lat": 28.417748999611494
},
{
"act": [
  {
    "lng": 49.22591120850278,
    "endTime": 1000,
    "type": "service",
    "serviceId": 5,
    "arrTime": 0.0399736179901069,
    "travelDurationTraffic": 500,
    "lat": 28.4497326780787,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.21200047283598,
    "endTime": 1000.0216844834507,
    "type": "service",
    "serviceId": 9,
    "arrTime": 1000.0216844834507,
    "travelDurationTraffic": 418,
    "lat": 28.441958930294764,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.21877101730139,
    "endTime": 1000.0417325096677,
    "type": "service",
    "serviceId": 8,
    "arrTime": 1000.0417325096677,
    "travelDurationTraffic": 556,
    "lat": 28.428681448543227,
    "timeUnit": "seconds"
  },
  {
    "lng": 49.24066032876325,
    "endTime": 1000.083605276303,
    "type": "service",
    "serviceId": 13,
    "arrTime": 1000.083605276303,
    "travelDurationTraffic": 576,
    "lat": 28.40869799336968,
    "timeUnit": "seconds"
  }
],
"driverId": "noDriver",
"lng": 49.224085597032904,
"start": 0,
"end": 1000.1030666862222,
"vehicleId": "Third",
"lat": 28.41158467155847
}
]
```


Рисунок 3.17 – Вихідна модель рішення задачі

Ця модель передається на сторону front-end частини, щоб вона, за допомогою Google Maps, графічно побудувала маршрут.

Після тестування системи, вона була запущеною на сервері, щоб нею можна було користуватися з інтернету, і не потрібно було локально її розгортати. Для цього використовувався сервіс Heroku. За посиланням https://vrp-solution.herokuapp.com/swagger-ui/ можна переглянути документацію в режимі он-лайн та з неї протестувати роботу системи.

3.7 Висновки за розділом 3

Були пояснені технічні сторони реалізації, пояснено чому використовувались ті чи інші інструменти, як саме вони використовувались, яке місце займають у проекті ті що саме вони роблять. Архітектура системи складається з таких модулів: моделі, контролери та допоміжні класи, такі як валідатор, конфігуратор документації. Тестування виконувалось за допомогою REST клієнту Insomnia, яке показало працездатність системи та її коректну роботу.

ВИСНОВКИ

В результаті виконання цієї роботи були систематизовані дані щодо поставленої задачі, виявленні особливості інших аналогічних задач та як вони можуть бути використанні для поставленої задачі, щоб зробити її більш гнучкою та адаптивною, також були розглянуті підходи для реалізації цих задач, щоб знайти серед них найбільш підходящий.

Після цього були розглянуті варіанти рішення поставленої задачі, такі як написання рішення з нуля, написання його за допомогою фреймворку OptaPlanner або фреймворку Jsprit. Цих три варіанти були розглянуті на основі сформованих критерій, та був обраний найліпший варіант, а саме Jsprit, тому що цей фреймворк дозволяє швидко розгорнути «екосистему» в якій можна легко будувати задачі пов’язані з VRP, налаштовувати рішення та їх поєднувати.

Після оцінки існуючих рішень була розпочата розробка серверної частини для сервісу рішення VRPTW, яка може масштабуватись до MDVRPTW. Для розробки були використані такі фреймворки як Maven, для автоматичної збірки проектів, Spring, для більш простого створення RESTful API та Jsprit для розробки самої бізнес-логіки. Розробка була на мові програмування Java. Це все дало змогу швидко написати проект, мінімізувати роботу с файлам конфігурацій проекту, дещо зменшити об’єм потрібного коду та мати можливість розгорнути систему у будь якому середовищі без потреби додаткових адаптацій.

У результаті роботи були розроблені: бізнес-логіка яка вирішує поставлену задачу, REST контролери для прийняття та відправки даних, валідатор даних та обробник помилок, також була створена документація. Вся системи була запущена на віддалений сервер, що дає змогу будь якому іншому сервісу використовувати цю систему.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Comtois Claude, Slack Brian, Rodrigue Jean-Paul. The geography of transport systems (3rd ed.). London, 2013.
2. Toth, P., Vigo D. The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications. Philadelphia, 2002.
3. Mahmud Nafix, Haque Md. Mokammel. Solving Multiple Depot Vehicle Routing Problem (MDVRP) using Genetic Algorithm. Конференція International Conference on Electrical, Computer and Communication Engineering (ECCE), 2019.
4. Baker, E.K. and J.R. Shaffer. Solution Improvement Heuristic for the Vehicle Routing and Scheduling Problem with Time Window Constrainats. // Mathematical and Managenent Sciences Vol. 6, P. 261–300, 1986.
5. Martin Desrochers, Jacques Desrosiers and Marius Solomon. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows, 1987.
6. G. A. Croes, A method for solving traveling salesman problems. // Operations Research, P. 791-812, 1958.
7. Fred W. Glover, Manuel Laguna. Tabu Search, 1997.
8. Ahn, J. The Generalized Location Routing Problem with Profits for Planetary Surface Exploration and Terrestrial Applications. Massachusetts, 2008.
9. N. Azi, M. Gendreau, J-Y. Potvin. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. // Operational Research, Vol. 202, P.756–763, 2010.
10. Schrage, L. Formulation and Structure of More Complex-Realistic Routing and Scheduling Problems. // Networks 1981, P. 229–232
11. S. Anily, “The Vehicle Routing Problem with Delivery and Back-haul Options”, // Naval Research Logistics, Vol. 43, 1996, P. 415-434.

12. Metropolis, N., Rosenbluth, A., Rosenbluth M., Teller, A., Teller, E. Equations of state calculations by fast computing machines. // Chemical Physics, Vol. 21, P.1087-1091, 1983.
13. Davis, A. Handbook of Genetic Algorithms, New York, 1991.
14. Or, I. Traveling salesman type combinatorial optimization problems and their relation to the logistics of blood banking. Evanston, 1976.
15. Bramle, J. Simchi-Levi, D. The Logic of Logistics: theory, algorithms, and applications for logistics management, New York, 1997.
16. L. Schrage. Formulation and structure of more complex/realistic routing and scheduling problems. // Networks, Vol.11, P. 229–232, 1981.
17. Z. Feng. Multi-period vehicle routing problem with recurring dynamic time windows. Конференція Digtial Object Identifier, с. 1–6, 2011.
18. M. L. Fisher. Optimal solution of vehicle routing problems using minimum K-trees. // Operations Research, Vol. 42, P. 626–642, 1994.
19. A. W. J. Kolen, A. H. G. R. Kan, та H. W. J. M. Trienekens. Vehicle routing with time windows. // Operations Research, Vol. 35, номер 2, P. 266–273, 1987.
20. L. Bodin, L. Berman. Routing and Scheduling of School Buses by Computer. // Transportation Science, Vol. 13, P. 113-129, 1979.
21. F. Alonso, M.J. Alvarez, J.E. Beasley. A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. // Operations Research, Vol. 59, P. 963–976, 2008.
22. R. Ayadi, A.E. Elldrissi, Y. Benadada, El Hilali Alaou. Evolutionary algorithm for a green vehicle routing problem with multiple trips. Міжнародна конференція з питань логістики та управління операціями, с. 148–154, 2014.
23. N. Azi, M. Gendreau, Potvin J-Y. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. // Computers & Operations Research Vol. 41, P.167–173, 2014.

24. J. Brandão, A.Mercer. A tabu search algorithm for the multi-trip vehicle routing problem. // European Journal of Operational Research Vol. 100, P.180–191, 1997.
25. Laporte, G., Gendreau, M. Classical and modern heuristics for the vehicle routing problem. Журнал International Transactions in Operational Research, випуск 7, с. 285–300, 2000.
26. Golden, B., Wasil, E. The open vehicle routing problem: Algorithms, large-scale test problems, and computational results. // Computers and Operations Research Vol. 34, P. 2198-2930, 2007.
27. Sariklis, D., Powell, S. A Heuristic Method for the Open Vehicle Routing Problem. // Journal of the Operational Research Society, Vol. 51, P. 564–573, 2000.
28. Brandao, J. A Tabu Search Algorithm for the Open Vehicle Routing Problem. // European Journal of Operational Research, Vol. 157, P. 552–564 2004.
29. A. Archer, M. Bateni, M. Hajiaghayi, H. Karloff. Improved approximation algorithms for prize-collecting steiner tree and tsp. // SIAM Journal on Computing, Vol. 40, P. 309–332, 2011.
30. C. Archetti, N. Bianchessi, A. Hertz, M. G. Speranza. Incomplete service and split deliveries in a routing problem with profits. // Networks, Vol. 63, P. 135–145, 2014.
31. C. Archetti, N. Bianchessi, M. G. Speranza. Optimal solutions for routing problems with profits. // Discrete Applied Mathematics, Vol.161, P. 547–557, 2013.
32. S. Anily. The Vehicle Routing Problem with Delivery and Back-haul Options. // Naval Research Logistics, Vol. 43, P. 415-434, 1996.
33. N. Bianchessi, G. Righini. Heuristic Algorithms for the Vehicle Routing Problem with Simultaneous Pick-up and Delivery. // Computers & Operations Research, Vol. 34, P. 578–594, 2007.

34. J. Brandão. A New Tabu Search Algorithm for the Vehicle Routing Problem with Backhauls. // European Journal of Operational Research, Vol. 173, P. 540-555, 2006.
35. J.-F. Chen, T.-H. Wu. Vehicle Routing Problem with Simultaneous Deliveries and Pickups. // Journal of the Operational Research Society, Vol. 57, P. 579-587, 2006.
36. T. Vidal, T.G. Crainic, M. Gendreau, N. Lahrichi, W. Rei. A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. // Operational Research, Vol. 60, P. 611-624, 2013.
37. J. Renaud, G. Laporte, F.F. Boctor. A tabu search heuristic for the multi-depot vehicle routing problem. Comput. // Computers & Operations Research, Vol. 23, P. 229-235, 1996.
38. J. Lysgaard, A.N. Letchford, R.W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. // Mathematical Programming, Vol. 100, P. 423-445, 2004.
39. M. Dror. Note on the complexity of the shortest path models for column generation in VRPTW. Журнал Operational Research, Vol. 42, P. 977-978, 1994.
40. M. Desrochers, J. Desrosiers, M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. // Operational Research, Vol. 40, P. 342-354, 1992.
41. Vehicle Routing Problems And How To Solve Them. URL: https://dev.to/iedmrc/vehicle-routing-problems-and-how-to-solve-them-8h3

ДОДАТОК А

Код моделей

VrpModel

```
package com.diplom.vrp.models;
import io.swagger.annotations.ApiModelProperty;
import java.util.List;
public class VrpModel {
    @ApiModelProperty(notes = "List of depots", required = true)
    private List<DepotModel> depots;
    @ApiModelProperty(notes = "List of location to serve", required = true)
    private List<ServiceModel> services;
    public List<DepotModel> getDepots() {
        return depots;
    }
    public void setDepots(List<DepotModel> depots) {
        this.depots = depots;
    }
    public List<ServiceModel> getServices() {
        return services;
    }
    public void setServices(List<ServiceModel> services) {
        this.services = services;
    }
    @Override
    public String toString() {
        return "VrpModel{" +
                "depots=" + depots.toString() +
                ", services=" + services.toString() +
                '}';
    }
}
```

DepotModel

```
package com.diplom.vrp.models;
import io.swagger.annotations.ApiModelProperty;
public class DepotModel {
    @ApiModelProperty(notes = "Vehicle type")
    private String vehicleType;
    @ApiModelProperty(notes = "X coordinate of the vehicle's start location", required = true)
```

```
private Double vehicleStartCoordinateX;
@ApiModelProperty(notes = "Y coordinate of the vehicle's start location", required = true)
private Double vehicleStartCoordinateY;
@ApiModelProperty(notes = "Vehicle capacity", required = true)
private Integer vehicleCapacity;
@ApiModelProperty(notes = "Cost per waiting time unit, for instance € per second", required = true)
private Double costPerWaitingTime;
public String getVehicleType() {
   return vehicleType;
}
public void setVehicleType(String vehicleType) {
   this.vehicleType = vehicleType;
}
public Double getVehicleStartCoordinateX() {
   return vehicleStartCoordinateX;
}
public void setVehicleStartCoordinateX(Double vehicleStartCoordinateX) {
   this.vehicleStartCoordinateX = vehicleStartCoordinateX;
}
public Double getVehicleStartCoordinateY() {
   return vehicleStartCoordinateY;
}
public void setVehicleStartCoordinateY(Double vehicleStartCoordinateY) {
   this.vehicleStartCoordinateY = vehicleStartCoordinateY;
}
public Integer getVehicleCapacity() {
   return vehicleCapacity;
}
public void setVehicleCapacity(Integer vehicleCapacity) {
   this.vehicleCapacity = vehicleCapacity;
}
public Double getCostPerWaitingTime() {
   return costPerWaitingTime;
}
public void setCostPerWaitingTime(Double costPerWaitingTime) {
   this.costPerWaitingTime = costPerWaitingTime;
}
@Override
public String toString() {
```

```
        return "DepotModel{" +
                "vehicleType='" + vehicleType + '\'' +
                ", vehicleStartCoordinateX=" + vehicleStartCoordinateX +
                ", vehicleStartCoordinateY=" + vehicleStartCoordinateY +
                ", vehicleCapacity=" + vehicleCapacity +
                ", costPerWaitingTime=" + costPerWaitingTime +
                '}';
    }
}
```

ServiceModel

```
package com.diplom.vrp.models;
import io.swagger.annotations.ApiModelProperty;
public class ServiceModel {
    @ApiModelProperty(notes = "Service ID", required = true)
    private String serviceId;
    @ApiModelProperty(notes = "Earliest time to arrive", required = true)
    private Double earliest;
    @ApiModelProperty(notes = "Latest time to arrive", required = true)
    private Double latest;
    @ApiModelProperty(notes = "Dimension value", required = true)
    private Integer dimensionValue;
    @ApiModelProperty(notes = "X coordinate of the location", required = true)
    private Double locationX;
    @ApiModelProperty(notes = "Y coordinate of the location", required = true)
    private Double locationY;
    public ServiceModel(String serviceId, double earliest, double latest, int dimensionValue, double locationX, double
locationY) {
        this.serviceId = serviceId;
        this.earliest = earliest;
        this.latest = latest;
        this.dimensionValue = dimensionValue;
        this.locationX = locationX;
        this.locationY = locationY;
    }
    public ServiceModel() {
    }
    public String getServiceId() {
        return serviceId;
    }
```

```
public void setServiceId(String serviceId) {
   this.serviceId = serviceId;
}
public Double getEarliest() {
   return earliest;
}
public void setEarliest(Double earliest) {
   this.earliest = earliest;
}
public Double getLatest() {
   return latest;
}
public void setLatest(Double latest) {
   this.latest = latest;
}
public Integer getDimensionValue() {
   return dimensionValue;
}
public void setDimensionValue(Integer dimensionValue) {
   this.dimensionValue = dimensionValue;
}
public Double getLocationX() {
   return locationX;
}
public void setLocationX(Double locationX) {
   this.locationX = locationX;
}
public Double getLocationY() {
   return locationY;
}
public void setLocationY(Double locationY) {
   this.locationY = locationY;
}
@Override
public String toString() {
   return "ServiceModel{" +
         "serviceId='" + serviceId + '\'' +
         ", earliest=" + earliest +
         ", latest=" + latest +
```

```
            ", dimensionValue=" + dimensionValue +
            ", locationX=" + locationX +
            ", locationY=" + locationY +
            '}';
    }
}
```

ДОДАТОК Б

Код обробника помилок

```
package com.diplom.vrp.exceptions;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;
import java.time.LocalDateTime;
import java.util.LinkedHashMap;
import java.util.Map;
@ControllerAdvice
public class ControllerAdvisor extends ResponseEntityExceptionHandler {
   private static Logger logger = LoggerFactory.getLogger(ControllerAdvisor.class);
   @ExceptionHandler(ParameterIsNullOrLessThanZeroException.class)
   public ResponseEntity<Object>
handleParameterIsNullOrLessThanZeroException(ParameterIsNullOrLessThanZeroException e){
      Map<String, Object> body = new LinkedHashMap<>();
      body.put("timestamp", LocalDateTime.now());
      body.put("message", e.getMessage());
      logger.error("ParameterIsNullOrLessThanZeroException " + e.getMessage());
      return new ResponseEntity<>(body, HttpStatus.UNPROCESSABLE_ENTITY);
   }
   @ExceptionHandler(EmptyResponseException.class)
   public ResponseEntity<Object> handleEmptyResponseException(EmptyResponseException e){
      Map<String, Object> body = new LinkedHashMap<>();
      body.put("timestamp", LocalDateTime.now());
      body.put("message", e.getMessage());
      logger.error("ParameterIsNullOrLessThanZeroException " + e.getMessage());
      return new ResponseEntity<>(body, HttpStatus.BAD_REQUEST);
   }
}
```

ДОДАТОК В

Код валідатора моделі

```
package com.diplom.vrp.utils;
import com.diplom.vrp.exceptions.ParameterIsNullOrLessThanZeroException;
import com.diplom.vrp.models.DepotModel;
import com.diplom.vrp.models.ServiceModel;
import com.diplom.vrp.models.VrpModel;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class ModelValidator {
    private static Logger logger = LoggerFactory.getLogger(ModelValidator.class);
    public VrpModel validateModel(VrpModel model){
        int depotCount = 0;
        for (DepotModel depotModel: model.getDepots()) {
            if (depotModel.getVehicleType() == null || depotModel.getVehicleType().isEmpty()) {
                logger.error("An error occurred with \"depots[" + depotCount + "].vehicleType\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("depots[" + depotCount + "].vehicleType");
            }
            if (depotModel.getVehicleStartCoordinateX() == null) {
                logger.error("An error occurred with \"depots[" + depotCount + "].vehicleStartCoordinateX\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("depots[" + depotCount + "].vehicleStartCoordinateX");
            }
            if (depotModel.getVehicleStartCoordinateY() == null) {
                logger.error("An error occurred with \"depots[" + depotCount + "].vehicleStartCoordinateY\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("depots[" + depotCount + "].vehicleStartCoordinateY");
            }
            if (depotModel.getVehicleCapacity() == null || depotModel.getVehicleCapacity() <= 0) {
                logger.error("An error occurred with \"depots[" + depotCount + "].vehicleCapacity\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("depots[" + depotCount + "].vehicleCapacity");
            }
            if (depotModel.getCostPerWaitingTime() == null || depotModel.getCostPerWaitingTime() <= 0) {
                logger.error("An error occurred with \"depots[" + depotCount + "].costPerWaitingTime\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("depots[" + depotCount + "].costPerWaitingTime");
            }
            depotCount++;
        }
        int serviceCount = 0;
        for (ServiceModel serviceModel: model.getServices()) {
```

```
            if (serviceModel.getServiceId() == null || serviceModel.getServiceId().isEmpty()) {
                logger.error("An error occurred with \"services[" + serviceCount + "].serviceId\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("services[" + serviceCount + "].serviceId");
            }
            if (serviceModel.getEarliest() == null || serviceModel.getEarliest() <= 0) {
                logger.error("An error occurred with \"services[" + serviceCount + "].earliest\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("services[" + serviceCount + "].earliest");
            }
            if (serviceModel.getLatest() == null || serviceModel.getLatest() <= 0) {
                logger.error("An error occurred with \"services[" + serviceCount + "].latest\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("services[" + serviceCount + "].latest");
            }
            if (serviceModel.getDimensionValue() == null || serviceModel.getDimensionValue() <= 0) {
                logger.error("An error occurred with \"services[" + serviceCount + "].dimensionValue\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("services[" + serviceCount + "].dimensionValue");
            }
            if (serviceModel.getLocationX() == null) {
                logger.error("An error occurred with \"services[" + serviceCount + "].locationX\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("services[" + serviceCount + "].locationX");
            }
            if (serviceModel.getLocationY() == null) {
                logger.error("An error occurred with \"services[" + serviceCount + "].locationY\" parameter");
                throw new ParameterIsNullOrLessThanZeroException("services[" + serviceCount + "].locationY");
            }
            serviceCount++;
        }
        return model;
    }
}
```

ДОДАТОК Г

Код конфігуратора документації

```
package com.diplom.vrp.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;
@Configuration
@EnableSwagger2
public class SpringFoxConfig {
  @Bean
  public Docket api() {
    return new Docket(DocumentationType.SWAGGER_2)
        .useDefaultResponseMessages(false)
        .select()
        .apis(RequestHandlerSelectors.basePackage("com.diplom.vrp.controllers"))
        .paths(PathSelectors.any())
        .build()
        .apiInfo(apiInfo());
  }
  private ApiInfo apiInfo() {
    return new ApiInfoBuilder().title("VRP-Solution API")
        .description("VRP-Solution API as a diploma work for Vasyl' Stus Donetsk National University")
        .license("Apache 2.0").contact(new Contact("Vlad Bezdsuhnyi",
            "https://www.linkedin.com/in/vlad-bezdushnii/", "vladibzd@gmail.com"))
        .licenseUrl("http://www.apache.org/licenses/LICENSE-2.0").version("1.0").build();
  }
}
```

ДОДАТОК Д

Код контролерів та бізнес-логіки

Ping Controller

```
package com.diplom.vrp.controllers;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@CrossOrigin
@Api(value = "Ping controller")
public class PingController {
  private static Logger logger = LoggerFactory.getLogger(PingController.class);
  @ApiOperation(value = "Check if server is up")
  @ApiResponses(value = {
      @ApiResponse(code = 500, message = "A monkey is trying to deal with this situation"),
      @ApiResponse(code = 200, message = "pong")
  })
  @PostMapping(path = "/ping", produces = "application/json")
  public String isAlivePost(){
    logger.info("Entering POST /ping endpoint");
    return "pong";
  }
  @ApiOperation(value = "Check if server is up")
  @ApiResponses(value = {
      @ApiResponse(code = 500, message = "A monkey is trying to deal with this situation"),
      @ApiResponse(code = 200, message = "pong")
  })
  @GetMapping(path = "/ping", produces = "application/json")
  public String isAliveGet(){
    logger.info("Entering GET /ping endpoint");
    return "pong";
```

```
    }
}
```

Vrp Controller

```
package com.diplom.vrp.controllers;
import com.diplom.vrp.models.OutputModel;
import com.diplom.vrp.models.VrpModel;
import com.diplom.vrp.utils.MultipleTimeWindowSolution;
import io.swagger.annotations.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.*;
@RestController
@CrossOrigin(maxAge = 3600)
@Api(value = "Main controller")
public class VrpController {
    private static Logger logger = LoggerFactory.getLogger(VrpController.class);
    @ApiImplicitParams({
            @ApiImplicitParam(name = "model", value = "A JSON representation of vehicle's parameters and services
data")
    })
    @ApiOperation(value = "Solve the VRP with TW", response = OutputModel.class)
    @ApiResponses(value = {
            @ApiResponse(code = 500, message = "A monkey is trying to deal with this situation"),
            @ApiResponse(code = 422, message = "`ParamName` is null or less than 0"),
            @ApiResponse(code = 200, message = "A JSON representation of optimal routes")
    })
    @PostMapping(path = "/solve", consumes = "application/json", produces = "application/json")
    public String solve(@RequestBody VrpModel model){
        logger.info("Entering /solve endpoint");
        MultipleTimeWindowSolution solution = new MultipleTimeWindowSolution();
        return solution.solve(model);
    }
}
```

Бізнес-логіка

```
package com.diplom.vrp.utils;
import com.diplom.vrp.exceptions.EmptyResponseException;
import com.diplom.vrp.exceptions.ParameterIsNullOrLessThanZeroException;
```

```
import com.diplom.vrp.models.DepotModel;
import com.diplom.vrp.models.ServiceModel;
import com.diplom.vrp.models.VrpModel;
import com.graphhopper.jsprit.core.algorithm.VehicleRoutingAlgorithm;
import com.graphhopper.jsprit.core.algorithm.box.Jsprit;
import com.graphhopper.jsprit.core.problem.Location;
import com.graphhopper.jsprit.core.problem.VehicleRoutingProblem;
import com.graphhopper.jsprit.core.problem.job.Service;
import com.graphhopper.jsprit.core.problem.solution.VehicleRoutingProblemSolution;
import com.graphhopper.jsprit.core.problem.vehicle.VehicleImpl;
import com.graphhopper.jsprit.core.problem.vehicle.VehicleTypeImpl;
import com.graphhopper.jsprit.core.reporting.SolutionPrinter;
import com.graphhopper.jsprit.core.util.ManhattanCosts;
import com.graphhopper.jsprit.core.util.Solutions;
import com.graphhopper.jsprit.io.problem.VrpXMLWriter;
import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.HttpHeaders;
import org.apache.http.NameValuePair;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.json.XML;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.File;
import java.io.FileNotFoundException;
import java.net.URI;
import java.util.*;
```

```
public class MultipleTimeWindowSolution {

   private static Logger logger = LoggerFactory.getLogger(MultipleTimeWindowSolution.class);
   private final CloseableHttpClient httpClient = HttpClients.createDefault();

   private static final String BASE_VIRTUAL_EARTH_URL = "http://dev.virtualearth.net/REST/v1/Routes";
   private final String API_KEY = "***";

   private static void wait(int ms)
   {
      try
      {
         Thread.sleep(ms);
      }
      catch(InterruptedException ex)
      {
         Thread.currentThread().interrupt();
      }
   }
   private int getTravelDurationWithTraffic(double x1, double y1, double x2, double y2) throws Exception{
      HttpGet request = new HttpGet(BASE_VIRTUAL_EARTH_URL);
      List<NameValuePair> urlParameters = new ArrayList<>();
      urlParameters.add(new BasicNameValuePair("waypoint.0", x1 + "," + y1));
      urlParameters.add(new BasicNameValuePair("waypoint.1", x2 + "," + y2));
      urlParameters.add(new BasicNameValuePair("optimize", "timeWithTraffic"));
      urlParameters.add(new BasicNameValuePair("key", API_KEY));
      URI uri = new URIBuilder(request.getURI()).setParameters(urlParameters).build();
      request.addHeader(HttpHeaders.ACCEPT, "application/json");
      request.setURI(uri);
      logger.info(String.valueOf(request.getURI()));
      int travelDuration = -1;
      try (CloseableHttpResponse response = httpClient.execute(request)){
         Header rateLimitHeader = response.getHeaders("X-MS-BM-WS-INFO")[0];
         if (rateLimitHeader.getValue().equals(String.valueOf(1))){
            wait(3000);
         }
         HttpEntity entity = response.getEntity();
         if (response.getStatusLine().getStatusCode() != 200) {
            logger.error("Error with VE API call. URI of the call: " + request.getURI());
```

```
            return travelDuration;
        }
        if (entity != null) {
            String result = EntityUtils.toString(entity);
            JSONObject jsonResult = new JSONObject(result);
            JSONObject trafficRoute =
jsonResult.getJSONArray("resourceSets").getJSONObject(0).getJSONArray("resources").getJSONObject(0);
            travelDuration = trafficRoute.getInt("travelDurationTraffic");
            return travelDuration;
        }
    }
    return travelDuration;
  }

  public String solve(VrpModel model){
    ModelValidator validator = new ModelValidator();
    model = validator.validateModel(model);
    logger.info("Input model: " + model.toString());
    final int WEIGHT_INDEX = 0; //0 means weight (e.g. 2700kg), 1 means volume (e.g. 17m^3)

    List<Service> serviceList = new ArrayList<>();
    List<ServiceModel> serviceModelList = model.getServices();

    for (ServiceModel serviceModel: serviceModelList) {
      Service service = Service.Builder.newInstance(serviceModel.getServiceId())
          .addTimeWindow(serviceModel.getEarliest(), serviceModel.getLatest())
          .addSizeDimension(WEIGHT_INDEX, serviceModel.getDimensionValue())
          .setLocation(Location.newInstance(serviceModel.getLocationX(), serviceModel.getLocationY()))
          .build();
      serviceList.add(service);
    }

    VehicleRoutingProblem.Builder vrpBuilder = VehicleRoutingProblem.Builder.newInstance();
    int depotCounter = 1;
    for (DepotModel depotModel: model.getDepots()) {
        VehicleTypeImpl vehicleType = VehicleTypeImpl.Builder.newInstance(depotCounter + "_type")
            .addCapacityDimension(WEIGHT_INDEX, depotModel.getVehicleCapacity())
            .setCostPerWaitingTime(depotModel.getCostPerWaitingTime()).build();
        VehicleImpl vehicle = VehicleImpl.Builder.newInstance(depotModel.getVehicleType())
```

```
                .setStartLocation(Location.newInstance(depotModel.getVehicleStartCoordinateX(),
depotModel.getVehicleStartCoordinateY()))
                .setType(vehicleType).build();
            vrpBuilder.addVehicle(vehicle);
        depotCounter++;
    }

    for (Service service: serviceList) {
        vrpBuilder.addJob(service);
    }

    vrpBuilder.setFleetSize(VehicleRoutingProblem.FleetSize.FINITE);
    vrpBuilder.setRoutingCost(new ManhattanCosts());
    VehicleRoutingProblem problem = vrpBuilder.build();

    VehicleRoutingAlgorithm algorithm = Jsprit.createAlgorithm(problem);

    Collection<VehicleRoutingProblemSolution> solutions = algorithm.searchSolutions();

    VehicleRoutingProblemSolution bestSolution = Solutions.bestOf(solutions);


    new VrpXMLWriter(problem, solutions).write("problem-with-solution.xml");
    JSONObject jspritXMLOutputInJSON = null;
    try {
        logger.info("Trying to convert XML to JSON");
        File myObj = new File("problem-with-solution.xml");
        Scanner myReader = new Scanner(myObj);
        String data = null;
        while (myReader.hasNextLine()) {
            data += myReader.nextLine();
        }
        String dataWithoutNull = Objects.requireNonNull(data).substring(4);
        jspritXMLOutputInJSON = XML.toJSONObject(dataWithoutNull);
        myReader.close();
    } catch (FileNotFoundException e) {
        logger.error("Converting failed. File is not found " + e);
    }
```

```
JSONObject obj = new JSONObject(jspritXMLOutputInJSON.toString());
JSONArray solutionArray = obj.getJSONObject("problem").getJSONObject("solutions").getJSONArray("solution");

int travelDurationWithTraffic = 0;

if (solutionArray.getJSONObject(0).has("routes")) {
  if (model.getDepots().size() == 1) {
    for (int i = 0; i < solutionArray.length(); i++) {
      JSONArray tmp = new JSONArray();
      try{
        tmp =
solutionArray.getJSONObject(i).getJSONObject("routes").getJSONObject("route").getJSONArray("act");
        solutionArray.getJSONObject(i).getJSONObject("routes").getJSONObject("route").put("lng",
model.getDepots().get(0).getVehicleStartCoordinateX());
        solutionArray.getJSONObject(i).getJSONObject("routes").getJSONObject("route").put("lat",
model.getDepots().get(0).getVehicleStartCoordinateY());
      } catch (JSONException e){
        logger.warn("Something wrong happened while getting an array " + e);

tmp.put(solutionArray.getJSONObject(i).getJSONObject("routes").getJSONObject("route").getJSONObject("act"));
      }
      travelDurationWithTraffic = 0;
      for (int j = 0; j < tmp.length(); j++) {

        JSONObject act = tmp.getJSONObject(j);
        int id = act.getInt("serviceId");

        JSONObject nextAct = null;
        int nextId = -1;
        if (j + 1 != tmp.length()) {
          nextAct = tmp.getJSONObject(j + 1);
          nextId = nextAct.getInt("serviceId");
        }
        for (ServiceModel serviceModel : serviceModelList) {
          if (serviceModel.getServiceId().equals(String.valueOf(id))) {
            if (j == 0) {
              try {
```

```
                    travelDurationWithTraffic =
getTravelDurationWithTraffic(model.getDepots().get(0).getVehicleStartCoordinateX(),
model.getDepots().get(0).getVehicleStartCoordinateY(),
                        serviceModel.getLocationX(), serviceModel.getLocationY());
                    act.put("travelDurationTraffic", travelDurationWithTraffic);
                    act.put("timeUnit", "seconds");
                    act.put("lng", serviceModel.getLocationX());
                    act.put("lat", serviceModel.getLocationY());
                    nextAct.put("travelDurationTraffic", travelDurationWithTraffic);
                    nextAct.put("timeUnit", "seconds");
                    nextAct.put("lng", serviceModel.getLocationX());
                    nextAct.put("lat", serviceModel.getLocationY());
                  } catch (Exception e) {
                    logger.error("Failed to send data to VE: " + e.getMessage());
                    e.printStackTrace();
                  }
                }
                for (ServiceModel nextModel : serviceModelList) {
                  if (nextId != -1 && nextModel.getServiceId().equals(String.valueOf(nextId))) {
                    try {
                      travelDurationWithTraffic = getTravelDurationWithTraffic(serviceModel.getLocationX(),
serviceModel.getLocationY(),
                          nextModel.getLocationX(), nextModel.getLocationY());
                      nextAct.put("travelDurationTraffic", travelDurationWithTraffic);
                      nextAct.put("timeUnit", "seconds");
                      nextAct.put("lng", nextModel.getLocationX());
                      nextAct.put("lat", nextModel.getLocationY());
                    } catch (Exception e) {
                      logger.error("Failed to send data to VE: " + e);
                    }
                  }
                }

              } else continue;
            }
          }
        }
      } else{
        JSONArray routes = new JSONArray();
```

```
try {
   routes = solutionArray.getJSONObject(0).getJSONObject("routes").getJSONArray("route");
} catch (JSONException e){
   logger.warn("Something wrong happened while getting an array " + e);
   routes.put(solutionArray.getJSONObject(0).getJSONObject("routes").getJSONObject("route"));
}
for (int i = 0; i < routes.length(); i++) {
   String vehicleId = null;
   JSONObject route = routes.getJSONObject(i);
   for (DepotModel depotModel: model.getDepots()) {
      if (depotModel.getVehicleType().equals(route.getString("vehicleId"))){
         vehicleId = depotModel.getVehicleType();
         route.put("lng", depotModel.getVehicleStartCoordinateX());
         route.put("lat", depotModel.getVehicleStartCoordinateY());
      }
   }
   JSONArray acts = new JSONArray();
   try{
      acts = route.getJSONArray("act");
   } catch (JSONException e){
      logger.warn("Something wrong happened while getting an array " + e);
      acts.put(route.getJSONObject("act"));
   }
   for (int j = 0; j < acts.length(); j++) {
      JSONObject act = acts.getJSONObject(j);
      int id = act.getInt("serviceId");
      JSONObject nextAct = null;
      int nextId = -1;
      if (j + 1 != acts.length()) {
         nextAct = acts.getJSONObject(j + 1);
         nextId = nextAct.getInt("serviceId");
      }
      for (ServiceModel serviceModel : serviceModelList) {
         if (serviceModel.getServiceId().equals(String.valueOf(id))) {
            if (j == 0) {
               for (DepotModel depotModel: model.getDepots()) {
                  if (depotModel.getVehicleType().equals(vehicleId)) {
                     try {
```

```
                        travelDurationWithTraffic =
getTravelDurationWithTraffic(depotModel.getVehicleStartCoordinateX(), depotModel.getVehicleStartCoordinateY(),
                                serviceModel.getLocationX(), serviceModel.getLocationY());
                        act.put("travelDurationTraffic", travelDurationWithTraffic);
                        act.put("timeUnit", "seconds");
                        act.put("lng", serviceModel.getLocationX());
                        act.put("lat", serviceModel.getLocationY());
                        nextAct.put("travelDurationTraffic", travelDurationWithTraffic);
                        nextAct.put("timeUnit", "seconds");
                        nextAct.put("lng", serviceModel.getLocationX());
                        nextAct.put("lat", serviceModel.getLocationY());
                      } catch (Exception e) {
                        logger.error("Failed to send data to VE: " + e);
                      }
                    }
                  }
                for (ServiceModel nextModel : serviceModelList) {
                  if (nextId != -1 && nextModel.getServiceId().equals(String.valueOf(nextId))) {
                    try {
                      travelDurationWithTraffic = getTravelDurationWithTraffic(serviceModel.getLocationX(),
serviceModel.getLocationY(),
                          nextModel.getLocationX(), nextModel.getLocationY());
                      nextAct.put("travelDurationTraffic", travelDurationWithTraffic);
                      nextAct.put("timeUnit", "seconds");
                      nextAct.put("lng", nextModel.getLocationX());
                      nextAct.put("lat", nextModel.getLocationY());
                    } catch (Exception e) {
                      logger.error("Failed to send data to VE: " + e);
                    }
                  }
                }

              } else continue;
            }
          }
        }
      }
```

```
        } else throw new EmptyResponseException("Nothing to show. None of the services could be assigned to provided
vehicle(s)");
        logger.info("Problem is solved, returning solution...");
        return obj.toString();
    }
}
```