Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Факультет інформаційних технологій
(факультет)

Кафедра системного аналізу та управління
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

Здобувача вищої освіти Линника Влада Юрійовича

академічної групи 124 - 21-1

спеціальності 124 Системний аналіз

за освітньо-професійною програмою Системний аналіз

на тему: «Прийняття рішень в процесах транспортування будівельних матеріалів в умовах підприємства ТОВ "Будін-Торг"»

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**Інституційною**	
кваліфікаційної роботи	*к.т.н., доц. Желдак Т.А.*			
розділів:				
Інформаційно-аналітичний	*к.т.н., доц. Желдак Т.А.*			
Спеціальний розділ	*к.т.н., доц. Желдак Т.А.*			
Рецензент				
Нормоконтролер	*к.ф.-м.н., доц. Хом'як Т.В.*			

Дніпро
2025

ЗАТВЕРДЖЕНО:

завідувач кафедри

Системного аналізу та управління
(повна назва)

_____ *к.т.н., доц. Желдак Т.А.*
(підпис) (прізвище, ініціали)

«____» _____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня *бакалавра*

здобувачу вищоъ освіти *Линнику В. Ю.* **академічної групи** *124 -21-1*
спеціальності: *124 Системний аналіз*
на тему «*Прийняття рішень в процесах транспортування будівельних матеріалів в умовах підприємства ТОВ "Будін-Торг"»,*
затверджену наказом ректора НТУ «Дніпровська політехніка»
від *05.05.2025 р. №336 - с*

Розділ	Зміст	Терміни виконання
1.Інформаційно-аналітичний розділ	*Проаналізувати структуру об'єкта дослідження. Визначити предметну область дослідження та проблему, що розв'язується. Обґрунтувати методи виконання поставлених завдань*	25.04.2025-10.05.2025
2. Спеціальний розділ	*Розв'язати поставлені задачі: розробити алгоритми та створити систему для автоматизації товарних перевезень, враховуючи різні фактори, які необхідні для зручного переміщення людей.*	10.05.2025-01.06.2025

Завдання видано _____ *доц. Желдак Т.А.*
(підпис) (прізвище, ініціали)

Дата видачі: 27.12.2024 р.

Дата подання до екзаменаційної комісії: 20.06.2025 р.

Прийнято до виконання _____ Линник В. Ю__.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 98 с., 45 рис., 11 табл., 3 додатки, 30 джерел.

Об'єктом дослідження в роботі є процес прийняття рішень у сфері транспортування будівельних матеріалів у виробничих умовах підприємства.

Предметом дослідження є логістичні алгоритми, математичні моделі та програмні засоби оптимізації маршрутів доставки будівельних матеріалів.

Метою роботи є підвищення ефективності товарної логістики підприємства шляхом розробки програми.

Методи дослідження: системно-аналітичний підхід, математичне моделювання задач маршрутизації, використання бібліотеки Google OR-Tools для реалізації алгоритмів оптимізації, принципи об'єктно-орієнтованого програмування мовою C# у середовищі WinForms.

В аналітичній частині проаналізовано особливості логістики в будівельній галузі, класифіковано транспортні задачі та проведено огляд існуючих програмних рішень. Обґрунтовано доцільність створення власного інструменту для вирішення задач оптимізації маршрутів.

У проєктній частині побудовано математичну модель маршрутизації, описано архітектуру системи, реалізовано ключові модулі, включаючи формування симуляцій, збереження маршрутів і розрахунок найкоротших шляхів.

Практичне значення одержаних результатів полягає в розробці прикладного програмного модуля для оптимізації маршрутів транспортування будівельних матеріалів, який уперше адаптовано до потреб малого підприємства з урахуванням обмежених ресурсів і реальних виробничих умов.

Ключові слова: ТРАНСПОРТНА ЗАДАЧА, ЛОГІСТИКА, ОПТИМІЗАЦІЯ МАРШРУТІВ, TSP, WINFORMS, C#, GOOGLE OR-TOOLS, СИМУЛЯЦІЯ.

ABSTRACT

Explanatory note: 98 pages, 45 figures, 11 tables, 3 appendices, 30 sources.

The object of the research is the decision-making process in the field of construction material transportation under real production conditions of an enterprise.

The subject of the research is logistics algorithms, mathematical models, and software tools for optimizing delivery routes for construction materials.

The aim of the work is to improve the efficiency of the company's goods logistics by developing a software solution.

Research methods: system-analytical approach, mathematical modeling of routing problems, use of the Google OR-Tools library for implementing optimization algorithms, and principles of object-oriented programming in C# using the WinForms environment.

The analytical section analyzes the specifics of logistics in the construction industry, classifies transportation tasks, and reviews existing software solutions. The rationale for creating a custom tool for solving routing optimization problems is provided.

The design section presents a mathematical model for routing, describes the system architecture, and implements key modules, including simulation generation, route saving, and shortest path calculation.

The practical significance of the results lies in the development of an application software module for optimizing the transportation routes of construction materials, adapted for the first time to the needs of a small enterprise considering limited resources and actual production conditions.

Keywords: TRANSPORTATION PROBLEM, LOGISTICS, ROUTE OPTIMIZATION, TSP, WINFORMS, C#, GOOGLE OR-TOOLS, SIMULATION.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах постійного розвитку інфраструктури та зростання обсягів будівництва питання ефективної організації транспортування будівельних матеріалів набуває особливого значення [1]. Особливо це стосується підприємств, які оперують у регіонах з обмеженою транспортною доступністю, сезонними ускладненнями руху та високими вимогами до строків постачання. Затримки у доставці цементу, щебню, цегли чи інших матеріалів можуть призвести до порушення графіків будівництва, збільшення витрат і ризику фінансових санкцій. Саме тому питання ухвалення обґрунтованих управлінських рішень у логістичних процесах стає критично важливим.

Існуючі системи планування маршрутів у галузі будівництва зазвичай базуються на ручному плануванні або використанні загальних транспортних рішень, що не враховують специфіки матеріалопотоків будівельної сфери. Наприклад, відомі платформи, як-от SAP Transportation Management або Oracle SCM Cloud, хоча й надають широкий спектр функцій, часто є надмірно складними й фінансово недоступними для вітчизняних малих і середніх підприємств [2]. Крім того, вони не завжди підтримують локалізовані карти, нестандартні умови перевезень або нестабільність дорожньої інфраструктури, характерну для багатьох регіонів України.

На тлі війни та перебудови логістичних ланцюгів критично зростає потреба в адаптивних, легких і доступних інструментах для моделювання і прийняття рішень, з урахуванням мінливих умов транспортування. Особливої актуальності набуває використання математичних моделей, що дозволяють мінімізувати витрати на перевезення, скорочувати час доставки та оптимізувати використання транспортних ресурсів. Одним із перспективних підходів у цьому напрямку є впровадження інструментарію оптимізації маршрутів на основі сучасних

бібліотек, таких як Google OR-Tools, що дають змогу вирішувати задачі типу комівояжера з урахуванням реальних обмежень.

У контексті України, де ресурси підприємств часто обмежені, а потреба в цифровій трансформації процесів зростає, розробка прикладних програмних рішень, орієнтованих на специфіку національного ринку, є надзвичайно доцільною. Такий підхід сприяє не лише зростанню ефективності окремих підприємств, а й зміцненню логістичної стійкості будівельної галузі загалом. Таким чином, дослідження і впровадження програмних засобів підтримки прийняття рішень у сфері транспортування будівельних матеріалів є важливим кроком у напрямі розвитку цифрової логістики в Україні.

Метою даної кваліфікаційної роботи є розробка та реалізація програмного модуля підтримки прийняття рішень у процесах транспортування будівельних матеріалів, що забезпечує оптимізацію маршрутів доставки з урахуванням особливостей діяльності підприємства ТОВ «Будін-Торг» та сучасних методів математичного моделювання.

Для досягнення поставленої мети необхідно виконати наступні основні задачі:

– проаналізувати особливості логістичних процесів у будівельній сфері та класифікувати типові задачі транспортування з позицій їх математичної складності й практичної реалізованості;

– дослідити сучасні підходи до оптимізації маршрутів постачання, зокрема методи розв’язання задачі комівояжера, та визначити доцільність використання бібліотеки Google OR-Tools у прикладних умовах підприємства;

– спроєктувати архітектуру інформаційної системи, що включає математичну модель, діаграми бізнес-процесів, схему бази даних, а також функціональні інтерфейси для введення вхідних даних і виведення результатів оптимізації;

– реалізувати програмний модуль для підтримки прийняття рішень, провести його тестування на основі реальних сценаріїв діяльності підприємства

ТОВ «Будін-Торг» та сформулювати рекомендації щодо впровадження результатів роботи у виробничу практику.

Об’єктом дослідження є процес прийняття рішень у сфері транспортування будівельних матеріалів у виробничих умовах підприємства.

Предметом дослідження є логістичні алгоритми, математичні моделі та програмні засоби оптимізації маршрутів доставки будівельних матеріалів.

Методи дослідження: методи системного аналізу – для вивчення логістичних процесів та постановки задачі транспортування; об’єктно-орієнтоване моделювання – для проєктування структури ПЗ та бізнес-процесів; математичне моделювання – для формалізації задачі оптимального маршруту; засоби алгоритмічного аналізу та інструменти Google OR-Tools – для реалізації моделі оптимізації у вигляді програмного модуля.

Наукова новизна одержаних результатів полягає в адаптації та практичному застосуванні алгоритмів оптимізації на основі Google OR-Tools для розв’язання задачі маршрутизації в умовах реального підприємства будівельного профілю з урахуванням специфіки вітчизняної логістичної інфраструктури.

Практичне значення одержаних результатів полягає в розробці прикладного програмного модуля для оптимізації маршрутів транспортування будівельних матеріалів, який уперше адаптовано до потреб малого підприємства з урахуванням обмежених ресурсів і реальних виробничих умов. Запропоноване рішення вдосконалює процес прийняття логістичних рішень шляхом автоматизації планування маршрутів із використанням математичних методів і відкритого програмного забезпечення.

1 ІНФОРМАЦІЙНО-АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Особливості логістичних процесів у будівельній сфері

Успішне функціонування сучасного будівельного підприємства значною мірою залежить від здатності ефективно управляти матеріальними та інформаційними потоками, що супроводжують виробничий цикл. В умовах інтенсивного розвитку будівельної інфраструктури та зростаючих вимог до дотримання строків і якості робіт логістика набуває стратегічного значення [3]. Забезпечення безперервності постачання, оптимальне використання транспортних засобів і ресурсів, зниження витрат на транспортування – усе це перетворює логістичні рішення на ключовий елемент системи управління будівництвом.

Особливістю будівельної галузі є складна структура виробничих процесів, що супроводжується необхідністю своєчасного забезпечення об'єктів великою кількістю вантажів різного призначення – від сипучих матеріалів до збірних конструкцій [4]. Крім того, характерною рисою є географічна розосередженість об'єктів, варіативність умов будівельного майданчика та високий рівень залежності від зовнішніх чинників. Усе це вимагає ретельно скоординованої логістичної діяльності, здатної забезпечити гнучкість і надійність процесів постачання.

На цьому тлі логістичні процеси в будівельній галузі охоплюють сукупність операцій, пов’язаних із плануванням, організацією, контролем і оптимізацією потоків будівельних матеріалів, конструкцій, техніки та допоміжного обладнання. Вони мають чітко виражену фазову структуру, де кожен етап будівництва вимагає своєчасного забезпечення необхідними ресурсами у визначених обсягах, у відповідному місці та з урахуванням часових обмежень.

На рис. 1.1 наведено узагальнену структуру логістичних процесів, що реалізуються у сфері будівництва.

Рисунок 1.1 – Основні логістичні процеси в будівельній галузі

Ключові напрями діяльності логістичних процесів охоплюють як операції постачання та транспортування матеріалів, так і планування, моніторинг, управління змінами та контроль ризиків. Особливу увагу приділено зв’язкам між доставкою, оптимізацією маршрутів і координацією з ресурсами складу, що відображає необхідність інтегрованого підходу до планування [5]. Окрім того, зазначено важливість врахування фаз будівництва, погодних умов та впливу аварійних ситуацій, що потребують постійного адаптивного реагування. Така структуризація дозволяє краще розуміти взаємозалежності між логістичними операціями та формує основу для їх подальшої автоматизації засобами інформаційних технологій.

1.2 Класифікація задач транспортування та методи їх розв’язання

Задачі транспортування є фундаментальним класом прикладних задач в галузі логістики та управління операціями. Їхньою основною метою є

забезпечення ефективного розподілу вантажів або ресурсів між джерелами (пунктами постачання) та споживачами (місцями призначення) при мінімальних витратах на перевезення, часу або іншому цільовому критерії [6]. У контексті будівельної логістики, де йдеться про доставку великогабаритних і важких матеріалів у різні точки будівництва, транспортування набуває особливого значення як критично важлива складова всього проєктного циклу.

На рис. 1.2 узагальнено подано основні категорії транспортних задач, що мають практичне значення для процесів постачання в межах будівництва.

Рисунок 1.2 – Категорії транспортних задач

У загальному випадку задачі транспортування поділяють на кілька категорій залежно від специфіки поставленого завдання та обмежень, що враховуються:

– класична транспортна задача [7]. Передбачає мінімізацію витрат на перевезення вантажу з ряду складів до ряду споживачів за відомими одиничними тарифами перевезення. Ця задача є лінійною і може бути вирішена методами симплекс-оптимізації, методом потенціалів або півплощинним методом. У

будівництві її можна застосовувати, коли потрібно розподілити запаси цементу чи бетону з центрального складу на кілька об'єктів;

– задача комівояжера (Traveling Salesman Problem, TSP). Формулюється як пошук найкоротшого маршруту, за яким водій повинен відвідати певну кількість пунктів лише один раз і повернутися до початкового [7]. У сфері транспортування будівельних матеріалів TSP використовується для планування маршрутів доставки, коли один транспортний засіб обслуговує кілька точок доставки в межах однієї поїздки. Задача є NP-складною і потребує застосування ефективних евристичних або комбінаторних алгоритмів.

– задача розподілу транспортних засобів (Vehicle Routing Problem, VRP) [8]. Розширення TSP, коли кілька транспортних засобів повинні здійснити обслуговування ряду замовників з урахуванням обмежень, як-от місткість, час роботи, вартість експлуатації. Це більш реалістична модель для будівельних підприємств, які мають автопарк і забезпечують доставку матеріалів на декілька майданчиків. Розв’язання VRP зазвичай потребує спеціалізованих бібліотек (наприклад, Google OR-Tools) або метаевристичних підходів (генетичні алгоритми, алгоритм мурашиних колоній, симульоване відпалу);

– динамічні транспортні задачі [9]. Виникають у ситуаціях, коли потреби у перевезенні змінюються з часом – наприклад, у процесі багатоденного будівництва, де зростають або зменшуються обсяги запитів на матеріали. Такі задачі враховують змінність параметрів і потребують адаптивних або імітаційних моделей для моделювання реального ходу подій.

У практиці будівельних підприємств, як показує аналіз, найчастіше постають задачі маршрутизації, які включають додаткові обмеження, пов’язані з часовими вікнами, вагою вантажу, типом матеріалу, станом доріг. Вирішення таких задач вручну є малоефективним, а класичні математичні методи не завжди забезпечують прийнятну продуктивність для задач реального масштабу.

Сучасні підходи до розв’язання зазначених задач базуються на використанні гібридних методів: поєднання лінійного програмування, евристик, стохастичних алгоритмів і бібліотек оптимізації. Одним з найбільш ефективних

інструментів для практичної реалізації моделей такого типу є Google OR-Tools, який надає набір алгоритмів для задач маршрутизації, у тому числі TSP, VRP, задач з часовими вікнами тощо.

Таким чином, вибір відповідного методу розв'язання транспортної задачі залежить від типу поставленої задачі, розміру вхідних даних, технічних обмежень і обраного критерію оптимальності. У випадку доставки будівельних матеріалів доцільно застосовувати адаптовані версії задачі комівояжера або VRP з урахуванням виробничих особливостей та практичних обмежень підприємства. Саме такі задачі мають ключове значення для розробки ефективного програмного модуля підтримки рішень у логістиці будівельної галузі.

1.3 Огляд сучасних програмних інструментів для транспортної оптимізації

У сучасній логістиці транспортування відіграє ключову роль у забезпеченні конкурентоспроможності підприємств, особливо у сферах, де своєчасна доставка матеріалів є критично важливою, як у будівництві. Ефективне управління маршрутами доставки, планування використання транспортного парку, зниження витрат на перевезення та дотримання строків постачання – усе це потребує застосування програмних засобів, що реалізують математичні моделі оптимізації.

Розвиток цифрових технологій упродовж останніх десятиліть призвів до появи широкого спектра інструментів, які дозволяють вирішувати задачі маршрутизації, навантаження транспорту, динамічного планування та моніторингу у режимі реального часу. Такі програмні засоби можуть бути інтегровані в системи управління ресурсами підприємства, функціонувати як автономні логістичні платформи або застосовуватись як програмні бібліотеки для розробки спеціалізованих додатків.

Залежно від масштабу підприємства, бюджету, технічної інфраструктури та специфіки задач, обираються різні інструменти – від комерційних комплексів до

відкритого програмного забезпечення. У подальшому розглянуто найпоширеніші та найперспективніші засоби транспортної оптимізації, які можуть бути застосовані для підтримки прийняття рішень у процесі доставки будівельних матеріалів.

TransCAD – це спеціалізоване геоінформаційне програмне забезпечення, розроблене компанією Caliper Corporation для потреб транспортного моделювання, планування та аналізу логістичних мереж (рис. 1.3). Основне призначення цього застосунку полягає в оптимізації транспортних потоків, побудові маршрутів, аналізі вантажообігу та формуванні сценаріїв розвитку транспортної інфраструктури [10]. Його активно використовують як державні установи, так і приватні логістичні компанії для підтримки стратегічного та тактичного планування.

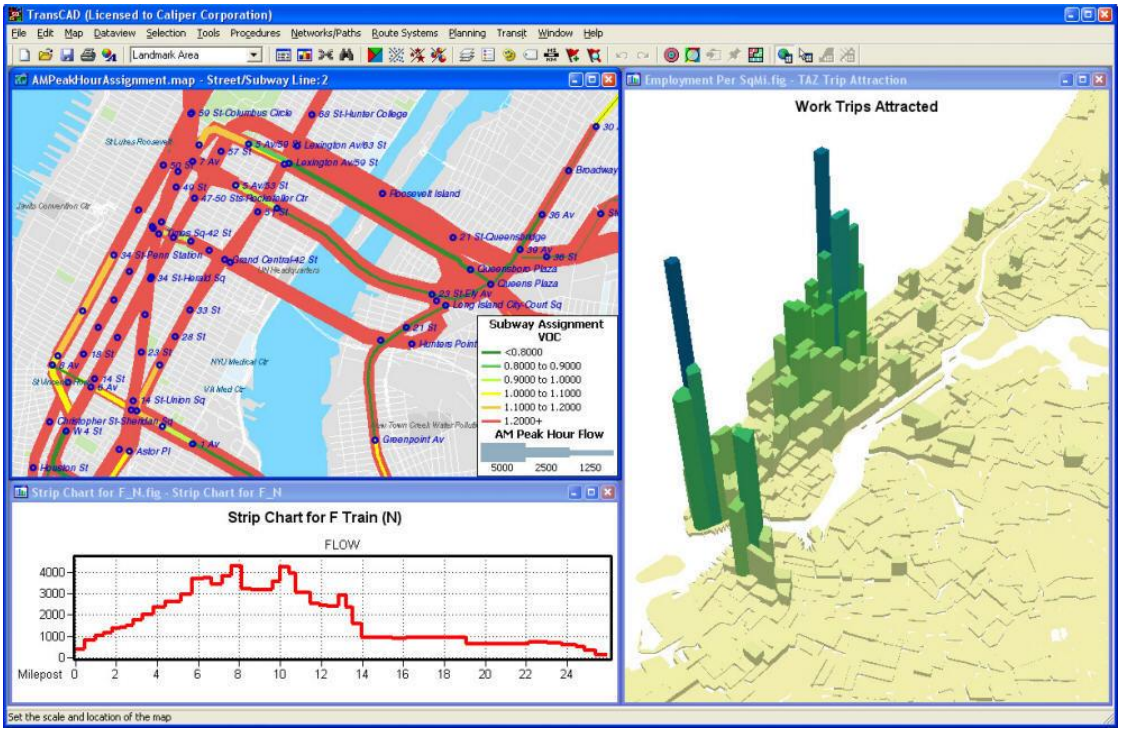


Рисунок 1.3 – Приклад інтерфейсу системи «TransCAD» [11]

TransCAD поєднує функціональність класичної геоінформаційної системи з інструментами транспортної аналітики. Його архітектура побудована на клієнт-серверному підході з використанням потужного картографічного ядра,

інтегрованого з аналітичними модулями. База даних транспортної мережі представлена у вигляді просторових об’єктів із атрибутивними характеристиками, що дозволяє поєднувати просторову візуалізацію з оптимізаційними обчисленнями. Система підтримує імпорт геоданих, таблиць витрат, матриць попиту та дозволяє будувати моделі графів для задач маршрутизації, зонального розподілу вантажів і аналізу транспортної доступності. Завдяки відкритій структурі користувач має змогу налаштовувати моделі під власні задачі, використовувати сценарне планування та формувати звіти з візуалізацією результатів на карті.

Переваги застосунку TransCAD:

– висока точність просторового моделювання завдяки інтеграції ГІС і транспортної аналітики [12];

– гнучка побудова та налаштування транспортних мереж і графів;

– підтримка складних сценаріїв планування і прогнозування вантажо- та пасажиропотоків [13];

– можливість роботи з великими наборами просторових і табличних даних у єдиному середовищі.

Недоліки застосунку TransCAD:

– висока вартість ліцензії [14], що обмежує доступність для малого бізнесу;

– вимогливість до апаратних ресурсів при роботі з великими геоданими;

– складність у навчанні через специфіку термінології та широкі можливості налаштувань [15];

– обмежена підтримка українських картографічних даних «з коробки».

Отже, TransCAD є потужним інструментом для професійного транспортного моделювання та аналітики, який особливо ефективний у стратегічному плануванні великих мереж. Проте через складність освоєння та високу вартість його доцільніше використовувати у великих організаціях або

державних проєктах, тоді як для малого та середнього бізнесу можуть бути ефективнішими адаптивніші та економічніші альтернативи.

Route4Me – це хмарне програмне забезпечення, призначене для автоматичного планування, оптимізації та управління маршрутами доставки в режимі реального часу (рис. 1.4). Основне його завдання полягає у швидкому формуванні найоптимальніших маршрутів для транспортних засобів із урахуванням кількості зупинок, дорожньої ситуації, часових вікон та інших логістичних обмежень [16]. Застосовується переважно у сфері кур'єрських служб, дистрибуції, сервісного обслуговування та логістики малого і середнього бізнесу.

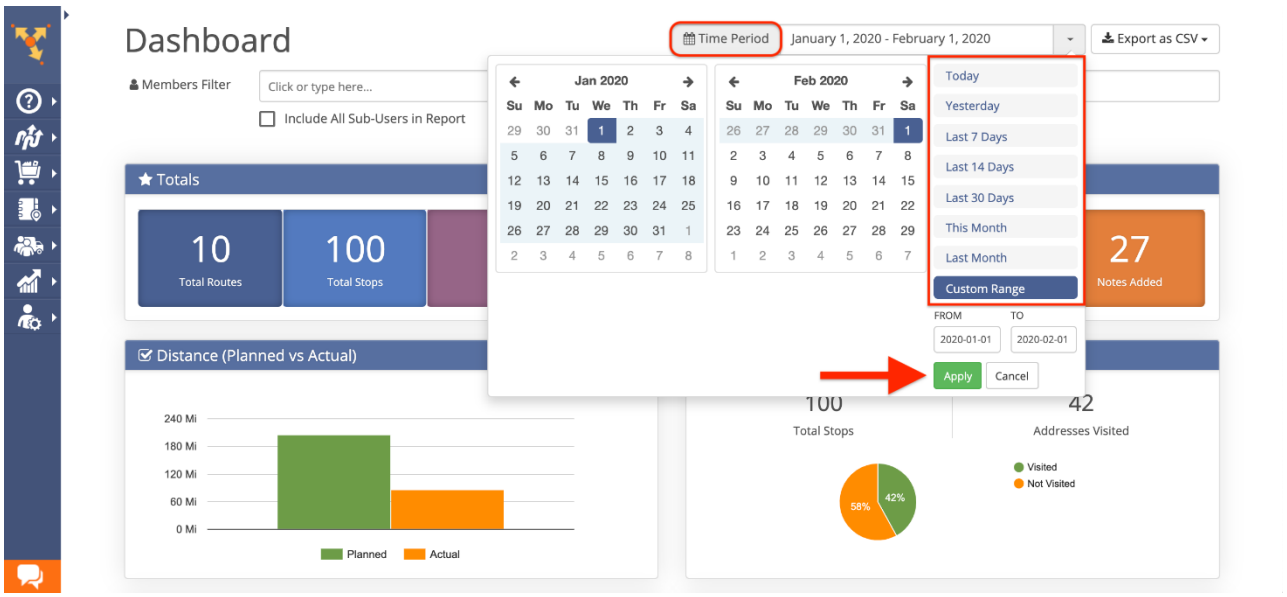


Рисунок 1.4 – Приклад інтерфейсу системи «Route4Me» [17]

Архітектура Route4Me базується на хмарній моделі з використанням архітектури типу SaaS (Software-as-a-Service), що дозволяє користувачам працювати через вебінтерфейс або мобільні додатки. Уся логіка обробки даних, збереження маршрутів, обчислення оптимальних шляхів і аналітики реалізується на стороні сервера. Взаємодія клієнта з системою здійснюється через API або через готові інтерфейси користувача, що забезпечують інтеграцію з GPS-трекерами, обліковими системами та базами замовлень. Така архітектура

дозволяє масштабувати систему без додаткових локальних інсталяцій, а також оперативно оновлювати інформацію про зміни в маршрутах, пробках чи пріоритетах доставки.

Переваги застосунку Route4Me:

– інтуїтивно зрозумілий веб-інтерфейс і мобільні додатки для різних платформ;

– швидка побудова маршрутів із врахуванням великої кількості параметрів (часові вікна, завантаження, пробки тощо) [18];

– хмарна архітектура, що не потребує локального розгортання й забезпечує високу доступність;

– легке підключення до GPS-трекерів та інтеграція з зовнішніми CRM/ERP-системами через API [19].

Недоліки застосунку Route4Me:

– обмежена функціональність у безкоштовній версії – більшість можливостей доступні лише в платних тарифах [20];

– призначення орієнтоване більше на останню милю, ніж на складні багаторівневі логістичні задачі;

– недостатня гнучкість для кастомізації алгоритмів оптимізації під специфіку великого підприємства [21];

– залежність від стабільного інтернет-з’єднання через повну хмарність системи.

Отже, Route4Me є ефективним та зручним рішенням для малого та середнього бізнесу, яке дозволяє швидко впровадити автоматизоване планування маршрутів без потреби у складному налаштуванні. Завдяки простоті використання та гнучкому API застосунок добре підходить для служб доставки та виїзного сервісу, однак для складніших логістичних сценаріїв або великомасштабних підприємств можуть знадобитися більш гнучкі інструменти з підтримкою локальної оптимізації.

OptimoRoute – це сучасний вебзастосунок для планування та оптимізації маршрутів, який орієнтований на бізнеси, що здійснюють щоденну доставку товарів, виїзне обслуговування клієнтів або регулярні логістичні операції (рис. 1.5). Його головною метою є скорочення часу в дорозі, підвищення продуктивності автопарку та зниження витрат на транспортування [22]. Система дозволяє враховувати обмеження щодо кількості замовлень, часових вікон, робочого часу водіїв, пріоритетності завдань і навіть типу транспортного засобу.

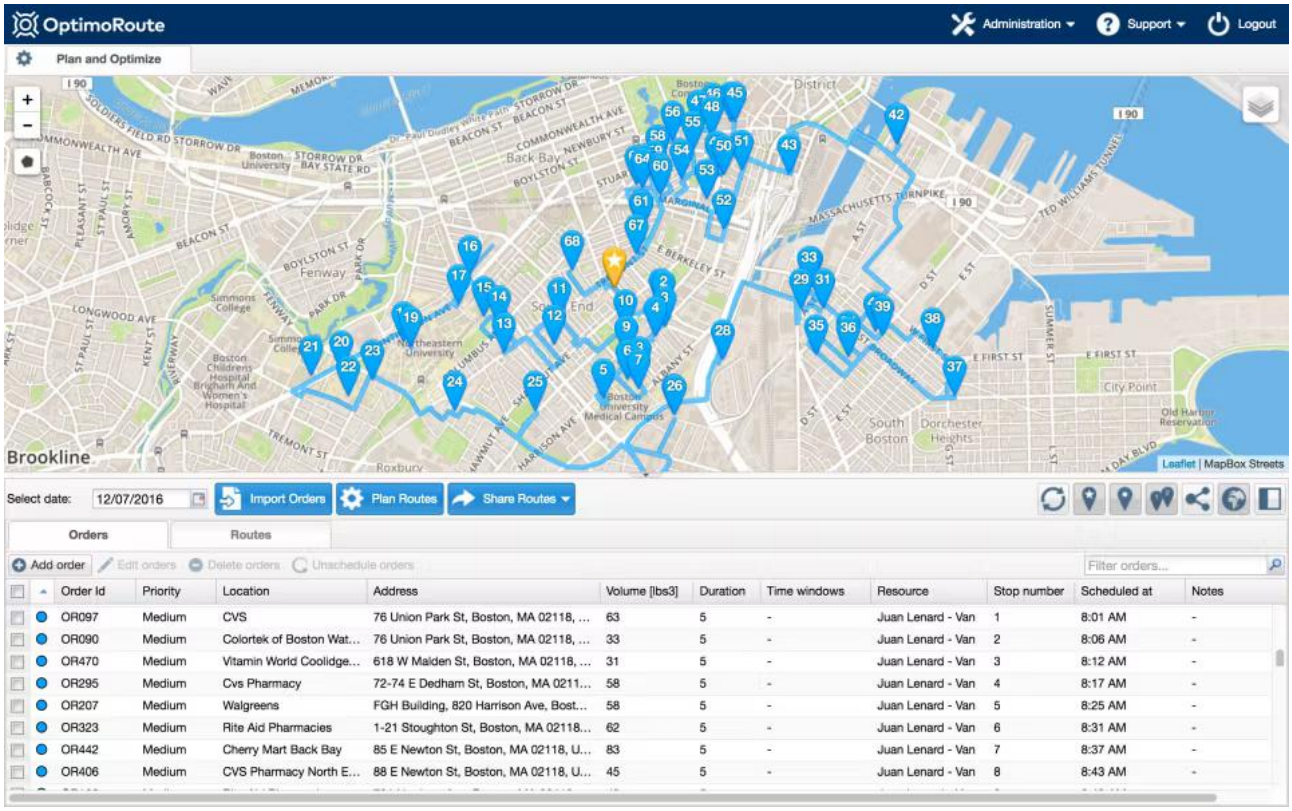


Рисунок 1.5 – Приклад інтерфейсу системи «OptimoRoute» [23]

Архітектура OptimoRoute базується на хмарній платформі з централізованою обробкою маршрутів на віддалених серверах. Користувачі взаємодіють із системою через вебінтерфейс, де завантажують дані у вигляді таблиць або інтегрують систему з власними обліковими або CRM-рішеннями. Обчислювальні алгоритми оптимізації виконуються в хмарі, після чого користувач отримує маршрути, готові до завантаження у мобільні застосунки водіїв або друку. Система підтримує зворотній зв’язок у реальному часі, що

дозволяє відслідковувати виконання маршрутів, перераховувати їх у разі змін і формувати звіти про продуктивність.

Переваги застосунку OptimoRoute:

– потужна оптимізація маршрутів з урахуванням часових вікон, завантаженості, типів транспорту і змін водіїв [24];

– простий і функціональний вебінтерфейс з підтримкою імпорту даних з Excel та інтеграцій через API [25];

– підтримка мобільних застосунків для водіїв з онлайн-оновленням маршрутів і зворотнім зв’язком;

– аналітичні звіти та відстеження виконання завдань у реальному часі.

Недоліки застосунку OptimoRoute:

– висока вартість тарифів при великій кількості користувачів або замовлень [26];

– обмежені можливості кастомізації логіки оптимізації під специфіку нестандартних підприємств [27];

– відсутність офлайн-режиму роботи у разі втрати підключення до Інтернету;

– інтерфейс орієнтований переважно на англомовну аудиторію, що може бути бар’єром для локальних користувачів.

Отже, OptimoRoute – це сучасне, потужне рішення для планування та оптимізації маршрутів, що ідеально підходить для малого та середнього бізнесу в галузі логістики, доставки або сервісного обслуговування. Його головною перевагою є простота впровадження у поєднанні з високою функціональністю. Водночас для підприємств із нестандартними логістичними сценаріями або обмеженим бюджетом можуть знадобитися альтернативні або більш гнучкі системи.

Проведений аналіз показав, що більшість сучасних програмних рішень для транспортної логістики мають складну структуру, орієнтовану на великі підприємства, та передбачають суттєві фінансові витрати на ліцензування.

Наявні системи не завжди враховують специфіку локальних умов, особливо в галузі будівництва, де потрібна адаптивність, простота використання та швидке розгортання без складної технічної інфраструктури. Крім того, багато рішень вимагають тривалого навчання персоналу або мають функціональні надлишки, що не використовуються в межах малого чи середнього підприємства. У зв'язку з цим виникає потреба у створенні легкого, доступного програмного модуля, який поєднує базові можливості оптимізації маршрутів із мінімальними вимогами до впровадження і підтримки, що особливо актуально для підприємств, подібних до ТОВ «Будін-Торг».

1.4 Характеристика діяльності підприємства ТОВ «Будін-Торг»

Підприємство ТОВ «Будін-Торг» є представником середнього бізнесу в будівельній галузі України, основною сферою діяльності якого є постачання, складування та транспортування широкого спектра будівельних матеріалів. Компанія здійснює обслуговування як приватних забудовників, так і великих підрядників, забезпечуючи своєчасну доставку товарів по місту, прилеглих районах та міжобласних напрямках. Її конкурентною перевагою є наявність власного автопарку, що включає вантажні автомобілі різної вантажопідйомності, а також гнучка логістична модель, яка дозволяє оперативно реагувати на зміну попиту, погодних умов, дорожньої ситуації та інших зовнішніх факторів.

Діяльність підприємства охоплює повний цикл обробки замовлень – від прийняття заявки до завершення доставки будівельних матеріалів кінцевому споживачу. При цьому особлива увага приділяється точності дотримання термінів поставок, відповідності асортименту та кількості замовлених матеріалів. Зокрема, компанія активно працює з такими матеріалами, як цемент, щебінь, пісок, цегла, блоки, металеві вироби, арматура, тепло- та гідроізоляційні суміші. Основні поставки здійснюються з центрального складу, розташованого в межах промислової зони, а логістичні маршрути охоплюють будівельні

майданчики в радіусі 150–200 км, що забезпечує оперативне реагування на потреби клієнтів.

З метою забезпечення безперебійної роботи компанія має внутрішньо регламентовану організаційну структуру, яка формує основу для планування, управління та контролю логістичних процесів. Кожен підрозділ підприємства виконує чітко визначені функції, спрямовані на забезпечення ефективності всього логістичного циклу – від прийому товару до його відвантаження та обліку. Узагальнене уявлення про взаємозв'язки структурних одиниць і їхні ролі в системі управління підприємством подано на рис. 1.6.

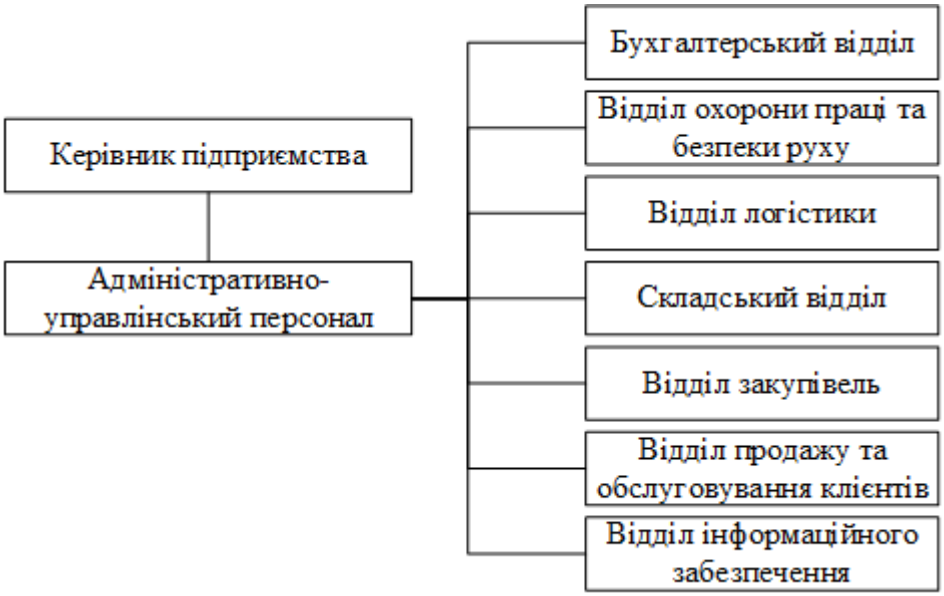


Рисунок 1.6 – Організаційна структура підприємства

Основні структурні підрозділи ТОВ «Будін-Торг» включають:

– керівник підприємства – здійснює загальне стратегічне управління, координує діяльність усіх підрозділів, ухвалює ключові фінансові та операційні рішення;

– бухгалтерський відділ – веде фінансовий облік, формує звітність, контролює оплату поставок, витрати на транспортування та взаєморозрахунки з клієнтами й постачальниками;

– відділ охорони праці та безпеки руху – відповідає за дотримання правил техніки безпеки, організацію інструктажів для водіїв і складського персоналу, а також контроль технічного стану транспорту.

– відділ логістики – відповідає за планування маршрутів доставки, координацію водіїв, розподіл транспортних засобів та моніторинг виконання поставок;

– складський відділ – здійснює приймання, облік, зберігання та відвантаження будівельних матеріалів згідно з накладними та графіками відвантаження;

– відділ закупівель – забезпечує поповнення запасів складу шляхом укладання угод з постачальниками, аналізу залишків та прогнозування потреб;

– відділ продажу та обслуговування клієнтів – опрацьовує замовлення клієнтів, надає консультації щодо асортименту, погоджує умови постачання;

– відділ інформаційного забезпечення – відповідає за обслуговування програмного забезпечення, зберігання логістичних даних, інтеграцію з обліковими системами;

– адміністративно-управлінський персонал – здійснює загальне керівництво підприємством, фінансове планування, аналіз ефективності логістичних рішень та прийняття управлінських рішень.

Така структура дозволяє ефективно координувати взаємодію між усіма етапами логістичного процесу та створює передумови для впровадження автоматизованих систем підтримки прийняття рішень у сфері транспортування.

1.5 Постановка задачі оптимізації

Після проведеного аналізу предметної області постає задача розробки інформаційної системи, яка дозволятиме ефективно приймати рішення щодо маршрутизації доставки будівельних матеріалів до замовників у різних населених пунктах. Основною вимогою є забезпечення побудови оптимального маршруту транспортування з урахуванням заданих обмежень, таких як кількість

пунктів доставки, відстань між містами, загальний час перевезення та можливість фіксації результатів симуляції для подальшого аналізу.

У рамках розробки системи необхідно реалізувати повний функціональний цикл оптимізації логістичних рішень, який включає: вибір підмножини населених пунктів, автоматичне формування відстаней між ними, побудову маршруту, що охоплює всі пункти лише один раз, та повернення до вихідного міста з мінімальними витратами. Задача повинна бути формалізована у вигляді задачі комівояжера (Traveling Salesman Problem), для розв’язання якої планується використання математичного інструментарію бібліотеки Google OR-Tools.

Поряд із основною задачею оптимізації необхідно реалізувати такі підсистеми:

– симуляція маршрутів: створення симуляцій на основі вибраних міст, автоматичне генерування матриці відстаней і часових оцінок, виконання обчислення оптимального маршруту та збереження результатів;

– завантаження симуляцій: можливість доступу до раніше збережених симуляцій для перегляду, редагування або повторного аналізу результатів оптимізації;

– управління довідником міст: формування та редагування переліку населених пунктів, які можуть брати участь у маршрутах доставки;

– облік користувачів і подій системи: впровадження базової системи авторизації з фіксацією дій користувача, а також зберігання інформації про вхід, запуск симуляцій та інші ключові дії в лог-файлі;

– персоналізація: реалізація функціоналу для зміни облікових даних користувачем ,

З технічного боку, у структурі проєкту необхідно реалізувати реляційну базу даних на основі SQLite, що повинна містити такі сутності: Users – для зберігання інформації про користувачів; Logs – для фіксації подій; City – довідник міст; Simulation – опис симуляцій; Distance – деталі відстаней між містами в межах конкретної симуляції.

Поставлена задача передбачає розробку програмного модуля, який дасть змогу здійснювати оптимізацію маршрутів постачання будівельних матеріалів шляхом інтеграції математичних методів, структурованих даних та інтерфейсу взаємодії з користувачем. Реалізація системи має забезпечити зниження логістичних витрат підприємства та підвищення оперативності ухвалення управлінських рішень.

1.6 Висновки до розділу 1

У рамках даного розділу було здійснено комплексний аналіз логістичних процесів у будівельній сфері з акцентом на специфіку постачання важких і об’ємних матеріалів у динамічних умовах виробничих майданчиків. Визначено ключові етапи логістики, серед яких особливу увагу приділено процесам планування маршрутів, координації поставок та управління ризиками. Проведено класифікацію транспортних задач, зокрема акцентовано на задачі комівояжера як найбільш релевантній до умов перевезення міжбудівельних об’єктів.

Проаналізовано сучасні програмні засоби транспортної оптимізації, серед яких детально розглянуто можливості TransCAD, Route4Me та OptimoRoute. Виділено їхні переваги (наприклад, гнучка маршрутизація, підтримка API, аналітика) та виявлено суттєві недоліки (висока вартість, надлишковий функціонал, складність кастомізації), що обґрунтувало доцільність створення нового прикладного рішення. Обґрунтовано потребу у розробці легкого, адаптивного та доступного програмного модуля, що відповідає реальним потребам підприємства середнього рівня в українських умовах.

Розглянуто особливості діяльності ТОВ «Будін-Торг», яке виступає прикладом цільового підприємства для застосування майбутньої системи. Детально охарактеризовано організаційну структуру підприємства, що дозволяє визначити логічне розмежування функцій між підрозділами та передбачити інтеграцію системи з внутрішніми бізнес-процесами.

На завершення розділу сформульовано чітку постановку задачі оптимізації маршрутів доставки, що передбачає реалізацію функціоналу моделювання, побудови матриць відстаней, збереження та завантаження симуляцій, а також підтримку обліку користувачів і подій. Отримані результати дозволяють перейти до наступного етапу – спеціального розділу, де буде безпосередньо реалізовано математичну модель, розроблено програмний модуль та проведено його тестування в умовах діяльності ТОВ «Будін-Торг».

2 СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Математична модель транспортної задачі та її особливості

Задача комівояжера в логістичних системах постає як класична задача на графі, де вершини відповідають населеним пунктам (або складам, об'єктам будівництва тощо), а ребра – відстаням чи часовим витратам на транспортування між ними. Вона широко застосовується в задачах розвезення вантажів, планування маршрутів та диспетчеризації, особливо коли мова йде про оптимізацію транспортних витрат у будівельному секторі.

Розглянемо орієнтований граф:

$$G = (V, E)\ , \tag{2.1}$$

де:

$\mathrm{V} = \{1,2, \dots, \mathrm{n}\}$ – множина вершин, що представляє населені пункти;

$\mathrm{E} \subseteq \mathrm{V} \times \mathrm{V}$ – множина дуг між пунктами;

$d_{ij} \in R_+$ – вага дуги (i,j), що визначає витрати на переміщення з пункту i до пункту j (як правило, відстань або час перевезення).

Задача полягає у знаходженні гамільтонового циклу мінімальної вартості:

$$min \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} d_{ij} \cdot x_{ij}\ , \tag{2.2}$$

де:

$x_{ij} \in \{0,1\}$ – бінарна змінна, що дорівнює 1, якщо обрано маршрут з i у j, і 0 – в іншому випадку.

Модель підлягає таким обмеженням:

1. Забезпечення виходу з кожного міста:

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1\ , \forall i \in \{1,2, \dots, n\}. \tag{2.3}$$

2. Забезпечення входу в кожне місто:

$$\sum_{i=1, j \neq i}^{n} x_{ij} = 1\ , \forall j \in \{1,2, \dots, n\}. \tag{2.4}$$

3. Уникнення підциклів (умова зв’язності):

Щоб виключити наявність ізольованих підциклів, які задовольняють попередні обмеження, але не утворюють єдиного циклу, запроваджується умова зв’язності. Одним із найбільш поширених підходів є метод Міллера–Такера–Земліна (MTZ), який вводить допоміжні змінні u_i для всіх i ∈ {2, … , n}, такі що:

$$u_i - u_j + n \cdot x_{ij} \leq n - 1, \forall i = j, i, j \in \{2, \dots, n\}, \quad (2.5)$$

де:

u_i – порядковий номер відвідування вершини i у маршруті (для забезпечення ациклічності часткових підтурів).

Зазначене обмеження у формі MTZ дозволяє зберегти зв’язність рішення та уникнути появи кількох окремих циклів, які могли б задовольнити умови входу й виходу з міст, але не формували б єдиний замкнений маршрут. У межах подальшого формалізованого аналізу доцільно також врахувати структурні відмінності між двома варіантами задачі комівояжера – симетричним та несиметричним – адже характер дорожньої інфраструктури та специфіка перевезень у будівельній сфері можуть істотно впливати на вибір математичної моделі.

Залежно від того, чи є витрати на транспортування між парами пунктів взаємно рівними, задачі поділяють на симетричні, що мають більш просту структуру, та асиметричні, які є складнішими в обчислювальному плані, але водночас більш адекватно відображають умови реального транспортування матеріалів.

У загальному вигляді задача комівояжера може бути подана у двох формулюваннях залежно від властивостей матриці відстаней:

1. Симетричний TSP

Цей випадок передбачає, що відстань з міста ii до міста j дорівнює відстані з міста j до міста i, тобто:

$$d_{ij} = d_{ji}, \forall i, j \in \{1, \dots, n\}, i = j. \quad (2.6)$$

Цей варіант найбільш характерний для логістичних сценаріїв, де використовуються прямі магістралі з однаковими витратами на транспортування в обох напрямках. У будівельній логістиці симетричний TSP може бути наближенням, якщо маршрути проходять переважно по дорогах з однорідними умовами та без вагових обмежень для зворотного ходу.

2. Несиметричний TSP

У випадку, коли:

$$d_{ij} \neq d_{ji}, \text{для деяких пар } i, j, \tag{2.7}$$

маємо справу з асиметричним формулюванням. Таке трапляється, наприклад, коли:

– наявні односторонні дороги;

– відстань у зворотному напрямку включає додаткові витрати (об'їзди, переправи, платні ділянки);

– час доставки враховує трафік, який змінюється за напрямком руху.

У реальних логістичних сценаріях у будівельній галузі часто трапляється саме асиметричний варіант, особливо при перевезенні сипучих матеріалів (бетон, пісок), які вимагають врахування зворотного порожнього ходу.

У класичній постановці TSP критерій оптимальності – це мінімізація сумарної довжини маршруту. Однак в задачах транспортування будівельних матеріалів більш коректним може бути використання узагальненої функції витрат, яка враховує:

– відстань d_{ij} між пунктами;

– витрати пального f_{ij};

– витрати часуt_{ij};

– вагу вантажу w_{ij}, що впливає на зношення техніки.

Таким чином, загальна вартість перевезення з i в j можна подати у вигляді функції:

$$c_{ij} = a \cdot d_{ij} + \beta \cdot t_{ij} + \gamma \cdot f_{ij} + \delta \cdot w_{ij}, \tag{2.8}$$

де:

α,β,γ,δ – вагові коефіцієнти, що визначають пріоритети критеріїв відповідно до бізнес-умов (наприклад, мінімізація часу чи пального).

Ці значення можуть задаватися експертно або отримуватись в результаті аналізу статистики витрат на перевезення на підприємстві.

Після цього вектор c_{ij} замінює d_{ij} у цільовій функції, тобто маємо:

$$min \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} \cdot x_{ij} , \quad (2.9)$$

де c_{ij} – узагальнені витрати на перевезення, що враховують комбінацію просторових, часових і експлуатаційних чинників.

Використання такого підходу забезпечує більш реалістичне моделювання вартості маршруту в умовах транспортної логістики будівельної галузі, де значення можуть залежати не лише від геометричної відстані, а й від таких чинників, як вантажопідйомність техніки, тип дорожнього покриття, рівень завантаженості, погодні умови тощо. Це дозволяє підвищити точність оптимізаційної моделі та забезпечити прийняття рішень, які краще відповідають виробничим і економічним обмеженням підприємства.

Для подальшого теоретичного аналізу зручно представити задачу комівояжера в матричній формі, що дозволяє спростити її комп'ютерну реалізацію та забезпечити можливість використання класичних методів лінійного програмування, комбінаторної оптимізації або евристичного пошуку.

Нехай:

$C = [c_{ij}] \in R^{n\times n}$ – матриця витрат (або узагальненої вартості транспортування);

$X = [x_{ij}] \in \{0,1\}^{n\times n}$ – матриця рішень, що кодує послідовність переходів між містами.

Тоді задачу (2.9) можна записати у вигляді:

$$\min(C, X) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij} \text{ за умови: } x_{ii} = 0, \forall i, \quad (2.10)$$

а обмеження на єдиний вхід та вихід з кожного міста відповідно приймають вигляд:

$$\sum_{j=1}^{n} x_{ij} = 1\,, \forall i \in \{1, \dots, n\}; \tag{2.11}$$

$$\sum_{i=1}^{n} x_{ij} = 1\,, \forall j \in \{1, \dots, n\}; \tag{2.12}$$

Застосування матричного запису спрощує реалізацію моделі у середовищах математичного програмування (наприклад, MATLAB, Python з бібліотекою OR-Tools або в спеціалізованих логістичних модулях), а також забезпечує ефективну інтеграцію з методами обробки великих обсягів вхідних даних, що є типовим у реальних логістичних системах.

Однак, попри наочність і формальну чіткість математичної моделі, задача комівояжера належить до класу NP-складних задач (від англ. nondeterministic polynomial time complete). Це означає, що для неї наразі не відомо жодного алгоритму, який би дозволяв знаходити точний розв’язок за поліноміальний час для довільного значення n. Час виконання будь-якого точного алгоритму (наприклад, повного перебору або методів цілочисельного програмування) зростає експоненційно з кількістю міст. Зокрема, кількість можливих маршрутів при n містах становить $(n-1)!/2$ у симетричному випадку та $(n-1)!$ в асиметричному.

Такий експоненційний характер обчислювальної складності означає, що для задач реального масштабу – наприклад, побудови маршрутів доставки між десятками або сотнями об’єктів – застосування точних методів є на практиці малоефективним або навіть непридатним без суттєвих обмежень на розмірність задачі.

У зв’язку з цим у прикладних задачах, зокрема в логістиці будівельних матеріалів, активно використовуються евристичні та метаевристичні методи, які дозволяють знайти близькі до оптимальних рішення за прийнятний обчислювальний час. До таких методів належать:

– жадібні алгоритми (Greedy Algorithms) та модифіковані варіанти пошуку найближчого сусіда;

– алгоритм 2-opt і його розширення (k-opt) для локального покращення маршруту;

– генетичні алгоритми, які імітують природний добір у просторі можливих рішень;

– метод імітованого відпалу (Simulated Annealing), що дозволяє уникати локальних мінімумів;

– пошук із використанням заборонених переходів (Tabu Search), який враховує історію нещодавніх рішень для уникнення зациклення та локальних мінімумів;

– алгоритми на основі колоній мурах (Ant Colony Optimization) та інші популяційні моделі.

Особливо ефективними є комбіновані підходи, що поєднують формалізовану постановку задачі з використанням обчислювальних бібліотек (наприклад, Google OR-Tools) та інтелектуальних евристик, адаптованих до особливостей конкретної предметної області – у даному випадку до будівельної логістики.

2.2 Основи роботи з бібліотекою Google OR-Tools

Для реалізації задачі маршрутизації в межах дослідження було обрано бібліотеку Google OR-Tools – універсальний інструмент оптимізації, розроблений для ефективного розв’язання задач дискретної комбінаційної природи, зокрема задачі комівояжера та її узагальнень (VRP, CVRP тощо). Ця бібліотека забезпечує інтуїтивно зрозуміле API, високу продуктивність та вбудовану підтримку обмежень і параметрів пошуку, що є надзвичайно важливими для практичної реалізації логістичних алгоритмів у реальному середовищі.

До основних компонентів OR-Tools, що використовуються для побудови маршруту, належать:

– RoutingIndexManager – забезпечує трансляцію між логічними індексами вузлів (міст) та внутрішніми індексами моделі;

– RoutingModel – ядро оптимізаційної моделі, яке формує граф перевезень, накладає обмеження та виконує розрахунок шляху;

– Cost Function Provider – визначає цільову функцію (наприклад, мінімізацію відстані);

– Constraint Handler – керує додатковими обмеженнями, якщо вони присутні (наприклад, місткість, час, часові вікна);

– Solver – запускає обчислювальний процес за обраною стратегією пошуку, наприклад, Path Cheapest Arc або Guided Local Search.

На рис. 2.1 наведено узагальнену схему взаємодії основних модулів розробленої системи оптимізації маршрутів на основі бібліотеки OR-Tools.

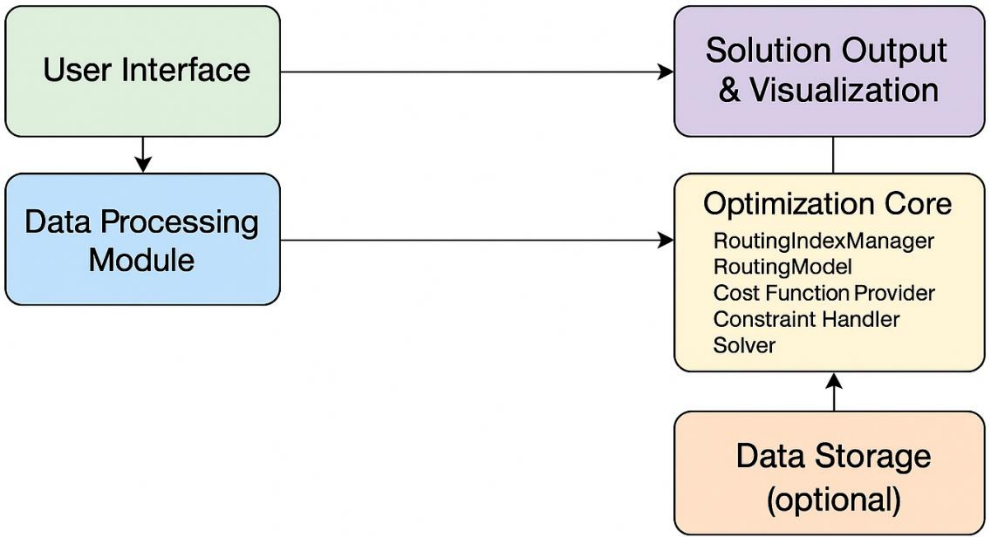


Рисунок 2.1 – Схема взаємодії модулів системи

У цій структурі ключову роль відіграє інтерфейс користувача, через який вводяться вихідні дані, зокрема перелік міст, параметри симуляції та додаткові налаштування маршруту. Ці дані передаються у модуль обробки даних, де формуються матриці відстаней і часових оцінок. Далі інформація надходить до ядра оптимізації, побудованого на компонентах OR-Tools, яке виконує основну обчислювальну роботу – побудову моделі, оцінку варіантів та вибір найкращого маршруту. За потреби, ядро звертається до модуля зберігання даних, де можуть

бути збережені або завантажені попередні симуляції. Після завершення розрахунків результати передаються у модуль виводу та візуалізації рішення, де відображається оптимальний маршрут, перелік послідовних міст і значення загальної довжини шляху. Така архітектура дозволяє досягнути гнучкої та модульної реалізації системи, у якій кожен компонент виконує конкретну функцію, а бібліотека OR-Tools забезпечує ефективне ядро обчислювальної логіки.

Побудована архітектура дозволяє логічно структурувати процес взаємодії між користувачем, обробкою вхідних даних та модулем оптимізації, забезпечуючи гнучкість, масштабованість і зручність подальшої інтеграції з базою даних. Проте, щоб зрозуміти порядок дій під час безпосередньої роботи з бібліотекою Google OR-Tools, доцільно перейти до покрокового алгоритму її використання, що відображає послідовність викликів основних методів і етапів моделювання. На рис. 2.2 наведено схему, яка відображає етапи реалізації алгоритму оптимізації маршруту з використанням інструментів OR-Tools.

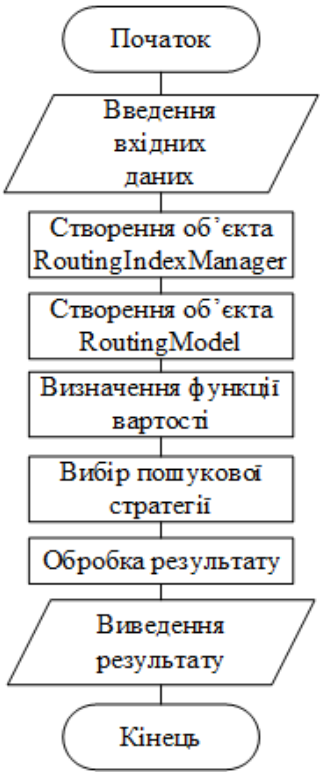


Рисунок 2.2 – Алгоритм роботи з бібліотекою Google OR-Tools

Алгоритм починається з введення вхідних даних, що включають список міст, а також матрицю відстаней між ними. Далі створюється об'єкт RoutingIndexManager, який забезпечує відповідність між логічними індексами міст та внутрішніми індексами маршрутизатора. Наступним кроком є ініціалізація об'єкта RoutingModel, який слугує основою для формування моделі перевезень та маршруту.

Після цього виконується визначення функції вартості, тобто обчислення відстані між кожною парою точок для цільової функції. Наступним етапом є вибір стратегії пошуку рішення – наприклад, Path Cheapest Arc, яка забезпечує базову ефективність при формуванні маршруту. Далі система обробляє результат, розшифровуючи послідовність індексів у конкретні назви міст, розраховує загальну відстань та підготовлює текстовий або візуальний звіт. На завершення відбувається виведення результату для користувача в інтерфейсі.

2.3 Проєктування діаграм основних бізнес процесів

Ефективне проєктування програмного забезпечення для автоматизації логістичних рішень вимагає чіткого формалізованого опису бізнес-процесів, що відображають логіку роботи підприємства. Такий опис дозволяє виявити ключові учасники взаємодії, визначити їхні функції, упорядкувати інформаційні потоки та встановити залежності між подіями, що відбуваються в системі. У контексті транспортування будівельних матеріалів особливу увагу слід приділити етапам формування маршруту доставки, збереження результатів симуляції та взаємодії користувача з функціональними модулями програмного середовища.

Графічний опис логіки роботи системи здійснено за допомогою діаграм послідовностей (sequence diagrams), що належать до нотації UML і дають змогу моделювати обмін повідомленнями між об’єктами під час виконання визначених сценаріїв [28]. Такий підхід забезпечує наочне представлення взаємодії між користувачем, модулями симуляції, обробки відстаней, ядром оптимізації та сховищем даних. Це не лише сприяє ефективному проєктуванню архітектури

програмного забезпечення, а й значно полегшує перевірку коректності реалізованої бізнес-логіки на етапах програмування та тестування. Одним із ключових сценаріїв, що моделювались у процесі проєктування, є автентифікація користувача – обов’язкова процедура перед доступом до основних функцій, таких як створення симуляцій чи побудова оптимізованих маршрутів. Відповідна діаграма послідовності відображає повний цикл цього процесу: від відкриття форми входу до підтвердження дійсності облікових даних і запуску головного вікна системи.

На рис. 2.3 наведено діаграму послідовності, яка демонструє обмін повідомленнями між користувачем та модулями системи під час виконання сценарію входу.

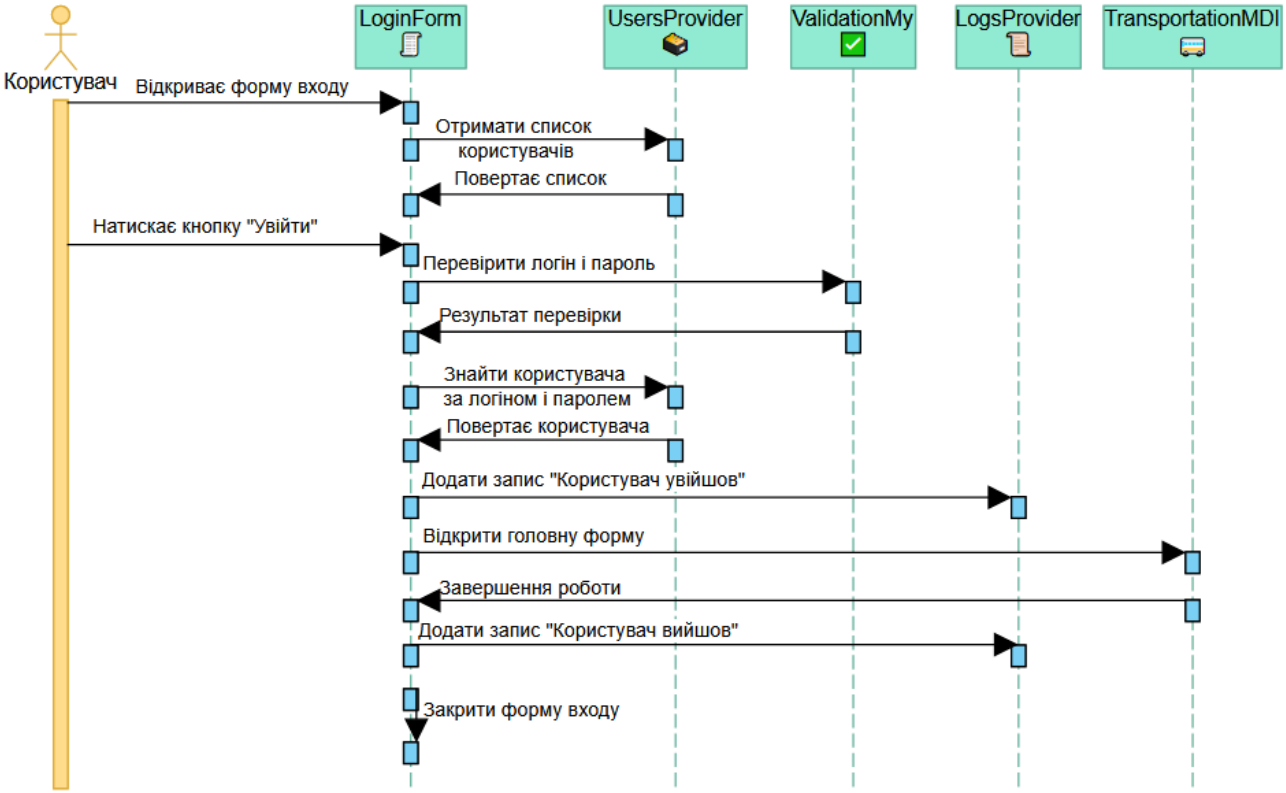


Рисунок 2.3 – Діаграма послідовності процесу авторизації

Процес розпочинається з ініціалізації форми входу, після чого користувач натискає кнопку «Увійти». Система звертається до провайдера користувачів для отримання списку облікових записів, далі відбувається перевірка правильності

логіна та пароля за допомогою модуля валідації. У разі успішного входу здійснюється пошук конкретного користувача за заданими обліковими даними та формування запису в журналі дій із повідомленням «Користувач увійшов». Після цього відкривається головна форма системи – TransportationMDI. По завершенні сеансу система додає ще один запис у журнал із повідомленням про вихід користувача, після чого форма входу закривається.

На рисунку 2.4 наведено діаграму послідовності процесу додавання населеного пункту до системи.

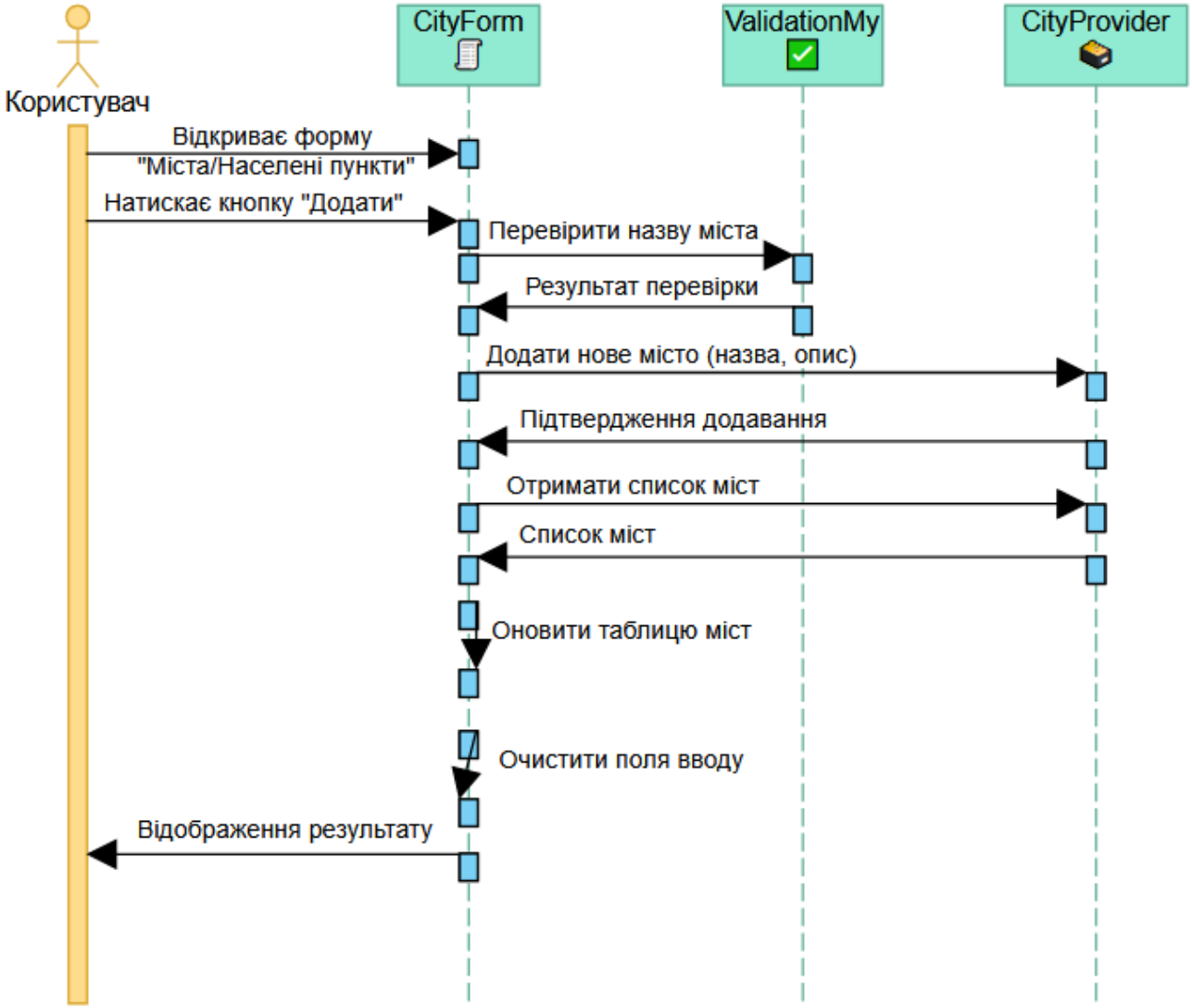


Рисунок 2.4 – Діаграма послідовності процесу додавання населеного пункту

Процес починається з відкриття користувачем форми "Міста/Населені пункти" та натискання кнопки «Додати». Система ініціює перевірку

правильності введеної назви міста за допомогою модуля валідації. Якщо результат перевірки успішний, формується запит на додавання нового запису до бази даних через модуль CityProvider, який обробляє введену назву й опис міста. Після отримання підтвердження про успішне збереження, система оновлює список міст і таблицю відображення, отримавши актуальні дані з бази. Завершальним етапом є очищення полів введення у формі та виведення результату для користувача у вигляді оновленого інтерфейсу.

Рис. 2.5 відоюражає діаграму послідовності процесу проведення симуляції маршруту доставки.

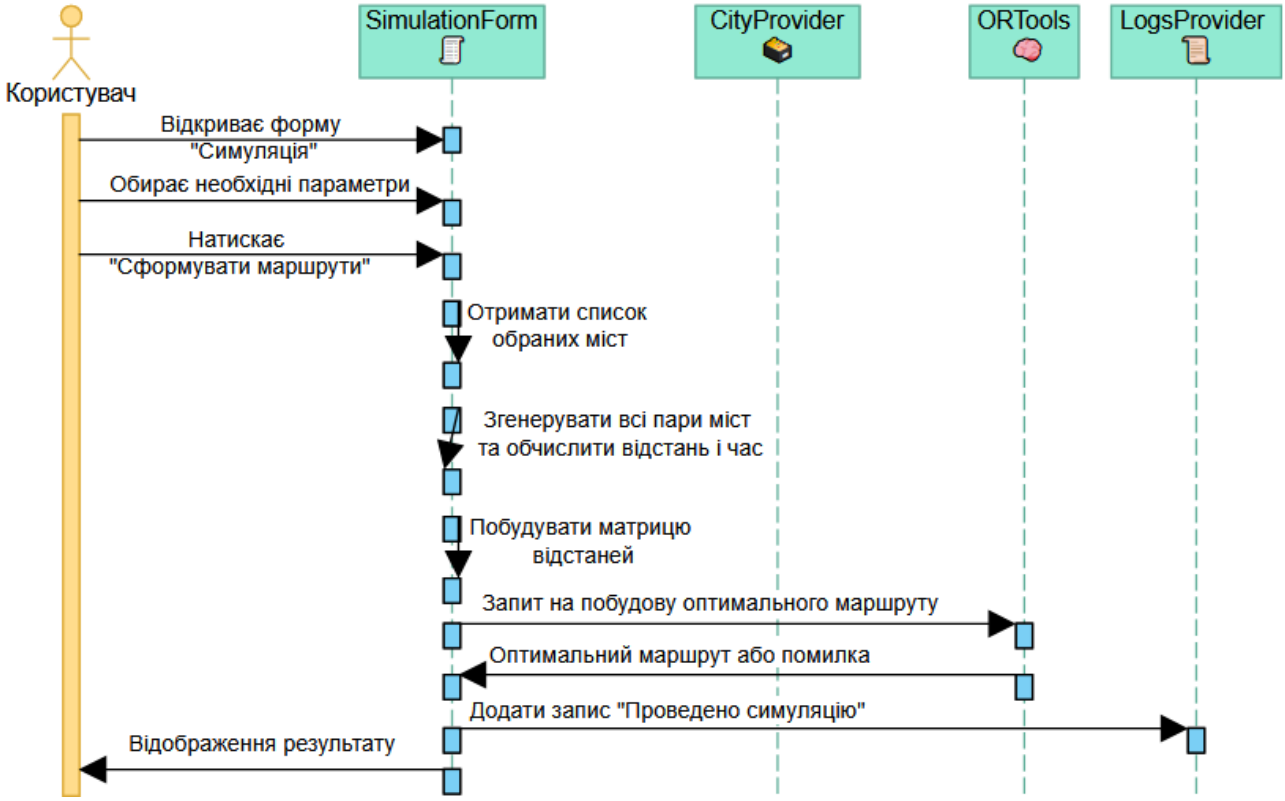


Рисунок 2.5 – Діаграма послідовності процесу проведення симуляції

Процес розпочинається з відкриття користувачем відповідної форми «Симуляція» та вибору набору необхідних параметрів, включаючи список міст. Після підтвердження дії (натискання кнопки «Сформувати маршрути»), система звертається до довідника для отримання списку обраних населених пунктів. Далі ініціюється генерація усіх можливих пар міст, між якими обчислюється відстань

і приблизний час подолання. На основі цих даних формується матриця відстаней, що передається до модуля оптимізації.

Модуль OR-Tools обробляє вхідні дані, будуючи оптимальний маршрут або повертає повідомлення про неможливість його побудови (наприклад, у разі недостатньої кількості точок). Результат обробки – у вигляді послідовності міст та загальної відстані – передається назад у форму симуляції, де відображається у зручному форматі для користувача. Паралельно в системний журнал вноситься запис «Проведено симуляцію», що забезпечує аудит дій користувача.

На рис. 2.6 зображено діаграму послідовності процесу завантаження симуляції з попередньо збереженого списку.

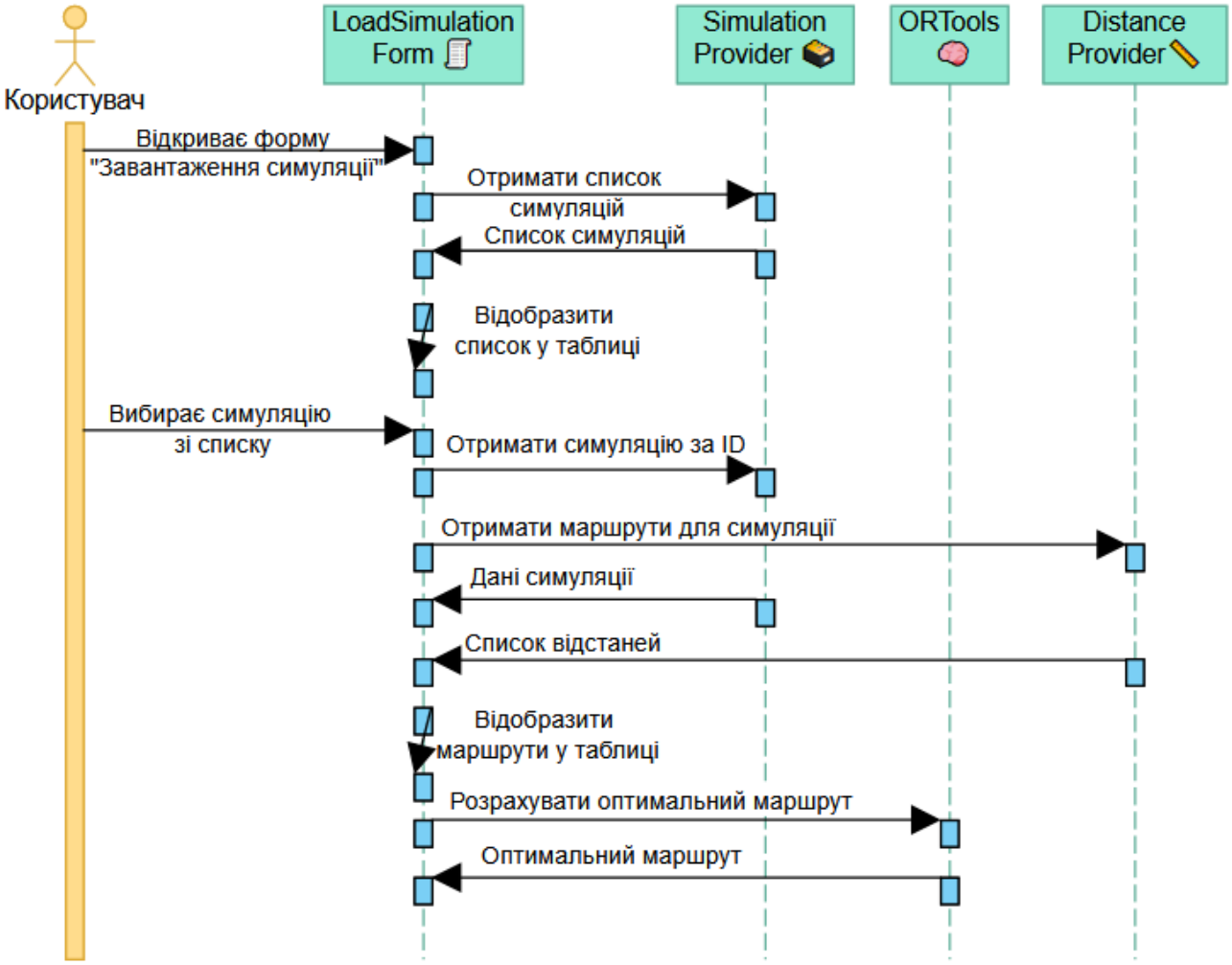


Рисунок 2.6 – Діаграма послідовності процесу завантаження симуляції

Користувач відкриває форму «Завантаження симуляції», після чого система звертається до провайдера симуляцій із запитом на отримання повного списку наявних симуляцій, який повертається та відображається у вигляді таблиці. Після вибору потрібного запису користувачем система надсилає запит на отримання симуляції за ідентифікатором, а також витягує відповідні маршрути із бази даних – тобто перелік міст і відповідні міжміські відстані. Ці дані передаються у таблицю, де відображаються для візуального аналізу.

Завершальним етапом є повторний запуск оптимізації маршруту на основі завантажених відстаней, для чого формується запит до модуля OR-Tools. Розрахований оптимальний маршрут передається назад у форму, де виводиться для користувача. Таким чином, система не лише зберігає попередні результати, а й дозволяє швидко відновлювати контекст моделювання для повторного аналізу або вдосконалення логістичного рішення.

2.4 Проєктування архітектури системи

Створення ефективного програмного забезпечення для підтримки прийняття рішень у логістиці вимагає не лише реалізації окремих алгоритмів оптимізації, а й ретельного проєктування архітектури системи [29]. Архітектура визначає логічну організацію компонентів, способи їх взаємодії та забезпечує узгоджену роботу всіх модулів у процесі обробки даних, обчислень і візуалізації результатів. У контексті задачі побудови оптимальних маршрутів для транспортування будівельних матеріалів особливої важливості набуває підтримка масштабованості, простоти підтримки коду та гнучкого розширення функціоналу в майбутньому.

З огляду на ці вимоги, для реалізації системи було обґрунтовано вибір трьохрівневої архітектури (three-tier architecture), яка чітко розділяє логіку взаємодії на три незалежні рівні: рівень представлення (інтерфейс користувача), рівень логіки застосунку (обробка даних, алгоритми оптимізації) та рівень доступу до даних (взаємодія з базою даних).

Перевагою такого підходу є те, що він забезпечує структурованість та модульність, завдяки чому можна незалежно змінювати інтерфейс або бізнес-логіку без впливу на інші компоненти. Це особливо важливо для системи, що виконує симуляції, зберігає попередні сценарії, працює з математичними моделями та потребує постійної взаємодії з базою даних. Рівень логіки додатку, реалізований окремими провайдерами, відповідає за обробку запитів, валідацію та виклик алгоритмів оптимізації. Рівень зберігання (на базі SQLite) забезпечує просте, автономне й кросплатформне зберігання всіх довідників і результатів симуляцій.

Для реалізації доступу до даних у межах трьохрівневої архітектури системи розроблено окремий рівень обробки запитів до бази даних – рівень провайдерів. Він ізольований від користувацького інтерфейсу та логіки оптимізації, забезпечуючи централізовану, узгоджену та безпечну роботу з даними. На рис. 2.7 зображено діаграму класів рівня даних, яка описує структуру основних класів-доступу до даних, їх поля та методи.

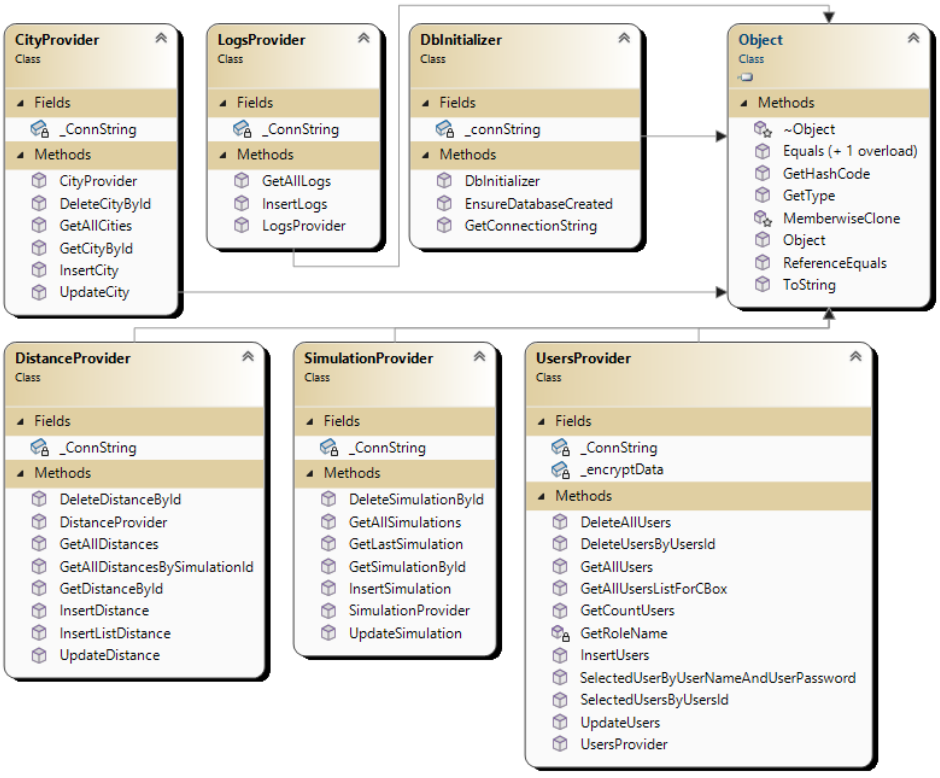


Рисунок 2.7 – Діаграма класів рівня даних

Діаграма класів рівня даних складається з:

– класу CityProvider, який відповідає за доступ до інформації про населені пункти. Він реалізує методи для отримання повного списку міст, пошуку за ідентифікатором, додавання нового міста та оновлення вже наявного запису;

– класу LogsProvider, який забезпечує фіксацію подій у системі. Містить методи для зчитування журналу подій та додавання нових записів про дії користувача;

– класу DbInitializer, який відповідає за ініціалізацію структури бази даних. Забезпечує створення таблиць при першому запуску та повертає рядок підключення до сховища SQLite;

– класу DistanceProvider, який реалізує доступ до інформації про міжміські відстані в контексті симуляцій. Він дозволяє додавати нові відстані, оновлювати існуючі, отримувати всі зв’язки або лише ті, що пов’язані з конкретною симуляцією, а також видаляти непотрібні дані;

– класу SimulationProvider, який працює з інформацією про симуляції маршрутів. Він забезпечує зберігання, оновлення, видалення симуляцій, пошук останньої симуляції, а також витяг усіх наявних у системі симуляцій для подальшого аналізу;

– класу UsersProvider, який керує інформацією про користувачів системи. Містить засоби для вставки, оновлення, видалення користувачів, перевірки облікових даних, отримання кількості зареєстрованих користувачів та їхнього списку для подальшої автентифікації або персоналізації доступу.

Усі класи мають спільне поле _ConnString для роботи з базою, що забезпечує уніфікований механізм підключення. Клас UsersProvider також містить додаткове поле _encryptData для обробки паролів, що підкреслює важливість безпеки на рівні доступу до даних. Уся ця структура формує надійний і масштабований фундамент для функціонування системи підтримки прийняття рішень у транспортній логістиці.

Для реалізації функціональності системи на рівні взаємодії з кінцевим користувачем розроблено набір спеціалізованих форм, кожна з яких відповідає за

окремий бізнес-процес: авторизацію, додавання та редагування даних, запуск симуляцій, персоналізацію інтерфейсу, роботу з логами тощо. Завдяки чіткій структуризації інтерфейс забезпечує інтуїтивне керування процесами транспортування будівельних матеріалів, підтримує простоту навігації та гнучкість використання функцій. На рис. 2.8 представлено діаграму класів рівня користувацького інтерфейсу системи, яка відображає основні форми, їх методи та спадкування від базового класу Form.

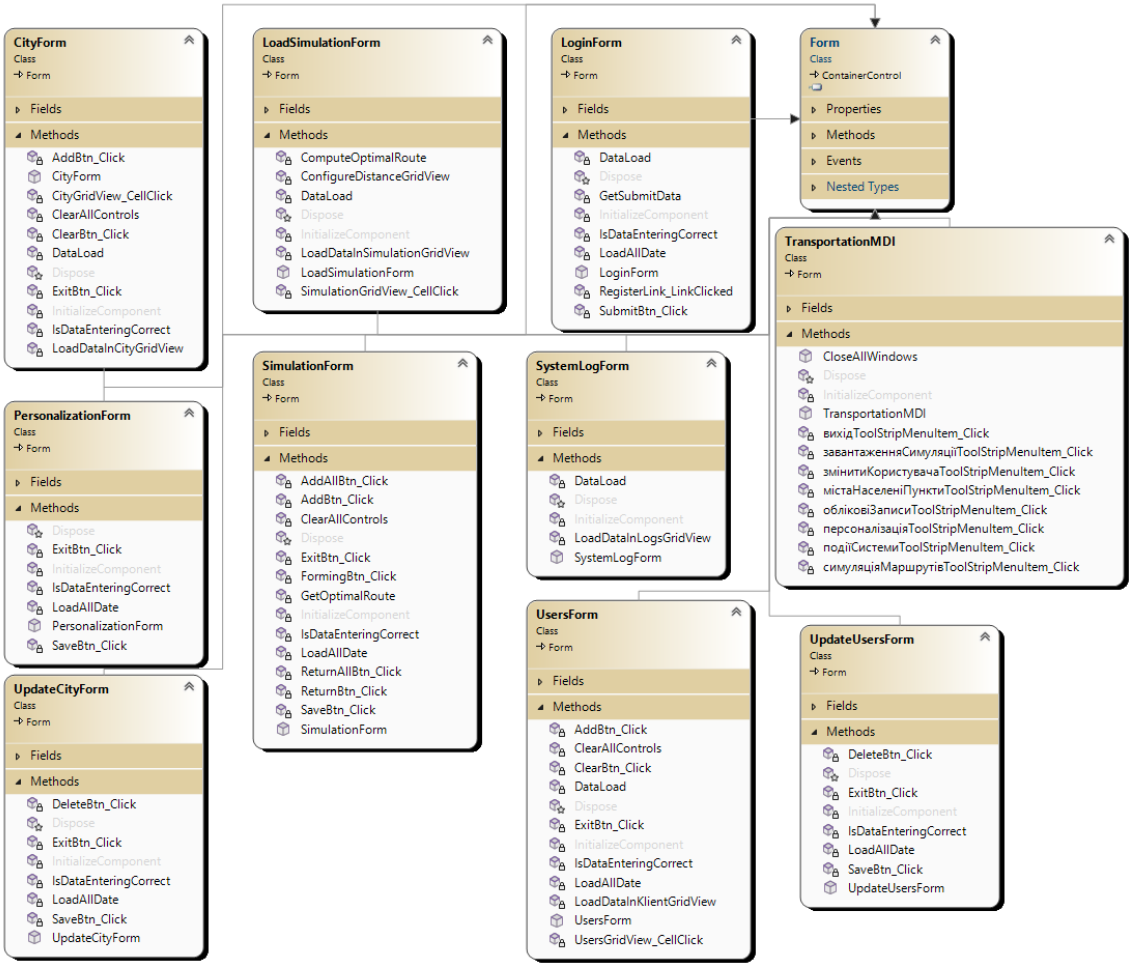


Рисунок 2.8 – Діаграма класів рівня користувацького інтерфейсу

Діаграма класів рівня користувацького інтерфейсу складається з:

– класу TransportationMDI, який є головною формою багатовіконного інтерфейсу. Через нього користувач має доступ до всіх функціональних підсистем: запуску симуляцій, перегляду логів, керування користувачами,

містами, налаштуваннями інтерфейсу. Він містить методи-обробники подій меню, які відкривають відповідні підформи системи;

– класу LoginForm, який реалізує механізм автентифікації користувача. Він містить методи перевірки даних для входу, завантаження списку користувачів та обробки подій введення. Після успішної перевірки відкриває головне вікно системи;

– класу SimulationForm, що призначений для запуску та управління симуляціями побудови маршрутів. Він забезпечує вибір міст, формування матриці відстаней, виконання оптимізації та відображення результату користувачеві;

– класу LoadSimulationForm, який реалізує функцію завантаження попередньо збережених симуляцій із бази даних, їх перегляд, повторну обробку й виведення оптимального маршруту на основі збережених параметрів;

– класу CityForm, який відповідає за перегляд, додавання нових міст і взаємодію з довідником населених пунктів. Містить методи перевірки введених даних, оновлення таблиці та керування подіями у формі;

– класу UpdateCityForm, який дозволяє редагувати вже створені записи про міста. Передбачає завантаження даних для редагування, оновлення значень у базі даних і обробку подій кнопок;

– класу UsersForm, що забезпечує управління обліковими записами: додавання, перегляд, видалення та пошук користувачів. Реалізує логіку перевірки введених даних та оновлення таблиці користувачів;

– класу UpdateUsersForm, який дозволяє редагувати дані про користувачів. Має методи для збереження змін, завантаження інформації та завершення роботи форми;

– класу SystemLogForm, який реалізує функціональність перегляду журналу подій системи. Забезпечує завантаження списку логів із бази даних та їх відображення у вигляді таблиці;

– класу PersonalizationForm, що надає користувачеві можливість змінювати налаштування інтерфейсу або зберігати персональні параметри роботи. Включає механізми перевірки коректності введення та збереження змін.

Рівень користувацького інтерфейсу побудований на принципах розділення відповідальностей, де кожна форма реалізує вузькоспеціалізований набір функцій, пов’язаних із певним логістичним процесом або етапом взаємодії. Це забезпечує зручність у використанні, покращує підтримку та розвиток системи, а також дозволяє гнучко масштабувати інтерфейс без зміни всієї структури. Завдяки використанню шаблонного підходу із загальною логікою подій, система демонструє стабільність у роботі та легкість у сприйнятті для кінцевого користувача.

У структурі розробленої системи рівень бізнес-логіки відіграє ключову роль, оскільки на ньому реалізовано основні правила обробки даних, перевірки правильності введення, шифрування та інші логічні дії, що забезпечують узгоджене функціонування інтерфейсу користувача з даними, які зберігаються у базі. На рис. 2.9 наведено діаграму класів рівня бізнес-логіки системи, яка охоплює класи для перевірки вхідних даних, генерації повідомлень, обробки шифрування та ролей користувача.

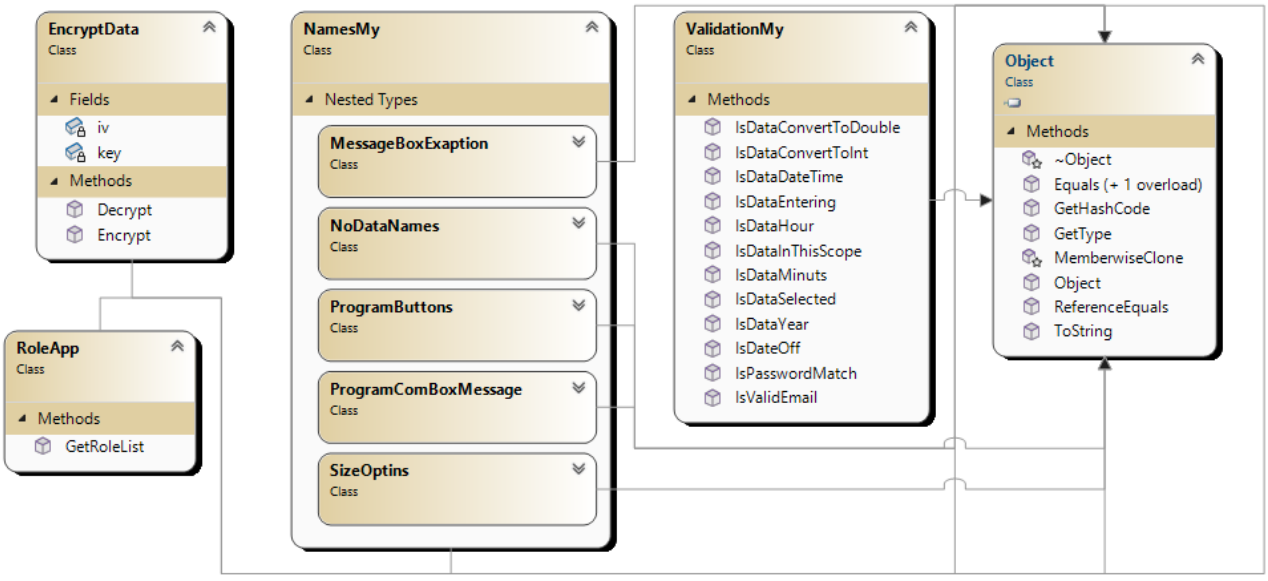


Рисунок 2.9 – Діаграма класів рівня користувацького інтерфейсу

Діаграма класів рівня бізнес-логіки складається з:

– класу ValidationMy, який виконує перевірку правильності введених користувачем даних у формах. Він містить набір методів для перевірки числових значень, дати, часу, електронної пошти, паролів та цілісності введених даних. Клас активно використовується на всіх рівнях інтерфейсу для забезпечення валідності інформації до її збереження у базі;

– класу EncryptData, який відповідає за шифрування та розшифрування конфіденційної інформації – зокрема, паролів користувачів. Має закриті поля iv та key, що відповідають за ініціалізацію алгоритму, і методи Encrypt та Decrypt для обробки рядкових значень;

– класу RoleApp, що забезпечує логіку ролей користувачів у системі. Він містить метод GetRoleList, який повертає перелік ролей, що можуть використовуватися при створенні облікових записів, і дозволяє здійснювати розмежування доступу до функцій системи;

– класу NamesMy, який реалізує набір вкладених типів, що містять текстові повідомлення та службові мітки, що відображаються користувачу. До вкладених класів належать.

Цей рівень системи забезпечує централізоване управління логікою перевірки, повідомлень і допоміжних обчислень. Він повністю відокремлений від візуального оформлення і роботи з базою даних, що дозволяє не дублювати функціональність у різних частинах програми. Завдяки цьому підходу досягається узгодженість перевірок, знижується ризик помилок, а система легко масштабується за рахунок розширення набору правил валідації або повідомлень.

2.5 Опис схеми бази даних

Побудова структури бази даних є одним із ключових етапів у процесі розробки програмного забезпечення, оскільки саме вона забезпечує надійне зберігання, доступ і обробку всієї необхідної інформації для виконання бізнес-логіки системи. Для задач, пов’язаних з оптимізацією маршрутів

транспортування будівельних матеріалів, база даних повинна підтримувати зберігання відомостей про населені пункти, симуляції, відстані між містами, облікові записи користувачів, а також системні події.

З урахуванням поставленої задачі, було спроєктовано логічну структуру реляційної бази даних на основі СУБД SQLite. Такий вибір обумовлений її компактністю, простотою розгортання та відсутністю потреби у встановленні серверного середовища, що повністю відповідає вимогам автономної роботи системи в умовах малого чи середнього підприємства. У межах розробленої системи створено такі основні таблиці:

Таблиця Users призначена для зберігання облікових даних користувачів системи, які взаємодіють з функціоналом модулів, пов’язаних із симуляціями маршрутів, переглядом логів, управлінням населеними пунктами тощо. У таблиці фіксуються персональні дані користувачів, інформація про права доступу, контактні дані, а також можливість залишення коментаря або опису до облікового запису (табл. 2.1).

Таблиця 2.1

Атрибути сутності «Users»

№	Найменування	Тип даних	Призначення
1	UsersId	INTEGER	Ідентифікатор користувача, первинний ключ
2	FirstName	TEXT	Ім’я користувача
3	LastName	TEXT	Прізвище користувача
4	UsersName	TEXT	Логін користувача, який використовується для авторизації
5	UsersPassword	TEXT	Пароль користувача у зашифрованому вигляді
6	RoleId	INTEGER	Ідентифікатор ролі користувача, що визначає його права в системі
7	Description	TEXT	Додатковий опис або коментар до облікового запису
8	Email	TEXT	Електронна адреса користувача для зв’язку або відновлення доступу

Таблиця Logs призначена для зберігання записів про події, що відбуваються у системі, зокрема авторизацій користувачів, запуску симуляцій, редагування даних тощо. Вона відіграє роль системного журналу (аудиту), що дозволяє відстежувати дії кожного користувача з прив'язкою до часу та сутності (табл. 2.2).

Таблиця 2.2

Атрибути сутності «Logs»

№	Найменування	Тип даних	Призначення
1	LogsId	INTEGER	Ідентифікатор події, первинний ключ
2	UsersId	INTEGER	Зовнішній ключ, що вказує на користувача, який виконав дію
3	EventNameShow	TEXT	Назва або короткий опис події
4	EventDate	TEXT	Дата й час фіксації події у системі

Таблиця Distance використовується для зберігання відстаней між містами в межах конкретної симуляції. Вона містить інформацію про пари населених пунктів, протяжність маршруту та розрахунковий час подолання. Усі записи пов'язуються з конкретною симуляцією через зовнішній ключ (табл. 2.3).

Таблиця 2.3

Атрибути сутності «Distance»

№	Найменування	Тип даних	Призначення
1	DistanceId	INTEGER	Ідентифікатор запису про відстань, первинний ключ
2	FromCity	TEXT	Назва міста, з якого починається маршрут
3	ToCity	TEXT	Назва міста, до якого пролягає маршрут
4	DistanceKm	INTEGER	Відстань між містами у кілометрах
5	EstimatedTimeMin	INTEGER	Орієнтовний час у хвилинах, необхідний для подолання маршруту
6	SimulationId	INTEGER	Зовнішній ключ, що вказує на симуляцію, до якої належить даний запис

Таблиця City використовується для зберігання довідника населених пунктів, між якими можуть бути побудовані маршрути доставки будівельних матеріалів. Міста з цієї таблиці беруть участь у симуляціях та при формуванні відстаней (табл. 2.4).

Таблиця 2.4

Атрибути сутності «City»

№	Найменування	Тип даних	Призначення
1	CityId	INTEGER	Ідентифікатор населеного пункту, первинний ключ
2	CityName	TEXT	Назва міста або населеного пункту
3	Description	TEXT	Додаткова інформація або коментар щодо міста

Таблиця Simulation призначена для зберігання загальної інформації про симуляції маршрутів, які створюються користувачем у процесі планування перевезень. Кожна симуляція містить унікальну назву, опис та використовується як основа для формування набору відстаней і побудови оптимального маршруту (табл. 2.5).

Таблиця 2.5

Атрибути сутності «Simulation»

№	Найменування	Тип даних	Призначення
1	SimulationId	INTEGER	Ідентифікатор симуляції, первинний ключ
2	SimulationName	TEXT	Назва симуляції, за якою її може ідентифікувати користувач
3	Description	TEXT	Додатковий опис, який пояснює мету або умови проведеної симуляції

У процесі розробки системи було спроєктовано базу даних, яка забезпечує збереження ключових сутностей, необхідних для моделювання логістичних процесів та управління симуляціями транспортування будівельних матеріалів. Структура охоплює інформацію про користувачів, населені пункти, симуляції маршрутів, лог подій та довідник відстаней, що забезпечує необхідну зв'язність

між об'єктами системи. Кожна таблиця виконує чітко окреслену функцію, а міжтабличні зв’язки реалізовано через зовнішні ключі, що підвищує цілісність даних. Узагальнена структура цих таблиць представлена у вигляді фізичної моделі бази даних на рис. 2.10.

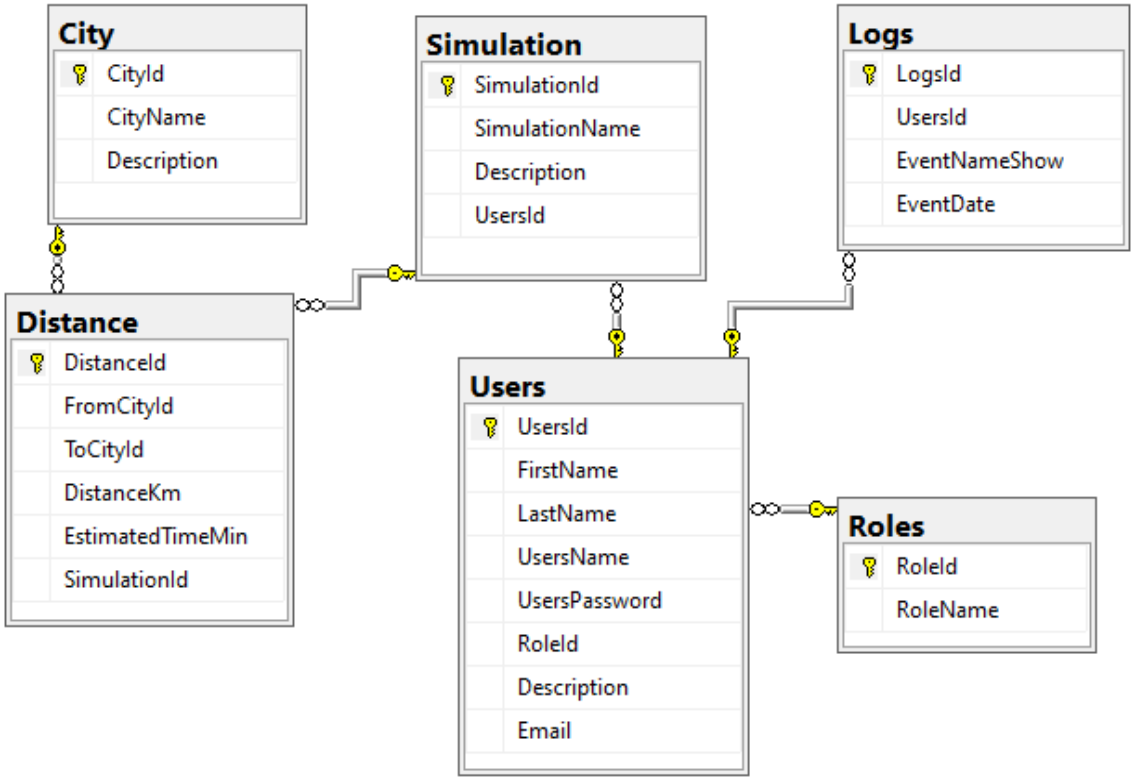


Рисунок 2.10 – Схема бази даних

Фізична модель бази даних демонструє логічно пов’язану схему, яка дозволяє зручно масштабувати систему, розширювати функціональність без порушення цілісності даних та забезпечує ефективне виконання запитів до бази. Така модель виступає базисом для побудови надійної платформи управління логістичними сценаріями в умовах змінного середовища.

2.6 Практична реалізація системи

Розробка програмної системи для підтримки прийняття рішень у процесах транспортування будівельних матеріалів передбачає реалізацію усіх визначених

функціональних складових із урахуванням трирівневої архітектури. Особлива увага приділяється забезпеченню взаємодії між рівнями, зокрема обміну даними між користувачем, логікою додатку та збереженням інформації у базі. Система має підтримувати повний цикл обробки логістичних симуляцій: створення вхідних даних, обробку маршрутів, збереження результатів, ведення журналу подій та адміністрування доступу користувачів.

Першим кроком практичної реалізації стала побудова рівня доступу до даних, який відповідає за роботу з таблицями бази даних. Реалізовано окремі класи-провайдери, що інкапсулюють запити до відповідних сутностей: користувачів, міст, симуляцій, маршрутів та логів. Такий підхід дозволяє централізовано обробляти операції створення, читання, оновлення та видалення (CRUD) з мінімізацією дублювання коду та забезпеченням високого рівня підтримки й масштабованості.

На рис. 2.11 зображено приклад коду, що відповідає за додавання нового міста або населеного пункту в таблицю City.

```
public void InsertCity(string cityName, string description) {
    using (var connection = new SqliteConnection(_ConnString)) {
        connection.Open();
        var command = connection.CreateCommand();
        command.CommandText = @"INSERT INTO City (CityName, Description)
                                VALUES (@CityName, @Description)";
        command.Parameters.AddWithValue("@CityName", cityName);
        command.Parameters.AddWithValue("@Description", description);
        command.ExecuteNonQuery();
    }
}
```

Рисунок 2.11 – Додавання нового міста або населеного пункту

Метод відкриває з’єднання з базою даних SQLite, створює SQL-команду з параметрами, що відповідають назві міста та його опису, й виконує її з метою збереження нових даних у відповідній таблиці. Такий підхід дозволяє гнучко працювати з довідником населених пунктів, гарантуючи коректну інтерпретацію

введених даних та їх подальше використання у формуванні логістичних сценаріїв.

Метод на рис. 2.12 відповідає за отримання повного списку міст, реалізує вибірку всіх записів з таблиці City, впорядкованих за алфавітом. Після відкриття з'єднання з базою даних формується SQL-команда, яка виконує відповідний запит. Результати обробляються за допомогою об'єкта SqliteDataReader, що дозволяє поетапно зчитувати кожен рядок даних. Кожне місто представляється як об'єкт класу City, де окремо присвоюються значення ідентифікатора, назви, опису та порядкового номера.

```
public List<City> GetAllCities() {
  List<City> result = new List<City>();
  using (var connection = new SqliteConnection(_ConnString)) {
    connection.Open();
    var command = connection.CreateCommand();
    command.CommandText = "SELECT * FROM City ORDER BY CityName ASC";
    using var reader = command.ExecuteReader();
    int i = 0;
    while (reader.Read()) {
      City one = new City {
        Number = ++i,
        CityId = reader.GetInt32(reader.GetOrdinal("CityId")),
        CityName = reader["CityName"]?.ToString() ?? "",
        Description = reader["Description"]?.ToString() ?? ""
      };
      result.Add(one);
    }
  }
```

Рисунок 2.12 – Отримання повного списку міст

Формування списку міст у структурованому вигляді дозволяє зручно відображати інформацію на рівні користувацького інтерфейсу, забезпечуючи при цьому її актуальність і коректність. Отримані дані можуть бути використані для формування випадаючих списків, вибору відправного та кінцевого пункту у маршрутах, а також для подальшого редагування або видалення записів, що підвищує гнучкість взаємодії з довідником населених пунктів.

На рис. 2.12 наведено метод, який реалізує вибірку одного конкретного міста за його унікальним ідентифікатором, переданим як параметр. Після

встановлення з’єднання з базою даних формується параметризований SQL-запит, що дозволяє уникнути потенційних атак типу SQL-ін’єкції. Команда звертається до таблиці City, де здійснюється пошук запису, значення поля CityId якого відповідає заданому значенню.

```
public City GetCityById(int cityId) {
    City city = new City();
    using (var connection = new SqliteConnection(_ConnString)) {
        connection.Open();
        var command = connection.CreateCommand();
        command.CommandText = "SELECT * FROM City WHERE CityId = @CityId";
        command.Parameters.AddWithValue("@CityId", cityId);
        using var reader = command.ExecuteReader();
        if (reader.Read()) {
            city.CityId = reader.GetInt32(reader.GetOrdinal("CityId"));
            city.CityName = reader["CityName"]?.ToString() ?? "";
            city.Description = reader["Description"]?.ToString() ?? "";
        } else {
            city.CityId = 0;
            city.Message = NamesMy.NoDataNames.NoDataInCity;
        }
    }
    return city;
}
```

Рисунок 2.13 – Вибірка одного конкретного міста за його унікальним ідентифікатором

У разі успішного знаходження відповідного запису, об’єкт City наповнюється значеннями зчитаних полів – зокрема, кодом міста, його назвою та описом. Якщо ж місто з таким ідентифікатором відсутнє, повертається об'єкт із кодом, що дорівнює нулю, та спеціальним повідомленням про відсутність даних. Такий підхід забезпечує коректну обробку виняткових ситуацій та дозволяє гнучко реагувати на спроби доступу до неактуальних або неіснуючих даних.

На рис. 2.14 наведено код методу, що відповідає за оновлення даних про місто або населений пункт у таблиці City. Метод реалізує оновлення існуючого запису за унікальним ідентифікатором, що передається як вхідний параметр. Після ініціалізації з’єднання із базою даних формується SQL-команда, яка оновлює значення полів CityName та Description відповідного запису. Параметри передаються безпосередньо через об’єкт команди, що дозволяє уникати SQL-

ін’єкцій і забезпечує надійність транзакцій. Після виконання команди база даних містить оновлену інформацію, що є ключовим кроком для підтримки актуального стану даних у системі.

```
public void UpdateCity(int cityId, string cityName, string description) {
    using (var connection = new SqliteConnection(_ConnString)) {
        connection.Open();
        var command = connection.CreateCommand();
        command.CommandText = @"UPDATE City
                                SET CityName = @CityName, Description = @Description
                                WHERE CityId = @CityId";
        command.Parameters.AddWithValue("@CityName", cityName);
        command.Parameters.AddWithValue("@Description", description);
        command.Parameters.AddWithValue("@CityId", cityId);
        command.ExecuteNonQuery();
    }
}
```

Рисунок 2.14 – Оновлення даних про місто або населений пункт

Метод, представлений на рис. 2.15, реалізує функціональність видалення запису з таблиці City за заданим ідентифікатором міста. На основі переданого параметра cityId створюється SQL-команда, яка формує запит типу DELETE, що дозволяє вилучити відповідний запис із бази даних.

```
public void DeleteCityById(int cityId) {
    using (var connection = new SqliteConnection(_ConnString)) {
        connection.Open();
        var command = connection.CreateCommand();
        command.CommandText = "DELETE FROM City WHERE CityId = @CityId";
        command.Parameters.AddWithValue("@CityId", cityId);
        command.ExecuteNonQuery();
    }
}
```

Рисунок 2.15 – Видалення запису з таблиці City

Після відкриття з’єднання з базою за допомогою бібліотеки SqliteConnection, об'єкт command налаштовується на виконання запиту з параметризованим значенням, що гарантує захист від SQL-ін’єкцій. Метод виконує команду без повернення результатів, адже операція має лише побічний

ефект – очищення таблиці від конкретного рядка. Таким чином, реалізується необхідна операція з підтримки цілісності й актуальності даних, що відображають географічні одиниці, задіяні в логістичних симуляціях.

Метод, зображений на рис. 2.16, відповідає за пакетне додавання відстаней між населеними пунктами до таблиці Distance у межах однієї симуляції. Для цього використовується структура List<Distance>, яка містить набір об’єктів із відповідними параметрами маршруту: початкове та кінцеве місто, протяжність маршруту в кілометрах, орієнтовний час у хвилинах, а також ідентифікатор симуляції.

```
public void InsertListDistance(List<Distance> list) {
  using (var connection = new SqliteConnection(_ConnString)) {
    connection.Open();
    using (var transaction = connection.BeginTransaction()) {
      foreach (var item in list) {
        var command = connection.CreateCommand();
        command.CommandText = @"INSERT INTO Distance (FromCity, ToCity,
                              DistanceKm, EstimatedTimeMin, SimulationId)
                              VALUES (@FromCity, @ToCity, @DistanceKm,
                              @EstimatedTimeMin, @SimulationId)";
        command.Parameters.AddWithValue("@FromCity", item.FromCity);
        command.Parameters.AddWithValue("@ToCity", item.ToCity);
        command.Parameters.AddWithValue("@DistanceKm", item.DistanceKm);
        command.Parameters.AddWithValue("@EstimatedTimeMin", item.EstimatedTimeMin);
        command.Parameters.AddWithValue("@SimulationId", item.SimulationId);
        command.ExecuteNonQuery();
      }
      transaction.Commit();
    }
  }
}
```

Рисунок 2.16 – Пакетне додавання відстаней між населеними пунктами

Після ініціалізації з'єднання з базою даних відкривається транзакція, що дозволяє вставити всі записи атомарно, забезпечуючи узгодженість та відсутність часткового збереження у разі помилки. Для кожного елемента списку створюється SQL-команда INSERT, параметри якої заповнюються зі структури Distance, після чого вона виконується в межах поточної транзакції. Завершальним етапом є виклик Commit, який фіксує всі зміни, що дозволяє

ефективно зберігати масив зв’язків між містами у рамках обраної симуляції. Такий підхід значно підвищує продуктивність при роботі з великими обсягами маршрутної інформації.

У межах реалізації рівня інтерфейсу користувача було спроєктовано набір функціональних форм, що відображають основні сценарії взаємодії з системою. Кожна форма орієнтована на виконання окремого завдання – від додавання населених пунктів і керування користувачами до запуску симуляцій і перегляду маршрутів. Основним принципом побудови графічного інтерфейсу стало забезпечення зручності, логічної структури та візуальної простоти, що дозволяє користувачеві інтуїтивно виконувати дії без потреби в додатковому навчанні.

На рис. 2.17 зображено форму, яка використовується для проведення симуляції маршруту на основі обраних населених пунктів.

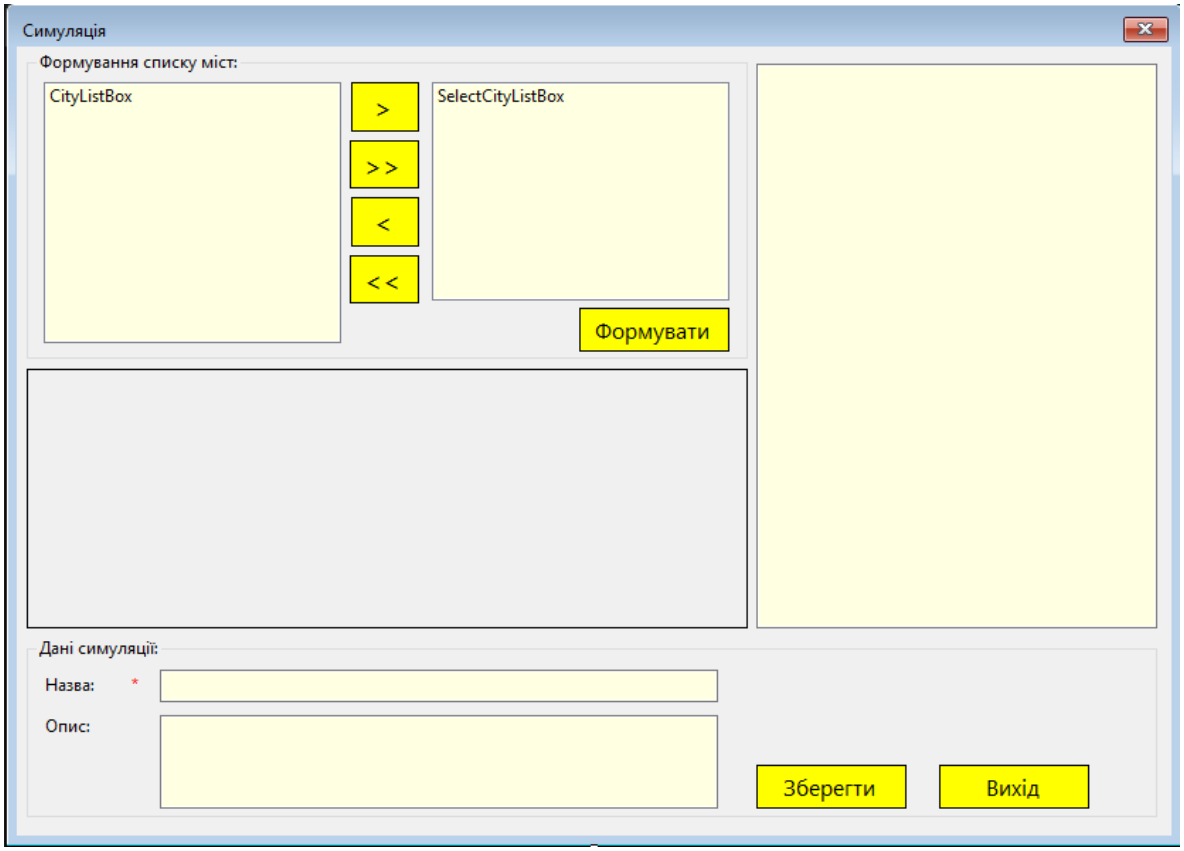


Рисунок 2.17 – Розробка форми для проведення симуляції

Форма побудована з урахуванням поетапного виконання завдань: ліворуч розміщено список доступних міст та кнопки для переміщення елементів до списку вибраних, праворуч – область для виводу результатів маршрутизації. Кнопка «Формувати» ініціює генерацію матриці відстаней між вибраними містами. Нижній блок призначено для введення назви та опису симуляції, а також для її збереження у базу даних. Завдяки такому компонуванню інтерфейс дозволяє користувачеві швидко сформувати початкові дані для оптимізації логістичного маршруту та зберегти їх для подальшого використання.

Метод, що викликається при ініціалізації форми симуляції, виконує завантаження списку всіх доступних міст із бази даних (рис. 2.18). Для цього викликається метод провайдера GetAllCities, який повертає повний перелік об’єктів типу City. Отриманий список зберігається у внутрішньому полі _CityList і прив’язується до елемента CityListBox як джерело даних. Паралельно налаштовуються властивості відображення – DisplayMember вказує, що користувач бачить назву міста, а ValueMember – що за кожною позицією зберігається унікальний ідентифікатор.

```
private void LoadAllDate() {
    _CityList = _CityProvider.GetAllCities();
    CityListBox.DataSource = new List<City>(_CityList);
    CityListBox.DisplayMember = "CityName";
    CityListBox.ValueMember = "CityId";
    SelectCityListBox.DataSource = new List<City>();
    SelectCityListBox.DisplayMember = "CityName";
    SelectCityListBox.ValueMember = "CityId";
}
```

Рисунок 2.18 – Завантаження списку всіх доступних міст із бази даних

Список SelectCityListBox, який використовується для відбору міст у межах симуляції, ініціалізується як порожній, але з аналогічною структурою відображення. Це формує основу для подальшої динамічної взаємодії з вибраними пунктами.

Метод AddBtn_Click реалізує логіку переміщення вибраного користувачем міста зі списку доступних міст до списку вибраних для формування маршруту

(рис. 2.19). Після визначення джерела (CityListBox) і приймача (SelectCityListBox) у вигляді списків типу List<City>, виконується перевірка: чи дійсно вибрано елемент, і чи місто ще не присутнє у списку призначення. Якщо обидві умови виконано, об'єкт типу City видаляється зі списку source і додається до списку target.

```
private void AddBtn_Click(object sender, EventArgs e) {
  var source = (List<City>)CityListBox.DataSource;
  var target = (List<City>)SelectCityListBox.DataSource;
  if (CityListBox.SelectedItem is City selectedCity &&
    !target.Any(c => c.CityId == selectedCity.CityId)) {
    source.Remove(selectedCity);
    target.Add(selectedCity);
    CityListBox.DataSource = new List<City>(source);
    SelectCityListBox.DataSource = new List<City>(target);
  }
}
```

Рисунок 2.19 – Переміщення вибраного міста зі списку доступних міст до списку вибраних

Після оновлення обох колекцій списки прив’язуються заново до відповідних візуальних компонентів, що дозволяє миттєво оновити інтерфейс без необхідності перезавантаження форми. Такий підхід дозволяє реалізувати зручне перетягування даних між двома списками, зберігаючи при цьому унікальність вибраних пунктів і підтримуючи правильність подальшої симуляції.

Метод AddAllBtn_Click реалізує масове переміщення всіх міст із початкового списку до списку вибраних для симуляції. Для цього список із CityListBox зчитується як джерело, а список із SelectCityListBox – як приймач.

```
private void AddAllBtn_Click(object sender, EventArgs e) {
  var source = source = (List<City>)CityListBox.DataSource;
  var target = target = (List<City>)SelectCityListBox.DataSource;
  source.AddRange(source);
  target.Clear();
  CityListBox.DataSource = new List<City>(source);
  SelectCityListBox.DataSource = new List<City>(target);
}
```

Рисунок 2.20 – Переміщення всіх міст

Після цього список джерела очищується методом Clear, що означає повне виведення всіх міст зі списку доступних. Обидва оновлені списки повторно прив’язуються до відповідних візуальних компонентів, забезпечуючи відображення змін у графічному інтерфейсі. Такий механізм спрощує взаємодію з великими списками та дозволяє користувачеві швидко обрати повний набір населених пунктів для подальшої обробки.

Метод FormingBtn_Click відповідає за генерацію повної матриці маршрутів між усіма вибраними містами, що беруть участь у симуляції (рис. 2.21). Зі списку SelectCityListBox отримується колекція міст, після чого за допомогою вкладених циклів формується кожна можлива пара напрямку з міста from у місто to, уникаючи випадку, коли обидва індекси збігаються. Для кожної такої пари генерується випадкова відстань у межах від 100 до 800 км, що моделює реальну довжину маршруту.

```
private void FormingBtn_Click(object sender, EventArgs e) {
  var selectedCities = (List<City>)SelectCityListBox.DataSource;
  int number = 1;
  for (int i = 0; i < selectedCities.Count; i++) {
    for (int j = 0; j < selectedCities.Count; j++) {
      if (i == j) continue;
      var from = selectedCities[i];
      var to = selectedCities[j];
      int distanceKm = new Random().Next(100, 800);
      int estimatedTime = (int)(distanceKm / 60.0 * 60); // припустимо 60 км/год
      _DistanceList.Add(new Distance {
        Number = number++,
        FromCity = from.CityName,
        ToCity = to.CityName,
        DistanceKm = distanceKm,
        EstimatedTimeMin = estimatedTime,
        SimulationId = 0
      });
    }
  }
```

Рисунок 2.21 – Генерація повної матриці маршрутів між усіма вибраними містами

На основі цієї відстані розраховується орієнтовний час пересування, виходячи з середньої швидкості руху – умовно прийнятої як 60 км/год. Для кожної згенерованої пари створюється новий об’єкт типу Distance, в якому

зберігаються всі параметри маршруту, включаючи номер, назви міст, розраховану відстань, час та симуляційний ідентифікатор. Цей об’єкт додається до внутрішнього списку _DistanceList, який надалі використовується для обробки та збереження результатів симуляції. Такий підхід дозволяє автоматизувати створення первинних даних маршруту, необхідних для виконання оптимізації.

Метод GetOptimalRoute реалізує первинну перевірку наявності вхідних даних перед запуском алгоритму оптимізаці (рис. 2.22)ї. Із графічного компонента DistanceGridView за допомогою приведення через BindingSource отримується зв'язаний список об'єктів типу Distance, що містить маршрути між містами. Ці дані є базовими для побудови матриці відстаней, яка в подальшому буде передана до алгоритму обчислення найоптимальнішого маршруту.

```
private void GetOptimalRoute() {
  // 1. Отримуємо список відстаней
  var distanceList =
    ((BindingSource)DistanceGridView.DataSource)?.DataSource as BindingList<Distance>;
  if (distanceList == null || distanceList.Count == 0) {
    ResultTBox.Text = "Дані відсутні або не коректні.";
    return;
  }
```

Рисунок 2.22 – Отримання списку відстаней

Після отримання колекції перевіряється її наявність та кількість записів. У разі, якщо список є порожнім або не ініціалізований, відображається відповідне повідомлення у текстовому полі ResultTBox. Така перевірка дозволяє уникнути помилок при обробці порожніх чи некоректних даних і гарантує, що алгоритм транспортної оптимізації буде запускатися лише за наявності валідної матриці маршрутів.

На наступному етапі формується повний перелік унікальних міст, що фігурують у вибраних маршрутах (рис. 2.23). Для цього застосовується метод SelectMany, який дозволяє об'єднати значення полів FromCity та ToCity з кожного елемента списку, утворюючи загальну колекцію назв усіх задіяних населених пунктів.

```
var allCities = distanceList
    .SelectMany(d => new[] { d.FromCity, d.ToCity })
    .Distinct()
    .ToList();
```

Рисунок 2.23 – Формування повного переліку унікальних міст

Подальше застосування методу Distinct усуває дублікати, залишаючи лише унікальні записи. В результаті формується впорядкований список allCities, який слугує основою для побудови індексної матриці відстаней та асоціюється з вершинами графа при реалізації алгоритмів пошуку оптимального маршруту. Такий підхід забезпечує коректне відображення логіки транспортної задачі, де кожне місто виступає окремим вузлом у моделі.

На рис. 2.24 визначається кількість міст, що беруть участь у симуляції, та на основі цього формується розмірність квадратної матриці відстаней. Для зручності подальших обчислень створюються два відображення: cityNameToIndex, яке зіставляє назву міста з його індексом у масиві, та indexToCity, яке дозволяє зворотне відображення – від індексу до назви міста. Це забезпечує двобічний зв’язок між символьними й числовими представленнями вузлів графа.

```
int n = allCities.Count;
var cityNameToIndex =
  allCities.Select((name, index) => new { name, index })
  .ToDictionary(x => x.name, x => x.index);
var indexToCity = allCities.ToList();
// 3. Створення матриці відстаней
int[,] distanceMatrix = new int[n, n];
foreach (var d in distanceList) {
  if (cityNameToIndex.ContainsKey(d.FromCity) &&
    cityNameToIndex.ContainsKey(d.ToCity)) {
    int fromIdx = cityNameToIndex[d.FromCity];
    int toIdx = cityNameToIndex[d.ToCity];
    distanceMatrix[fromIdx, toIdx] = d.DistanceKm;
  }
}
```

Рисунок 2.24 – Створення матриці відстаней

На основі отриманих маршрутів та цих відповідностей ініціалізується двовимірна цілочисельна матриця distanceMatrix, у якій кожна комірка [i, j] зберігає значення відстані між містами з індексами i та j. Дані до цієї матриці заповнюються шляхом проходу по списку маршрутів: для кожного з них визначаються індекси початкового (FromCity) та кінцевого (ToCity) міст, після чого відповідна комірка в матриці отримує значення відстані в кілометрах. Така структура є ключовою для подальшого використання бібліотеки Google OR-Tools, яка вимагає саме матричного подання ваг графа при реалізації задач маршрутизації.

На рис. 2.25 ініціалізується об’єкт RoutingIndexManager, який відповідає за управління індексами вузлів у маршрутизаторі. У даному випадку вказується кількість вузлів n, кількість транспортних засобів 1, а також стартова точка – індекс 0, що означає, що маршрут розпочинається з першого міста у списку. Далі створюється основний об’єкт маршрутизатора RoutingModel, який працює з графом можливих переміщень між містами на основі заданої матриці відстаней.

```
var manager = new RoutingIndexManager(n, 1, 0);
var routing = new RoutingModel(manager);
int transitCallbackIndex =
  routing.RegisterTransitCallback((long fromIndex, long toIndex) => {
  int from = manager.IndexToNode(fromIndex);
  int to = manager.IndexToNode(toIndex);
  return distanceMatrix[from, to];
});
routing.SetArcCostEvaluatorOfAllVehicles(transitCallbackIndex);
var searchParameters =
  operations_research_constraint_solver.DefaultRoutingSearchParameters();
searchParameters.FirstSolutionStrategy =
  FirstSolutionStrategy.Types.Value.PathCheapestArc;
var solution = routing.SolveWithParameters(searchParameters);
if (solution == null) {
  ResultTBox.Text = "Маршрут не знайдено.";
  return;
}
```

Рисунок 2.25 – Побудова маршруту

Для того щоб маршрутизатор міг використовувати конкретні значення відстаней між містами, реєструється функція зворотного виклику

RegisterTransitCallback. Вона приймає на вхід індекси вузлів у внутрішньому представленні OR-Tools (fromIndex, toIndex), конвертує їх у відповідні індекси міст за допомогою IndexToNode, і повертає відстань між цими містами з матриці distanceMatrix. Ця функція встановлюється як оцінка вартості дуг (arc cost evaluator) для всіх транспортних засобів у задачі.

Після цього формуються параметри пошуку DefaultRoutingSearchParameters, де як початкову стратегію побудови рішення обирається PathCheapestArc, що передбачає побудову найменш витратного шляху від поточного вузла до наступного. Нарешті викликається метод SolveWithParameters, який запускає пошук оптимального маршруту. Якщо рішення знайти не вдалося, у текстовому полі виводиться відповідне повідомлення, що маршрут не знайдено, і обчислення припиняється.

У завершальному етапі реалізується побудова текстового представлення оптимального маршруту для виведення на інтерфейс (рис. 2.26). За допомогою об'єкта StringBuilder формується структурований текст, який починається з заголовка «Оптимальний маршрут:». Змінна index ініціалізується як початкова позиція маршруту, а змінна totalDistance використовується для накопичення сумарної відстані.

```
    var sb = new StringBuilder();
    sb.AppendLine("Оптимальний маршрут:");
    long index = routing.Start(0);
    int totalDistance = 0;
    while (!routing.IsEnd(index)) {
      int fromNode = manager.IndexToNode(index);
      long nextIndex = solution.Value(routing.NextVar(index));
      int toNode = manager.IndexToNode(nextIndex);
      int legDistance = distanceMatrix[fromNode, toNode];
      totalDistance += legDistance;
      sb.AppendLine($"{indexToCity[fromNode]} → " +
        $"{indexToCity[toNode]} ({legDistance} км)");
      index = nextIndex;
    }
    sb.AppendLine($"Загальна відстань: {totalDistance} км");
    ResultTBox.Text = sb.ToString();
}
```

Рисунок 2.26 – Побудова текстового представлення оптимального маршруту

У циклі while, який виконується доти, доки не досягнуто кінцевого вузла маршруту, за допомогою IndexToNode визначається поточний вузол (fromNode) та наступний вузол (toNode), на який буде здійснено перехід згідно з рішенням, сформованим маршрутизатором. Відповідна відстань між цими вузлами зчитується з матриці відстаней distanceMatrix, після чого додається до загального підсумку та записується у вигляді окремого рядка з відображенням назви міст та відповідної відстані.

Після завершення обходу всіх сегментів маршруту до тексту додається рядок із загальною сумою пройдених кілометрів. Сформований текст присвоюється елементу ResultTBox, забезпечуючи користувача зрозумілою і повною інформацією про оптимальний шлях між обраними містами. Такий підхід дозволяє інтерактивно проаналізувати маршрут, що було побудовано на основі симуляційних даних.

Під час обробки натискання кнопки збереження симуляції спочатку перевіряється коректність введених даних за допомогою методу IsDataEnteringCorrect (рис. 2.27).

```
private void SaveBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    // Збереження симуляції
    _SimulationProvider.InsertSimulation(SimulationNameTBox.Text, DescriptionTBox.Text);
    //Отримання останньої симуляції
    Simulation lastSim = _SimulationProvider.GetLastSimulation();
    //Прив'язка SimulationId до всіх відстаней
    for (int i = 0; i < _DistanceList.Count; i++) {
      _DistanceList[i].SimulationId = lastSim.SimulationId;
    }
    //Збереження списку відстаней
    _DistanceProvider.InsertListDistance(_DistanceList);
    //Повідомлення користувачу
    MessageBox.Show("Симуляцію успішно збережено.", "Інформація",
      MessageBoxButtons.OK, MessageBoxIcon.Information);
    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId, "Симуляцію "
      + SimulationNameTBox.Text + " збережено", DateTime.Now);
    //Очищення інтерфейсу
    ClearAllControls();
  }
}
```

Рисунок 2.27 – Збереження симуляції

У випадку позитивного результату здійснюється збереження введеної симуляції, яка описується назвою та текстовим описом. Для цього викликається метод InsertSimulation, що додає новий запис до відповідної таблиці в базі даних.

Далі отримується остання додана симуляція за допомогою методу GetLastSimulation. Отриманий ідентифікатор симуляції (SimulationId) використовується для оновлення відповідного поля у всіх об'єктах відстаней, які було згенеровано в межах поточної симуляції. Це забезпечує коректне логічне зв’язування кожного елемента маршруту з конкретною симуляцією.

Після цього викликається метод InsertListDistance, який здійснює збереження оновленого списку відстаней до бази даних у межах транзакції. Успішне завершення операції супроводжується повідомленням для користувача про збереження симуляції, після чого до системного журналу вносяться відомості про подію з фіксацією часу та ідентифікатора користувача, що її ініціював. Наприкінці відбувається очищення полів інтерфейсу з метою підготовки форми до нових дій.

2.7 Тестування системи на прикладі реальних даних

Оцінювання працездатності створеної інформаційної системи для вибору оптимального маршруту транспортування будівельних матеріалів є ключовим етапом завершення розробки. Метою цього процесу є перевірка адекватності реалізованої логіки обробки даних, стабільності взаємодії між програмними компонентами та відповідності поведінки системи очікуваним результатам при роботі з фактичними або максимально наближеними до реальних вхідними даними. Тестування дозволяє виявити та усунути потенційні помилки, що могли залишитися непоміченими на етапі реалізації, зокрема пов’язані з формуванням відстаней, коректністю обрахунку симуляцій або алгоритмом побудови оптимального маршруту.

Для демонстрації цього нижче наведено приклади типових тестових сценаріїв, реалізованих у межах ключових функціональних модулів.

Тестовий приклад: №1

Призначення: перевірити коректність авторизації користувача в системі за введеними логіном і паролем.

Тест-вимоги, що перевіряються: наявність валідації обов'язкових полів, перевірка відповідності логіна і пароля даним у базі, коректне відображення повідомлення у випадку помилки або успішної авторизації, перехід до головної форми після входу.

Передумови для тесту: у базі повинен бути зареєстрований обліковий запис користувача admin із відповідним паролем; програма запущена і активна форма авторизації.

Критерій проходження тесту: при правильному введенні логіна і пароля система повинна здійснити вхід і відкрити головну форму, а при неправильних даних – відобразити відповідне повідомлення про помилку.

Таблиця 2.6

Тестування сценарію авторизації користувача в системі

№ з/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка (Так/Ні)
1	Ввести логін admin і правильний пароль	Здійснюється успішна авторизація, відкривається головне вікно	Авторизація виконана, головне меню відкрите	Так
2	Ввести логін admin і неправильний пароль	Відображається повідомлення: «Невірний логін або пароль»	Повідомлення про помилку з'явилося	Так
3	Залишити поле пароля порожнім і натиснути «Підтвердити»	Відображається повідомлення про необхідність заповнення обов'язкового поля	Повідомлення: «Поле пароля є обов'язковим»	Так
4	Натиснути посилання «Реєстрація»	Відкривається форма реєстрації нового користувача	Форма реєстрації відкрилася	Так

На рис. 2.24 зображено вікно авторизації користувача в системі після введення логіна та пароля.

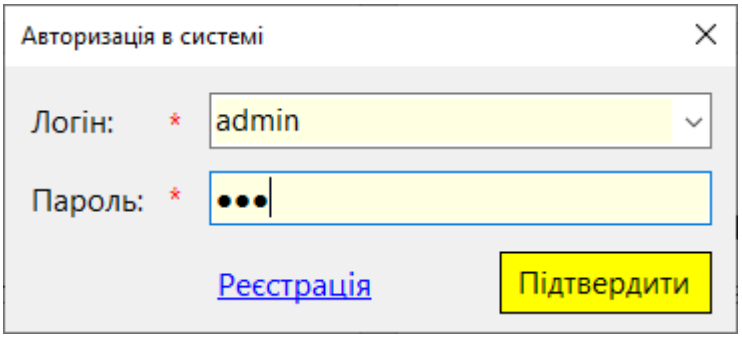


Рисунок 2.28 – Авторизація користувача

Проведений сценарій підтверджує коректну роботу механізму перевірки облікових даних та логіку переходу до головного вікна після успішного входу. Усі передбачені перевірки виконано без збоїв, що свідчить про надійність реалізованого механізму авторизації.

Тестовий приклад: №2

Призначення: перевірити коректність роботи механізму симуляції маршрутів між містами, включаючи формування матриці відстаней та побудову оптимального маршруту.

Тест-вимоги, що перевіряються: правильність формування маршруту відповідно до введеного списку міст, обчислення відстаней і часу, виведення оптимального маршруту у правій панелі, відповідність сумарної відстані сумі окремих сегментів.

Передумови для тесту: обліковий запис користувача авторизований у системі, форма симуляції відкрита, довідник міст заповнений, доступні не менше 5 міст.

Критерій проходження тесту: після вибору міст і натискання кнопки «Формувати» повинен з’явитися список усіх маршрутів у таблиці, а в правій частині – текст оптимального маршруту з правильною послідовністю і розрахованою сумарною відстанню.

Таблиця 2.7

Тестування сценарію побудови симуляції маршруту

№ з/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка (Так/Ні)
1	Обрати 5 міст зі списку зліва та перенести до правого списку	Міста повинні з’явитися у списку вибраних, лівий список оновлюється	Обрані міста відображено праворуч, лівий список оновлено	Так
2	Натиснути кнопку «Формувати»	Таблиця заповнюється маршрутами між усіма парами міст, формується матриця відстаней	Таблиця містить усі унікальні комбінації пар, кожна з відстанню та часом	Так
3	Перевірити правильність побудови оптимального маршруту	У правій частині відображається найкоротший шлях з урахуванням усіх міст, маршрут замкнений	Відображено маршрут: Запоріжжя → Ужгород → Миколаїв → Харків → Львів → Запоріжжя, 1313 км	Так
4	Перевірити сумарну відстань маршруту за окремими сегментами з таблиці	Сума окремих ділянок у маршруті повинна дорівнювати загальній відстані	Сума ділянок: 211 + 386 + 141 + 289 + 286 = 1313 км	Так
5	Ввести назву симуляції та натиснути «Зберегти»	Симуляція записується у базу, користувачу виводиться підтвердження, поля очищаються	Симуляцію збережено, повідомлення виведено, поля очищені	Так

На рис. 2.25 зображено результат симуляції для п’яти міст, у якому видно таблицю маршрутів, оптимальний шлях і введену назву симуляції.

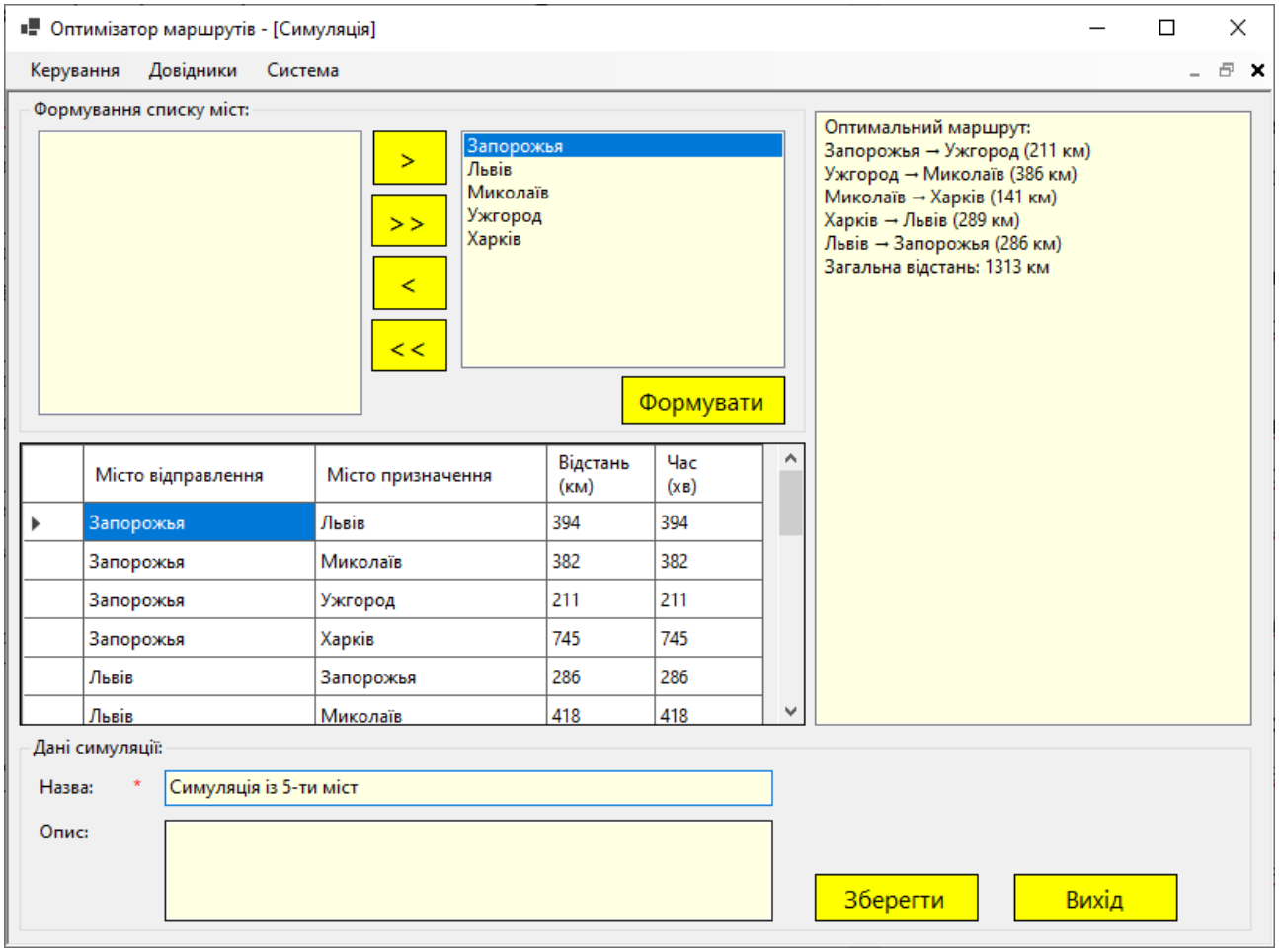


Рисунок 2.29 – Симуляція маршрутів для 5-ти міст України

Для оцінювання відповідності роботи створеної системи обробки та оптимізації маршрутів очікуваному функціоналу було проведено функціональне тестування на основі типових сценаріїв взаємодії користувача з інтерфейсом. Кожен тест виконувався вручну згідно з підготовленими послідовностями дій, що дозволило імітувати реальні умови експлуатації програмного забезпечення. Результати перевірок фіксувалися через візуальне підтвердження та зіставлення фактичної поведінки системи з очікуваною.

Для проведення тестування використовувалося таке апаратне забезпечення:

– ноутбук: Lenovo IdeaPad 5 Pro 16ACH6;
– процесор: AMD Ryzen 5 5600H (6 ядер, 12 потоків);
– оперативна пам’ять: 16 ГБ DDR4;
– накопичувач: SSD 512 ГБ PCIe NVMe.

Програмне середовище, у якому виконувалося тестування:

– операційна система: Windows 11 Pro x64, версія 23H2;
– середовище розробки: Visual Studio 2022;
– програмна платформа: .NET Framework 4.8;
– база даних: SQLite;
– тип застосунку: настільний WinForms-додаток.

У таблиці 2.8 наведено узагальнені результати тестування функціональних можливостей системи.

Таблиця 2.8

Результати функціонального тестування

Номер тест-кейсу	Очікуваний результат	Отриманий результат	Коментар щодо збігу результатів
1	Після введення коректних логіна і пароля відкривається головне вікно системи	Авторизація успішна, відкрито головну форму з активними функціями	Збіг – механізм автентифікації працює стабільно
2	Додавання нової симуляції маршрутів завершується збереженням до бази даних	Симуляція додана, інформація про маршрути збережена	Збіг – створення нової симуляції реалізовано коректно
3	Побудова оптимального маршруту завершується текстовим результатом	Виведено маршрут з усіма сегментами та сумарною відстанню	Збіг – обчислення відбулося успішно
4	Видалення населеного пункту призводить до його зникнення зі списку та бази даних	Місто видалено, зміни відображено у списку	Збіг – видалення запису працює правильно
5	Після редагування опису симуляції дані мають оновлюватися в таблиці	Дані оновлено, оновлений опис збережено в БД	Збіг – редагування працює відповідно до очікувань

Результати проведених перевірок підтвердили функціональну відповідність програмного забезпечення очікуваній поведінці в межах заданих

сценаріїв. Ключові операції – від авторизації користувача до побудови та збереження симуляцій – виконуються без збоїв, а взаємодія з інтерфейсом є стабільною й послідовною. Дані передаються, обробляються та зберігаються відповідно до логіки роботи системи, що дозволяє вважати програму готовою до практичного використання.

Тестування окремих модулів системи стало важливою складовою забезпечення її якості, оскільки дозволяє перевірити надійність роботи кожного логічного компонента до моменту їх інтеграції. У межах реалізованого рішення було протестовано компоненти, що відповідають за взаємодію з базою даних, обробку логіки бізнес-рівня та елементи взаємодії з графічним інтерфейсом.

Для проведення перевірок використовувався модульний фреймворк MSTest, вбудований у середовище Visual Studio 2022, що забезпечив зручну організацію тестових класів і запуск сценаріїв. На рис. 2.30 продемонстровано результати запуску тестів, які успішно пройдено без виявлених критичних відхилень.

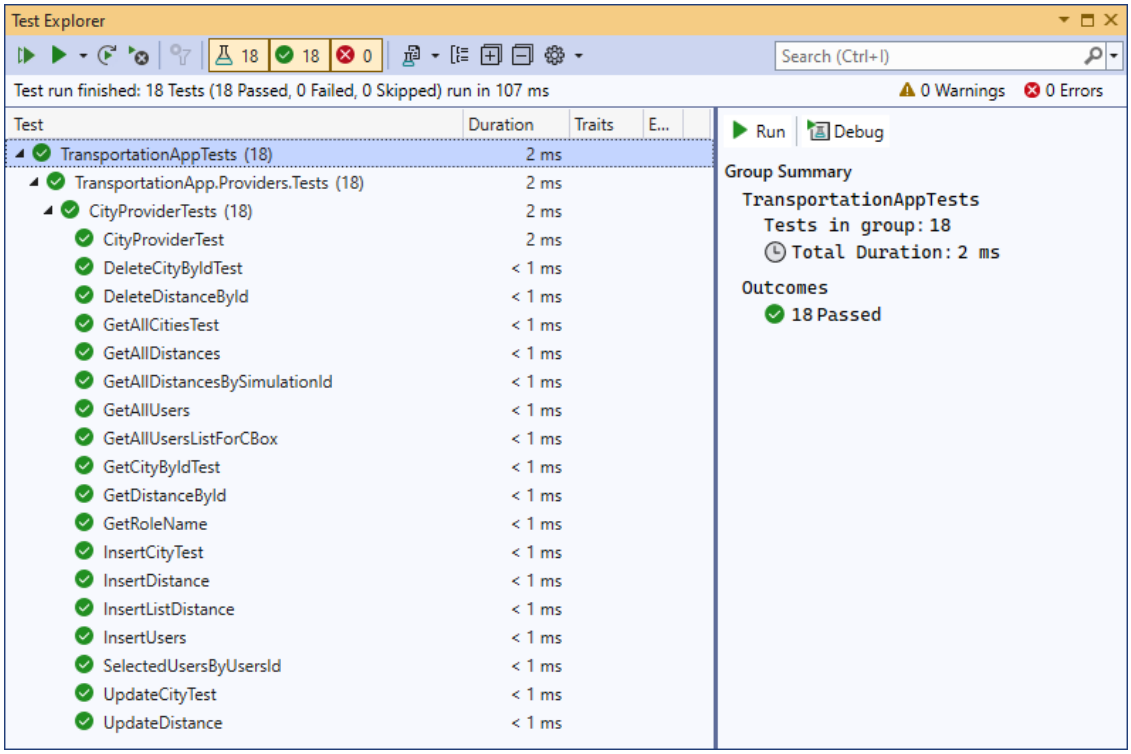


Рисунок 2.30 – Результати модульного тестування

Згідно з результатами, наведеними на рисунку, було виконано 18 юніт-тестів, кожен з яких завершився успішно без жодного збою чи помилки. Тестування охопило ключові функціональні методи, включаючи операції додавання, видалення, оновлення та вибірки даних з таблиць міст, симуляцій і відстаней. Серед перевірених методів – InsertCityTest, GetAllDistances, DeleteDistanceById, GetRoleName тощо. Загальний час виконання всієї серії тестів склав лише 2 мілісекунди, що вказує на високу швидкодію системи та добру оптимізацію логіки окремих компонентів. Отриманий результат підтверджує правильність реалізації бізнес-логіки та стабільність роботи класів-провайдерів у рамках тестованого середовища.

Оцінка навантажувальної стійкості є невіддільною частиною перевірки працездатності системи в умовах інтенсивної експлуатації, особливо якщо передбачається обслуговування значної кількості одночасних користувачів. У контексті розробки системи для управління транспортною логістикою, де можливе паралельне звернення до модуля побудови маршрутів, до бази даних або до механізмів обчислення, таке тестування дозволяє виявити критичні ділянки навантаження, оцінити ефективність роботи за високої активності та визначити межу збереження стабільного часу відповіді.

Метою проведеного навантажувального експерименту було:

– визначити рівень одночасного навантаження, за якого система функціонує без збоїв;

– встановити залежність між кількістю запитів і затримкою у відповіді на них;

– перевірити поведінку програми в умовах різкого зростання кількості користувачів;

– зафіксувати поріг, після якого спостерігається суттєве погіршення продуктивності.

Для імітації одночасних звернень було створено консольний тестовий застосунок на C#, який за допомогою ThreadPool імітує дії користувачів: запуск симуляцій маршрутів, звернення до бази даних і побудову оптимального шляху.

Кожен віртуальний сеанс виконував послідовність дій: ініціалізація підключення, запуск обчислень та логування події.

Таблиця 2.9

Результати вимірювання часу відгуку

Тест-кейс	Кількість користувачів	Очікуваний час відгуку (мс)	Отриманий час відгуку (мс)
ТС-1	10	≤ 2000	120
ТС-2	20	≤ 2000	210
ТС-3	30	≤ 2000	350
ТС-4	50	≤ 2000	690
ТС-5	70	≤ 2000	1100
ТС-6	100	≤ 2000	1830
ТС-7	120	≤ 2000	2450

Рис. 2.31 висвітлює залежність середнього часу відгуку від кількості користувачів.

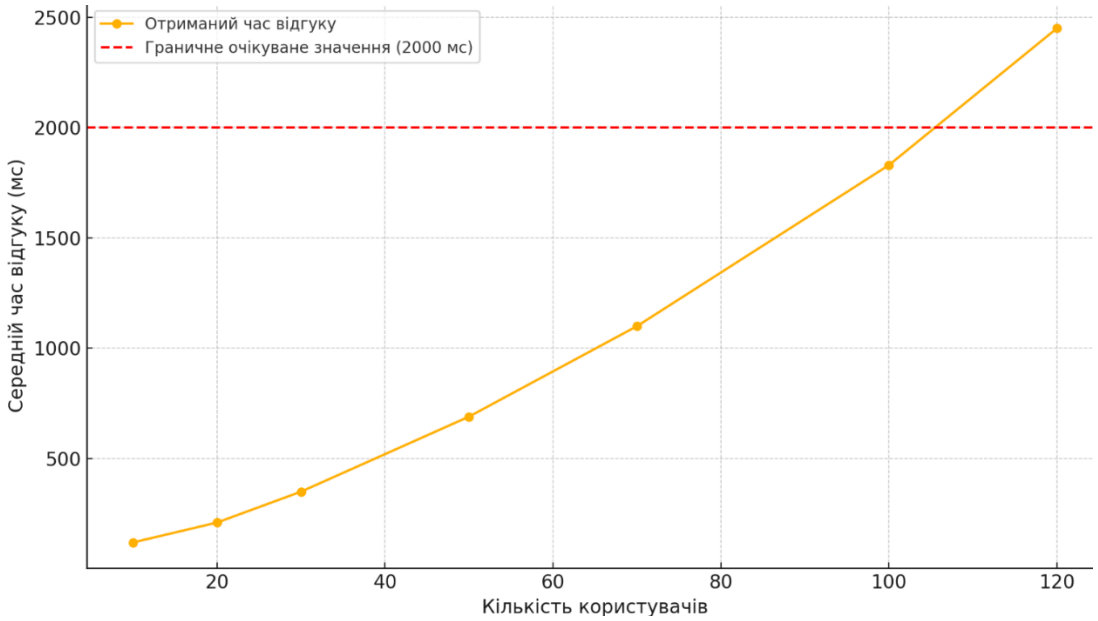


Рисунок 2.31 – Залежність середнього часу відгуку від кількості користувачів

Аналіз наведених даних дозволяє зробити висновок, що при навантаженні до 100 користувачів система утримує час відповіді в межах допустимих

нормативів. Погіршення продуктивності починає проявлятись за 120 паралельних звернень, що може бути підставою для подальшої оптимізації або масштабування.

Об’ємне тестування проводилося з метою оцінки здатності розробленої системи маршрутної логістики ефективно працювати з великими обсягами інформації в умовах інтенсивної взаємодії з базою даних. Для імітації великої кількості даних база даних була попередньо наповнена синтетично згенерованими значеннями з використанням автоматизованого скрипту, що дозволило досягти наступних обсягів:

– загальна кількість міст – 5 000;
– загальна кількість маршрутів (відстаней між містами) – понад 250 000;
– кількість симуляцій – 10 000;
– кількість логів взаємодії з системою – понад 1 000 000 записів.

Генеровані дані були імпортовані до сховища, що базується на SQLite, оптимізованому для роботи у вбудованому середовищі без серверної інфраструктури. Після заповнення було виконано серію тестових кейсів, що дозволили оцінити швидкодію ключових функцій при роботі з великими наборами даних (табл. 2.10).

Таблиця 2.10

Результати тестування системи при великому обсязі даних

Тест-кейс	Очікуваний час відгуку (мс)	Отриманий час відгуку (мс)
ТС-1: Пошук міста серед 5 000 записів	≤ 2000	980
ТС-2: Побудова симуляції для 50+ міст із 2500 маршрутів	≤ 3000	2750
ТС-3: Збереження симуляції з масивом з 1000 відстаней	≤ 2500	1920
ТС-4: Відображення історії дій користувачів з фільтрацією по подіях	≤ 2000	2280

Аналіз отриманих результатів показав, що система загалом справляється з навантаженням, хоча при складних запитах, які охоплюють великі обсяги логів або маршрутних комбінацій, спостерігається незначне перевищення допустимих меж. Це вказує на доцільність оптимізації окремих запитів або переходу на продуктивніше сховище у разі масштабування.

Проведене тестування з великою кількістю записів підтвердило загальну стабільність функціонування системи при високих обсягах даних, однак у деяких випадках було зафіксовано перевищення допустимого часу відповіді. Зокрема, при виконанні запитів на формування розширених звітів, таких як побудова аналітики за період, система демонструвала затримки понад 3 секунди (зокрема, ТС-3 – 3560 мс), що вказувало на наявність вузьких місць у продуктивності. Подібна ситуація спостерігалась і під час фільтрації у великому журналі подій. Основною причиною виявилася відсутність оптимальних індексацій у таблицях, які найчастіше зазнають навантаження.

Для підвищення швидкодії були внесені зміни до схеми бази даних та логіки обробки запитів:

– запроваджено індексацію поля EventDate у таблиці логів подій для прискорення вибірки за періодом;

– створено складений індекс за полями SimulationId і FromCity у таблиці Distance, що суттєво зменшив витрати часу на агрегацію даних при побудові оптимального маршруту;

– переглянуто SQL-запити з метою зменшення кількості вкладених операторів JOIN, що дозволило скоротити навантаження на СУБД при обробці запитів користувачів;

– реалізовано посторінкове виведення великих масивів даних у формах із візуалізацією, що покращило загальну чуйність інтерфейсу користувача навіть при роботі з десятками тисяч записів.

Окрім того, було оновлено конфігурацію SQL Server, зокрема активовано кешування планів запитів, що також сприяло зменшенню часу відповіді при повторному виконанні схожих запитів. Проведені дії дали змогу суттєво

покращити продуктивність системи без зміни апаратної конфігурації, що є особливо важливим для розгортання у середовищах із обмеженими ресурсами.

Повторне тестування після оптимізації дало позитивні результати, що наведено в табл. 2.11.

Таблиця 2.11

Результати повторного тестування після оптимізації

Тест-кейс	Очікуваний час відгуку (мс)	Отриманий час відгуку (мс)
ТС-1: Пошук міста серед 5 000 записів	≤ 2000	620
ТС-2: Побудова симуляції для 50+ міст із 2500 маршрутів	≤ 3000	1320
ТС-3: Побудова звіту за період (оптимізовано)	≤ 3000	1290
ТС-4: Фільтрація подій у журналі (оптимізовано)	≤ 2000	1170

Отже, можна зазначити, що система демонструє стійку роботу навіть при обробці великих обсягів інформації, за умови належного структурування та оптимізації бази даних. Внесені технічні зміни дозволили зменшити час відповіді більш ніж удвічі, що створює передумови для комфортної роботи користувачів у режимі інтенсивної взаємодії з системою.

2.8 Інструкція використання системи

Успішне впровадження інформаційної системи для оптимізації маршрутів транспортування будівельних матеріалів передбачає не лише її технічну реалізацію, а й забезпечення зручності взаємодії кінцевого користувача з усіма функціональними модулями. З цією метою розроблено набір форм, що забезпечують покроковий супровід у процесі роботи з системою – від моменту авторизації до здійснення симуляцій і аналізу результатів. Для полегшення засвоєння функціоналу користувачеві надається інструкція, що описує основні дії в межах типових сценаріїв.

На рис. 2.32 зображено стартове вікно системи, яке реалізує процес авторизації користувача.

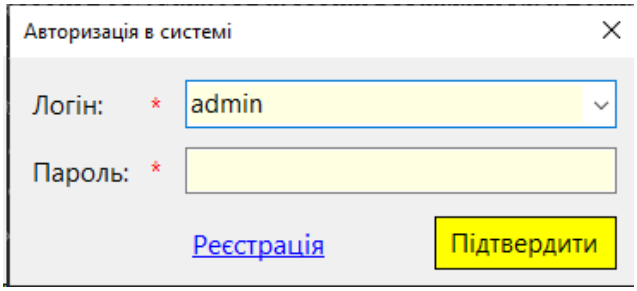


Рисунок 2.32 – Форма авторизації користувача

Після запуску програми користувачеві необхідно обрати своє ім’я зі списку, що містить зареєстровані облікові записи. У відповідне текстове поле слід ввести пароль, призначений під час реєстрації або отриманий від адміністратора системи. У разі правильного введення облікових даних кнопка «Підтвердити» активує вхід до системи та відкриває доступ до головного вікна програми відповідно до наданих прав. Якщо облікового запису ще не існує, користувач може скористатися посиланням «Реєстрація» для створення нового профілю.

Після натискання на посилання «Реєстрація» відкривається спеціальна форма, призначена для створення нового облікового запису (рис. 2.33).

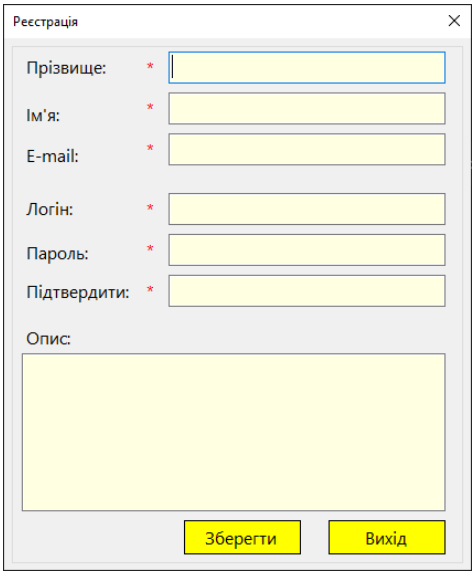


Рисунок 2.33 – Форма реєстрації нового користувача

Для успішного створення облікового запису користувач повинен послідовно заповнити всі обов’язкові поля форми. Зокрема, вказується прізвище та ім’я, що є ідентифікаційною інформацією в системі. У полі «E-mail» вводиться електронна адреса, яка може бути використана для внутрішніх повідомлень або відновлення доступу. У полях «Логін» та «Пароль» слід ввести бажані облікові дані, після чого необхідно підтвердити пароль у відповідному полі для уникнення помилок введення. За потреби можна додати описову інформацію у вільному текстовому полі «Опис». Натискання кнопки «Зберегти» ініціює перевірку валідності введених даних і, у разі успіху, реєструє нового користувача в системі. Кнопка «Вихід» дозволяє скасувати реєстрацію та повернутися до попереднього вікна без збереження введеної інформації.

Після авторизації користувач отримує доступ до головного функціоналу системи, зокрема до модуля управління населеними пунктами, що є базовим етапом перед формуванням симуляцій маршрутів. Інтерфейс форми для роботи з містами наведено на рис. 2.34.

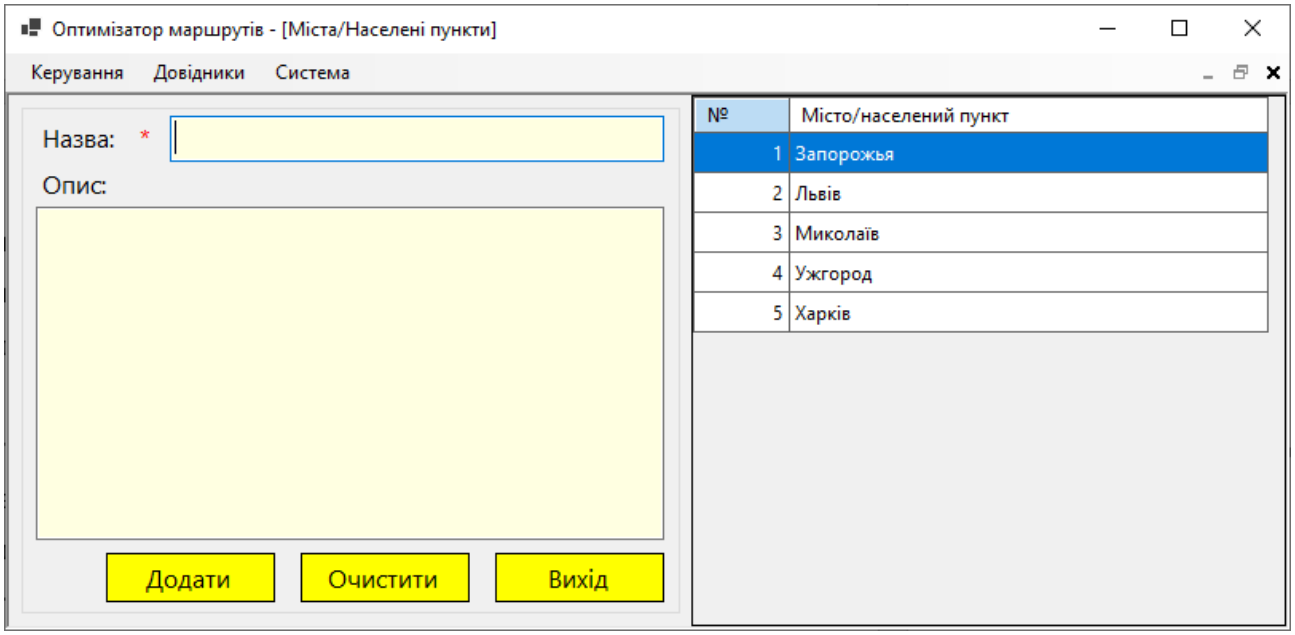


Рисунок 2.34 – Форма управління населеними пунктами

У разі потреби редагування вже наявного населеного пункту користувач має можливість відкрити спеціальну форму для оновлення або видалення

відповідного запису. На рис. 2.35 показано вікно редагування даних про місто, яке забезпечує інтуїтивно зрозумілий інтерфейс для роботи з конкретним елементом довідника.

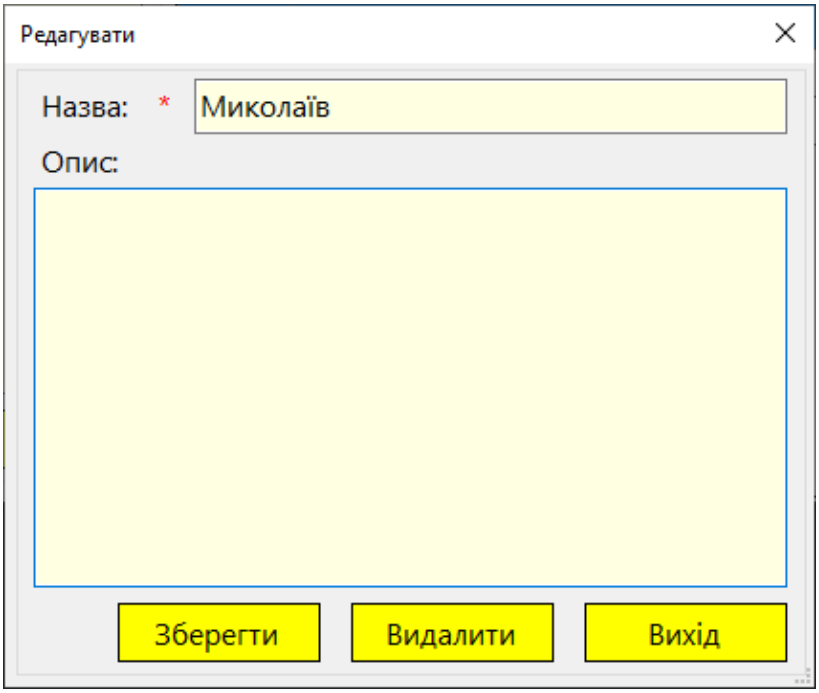


Рисунок 2.35 – Форма редагування населеного пункту

У даному вікні користувач може змінити назву міста, а також за потреби оновити описову інформацію. Поле з назвою є обов’язковим і вимагає заповнення перед збереженням. Кнопка «Зберегти» призначена для підтвердження внесених змін, після чого інформація оновлюється у базі даних. Якщо запис більше не є актуальним, передбачена функція його видалення через кнопку «Видалити». Кнопка «Вихід» дозволяє закрити форму без внесення змін. Такий підхід забезпечує повноцінне управління довідником населених пунктів у межах системи.

Одним із центральних етапів взаємодії з системою є формування симуляції доставки, яка дозволяє змоделювати можливі маршрути транспортування між заданими містами та побудувати оптимальний шлях на основі обчислених відстаней. Інтерфейс відповідної форми представлено на рис. 2.36.

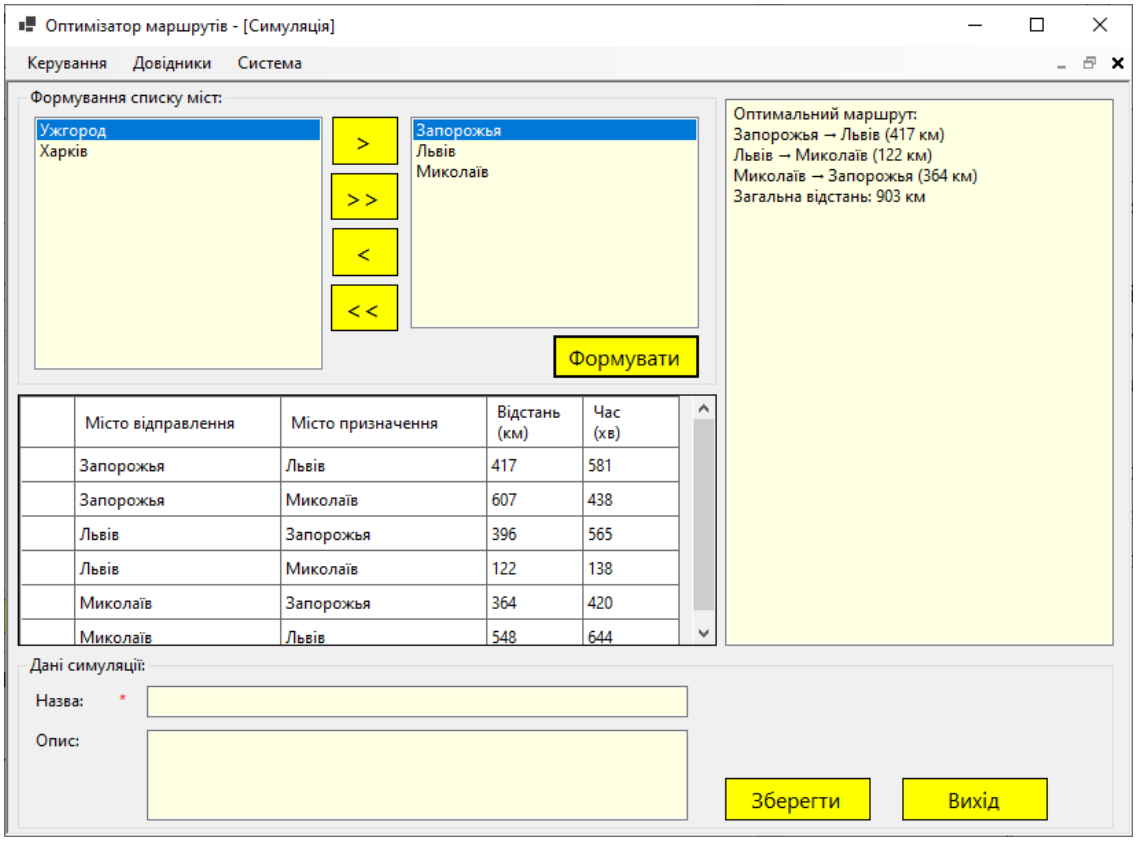


Рисунок 2.36 – Форма створення та аналізу симуляції маршрутів

На екрані користувач обирає з доступного переліку населених пунктів ті, що беруть участь у симуляції. За допомогою кнопок переміщення формується список вибраних міст, який надалі використовується для генерації всіх можливих пар відправлення та призначення. Натискання кнопки «Формувати» ініціює розрахунок матриці відстаней між кожною парою міст, після чого система застосовує алгоритм оптимізації маршруту з метою мінімізації сумарного шляху.

Результати симуляції відображаються у вигляді таблиці з параметрами: місто відправлення, місто призначення, відстань (у кілометрах) і приблизний час у дорозі (у хвилинах). У правій частині форми виводиться оптимальний маршрут з поетапним описом усіх переходів та загальною довжиною шляху. У нижньому блоці користувач має змогу вказати назву та опис симуляції, після чого зберегти її в систему для подальшого аналізу або повторного використання.

Користувачеві надається можливість переглядати раніше створені симуляції, аналізувати збережені маршрути та результати оптимізації без потреби

у повторному розрахунку. Такий функціонал реалізовано у спеціальній формі завантаження симуляцій, зображеній на рис. 2.37.

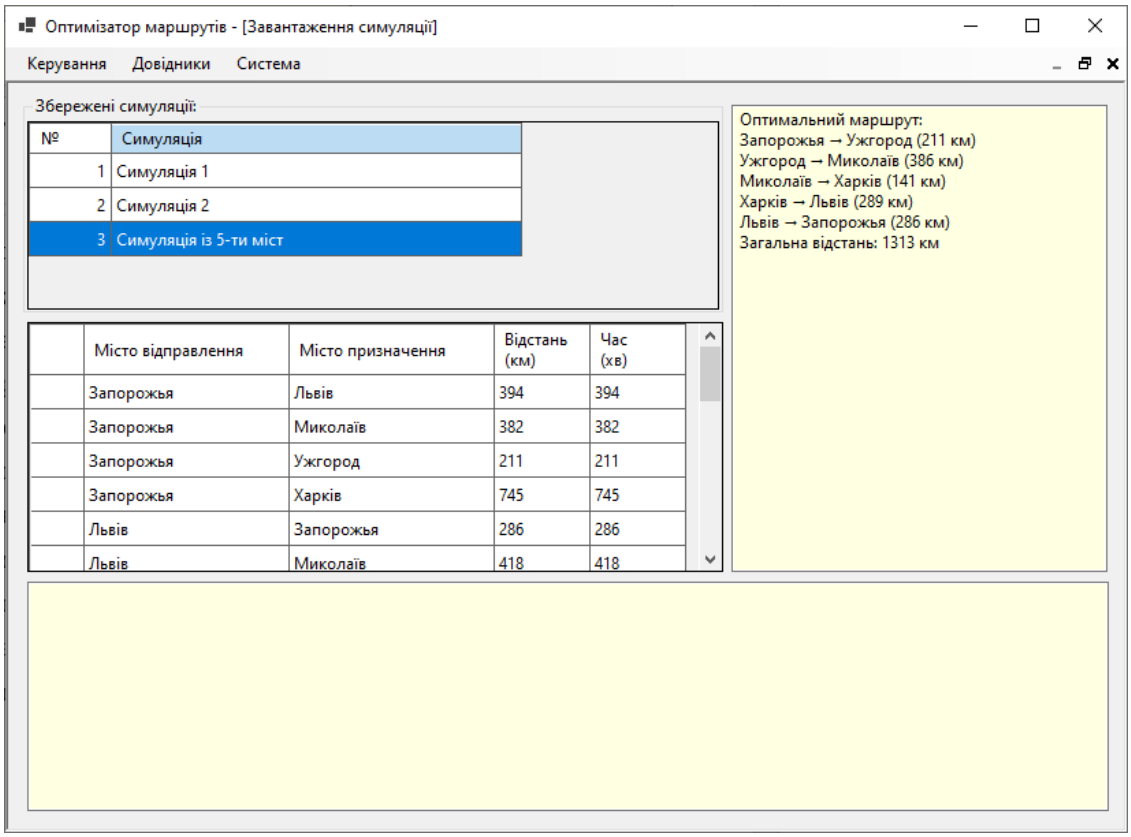


Рисунок 2.37 – Форма завантаження збережених симуляцій

У верхній частині вікна представлено таблицю з переліком усіх наявних симуляцій, що були збережені користувачем раніше. При виборі одного з записів система автоматично завантажує відповідні дані, які відображаються у двох інформаційних блоках. Зліва розташована таблиця маршрутів, що містить пари міст, між якими здійснювався розрахунок, а також значення відстаней і орієнтовного часу в дорозі. Праворуч відображається оптимальний маршрут, побудований під час первинної симуляції, із зазначенням послідовності переходів і загальної протяжності шляху.

Ця форма дозволяє зручно повертатися до попередніх результатів, проводити порівняльний аналіз альтернативних сценаріїв доставки та здійснювати візуальну перевірку точності обчислень. Така реалізація є важливою для підтримки прийняття обґрунтованих рішень у логістичних операціях.

Окремі дії в інформаційній системі є доступними лише користувачам з адміністративними повноваженнями. Серед них – управління обліковими записами інших учасників системи. На рис. 2.38 наведено інтерфейс керування користувачами, який доступний лише адміністраторам.

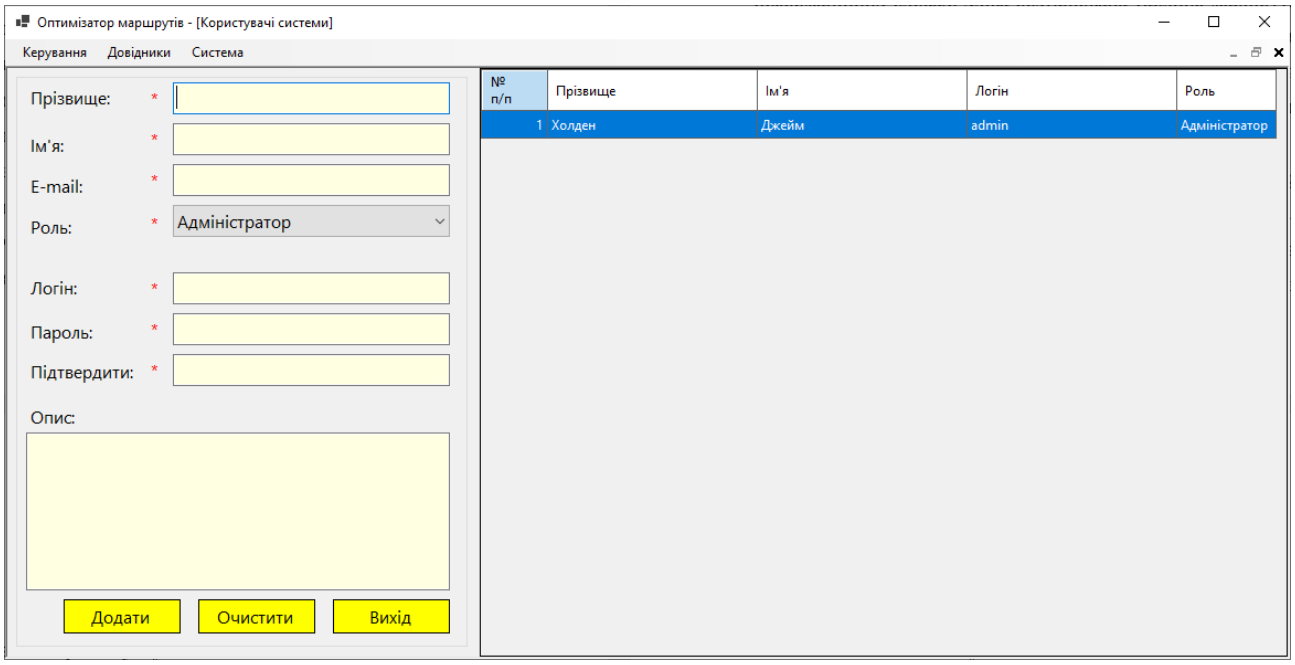


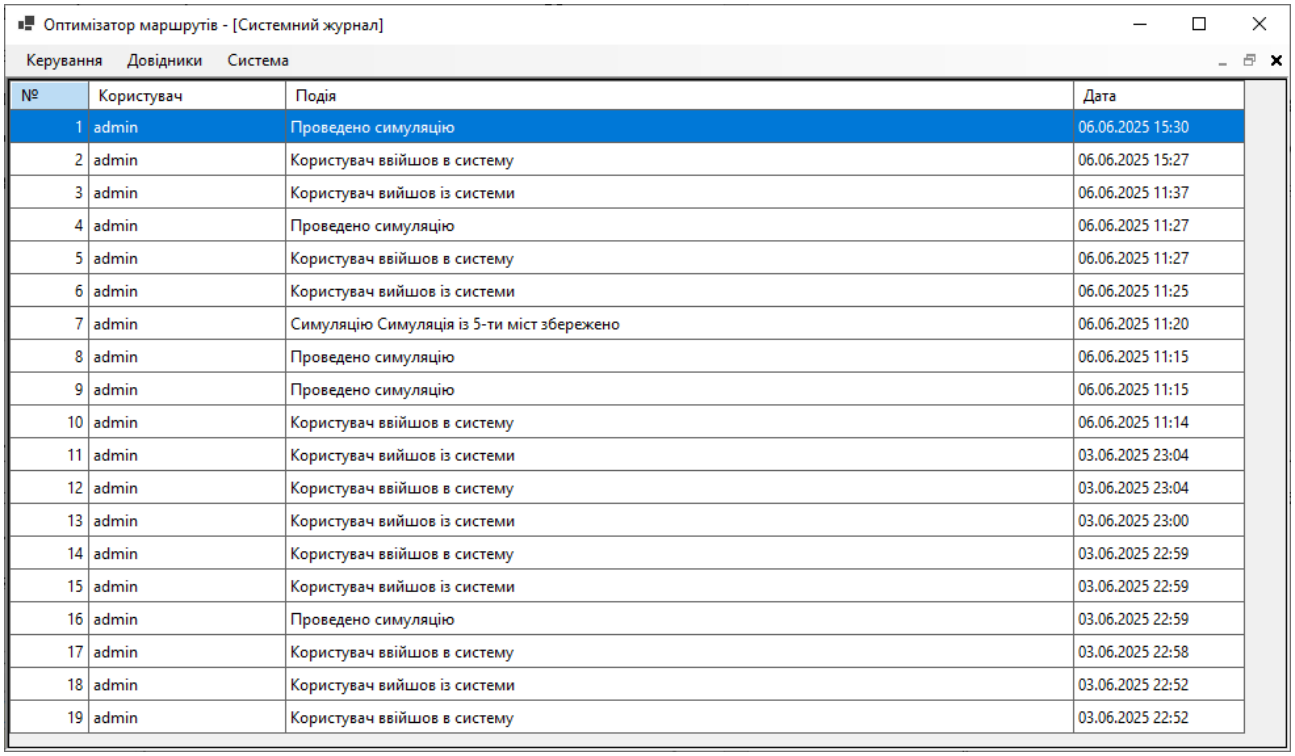
Рисунок 2.38 – Форма керування обліковими записами

У лівій частині форми розміщені поля для введення нових облікових даних, що включають прізвище, ім’я, електронну адресу, логін, пароль і підтвердження пароля. Роль користувача встановлюється за допомогою випадаючого списку, при цьому значення "Адміністратор" зазвичай зарезервоване для обмеженого кола осіб. Описове поле дозволяє ввести додаткову інформацію щодо створюваного запису.

У правій частині інтерфейсу відображається список зареєстрованих у системі користувачів, включаючи їхні прізвища, імена, логіни та ролі. Для виконання дій передбачено кнопки: «Додати» – створює новий запис у базі даних, «Очистити» – скидає вміст усіх полів введення, «Вихід» – закриває форму без збереження змін.

Такий функціонал надає адміністраторам можливість централізовано контролювати доступ до системи, забезпечуючи безпеку та структурованість при управлінні правами користувачів.

Ще одна функція, доступна адміністраторам, – це перегляд системного журналу, в якому фіксуються всі ключові події взаємодії користувачів із програмним забезпеченням. На рис. 2.39 представлено інтерфейс форми документування активностей.



№	Користувач	Подія	Дата
1	admin	Проведено симуляцію	06.06.2025 15:30
2	admin	Користувач ввійшов в систему	06.06.2025 15:27
3	admin	Користувач вийшов із системи	06.06.2025 11:37
4	admin	Проведено симуляцію	06.06.2025 11:27
5	admin	Користувач ввійшов в систему	06.06.2025 11:27
6	admin	Користувач вийшов із системи	06.06.2025 11:25
7	admin	Симуляцію Симуляція із 5-ти міст збережено	06.06.2025 11:20
8	admin	Проведено симуляцію	06.06.2025 11:15
9	admin	Проведено симуляцію	06.06.2025 11:15
10	admin	Користувач ввійшов в систему	06.06.2025 11:14
11	admin	Користувач вийшов із системи	03.06.2025 23:04
12	admin	Користувач ввійшов в систему	03.06.2025 23:04
13	admin	Користувач вийшов із системи	03.06.2025 23:00
14	admin	Користувач ввійшов в систему	03.06.2025 22:59
15	admin	Користувач вийшов із системи	03.06.2025 22:59
16	admin	Проведено симуляцію	03.06.2025 22:59
17	admin	Користувач ввійшов в систему	03.06.2025 22:58
18	admin	Користувач вийшов із системи	03.06.2025 22:52
19	admin	Користувач ввійшов в систему	03.06.2025 22:52

Рисунок 2.39 – Системний журнал подій у середовищі адміністратора

Інтерфейс складається з табличного представлення, яке містить номер запису, логін користувача, опис події та точну дату її здійснення. Всі записи упорядковані в хронологічному порядку, що забезпечує зручність аналізу історії дій. Серед зафіксованих подій – входи та виходи з системи, запуск симуляцій, збереження результатів тощо.

Завдяки такому модулю адміністратор може не лише здійснювати моніторинг активності, а й виявляти потенційно небажані або аномальні дії

користувачів. Це забезпечує прозорість роботи системи та підвищує рівень її захищеності.

2.8 Висновки до розділу 2

У рамках даного розділу було реалізовано повний цикл технічного проєктування та практичного втілення програмної системи для прийняття рішень у сфері транспортування будівельних матеріалів. Проведено детальне обґрунтування математичної моделі задачі комівояжера, з урахуванням її симетричних і несиметричних варіантів. Проаналізовано принципи функціонування бібліотеки Google OR-Tools, побудовано схему взаємодії її модулів і представлено покроковий алгоритм застосування інструментів бібліотеки для побудови оптимального маршруту.

Виконано проєктування ключових бізнес-процесів системи на основі діаграм послідовностей. На основі концепції трьохрівневої архітектури побудовано діаграми класів кожного рівня, що забезпечує чітке розмежування логіки, зберігання та представлення даних. Розроблено структуру бази даних: описано атрибути кожної таблиці та побудовано фізичну модель сховища для забезпечення цілісності даних. Реалізовано основні програмні методи, зокрема для додавання, оновлення, вибірки та видалення об’єктів, а також алгоритм формування оптимального маршруту з використанням OR-Tools.

Система була протестована в реальних умовах із використанням типових сценаріїв. Функціональне тестування підтвердило відповідність поведінки системи очікуваним результатам, а модульне — продемонструвало стабільність логіки кожного компонента. Проведено навантажувальне й об’ємне тестування, яке виявило критичні місця та дозволило провести оптимізацію, що в підсумку скоротила час відгуку системи більш ніж удвічі. Для користувачів розроблено повноцінну інструкцію з використання, яка охоплює всі основні дії: від авторизації до перегляду системного журналу.

ВИСНОВКИ

У рамках виконання кваліфікаційної роботи було розроблено програмне забезпечення для підтримки прийняття рішень у процесах транспортування будівельних матеріалів. Проведено аналіз логістичних процесів, що характерні для будівельної галузі, із виокремленням типових транспортних задач, серед яких особливу увагу приділено задачі комівояжера та варіантам її практичного застосування. Побудована математична модель на основі симетричного варіанту TSP стала основою для реалізації обчислювального модуля оптимізації з використанням бібліотеки Google OR-Tools.

Порівняльний огляд сучасних програмних засобів для транспортного планування дозволив виявити їхні переваги та обмеження. На основі цього сформульовано вимоги до власної розробки, з акцентом на простоту, доступність і адаптивність. Реалізована система підтримує формування списків міст, генерацію симуляцій між усіма парами, побудову оптимального маршруту за заданими параметрами, збереження та повторне завантаження результатів, що значно полегшує аналітичну роботу диспетчерського персоналу.

У межах архітектурного проєктування реалізовано трьохрівневу структуру програмного забезпечення, яка забезпечує розмежування відповідальностей між рівнями інтерфейсу, логіки та доступу до бази даних. Всі компоненти реалізовано на мові програмування C# із використанням середовища WinForms та SQLite. Описано структуру бази даних, включно з таблицями користувачів, подій, симуляцій, відстаней та міст. Побудовано фізичну модель сховища даних, що лягла в основу механізмів взаємодії між компонентами.

Здійснено функціональне, модульне, навантажувальне та об’ємне тестування системи. В результаті функціонального тестування було підтверджено коректну роботу всіх основних сценаріїв, зокрема авторизації, реєстрації, створення та моделювання симуляцій, а також управління обліковими записами. Модульне тестування, проведене у середовищі Visual Studio із

використанням MSTest, показало стовідсотковий успіх усіх одиничних перевірок. За результатами навантажувального тестування визначено граничні значення часу відгуку, при яких система зберігає працездатність. Додаткове об’ємне тестування продемонструвало здатність обробляти великі масиви даних, що стало можливим завдяки проведеним оптимізаціям структури бази даних.

Розроблене програмне забезпечення може бути застосоване на підприємствах будівельної галузі, які мають потребу в плануванні маршрутів доставки ресурсів. Завдяки відкритій архітектурі та модульному підходу система може бути доопрацьована для роботи з реальними геоданими, інтеграції з картографічними сервісами, підключення GPS-навігації або розширення обмежень задач маршрутизації.

Отримані результати свідчать про практичну придатність реалізованої системи до використання у виробничому середовищі, а також про можливість її подальшого вдосконалення з метою розширення функціональних можливостей у суміжних напрямах логістичної автоматизації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кваліфікаційна робота бакалавра [Електронний ресурс] : методичні рекомендації для здобувачів ступеня бакалавра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / уклад.: Т. А. Желдак, Т. В. Хом'як, А. В. Малієнко ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2025. – 32 с. url: https://ir.nmu.org.ua/handle/123456789/170863

1. Bapat H., Sarkar D., Gujar R. Selection of sustainable materials for energy savings of infrastructure-transportation project in Ahmedabad, India using BIM and FCM. Journal of Construction in Developing Countries. 2018. Vol. 26, No 2, pp. 135-161.

2. Verma A., Adichwal N., Sharma P. Implementation and Scope of Business Intelligence and Oracle Transportation Management System in Tata Steel Supply Chain. In Business Intelligence and Human Resource Management. 2022. Vol. 26, No 2, pp. 207-220.

3. Rodchenko V. (Digital technologies in logistics and supply chain management. FACTA UNIVERSITATIS-Economics and Organization. 2023. Vol. 20, No 3, pp. 191-203.

4. Ravi V., Jampani S. Blockchain Integration in SAP for Supply Chain Transparency. Integrated Journal for Research in Arts and Humanities. 2024. Vol. 4, No 6, pp. 10-55.

5. Feng B., Ye Q. Operations management of smart logistics: A literature review and future research. Frontiers of Engineering Management. 2021. Vol. 8, pp. 344-355.

6. Parmentier A. Learning to approximate industrial problems by operations research classic problems. Operations Research. 2022. Vol. 70, No 1, pp. 606-623.

7. Roberti R., Ruthmair M. Exact methods for the traveling salesman problem with drone. Transportation Science. 2021. Vol. 55, No 2, pp. 315-335.

8. Bogyrbayeva A., Meraliyev M., Mustakhov T., Dauletbayev B. Machine learning to solve vehicle routing problems: A survey. IEEE Transactions on Intelligent Transportation Systems. 2024. Vol. 25, No 6, pp. 754-772.

9. Chavhan S., Gupta D., Chidambaram R., Khanna A., Rodrigues J. (2020). A novel emergent intelligence technique for public transport vehicle allocation problem in a dynamic transportation system. IEEE Transactions on Intelligent Transportation Systems. 2020. Vol. 22, No 8, pp. 389-402.

10. TransCAD Reviews 2025: Details, Pricing, & Features. URL: https://www.g2.com/products/transcad/reviews (дата звернення 04.06.2025).

11. CEE 123 Transportation Systems III: Planning and Forecasting -- TransCAD Videos. URL: https://transcad.software.informer.com/5.0/ (дата звернення 04.06.2025).

12. TransCAD Transportation Planning Customer Reviews. URL: https://www.infotech.com/software-reviews/products/transcad-transportation-planning?c_id=48 (дата звернення 04.06.2025).

13. TransCAD Reviews in 2025. URL: https://sourceforge.net/software/product/TransCAD/ (дата звернення 04.06.2025).

14. TransCAD Pricing 2025. URL: https://www.trustradius.com/products/transcad/pricing (дата звернення 04.06.2025).

15. TransCAD: A Hierarchical Transformer for CAD Sequence. URL: https://openreview.net/forum?id=sRCEWsFNvM (дата звернення 04.06.2025).

16. Route4Me Reviews 2025: Details, Pricing, & Features. URL: https://www.g2.com/products/route4me/reviews (дата звернення 04.06.2025).

17. Route4Me Review: Pricing, Pros, Cons & Features | CompareCamp.com. URL: https://comparecamp.com/route4me-review-pricing-pros-cons-features/ (дата звернення 04.06.2025).

18. Route4Me Reviews - Pros & Cons, Ratings & more - GetApp. URL: https://www.getapp.com/transportation-logistics-software/a/route4me/reviews/ (дата звернення 04.06.2025).

19. Route4Me Software Reviews, Pros and Cons. URL: https://www.softwareadvice.com/fleet-management/route4me-profile/reviews/ (дата звернення 04.06.2025).

20. Route4Me Pricing, Alternatives & More 2025. URL: https://www.capterra.com/p/149633/Route4Me/ (дата звернення 04.06.2025).

21. Route4Me Reviews 2025. Verified Reviews, Pros & Cons. URL: https://www.capterra.com/p/149633/Route4Me/reviews/ (дата звернення 04.06.2025).

22. Read Customer Service Reviews of optimoroute.com. URL: https://www.trustpilot.com/review/optimoroute.com (дата звернення 04.06.2025).

23. OptimoRoute Reviews, Pricing & Ratings | GetApp NZ 2025. URL: https://www.getapp.co.nz/software/107281/optimoroute (дата звернення 04.06.2025).

24. OptimoRoute Reviews 2025: Details, Pricing, & Features. URL: https://www.g2.com/products/optimoroute/reviews (дата звернення 04.06.2025).

Хом'як, Т. В., Шевченко, Ю. О., & Гаранжа, Д. М. ПРОГРАМУВАННЯ ТА АЛГОРИТМІЧНІ МОВИ. https://ir.nmu.org.ua/handle/123456789/173373

25. OptimoRoute Reviews 2025. Verified Reviews, Pros & Cons. URL: https://www.capterra.com/p/161579/OptimoRoute/reviews/ (дата звернення 04.06.2025).

26. OptimoRoute Software Reviews, Pros and Cons. URL: https://www.softwareadvice.com/fleet-management/optimoroute-profile/reviews/ (дата звернення 04.06.2025).

27. OptimoRoute Pricing, Alternatives & More 2025. URL: https://www.capterra.com/p/161579/OptimoRoute/ (дата звернення 04.06.2025).

28. Томашевський О. М. Інформаційні технології та моделювання бізнес-процесів : навч. посіб. / О. М. Томашевський, Г. Г. Цигелик, М. Б. Вітер, В. І. Дудук. – К. : Центр учбової літератури, 2012. 296 с.

29. Villalobos, K., Ramirez-Duran, V. J., Diez, B., Blanco, J. M., Goni, A., & Illarramendi, A. A three level hierarchical architecture for an efficient storage of industry 4.0 data. Computers in Industry, 2020. – 121p.

30. Мінєєв, О. С., & Шевченко, Ю. О. (2026). Аналіз програмного забезпечення: методичні рекомендації до виконання практичних робіт для здобувачів ступеня бакалавра галузі знань 12 (F) Інформаційні технології. https://ir.nmu.org.ua/handle/123456789/173326

31. Коряшкіна, Л. С., Алєксєєв, О. М., & Гаранжа, Д. М. (2025). Навчальна практика з обчислень: методичні рекомендації для здобувачів ступеня бакалавра освітньо-професійної програми «Системний аналіз» спеціальності 124 Системний аналіз. https://ir.nmu.org.ua/handle/123456789/173197

32. Хом'як, Т. В., Шевченко, Ю. О., & Гаранжа, Д. М. (2026). Програмування та алгоритмічні мови: методичні рекомендації до виконання лабораторних робіт для здобувачів ступеня бакалавра ОПП «Системний аналіз» зі спеціальності 124 Системний аналіз Ч 1. https://ir.nmu.org.ua/handle/123456789/173373

33. Коряшкіна, Л. С., Станіна, О. Д., & Шевченко, Ю. О. (2024). Практикум з диференційних рівнянь. https://ir.nmu.org.ua/handle/123456789/167658

34. Виробнича практика [Електронний ресурс] : методичні рекомендації для здобувачів ступеня бакалавра освітньо-професійної програми «Системний аналіз» спеціальності 124 Системний аналіз / уклад.: Т.А. Желдак, Л.С. Коряшкіна, С.А. Ус ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2025. – 25 с. https://ir.nmu.org.ua/handle/123456789/173193

35. Передатестаційна практика [Електронний ресурс] : методичні рекомендації для здобувачів ступеня бакалавра освітньо-професійної програми «Системний аналіз» спеціальності 124 Системний аналіз / уклад.: Т.А. Желдак, А.В. Малієнко, О.Д. Станіна ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2025. – 24 с. https://ir.nmu.org.ua/handle/123456789/173256

Додаток А. Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Найменування	Кількість аркушів	Примітки
1				
2		Документація		
3				
4	САУ.КР.25.47.ПЗ	Пояснювальна записка	90	Формат А4
5				
6	САУ.КР.25.47.ДМ	Демонстраційний матеріал	15	Презентація на CD-R
7				
8	САУ.КР.25.47.КР	Копія роботи	1	Диск CD-R
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				

					САУ.КР.25.53.ДА.ПЗ.
Змін.	Аркуш	№ докум.	Підпис	Дата	

Розроб.	Линник В.Ю.			Матеріали кваліфікаційної роботи	Літ.	Аркуш	Аркушів
К. розд.	Желдак Т. А.						
Керівн.	Желдак Т. А.				**НТУ «ДП», 12; 124-21-1**		
Н.контр.	Хом'як Т. В.						
Зав. каф.	Желдак Т. А.						

Додаток В. Відгук на кваліфікаційну роботу

Відгук
на кваліфікаційну роботу бакалавра
здобувача вищої освіти групи *124 – 21 – 1*
спеціальності *124 Системний аналіз*
Линника Влада Юрійовича

Тема кваліфікаційної роботи: Прийняття рішень в процесах транспортування будівельних матеріалів в умовах підприємства ТОВ "Будін-Торг"

Обсяг кваліфікаційної роботи 98 стор.

Мета кваліфікаційної роботи: підвищення ефективності товарної логістики підприємства шляхом розробки та реалізації програмного модуля підтримки прийняття рішень у процесах транспортування будівельних матеріалів.

Актуальність теми полягає у необхідності використання математичних моделей, що дозволяють мінімізувати витрати на перевезення, скорочувати час доставки та оптимізувати використання транспортних ресурсів

Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра спеціальності 124 Системний аналіз, оскільки в роботі використані системно-аналітичний підхід, математичне моделювання задач маршрутизації, використання бібліотеки Google OR-Tools для реалізації алгоритмів оптимізації, принципи об’єктно-орієнтованого програмування мовою C# у середовищі WinForms. Виконані в кваліфікаційній роботі завдання відповідають вимогам ступеня бакалавра.

Практичне значення результатів кваліфікаційної роботи полягає в розробці прикладного програмного модуля для оптимізації маршрутів транспортування будівельних матеріалів, який уперше адаптовано до потреб малого підприємства з урахуванням обмежених ресурсів і реальних виробничих умов.

Висновки підтверджують можливість використання результатів роботи в практиці широкого кола підприємств, які самостійно займаються збутом готової продукції.

Оформлення пояснювальної записки та демонстраційного матеріалу до неї виконано згідно з вимогами. Роботу виконано самостійно, відповідно до завдання та у повному обсязі .

У роботі **не відзначено недоліків**, які можуть суттєво вплинути на її оцінювання.

Кваліфікаційна робота в цілому заслуговує оцінки: «**відмінно**» (95 балів).

З урахуванням висловлених зауважень автор заслуговує присвоєння кваліфікації «бакалавр з системного аналізу».

Керівник кваліфікаційної роботи,
К.т.н., доц., зав.каф. САУ _____ / Т.А. Желдак